KÁTAI ZOLTÁN

GRÁFELMÉLETI ALGORITMUSOK

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MŰSZAKI ÉS HUMÁNTUDOMÁNYOK KAR
MATEMATIKA–INFORMATIKA TANSZÉK

A kiadvány megjelenését a Sapientia Alapítvány támogatta.

KÁTAI ZOLTÁN

GRÁFELMÉLETI ALGORITMUSOK

Scientia Kiadó
Kolozsvár·2008

A kiadvány megjelenését támogatta:

SAPIENTIA ALAPÍTVÁNY

SZÜLŐFÖLD ALAP

Lektor:
Ionescu Klára (Kolozsvár)

Sorozatborító:
Miklósi Dénes

Descrierea CIP a Bibliotecii Naționale a României
KÁTAI ZOLTÁN
Gráfelméleti algoritmusok / Kátai Zoltán. – Cluj-Napoca: Scientia, 2008.
Bibliogr.
ISBN 978-973-7953-95-7

TARTALOM

CONTENTS

CUPRINS

ELŐSZÓ

A Gráfelméleti algoritmusok című könyvet (egyetemi jegyzetet) bevezetőnek szántuk a gráfelmélet témakörébe. Ahogy a címe is utal rá, a könyv egyik jellegzetessége, hogy súlypontját inkább az informatika, mint a matematika irányába toltuk el. Ezzel összhangban nagy hangsúlyt fektettünk a gráfelméleti algoritmusok gondos bemutatására. Másfelől a fogalmak és algoritmusok matematikai hátterét sem hanyagoltuk el. A legtöbb tételt nemcsak kijelentjük, hanem be is bizonyítjuk (vagy legalább belátjuk a bizonyítás lényegét).

A 2–11. fejezetek inkább informatika-orientáltak. Olyan témákkal foglalkoznak, mint: gráfok bejárási algoritmusai és ezek alkalmazásai, minimális feszítőfát meghatározó algoritmusok, optimális utakat kereső algoritmusok, maximális folyam-problémát megoldó algoritmusok és ezek alkalmazásai stb. A 12–14. fejezetek hangsúlyosabban matematikai jellegűek: Euler- és Hamilton-gráfok, gráfok síkbarajzolhatósága, gráfok színezési problémái stb. Kifejezetten törekedtünk arra, hogy a magyar gráfelméleti iskola minél több eredményét beleszőjük a könyv anyagába. (Erdős, Dirac, Gallai, Rédei, Kőnig, Lovász, Pósa, Turán, „magyar módszer” stb.)

A könyv első két függeléke az algoritmusok bonyolultsága és a programozási technikák területére kalauzolja el az olvasót. Az itt található anyagok kitűnő háttér-információval szolgálhatnak számos fejezet alaposabb megértéséhez. A harmadik függelék egy speciális gráftípust (d_gráfok) mutat be, amely lehetővé teszi számos dinamikus programozásos feladat egységes tanulmányozását. A d_gráfok fogalma a szerző saját kutatási eredményei közé tartozik. A negyedik függelék a magyar gráfelméleti iskola kiemelkedő alakjairól tartalmaz rövid ismertetéseket, az utolsóban pedig néhány gráfelméleti fogalom román és angol megfelelőit közöljük.

A könyv további jellegzetessége a felépítése. Először a gráfok két alapvető bejárási algoritmusát mutatjuk be (3. fejezet), majd ezt követően számos algoritmust úgy tárgyalunk, mint amelyek ezeknek válfajai, alkalmazásai (4., 5.2., 6.1., 8., 9., 10. fejezetek). Ez a megközelítés segíthet egyes algoritmusok jobb megértésében. Egy másik jellegzetesség az, hogy

a könyv felépítése is gráfra emlékeztet. E gráf pontjainak az egyes témaköröket kifejtő fejezetek (alfejezetek), az éleknek pedig az ezek közötti utalások tekinthetők (a 6.1., 6.2., 6.3., 6.4., 7. fejezetek csoportja a gráf egy klikkjeként is felfogható). A könyv egy további sajátossága számos fejezet logikai felépítésében rejlik: felvetünk egy gyakorlati problémát, lefektetjük a probléma megoldásához szükséges elméleti alapokat, felvázoljuk a megoldási stratégiát és közöljük a megoldási algoritmust. Mindezzel az a célunk, hogy kiemeljük a gráfalgoritmusok gyakorlati értékét. Konkrét alkalmazási területként foglalkozunk a kétpólusú alkatrészekből összerakott villamos hálózatok gráfelméleti vonatkozásaival (2.4–6., 13.1.1. alfejezetek).

A könyv informatikaközeli jellegével összhangban folyamatosan utalunk az egyes gráfalgoritmusok és az algoritmustervezési stratégiák (programozási technikák) közötti kapcsolatokra. Kiemelkedő e tekintetben a 6.4. alfejezet, amely a legrövidebb út algoritmusokat úgy mutatja be, mint egyazon dinamikus programozási stratégia különböző vetületeit.

Nagy hangsúlyt fektettünk a szemléletességre (ábrák, táblázatok, gyakorlati példák, szemléltetések stb.), hogy a könyv könnyen használható legyen önálló tanuláshoz is. Remélhetőleg mind az informatikatanárok, mind az informatikus diákok (középiskolások és egyetemi hallgatók) egyformán fogják a könyvet egyszerűnek és mélynek találni. Örülnénk, ha érdekesnek és gyakorlatiasnak is találnák. A könyv a matematikusok érdeklődésére is számot tart, akik gráfelméleti problémákat számítógép segítségével szeretnének megoldani. Egyes részek a villamosmérnököknek is érdekesek lehetnek.

A szerző

BEVEZETÉS

Hogyan jutunk el a gráf fogalmához? Sok, valós életben felmerülő feladat megoldását megkönnyíti az, ha a feladat lényegét valamilyen módon szemléltetni, ábrázolni tudjuk. Ha akár teljesen különböző természetű feladatok esetében is sikerül egy közös ábrázolási módot találni, akkor azt tapasztalhatjuk, hogy az első látásra különbözőnek tűnő feladatok meglepően sok mindenben hasonlítanak egymásra. A következőkben néhány ilyen, egymástól teljesen különböző problémához rendelünk azonos szemléltetési módot: pontokból és az azokat összekötő szakaszokból, esetenként irányított szakaszokból összeállított ábrázolást. Íme néhány példa:

- *Úthálózatok:* Egy terület úthálózata szemléletesen ábrázolható pontok és élek segítségével. A pontok a helységeket, az élek az őket összekötő útvonalakat jelentik. Egy város úthálózatát hasonlóképpen ábrázolhatjuk. Itt az útkereszteződések lehetnek a pontok, és az őket összekötő utak az élek. Ha irányított éleket használunk, akkor akár az utak irányítása is, például az egyirányú utak is, feltüntethetők.
- *Elektromos hálózatok:* Telepek, ellenállások, kodenzátorok, tekercsek és különböző fogyasztók összekapcsolásából származó hálózat is jól ábrázolható pontokkal és élekkel. A pontok az alkatrészek csatlakozásainak felelnek meg, míg az élek az alkatrészeket képviselik.
- *Vegyületek molekuláris modelljei:* Például a parafinmolekulákat, melyek n számú szénatomból és $2n + 2$ számú hidrogénatomból állnak, szintén ábrázolhatjuk így, ha az egyes atomokat pontokkal, a vegyértékkapcsolatokat pedig szakaszokkal jelöljük.
- *Társadalmi kapcsolatok:* Ha az egyéneket pontokkal, a köztük lévő kapcsolatokat pedig élekkel ábrázoljuk, akkor az emberek között fennálló legkülönbözőbb kapcsolatok is jól szemléltethetővé válnak.
- *Gazdasági tevékenységek:* Ha egy építkezési vállalat megbízatást kap bizonyos munkálat elvégzésére, akkor az egész munkálatot munkaszakaszokra osztja. Az egyes munkaszakaszok, valamint egymástól való függőségük jól szemléltethetők, ha minden

munkaszakaszt egy-egy irányított éllel ábrázolunk, a munkaszakaszok kezdetét és végét jelentik a pontok, a munkaszakaszok előírt sorrendjét pedig a megfelelő élek egymáshoz illesztésével mutatjuk be.

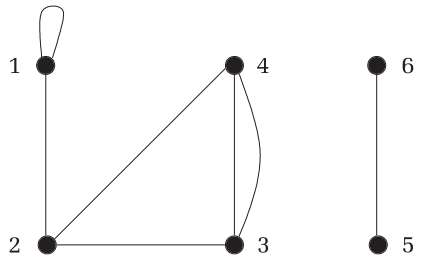
Lássunk ízelítőül néhány példát gráfokkal megoldható feladatokra is:

- *Feladat:* Legyen n, gazdasági szempontból fontos város, amelyek között korszerűsíteni szeretnénk az úthálózatot. Ha ismert a korszerűsítés költsége az úthálózat minden egyes szakaszára, számítsuk ki, hogy az 1. városból az n-be vezető melyik útvonal korszerűsítése a leggazdaságosabb.
  Megoldás: Ha a költségeket az útszakaszoknak megfelelő élek hosszúságaként fogjuk fel, akkor a kérdés a következő általános alakot ölti: Melyik az 1-ből n-be vezető *legrövidebb út*? Ez már egy gráfelméleti feladat.
- *Feladat:* Egy építkezési vállalat, miután egy munkálatot munkaszakaszokra bontott, a tapasztalat alapján megállapítja az egyes munkaszakaszok elvégzéséhez szükséges időt. Mekkora a legrövidebb idő, amely alatt az egész munkálat befejezhető?
  Megoldás: Ha az egyes munkaszakaszok kezdetét és végét pontok, míg a munkálat elvégzéséhez szükséges időtartamot a megfelelő hosszúságú élek jelentik, akkor az egész munkálatra felépíthető egy gráf. Ebben a gráfban a kérdés a következő általános alakot kapja: Melyik, a munkálat kezdetét jelentő és a munkálat végét jelentő két csomópont között a *leghosszabb út*?
- *Feladat:* n szakképesített munkást n különböző szerszámgépre kell beosztani. Mindegyik munkást felkészültsége, ügyessége és gyakorlata alapján a normához viszonyított százalékban kifejezett, különböző termelékenységi mutatóval oszthatjuk be az egyes szerszámgépekhez. Határozzuk meg az n munkás leggazdaságosabb beosztását az egyes gépekhez.
  Megoldás: Jelöljék a munkásokat és a gépeket pontok, a beosztási lehetőségeket a munkásokat képviselő pontoktól a gépeket képviselő pontokhoz vezető élek, amelyeknek hosszát a termelékenységi mutató adja. A probléma a következőképpen általánosítható: Melyik az így kapott gráfnak az az n éle, amelyek mindegyike különböző pontból indul ki és különböző pontba érkezik, összhosszúságuk pedig a legnagyobb? A gráfelmélet nyelvén: határozzuk meg a *maximális párosítást*.

1. FEJEZET

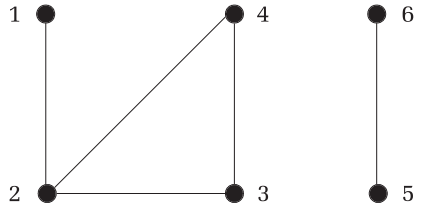
ALAPFOGALMAK

– Egy *gráf* egy rendezett pár, $G = (V, E)$, ahol V egy nem üres halmaz. V elemeit (csomó)pontoknak vagy csúcsoknak, E elemeit pedig éleknek nevezzük. Egy gráf pontjainak számát n-nel, éleinek számát pedig m-mel fogjuk jelölni.



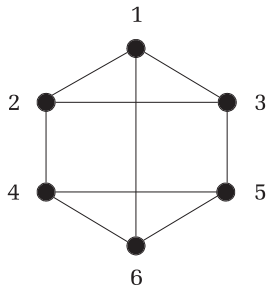
1.1. ábra. *Egy 6 pontú és 7 élű irányítatlan gráf.*

– Ha egy él végpontjai azonosak, akkor *hurokél*ről beszélünk. Az 1.1. ábrán az (1, 1) él hurokél.
– Ha két különböző nem hurokél végpontjai azonosak, akkor ezeket *párhuzamos* vagy *többszörös* éleknek nevezzük. Az 1.1. ábrán a 3-as és 4-es pontok között párhuzamos élek vannak.
– *Egyszerű gráf* az, amelyik nem tartalmaz sem hurokélt, sem többszörös élt. (Az egyszerű gráfok élei azonosíthatók a végpontjaik által.)



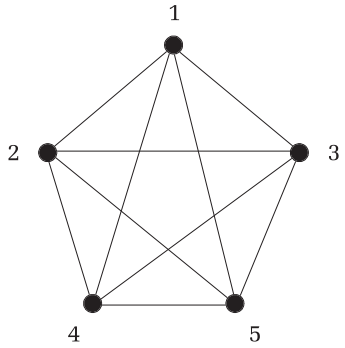
1.2. ábra. *Egy 6 pontú és 5 élű egyszerű irányítatlan gráf. V = {1, 2, 3, 4, 5, 6}, E = {(1, 2), (2, 4), (2, 3), (3, 4), (5, 6)}*

– Két pont akkor *szomszédos*, ha van él közöttük. Két él akkor szomszédos, ha valamelyik végpontjuk közös. Például az 1.2. ábrán az 1-es és 2-es pontok szomszédosak, de a 4-es és 5-ös pontok nem. Továbbá az (1, 2) és (2, 4) élek szomszédosak, de az (1, 2) és (3, 4) élek nem.
– Ha egy csomópont végpontja egy adott élnek, akkor azt mondjuk, hogy *illeszkedik* rá (és fordítva: az él illeszkedik az illető csomópontra). Például az 1.2. ábrán a 2-es pontra illeszkedő élek: (1, 2), (2, 3), (2, 4).
– *Izolált pont* az, amelyik nem illeszkedik egyetlen élre sem. Ha az 1.2. ábrán törölnénk az 1-es és 2-es pontok közötti élet, akkor az 1-es pont izolált maradna.
– Egy v pontra illeszkedő élek száma az illető pont *fokszám*át adja meg és $d(v)$-vel jelöljük. Egy gráf maximális fokszámát Δ-val, a minimálisat pedig δ-val fogjuk jelölni. Az 1.2. ábrán látható gráf esetén $\Delta = 3$ és $\delta = 1$, mert $d(2) = 3$ és $d(1) = d(5) = d(5) = 1$.
– Egy gráf *k-reguláris*, ha minden pontjának foka k.

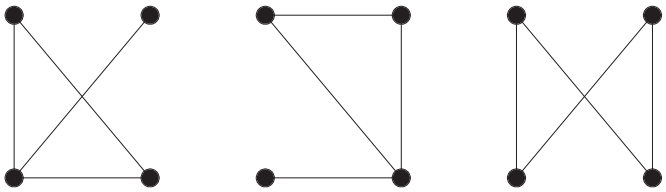


1.3. ábra. *3-reguláris gráf*

– Ha egy n pontú egyszerű gráf bármely két pontja szomszédos, akkor n pontú *teljes gráf*ról beszélünk és K_n-nel jelöljük. A gráf K_n éleinek száma: $n(n-1)/2$ (1.4. ábra).
– A $G(V, E)$ és a $G'(V', E')$ gráfok *izomorfak*, ha van olyan egy-egy értelmű megfeleltetés (bijekció) V és V' között, hogy G-ben pontosan akkor szomszédos két pont, ha G'-ben a nekik megfelelő pontok szomszédosak, és szomszédos pontpárok esetén ugyanannyi él fut közöttük (1.5. ábra).
– Egy $(v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k)$ sorozatot *élsorozat*nak vagy *sétá*nak nevezünk, ha e_i a v_{i-1}-et és v_i-t összekötő él. Ha $v_0 = v_k$,



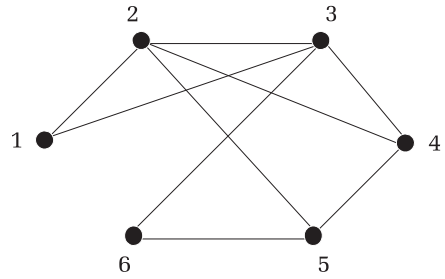
1.4. ábra. *5 pontú teljes gráf*



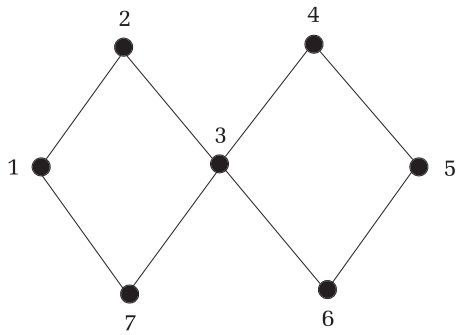
1.5. ábra. *Az első két gráf izomorf egymással, a harmadik viszont nem az velük*

az élsorozat *zárt*. Ha a csúcsok mind különbözők, akkor *út*ról beszélünk. A zárt utat *kör*nek is nevezzük. Egyszerű gráfban az utat $(v_0, v_1, v_2, \ldots, v_{k-1}, v_k)$-val írjuk le. Ha egy élsorozaton minden él különböző, akkor *vonal*ról (illetve zárt vonalról) beszélünk.

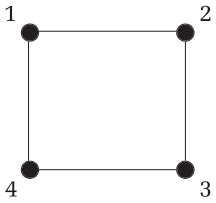
– A *Hamilton-út* a gráf minden pontját tartalmazza (egyszer és csakis egyszer). A *Hamilton-kör* a gráf minden pontját pontosan egyszer tartalmazza. Ha egy gráf tartalmaz Hamilton-kört, akkor *Hamilton-gráf*nak nevezzük.
– A *nyílt Euler-vonal* a gráf minden élét tartalmazza (egyszer és csakis egyszer). A *zárt Euler-vonal* kezdőpontja megegyezik a végpontjával és a gráf minden élét pontosan egyszer tartalmazza. Ha egy gráf tartalmaz zárt Euler-vonalat, akkor *Euler-gráf*nak nevezzük (1.6–1.9. ábra).
– Egy gráfból úgy kapjuk valamely *részgráf*ját, ha törlünk belőle éleket vagy pontokat a hozzájuk tartozó élekkel együtt. Ha csak éleket törlünk, akkor *feszítő*, ha csak pontokat törlünk, akkor *feszített* részgráfról beszélünk (1.10–1.13. ábra).



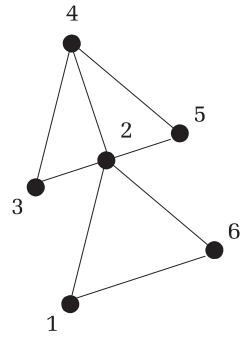
1.6. ábra. *Hamilton-gráf, amely nem Euler-gráf is. Hamilton-kör: (1, 2, 4, 5, 6, 3, 1), Hamilton-út: (1, 2, 3, 4, 5, 6), kör (vonal is): (1, 2, 4, 3, 1), zárt vonal (nem kör): (1, 2, 5, 6, 3, 2, 4, 3, 1)*



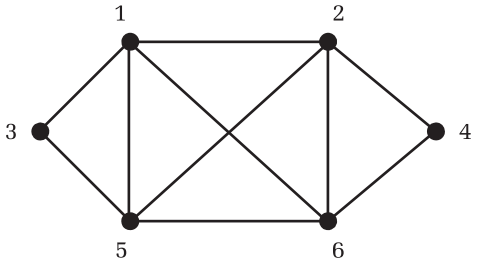
1.7. ábra. *Euler-gráf, amely nem Hamilton-gráf is. Zárt Euler-vonal (nem kör): (1, 2, 3, 6, 5, 4, 3, 7, 1), zárt vonal (kör is): (1, 2, 3, 7, 1), vonal (nem út): (1, 2, 3, 4, 5, 6, 3, 7)*



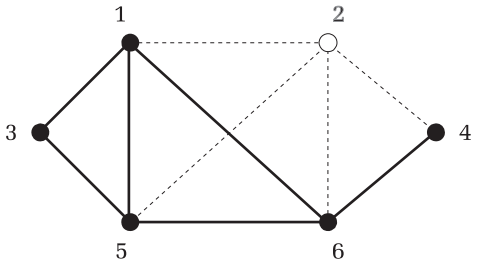
1.8. ábra. *Euler- és Hamilton-gráf*



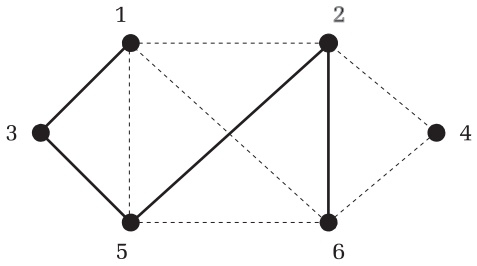
1.9. ábra. *Gráf, amely se nem Hamilton-gráf, se nem Euler-gráf. Zárt séta: (1, 2, 4, 3, 2, 5, 4, 2, 6, 1)*



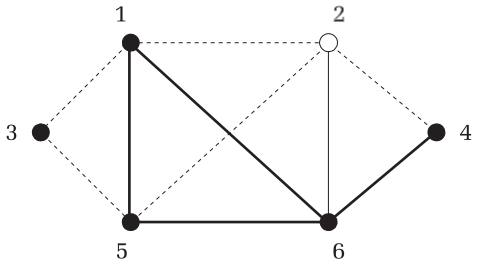
1.10. ábra. *Példagráf a részgráf fogalmának szemléltetéséhez*



1.11. ábra. *Feszítő részgráf (pontok törlésével nyerjük)*

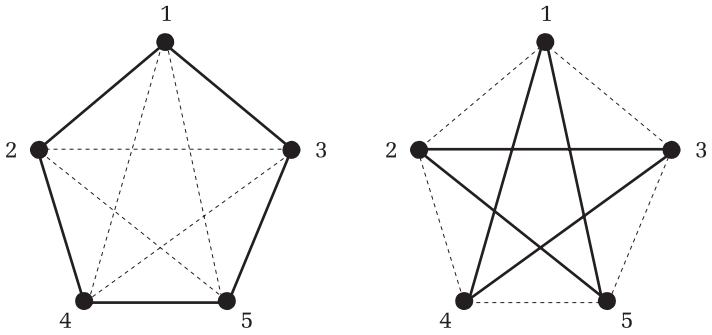


1.12. ábra. *Feszített részgráf (élek törlésével nyerjük)*



1.13. ábra. *Általános részgráf (pontok és élek törlésével nyerjük)*

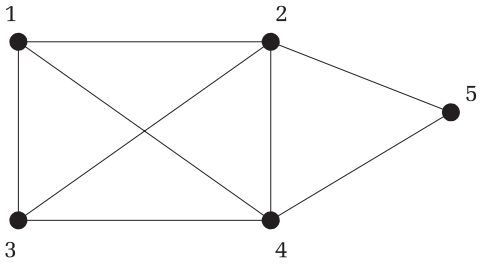
– A G gráf *komplementer gráf*jában azok a pontpárok vannak összekötve, amelyek G-ben nincsenek összekötve.



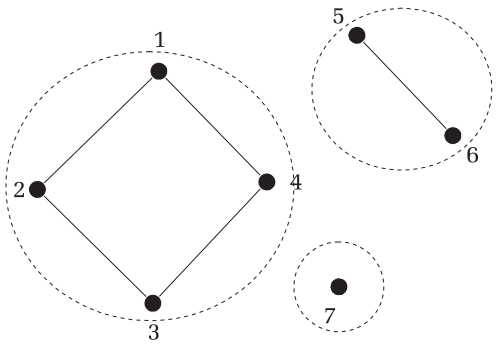
1.14. ábra. *Komplementer gráfok (A pontozott élek a hiányzó élek)*

– Egy gráf *összefüggő*, ha bármely két pontja között létezik út. Ha egy gráf nem összefüggő, akkor beszélhetünk az *összefüggő komponens*eiről. Egy gráf összefüggő komponensei a gráf azon

összefüggő *feszített* részgráfjai, amelyek *maximálisak* e tulajdonságra nézve.

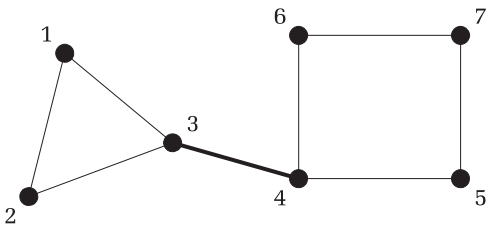


1.15. ábra. *Összefüggő gráf*

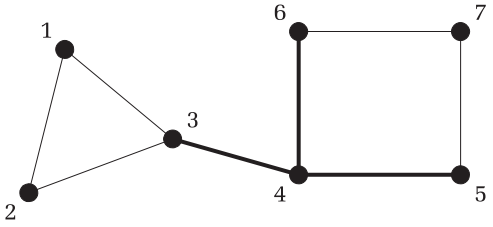


1.16. ábra. *Az ábrán látható gráfnak három összefüggő komponense van: (1, 2, 3, 4); (5, 6); (7)*

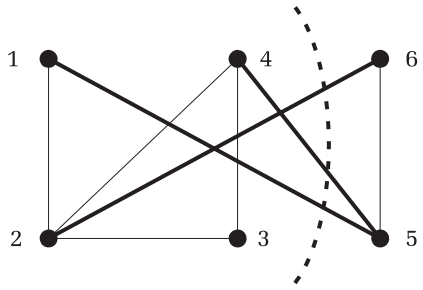
- Egy élet *elvágó él*nek nevezünk (*híd*), ha elhagyásával nő a gráf összefüggő komponenseinek száma. Egy pontot *elvágó pont*nak nevezünk, ha elhagyásával (a hozzátartozó élekkel együtt) nő a gráf összefüggő komponenseinek száma (1.17–1.18. ábra).
- Egy élhalmazt, amelynek törlésével nő a komponensek száma, *elvágó élhalmaz*nak nevezünk. Ha egy élhalmaz elvágó, de egyetlen valódi részhalmaza sem az, akkor *vágás*ról beszélünk (1.19. ábra).
- Az összefüggő körmentes gráfokat *fagráf*oknak (vagy *fák*nak) nevezzük (1.20. ábra).
- A körmentes gráfokat *erdő*nek nevezzük (1.21. ábra).
- A fák (erdők) egy fokszámú pontjait *level*eknek nevezzük.



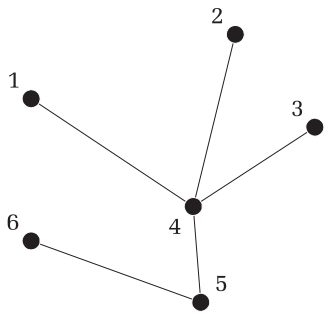
1.17. ábra. *Összefüggő gráf elvágó éle (a (3, 4) él)*



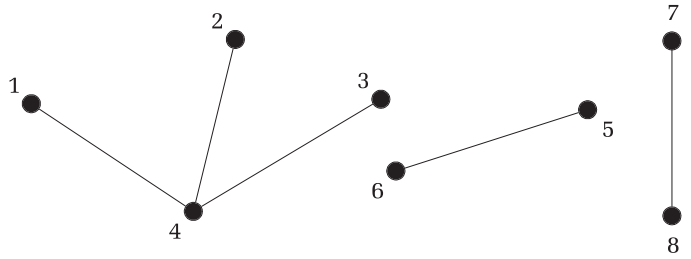
1.18. ábra. *Összefüggő gráf elvágó pontja (a 4-es pont)*



1.19. ábra. *Az {(1, 5), (2, 6), (4, 5)} elvágó élhalmaz vágást alkot*

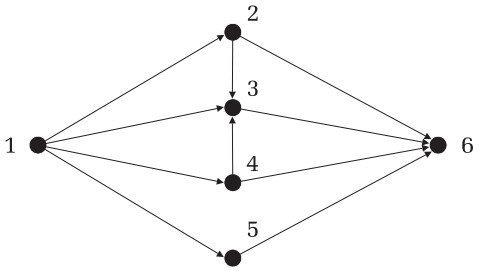


1.20. ábra. *6 pontú fa*



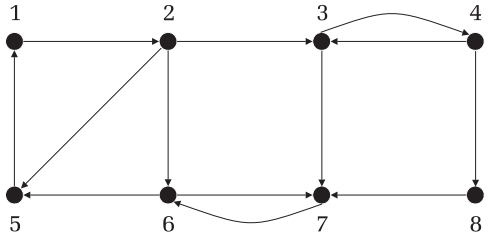
1.21. ábra. *Erdő, amely három összefüggő komponensből (fából) áll*

– Ha egy gráf éleit úgy definiáljuk mint *rendezett* pontpárokat, akkor *irányított gráf*fal van dolgunk. Az irányított éleknek van kezdő- és végpontjuk. Egy irányított gráf esetében beszélhetünk a pontok *be-fokszám*áról ($d_-(v)$), illetve *ki-fokszám*áról ($d_+(v)$). $d(v) = d_+(v) - d_-(v)$.
– Egy *forrás*nak csak ki-szomszédjai, egy *nyelő*nek pedig csak beszomszédjai vannak. Ha egy forrásból minden más ponthoz indul ki-él, akkor *szuperforrás*ról van szó. Ha egy nyelőbe minden más pontból érkezik be-él, akkor *szupernyelő*ről beszélünk.

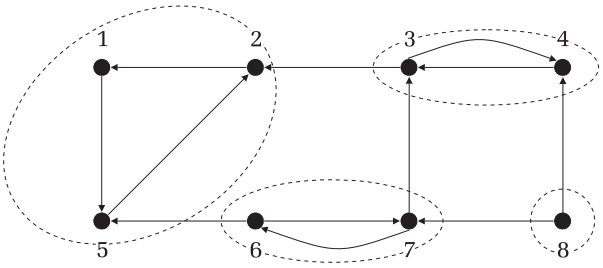


1.22. ábra. *Az ábrán látható irányított gráfban az 1-es pont forrás, a 6-os pedig nyelő*

– Egy irányított gráf akkor *erősen összefüggő*, ha bármely két pontja közt van oda-vissza irányított út (1.23–1.24. ábra).
– Ha az irányított teljes gráfot úgy definiáljuk, hogy bármely két pontja között van oda-, vagy vissza- vagy oda-vissza él, akkor $3^{C(n,2)}$ n pontú *irányított teljes gráf* létezik.



1.23. ábra. *Erősen összefüggő gráf*



1.24. ábra. *Az ábrán látható gráfnak négy erősen összefüggő komponense van: (1, 2, 5), (3, 4), (6, 7), (8)*

Néhány alaptétel

1.1. tétel. Minden gráfra igaz, hogy a fokszámok összege az élszám kétszerese. (Ebből következően a fokszámok összege páros szám.)

A tétel belátása: Gondolatban távolítsuk el a gráfból az összes élet. Az így létrejött gráf pusztán izolált pontokból áll, ezért fokszámösszegük nyilván nulla. Most tegyük vissza az éleket egyenként. Minden visszatett él eggyel növeli a végpontjainak fokszámait, azaz kettővel növeli az összfokszámot.

1.2. tétel. Minden, legalább kétpontú fában van legalább két elsőfokú pont.

A tétel belátása: Gondolatban távolítsuk el a fa összes élét, majd építsük fel a fát újra úgy, hogy egyenként visszatesszük az éleket. Mivel bármely fa összefüggő, ezért létezik az éleknek egy olyan visszarakási sorrendje, hogy az „építmény" minden köztes állapotában fa legyen. Amikor a fa még csak egy élből áll, nyilvánvalóan van két első fokú pontja. Ezután minden hozzácsatolt él egy újabb első fokú pontot hoz a

fába (különben kör alakulna ki), és legfennebb egyet számol fel. Tehát a „növekvő" fának folyamatosan van legalább két első fokú pontja.

1.3. tétel. Egy n pontú fa éleinek száma $n-1$.

A tétel belátása: Gondolatban távolítsuk el a fa összes élét, kivéve egyet. Az így kapott gráfnak lesz egy kétpontú (egyélű) fa-részgráfja és $n-2$ izolált pontja. Építsük vissza a fát, olyan sorrendben téve vissza az éleit, ahogy az előbbi tétel esetében is eljártunk. Minden visszatett él egy újabb izolált pontot csatol a „növekvő" fához. Ahhoz, hogy mind az $n-2$ izolált pont visszacsatolható legyen, legalább $n-2$ eltávolított élnek kellett lennie. Sőt, pontosan $(n-2)$-nek, hiszen minden további él kört eredményezne. A visszatett $n-2$ él a meghagyott eggyel összesen $n-1$ élet jelent.

1.4. tétel. Egy n pontú k komponensű erdő éleinek száma $n-k$.

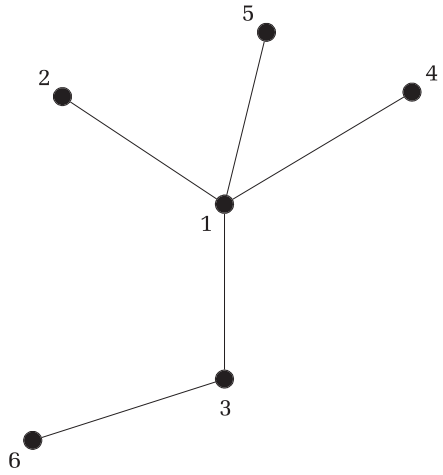
A tétel belátása: Hány plusz élre lenne szükség ahhoz, hogy a k komponensű erdő egyetlen fává „nőjön össze"? Nyilván $k-1$ hídra (elvágó élre). Mivel az így kapott fának az előbbi tétel értelmében $n-1$ éle lenne, világos, hogy az erdőnek $(n-1)-(k-1)=n-k$ éle volt.

1.5. tétel (Cayley). n^{n-2} különböző n pontú fa létezik.

Prüfer-kód: Minden n pontú fa, amelynek a csúcsait az $1,2,\ldots,n$ természetes számokkal azonosítjuk, kódolható egy $(v_1,v_2,\ldots,v_{n-2})$ számsorozattal, ahol v_i eleme az $\{1,2,\ldots,n\}$ halmaznak.

Kódolási folyamat ($n-1$ lépéses eljárás): Minden lépésben eltávolítjuk a fa legkisebb címkéjű levelét, és leírjuk annak a pontnak az azonosítóját, amelyről „levágtuk". Az utolsó lépésben, bármely n pontú fa esetén, egészen biztosan az n címkéjű pont kerül leírásra. Az első $n-2$ leírt érték lesz a fa Prüfer-kódja (1.25. ábra).

Dekódolási folyamat ($n-1$ lépéses eljárás): Egészítsük ki a kódot az n értékkel: $(v_1,v_2,\ldots,v_{n-2},n)$. Próbáljuk rekonstruálni a kódolási eljárást, kitalálva, hogy mely $(w_1,w_2,\ldots,w_n)$ levélsor eltávolítása nyomán került leírásra a $(v_1,v_2,\ldots,v_{n-2},n)$ (kiegészített) kód. Emlékezzünk, hogy minden lépésben a legkisebb azonosítójú levelet vágtuk le a fáról. Egy adott pillanatban azok a pontok számítottak levélnek, amely csúcsok nem voltak *már levágva* a fáról és amelyekről nem kellett *azután levágni* levelet. Úgy is fogalmazhatunk, hogy az i-edik lépésben eltávolításra kerülő w_i pont a legkisebb címkéjű azon pontok között, amelyek *nem* szerepelnek sem a már levágott $(w_1,w_2,\ldots,w_{i-1})$ pontok között, sem azok között a



Levágjuk a 2-es levelet az 1-es pontról. Leírjuk az 1-est.
Levágjuk a 4-es levelet az 1-es pontról. Leírjuk az 1-est.
Levágjuk az 5-ös levelet az 1-es pontról. Leírjuk az 1-est.
Levágjuk a 1-es levelet a 3-as pontról. Leírjuk a 3-ast.
Levágjuk a 3-as levelet a 6-os pontról. Megmarad a 6-os pont.

A Prüfer-kód: (1, 1, 1, 3).
A kiegészített kód: (1, 1, 1, 3, 6).

1.25. ábra. *Prüfer-kód. Kódolási folyamat*

v	1	1	1	3	6
w	**2**				

v	1	1	1	3	6
w	2	**4**			

v	1	1	1	3	6
w	2	4	**5**		

v	1	1	1	3	6
w	2	4	5	**1**	

v	1	1	1	3	6
w	2	4	5	1	**3**

1.26. ábra. *Prüfer-kód. Dekódolási folyamat. A vastagon szedett szám (az éppen levágandó levél) egyenlő a szürke elemek közt nem szereplő legkisebb értékkel. A w tömb szürke elemei a már levágott levelek. A v tömb szürke elemei: pontok, melyekről ezután kell levágnunk levelet. A rekonstruált fa mint éllista: (1,2), (1,4), (1,5), (3,1), (6,3)*

$(v_i, v_{i+1}, \ldots, v_{n-2}, n)$ pontok között, amelyekről a jelenben vagy a jövőben még levágásra kerül levél (és ezért kerülnek leírásra a Prüfer-kódban). Más szóval, w_i a $(w_1, w_2, \ldots, w_{i-1}, v_i, v_{i+1}, \ldots, v_{n-2}, n)$ $n-1$ elemű sorozatban nem szereplő legkisebb címke (ahol $i = 1, \ldots, n-1$) (1.26. ábra).

A Cayley-tétel belátása: Mivel minden fának van legalább két levele, a kódolási folyamat minden lépése bármely fára elvégezhető. Továbbá, mivel az $\{1, 2, \ldots, n\}$ halmaz elemeiből kialakított bármely $(n-1)$ elemű számsorozatból hiányozni fog a halmazok legalább egy eleme, ezért a dekódolási folyamatnak is minden lépése megvalósítható. Tehát minden n pontú fának megfelel egy $(v_1, v_2, \ldots, v_{n-2})$ alakú Prüfer-kód, és fordítva. Ebből következik, hogy a különböző n pontú fák száma n^{n-2}, azaz egyenlő azon $n-2$ elemű, egymástól különböző kódok számával, amelyeknek elemei 1 és n közötti természetes számok.

1.1. A könyvben használt pszeudokód nyelv leírása

Az algoritmusok leírására az alábbi pszeudokód nyelvet használjuk:

Operátorok

- aritmetikai operátorok: `+`, `-`, `*`, `/`, `DIV`, `MOD`
  Megjegyzés: Ha mindkét operandus egész szám, a `/` operátor egész osztást, különben valós osztást jelent.
- összehasonlítási operátorok: `<`, $\leqslant$, `>`, $\geqslant$, `=`, $\neq$
- logikai operátorok: `ÉS`, `VAGY`, `NEM`

Kommentek (megjegyzések)

Ha egy algoritmus valamely sorát dupla slash jel (`//`) előzi meg, akkor megjegyzésnek tekintendő.

Műveletek

Értékadás:

```
<változó> ← <kifejezés>
```

Elágazás:

```
ha <feltétel> akkor
   <műveletek1>
különben
   <műveletek2>
vége ha
```

Elöltesztelő ciklus:

```
amíg <feltétel> végezd
   <műveletek>
vége amíg
```

Hátultesztelő ciklus:

```
végezd
   <műveletek>
amíg <feltétel>
```

Megjegyzés: Mindkét `amíg` ciklusból akkor lépünk ki, ha a feltétel hamissá vált.

Ismert lépésszámú ciklus:

```
minden <változó> ← <kezdőérték>,<végsőérték>,<lépés> végezd
   <műveletek>
vége minden
```

Megjegyzés: Ha a lépésszám hiányzik, akkor 1-nek tekintjük.

Ugró utasítások:

ugorj – kiugrás az aktuális ciklusból
vissza – visszatérés eljárásból/függvényből

Beolvasási művelet:

```
beolvas: <változó lista>
```

Kiírási művelet:

```
kiír: <kifejezés lista>
```

Konstansok (példák)

`13, -524` (egész)
`-12.027, 0.22` (valós)
`'A', 'c' , '1' , '!'` (karakterek)
`"alma", "123", "23 almafa"` (karakterlánc)
`IGAZ, HAMIS` (logikai)

Adatszerkezetek

Egydimenziós tömb (vektor) (példák):
- `a[]` – egydimenziós tömb
- `a[1..n]` – egydimenziós tömb, amelynek elemei `1`-től `n`-ig vannak indexelve
- `a[i]` – hivatkozás egy egydimenziós tömb `i`-edik elemére

Kétdimenziós tömb (példák):
- `b[][]` – kétdimenziós tömb
- `b[1..n][1..m]` – kétdimenziós tömb, amelynek sorai `1`-től `n`-ig, oszlopai pedig `1`-től `m`-ig vannak indexelve
- `b[i][j]` – hivatkozás egy kétdimenziós tömb `i`-edik sorának `j`-edik oszlopbeli elemére

Bejegyzés (rekord, struktúra) (példák):
- `r.m` – hivatkozás az `r` bejegyzés típusú változó `m` mezőjére
- `a[i].x` – az `a` bejegyzés típusú tömb `i`-edik elemének az `x` mezője

Eljárás

```
<eljárás neve> (<formális paraméter lista>)
   <műveletek>
vége <eljárás neve>
```

Megjegyzés: Egy eljárásnak lehet több visszatérési pontja is (tartalmazhat üres `vissza` utasítást).

Függvény

```
<függvény neve> (<formális paraméter lista>)
   <műveletek>
   vissza <eredmény>
vége <függvény neve>
```

Megjegyzés: Egy függvénynek lehet több visszatérési pontja is.

Paraméterátadás: A formális paraméterek listájában jelezni fogjuk, mely paraméterek egyszerű típusúak és melyek tömbök. Az érték szerinti és cím szerinti paraméterátadás között úgy teszünk különbséget, hogy az utóbbi esetében a formális paramétereket félkövér karakterekkel írjuk.

Példa:

```
eljárás(x[],y[][],a,b,c[])
   ...
vége eljárás
```

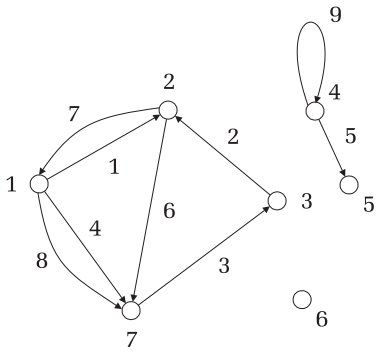
Megjegyzések:

1. a és b egyszerű típusú változók, x és c egydimenziós tömbök, y pedig kétdimenziós tömb. Az x, y és b paraméterek érték szerint, a és c cím szerint kerül átadásra.
2. Amikor tömböket adunk át, ha az algoritmus szempontjából nem lényeges, melyikfajta paraméterátadást használjuk, akkor az implementálásnál – memóriatakarékossági megfontolásból – célszerű a cím szerinti paraméterátadást használni. Egyes programozási nyelvekben (például a C nyelvben) a tömbök implicit cím szerint adódnak át.

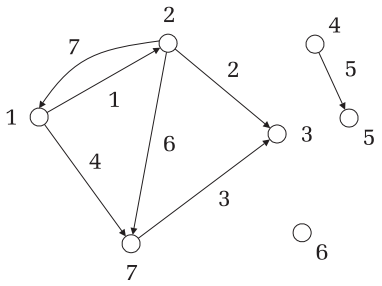
2. FEJEZET

GRÁFOK ÁBRÁZOLÁSA (A SZÁMÍTÓGÉP MEMÓRIÁJÁBAN)

Példagráfok a fejezetben tárgyalt fogalmak szemléltetéséhez:



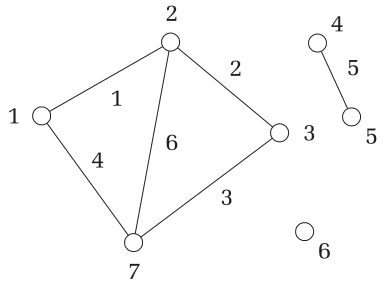
2.1. ábra. *Általános gráf* ($n = 7, m = 9$)



2.2. ábra. *Irányított egyszerű gráf* ($n = 7, m = 7$)

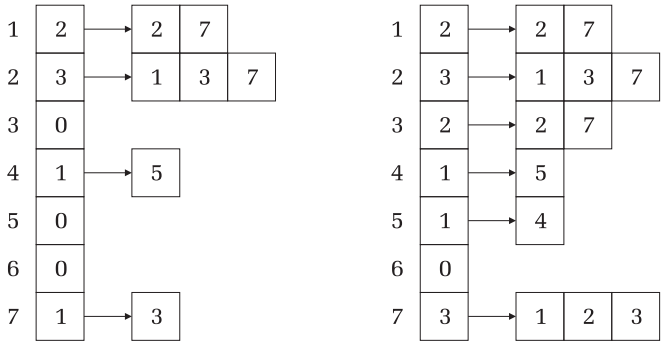
2.1. Szomszédsági listák

A 2.4. ábrák a példagráfok (2.2–2.3. ábrák, egyszerű irányított és irányítatlan gráfok) *szomszédsági listáit* ábrázolják. Néhány jellemzőjük:



2.3. ábra. *Irányítatlan egyszerű gráf* ($n = 7$, $m = 6$)

- Minden csúcsnak tároljuk a szomszédjait.
- Irányítatlan esetben, ha i szomszédja j-nek, akkor j is szomszédja i-nek.
- Irányított esetben a j pont akkor szomszédja az i pontnak, ha van irányított él i-től j-hez.
- A listák mind statikusan, mind dinamikusan implementálhatók.
- Gyakran érdemes tárolni a csúcsok fokszámait is.
- A gráfok tárolása szomszédsági listákkal az egyik leghatékonyabb tárolási mód.



2.4. ábra. *Szomszédsági listák*

2.2. Éllista

A 2.1. táblázatok a példagráfok (2.2–2.3. ábrák, egyszerű irányított és irányítatlan gráfok) *éllistáit* ábrázolják. Néhány jellemzőjük:

2.1. táblázat. *Éllisták*

1	2	3	4	5	6	7
1	**2**	**7**	**1**	**4**	**2**	**2**
2	**3**	**3**	**7**	**5**	**7**	**1**

1	2	3	4	5	6
1	**2**	**7**	**1**	**4**	**2**
2	**3**	**3**	**7**	**5**	**7**

– Egy tömböt használunk, ahol a tömb elemei egy-egy élet tárolnak az él két végpontja által.
– Általában ritka gráfok esetén használják, amelyeknek sok csúcsuk, de kevés élük van.
– Azért, hogy egy él visszakeresése logaritmikus időben történjen (bináris kereséssel), célszerű lehet az élek előzetes rendezése. Irányított gráfok esetén a rendezés történhet az élek kezdőpontja szerint, azonos kezdőpontúak esetén pedig a végpontjuk alapján. Irányítatlan gráfoknál leszögezhetjük, hogy a kisebb címkéjű pont lesz a kezdőpont, a nagyobb címkéjű pedig a végpont. Amennyiben a gráf élsúlyozott (minden éléhez rendeltünk egy valós számot, amelyet az él súlyának, költségének vagy hosszának tekintünk), javíthat a futási időn, ha az éllistánk e súlyértékek szerint rendezett.

2.3. Szomszédsági mátrix

Az $A(G) = (a_{ij})$ $n \times n$ méretű *szomszédsági mátrix* elemeit az alábbi módon értelmezzük:

$$a_{ij} = \begin{cases} 0, & \text{ha az } i \text{ pont és a } j \text{ pont között nem halad él} \\ k, & \text{ha az } i \text{ pont és a } j \text{ pont között } k \text{ párhuzamos él halad} \\ h, & \text{ha } i = j \text{ és az } i \text{ ponthoz } h \text{ hurokél illeszkedik} \end{cases}$$

Irányított gráf esetén a_{ij} nem föltétlen egyenlő a_{ji}-vel.

Egyszerű gráf esetén $h = 0$ és $k = 1$.

Az alábbi mátrixok a példagráfok (2.1–2.3. ábrák) szomszédsági mátrixait mutatják be (2.2. táblázat).

2.1. tétel. A szomszédsági mátrix t-edik hatványának elemei megadják minden (i, j) pontpár között a t hosszú élsorozatok számát. Ezen élsorozatok között nemcsak az utakat, hanem az azonos ponton többször is átmenő sorozatokat is számoljuk.

2.2. táblázat. *Szomszédsági mátrixok*

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	**2**
2	1	0	0	0	0	0	1
3	0	1	0	0	0	0	0
4	0	0	0	**1**	1	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	1
2	1	0	1	0	0	0	1
3	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	1
2	1	0	1	0	0	0	1
3	0	1	0	0	0	0	1
4	0	0	0	0	1	0	0
5	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0
7	1	1	1	0	0	0	0

A tétel belátása (A bizonyítás belátható a mátrixszorzás definíciójából): Teljes indukcióval bizonyítunk. A $t = 2$ esetben az A^2 mátrix $a_{ij}^{(2)}$ elemének értéke egyenlő a $\Sigma a_{ik}a_{kj}$ (ahol $k = 1, 2, \ldots, n$) összeggel. Ezen összeg azon tagjai lesznek 1-gyel egyenlők, amelyekre mind az a_{ik} érték, mind az a_{kj} érték egyenlő 1-gyel, azaz léteznek mind az (i, k), mind a (k, j) élek. Amikor $k = i$, illetve $k = j$, számításba vesszük az i, illetve j pontok esetleges hurokéleit is. Tehát az $a_{ij}^{(2)}$ érték az i és j pontok közötti két hosszú élsorozat számát adja meg.

Definíció szerint $A^t = A^{t-1} \times A$, és feltételezzük, hogy az A^{t-1} mátrix $a_{ij}^{(t-1)}$ eleme az i és j pontok közötti, $(t-1)$ hosszú élsorozatok számával egyenlő. Az előbbi gondolatmenetet követve, az $a_{ij}^{(t)} = \Sigma a_{ik}^{(t-1)} a_{kj}$ (ahol $k = 1, 2, \ldots, n$) összeg azon tagjai lesznek nullától különbözőek, amelyekre az $a_{ik}^{(t-1)}$ érték nagyobb, mint nulla, az a_{kj} elem pedig 1. Ez azt jelenti, hogy az i és k pontok között $a_{ik}^{(t-1)}$ darab $(t-1)$ hosszú élsorozat létezik, és k-ból j-be is van él. Más szóval: az i és j pontok között $a_{ik}^{(t-1)} a_{kj}$ darab olyan t elemű élsorozat van, amelyen utolsó előtti állomás a k pont. Szem előtt tartva ezt, nyilvánvaló, hogy az $a_{ij}^{(t)}$ érték az i és j pontok között létező összes t hosszú élsorozat számát jelenti.

Megjegyzés: Például egy hurokmentes irányítatlan gráf esetén A^3 diagonális elemeinek az összege a három hosszú körök számának a hatszorosa lesz. (Minden kör minden pontjából kiindulva mindkét irányban bejárható.)

2.4. Illeszkedési mátrix

A $B(G) = (b_{ij})$ $n \times m$ méretű *illeszkedési mátrix* elemeit az alábbi módon értelmezzük:

$$b_{ij} = \begin{cases} 0, & \text{ha a } j \text{ él nem illeszkedik az } i \text{ ponthoz} \\ 1, & \text{ha a } j \text{ él kezdőpontja az } i \text{ pont} \\ -1, & \text{ha a } j \text{ él végpontja az } i \text{ pont} \end{cases}$$

Ha a j él az i ponthoz illeszkedő hurokél, akkor is (megállapodás szerint) $b_{ij} = 1$. Irányítatlan esetben, definíció szerint, a j él mindkét végpontjának megfelelő mátrixelem értéke 1.

Az alábbi mátrixok a példagráfok (2.1–2.3. ábrák) illeszkedési mátrixait mutatják be.

2.2. tétel. Egy n pontú, c darab összefüggő komponensből álló, hurokélmentes irányított gráf illeszkedési mátrixának rangja[1] $n - c$.

A tétel belátása: Ha a komponensek száma nagyobb, mint 1, akkor a gráf pontjait és éleit komponensekként sorszámozva, a $B(G)$ mátrix blokkdiagonális lesz. Egy n_i pontú komponensre (n_i az i-edik komponens pontjainak száma, ahol $i = 1, 2, \ldots, c$; $\Sigma n_i = n$) tehát elég belátni, hogy a neki megfelelő blokk rangja $(n_i - 1)$. Mivel a szóban forgó blokk n_i soros és sorainak összege a nulla vektor (minden élenek megfelelő oszlopban pontosan egy darab 1-es és egy darab (–1)-es van, a többi elem pedig nulla; nincs a gráfnak hurokéle), világos, hogy a rang legfennebb $(n_i - 1)$ lehet. Bebizonyítjuk, hogy pontosan $(n_i - 1)$.

Legyen F az i-edik komponensnek egy n_i pontú, $(n_i - 1)$ élű feszítőfája. Legyen továbbá F-nek egy v_1 levele és e_1 a hozzákapcsolódó él. Hasonlóképpen legyen v_2 egy levele az $F - \{v_1\}$ fának és e_2 az az él, amelyik a fához kapcsolja ezt, és így tovább. Ha a blokk sorait v_1, v_2, ... sorrendben soroljuk fel, az oszlopait pedig az e_1, e_2, ... élsorrendnek

1 Egy mátrix rangja a lineárisan független oszlopainak maximális száma, ami megegyezik a lineárisan független sorainak maximális számával.

2.3. táblázat. *Illeszkedési mátrixok*

	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	0	−1	1	0
2	−1	−1	0	0	0	1	1	0	0
3	0	1	−1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	1
5	0	0	0	0	−1	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	1	−1	0	−1	0	−1	0

	1	2	3	4	5	6	7
1	1	0	0	1	0	0	−1
2	−1	1	0	0	0	1	1
3	0	−1	−1	0	0	0	0
4	0	0	0	0	1	0	0
5	0	0	0	0	−1	0	0
6	0	0	0	0	0	0	0
7	0	0	1	−1	0	−1	0

	1	2	3	4	5	6
1	1	0	0	1	0	0
2	−1	1	0	0	0	1
3	0	−1	−1	0	0	0
4	0	0	0	0	1	0
5	0	0	0	0	−1	0
6	0	0	0	0	0	0
7	0	0	1	−1	0	−1

megfelelően helyezzük el, akkor egy olyan $n_i \times (n_i - 1)$ méretű részmátrixot kapunk, amelynek ha elhagyjuk az utolsó sorát, akkor az így nyert négyzetes mátrix főátlóra eső elemei +/−1 értékűek, főátló feletti elemei pedig mind nullák. Mivel találtunk $(n_i - 1)$ darab lineárisan független oszlopot, ezért a szóban forgó blokk rangja $(n_i - 1)$. Összeadva komponensenként az így kapott értékeket, a $\Sigma(n_i - 1) = n - c$ (ahol $i = 1, 2, \ldots, c$) eredményhez jutunk.

2.3. tétel. Egy n pontú, összefüggő, hurokélmentes irányított gráf illeszkedési mátrixában válasszunk ki $n - 1$ oszlopot. Ezek akkor és

csakis akkor lesznek lineárisan függetlenek, ha a megfelelő $n-1$ él a gráf egy fáját alkotja.

A tétel belátása: Az előbbi tétel bizonyításakor beláttuk, hogy a fát alkotó éleknek megfelelő oszlopok lineárisan függetlenek. Most e mellé még bebizonyítjuk, hogy bármely kör esetén az ezt alkotó éleknek megfelelő oszlopok lineárisan függők. Nem nehéz átlátni, hogy ha egy kör pontjainak sorait és éleinek oszlopait egymás után soroljuk fel az illeszkedési mátrixban, és ha ezeket ráadásul a körön való megjelenésük szerinti sorrendben sorszámozzuk, akkor a körnek megfelelő diagonális blokk az alábbi alakú lesz (p a kör hossza):

$$\begin{matrix} x_1 & 0 & \dots & -x_p \\ -x_1 & x_2 & \dots & 0 \\ 0 & -x_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & x_p, \end{matrix}$$

ahol az x_i ($i = 1, 2, \dots, p$) értékek mindenike vagy 1 vagy (-1). Képezzük az oszlopvektoroknak azt a lineáris kombinációját, amelyben az i-edik oszlop együtthatója 1, ha $x_i = 1$, és (-1), ha $x_i = -1$.

2.4. tétel. Hagyjunk el az n pontú összefüggő irányított G gráf illeszkedési mátrixából egy tetszőleges sort. Az így kapott B' mátrixból képezhető $B' \cdot B'^T$ négyzetes mátrix determinánsa G feszítőfáinak a számával lesz egyenlő. (Bizonyítás végett lásd a [6], 34. oldalt.)

2.5. Körmátrix

Egy G irányított gráf (n pontú, m élű) minden körének (n_c: körök száma) válasszunk egy körbejárási irányt (például az óra járásával ellentétes irányt). Ezek után a következőképen definiáljuk a gráf $C(G)$ *körmátrixát* (mérete: $n_c \times m$).

$$c_{ij} = \begin{cases} 0, & \text{ha a } j \text{ él nem eleme az } i \text{ körnek} \\ 1, & \text{ha a } j \text{ él eleme az } i \text{ körnek, és irányítása megegyezik} \\ & \text{a körbejárási iránnyal} \\ -1, & \text{ha a } j \text{ él eleme az } i \text{ körnek, és irányítása ellentétes} \\ & \text{a körbejárási iránnyal} \end{cases}$$

1: (1, 7)
2: (8, 4)
3: (4, 6, 1)
4: (6, 3, 2)
5: (4, 3, 2, 1)
6: (8, 6, 1)
7: (8, 3, 2, 1)
8: (4, 6, 7)
9: (4, 3, 2, 7)
10: (8, 6, 7)
11: (8, 3, 2, 7)
12: (9)

A 2.1. gráf körei mint éllisták (eltekintünk az irányítástól; a körbejárási irány az óra járásával ellentétes irány)

2.4. táblázat.

A 2.1. gráf körmátrixa

	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	−1	0	0
2	0	0	0	−1	0	0	0	1	0
3	−1	0	0	1	0	−1	0	0	0
4	0	1	1	0	0	1	0	0	0
5	−1	1	1	1	0	0	0	0	0
6	−1	0	0	0	0	−1	0	1	0
7	−1	1	1	0	0	0	0	1	0
8	0	0	0	1	0	−1	1	0	0
9	0	1	1	1	0	0	1	0	0
10	0	0	0	0	0	−1	1	1	0
11	0	1	1	0	0	0	1	1	0
12	0	0	0	0	0	0	0	0	1

2.5. tétel. Ha G egy összefüggő irányított gráf, akkor $rang(C(G)) = m - n + 1$.

2.6. tétel. Egy G összefüggő irányított gráf $(m - n + 1)$ különböző oszlopa akkor és csakis akkor lineárisan független, ha a nekik megfelelő élek a G gráf egy feszítő fájának komplementerét alkotják. (A feszítő fa $(n - 1)$ élű, a komplementere pedig éppen $(m - n + 1)$ élet tartalmaz.)

Ha egy összefüggő G gráf valamely lerögzített F feszítő fájához az összes lehetséges módon hozzáveszünk egy-egy további élet, akkor az így kapott feszítő részgráfok mindenike pontosan egy kört fog tartalmazni. E körök összességét az F feszítőfához tartozó *alapkörrendszer*nek nevezzük.

Nem nehéz átlátni, hogy az F feszítőfához tartozó alapkörrendszer köreit képviselő $(m - n + 1)$ sor és az F-hez *nem* tartozó $(m - n + 1)$ élnek megfelelő oszlop alkotta négyzetes részmátrix (a sorok és oszlopok esetleges permutációi után) az előjelektől eltekintve egységmátrix.

2.6. Vágásmátrix

A körmátrixhoz hasonlóan definiálható a $Q(G)$ *vágásmátrix*, amennyiben a gráf minden vágásának választunk egy „irányítást". Egy vágást alkotó élek mind a gráf ugyanazon komponensében vannak, szétválasztva a komponens pontjait két nem üres V_1 és V_2 diszjunkt részhalmazra. Ha egy adott él kezdőpontja V_1-ben, a végpontja pedig V_2-ben van, akkor azt mondjuk, hogy a szóban forgó él irányítása megegyezik a vágás irányításával (különben ellentétes irányítású a vágással).

Az alábbiakban felsoroljuk a 2.1. gráf vágásait (mint ponthalmazpárt és mint élhalmazt). V_1 halmaznak mindig azt választottuk, amelyikben az illető komponens legkisebb címkéjű pontja van. A vágások irányítása V_1-től V_2 fele van.

1: {1} ∪ {2, 3, 7}; {1, 4, 7, 8}
2: {1, 3, 7} ∪ {2}; {1, 2, 6, 7}
3: {1, 2, 7} ∪ {3}; {2, 3}
4: {1, 2, 3} ∪ {7}; {3, 4, 6, 8}
5: {4} ∪ {5}; {5}

2.5. táblázat. *A 2.1. gráf vágásmátrixa*

	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	0	−1	1	0
2	1	1	0	0	0	−1	−1	0	0
3	0	−1	1	0	0	0	0	0	0
4	0	0	−1	1	0	1	0	1	0
5	0	0	0	0	1	0	0	0	0

2.7. tétel. Egy n pontú, c darab összefüggő komponensből álló, irányított gráf vágásmátrixának rangja $n - c$.

2.8. tétel. Egy n pontú, összefüggő, irányított gráf vágásmátrixában válasszunk ki $n - 1$ oszlopot. Ezek akkor és csakis akkor lesznek lineárisan függetlenek, ha a megfelelő $n - 1$ él a gráf egy fáját alkotja.

2.9. tétel. Ha B, C és Q a G hurokmentes irányított gráf illeszkedési, kör- és vágásmátrixai (feltételezzük, hogy a mátrixok oszlopai ugyanabban a sorrendben felelnek meg G éleinek), akkor:

$$B \cdot C^T = \mathbf{0} \quad \text{és} \quad Q \cdot C^T = \mathbf{0}.$$

Ellenőrizzük a fenti tételt a 2.3.1. (B), 2.4. (C) és 2.5. (Q) mátrixokra. A hurokmentességre vonatkozó kitételre való tekintettel töröljük mindhárom mátrix 9-edik oszlopát, illetve a C mátrix 12-edik sorát.

Megjegyzések:

- Gráfalgoritmusok implementálásánál nem tanácsos az illeszkedési, kör- és vágásmátrixokat használni mint gráfreprezentációkat. Az utóbbi kettő nem is alkalmas a gráfok ábrázolására, hiszen nem izomorf gráfoknak lehet azonos kör-, illetve vágásmátrixa. Sőt, általában kerülendő a szomszédsági mátrix használata is. Ezzel összhangban, a könyvben található algoritmusok rendszerint szomszédsági listát és/vagy éllistát használnak.
- Fontos gyakorlati alkalmazásra lelnek az illeszkedési és körmátrixok a villamos hálózatok tanulmányozásában. Ha egy villamos hálózat minden alkatrésze kétpólusú, akkor a hálózat kapcsolási rajza természetes módon megadható egy G irányított gráffal, ahol az élek irányítása a „mérőiránynak" felel meg. Ez esetben a Kirchhoff-féle áramegyenletek tömören $B\boldsymbol{i} = \mathbf{0}$ alakba írhatók, ahol az $\boldsymbol{i}$ vektor elemei az alkatrészeken átfolyó áramértékek. Hasonlóképpen a $C\boldsymbol{u} = \mathbf{0}$ egyenlet a Kirchhoff-féle feszültségegyenleteknek felel meg (az $\boldsymbol{u}$ vektor elemei az alkatrészeken átfolyó feszültségértékeknek felelnek meg).
- Ha egy villamoshálózat-gráf minden pontjára felírnánk egy-egy Kirchhoff-féle áramegyenletet, akkor n egyenletet kapnánk. A 2.7. tételből láthattuk, hogy ezek közül csak $(n - c)$ darab lineárisan független. Ezért a gyakorlatban a villamoshálózat-gráfok minden komponensében egy áramegyenletet figyelmen kívül szoktak hagyni. A feszültségegyenleteknél a helyzet valamivel bonyolultabb. Megtörténhet, hogy egy n pontú gráfnak közel $n!$ köre van, amelyeknek megfelelő feszültségegyenletekből csak $(m - n + 1)$ lesz lineárisan független. Például a K_6 gráfnak megfelelő hálózat 6 áramegyenletéből 1, viszont 165 feszültségegyenletéből 150 lenne fölösleges. Ezért a villamosmérnöki gyakorlatban a hálózati gráfok egy alapkörrendszerére érdemes felírni a Kirchhoff-féle feszültségegyenleteket.

2.7. Gyökeres fák ábrázolása

Egy n pontú gyökeres fa ábrázolható egy `apa[1..n]` tömb segítségével. Minden csomópontnak eltároljuk az apa-csomópontját. Mivel a gyökérnek nincs apja, ezért az `apa[gyökér]` értéket nullára állíthatjuk.

Az alábbi táblázat a 3.3.a ábrán látható gyökeres fa apa-tömbjét mutatja be:

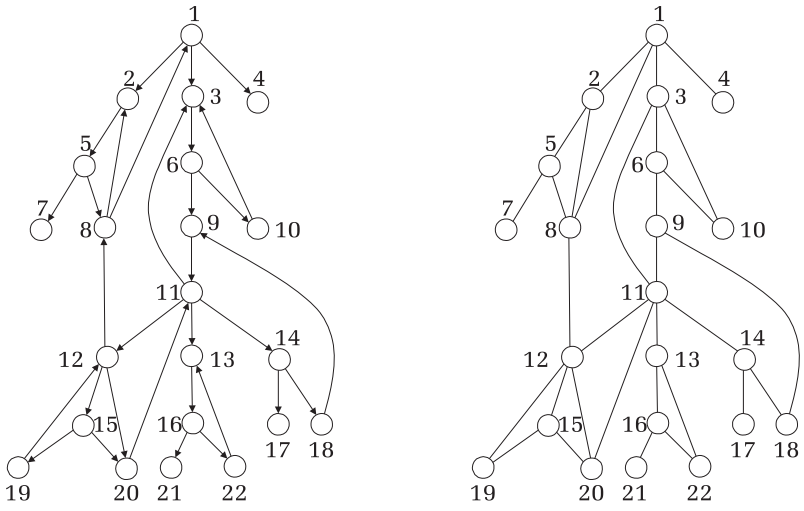
2.6. táblázat. *A 3.3. ábrán látható gyökeres fa apa-tömbje*

0	1	1	1	2	3	5	1	6	3	3	8	11	11	12	13	16	9	12	11	16	13
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

3. FEJEZET

GRÁFOK BEJÁRÁSAI

Példagráfok a bejárási algoritmusok szemléltetéséhez:



3.1. ábra. *a) Irányított gráf; b) irányítatlan gráf*

Bejárni egy gráfot azt jelenti, hogy egy bizonyos stratégia szerint, a gráf élein haladva, meglátogatjuk a csomópontjait. Alapvetően két bejárási algoritmus létezik: szélességi (DFS) és mélységi (BFS) bejárás. Az alábbiakban a két algoritmus leírásában használt közös jelöléseket közöljük.

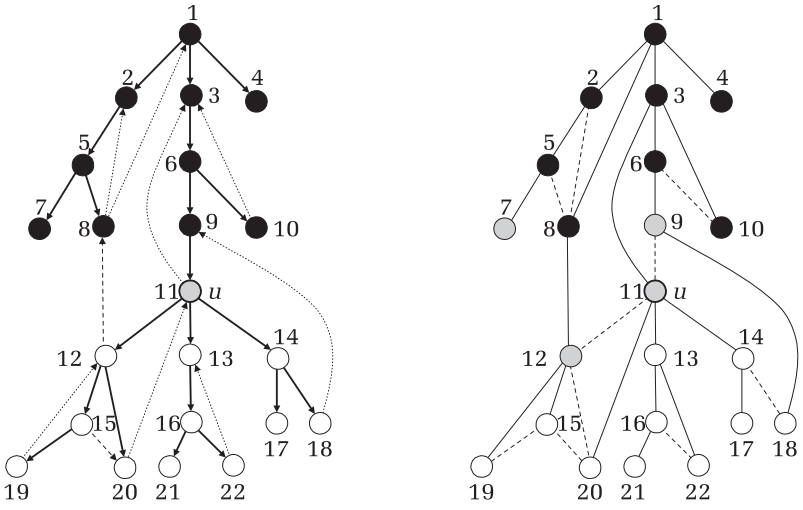
Jelölések:

`G` – gráf
`V(G)` – csúcsok halmaza: $\{1, 2, \ldots, n\}$
`n` – csúcsok száma
`E(G)` – élek halmaza
`m` – élek száma
`Szomszéd(u)` – az u pont szomszédainak halmaza

szín[u] – FEHÉR (érintetlen csúcs), SZÜRKE (elért csúcs), FEKETE (elhagyott csúcs)

apa[u] – az u csúcs apa-csúcsa a mélységi, illetve szélességi fában (lásd lennebb). Az illető fák egyfajta ábrázolása.

3.1. Szélességi bejárás (BFS – Breadth First Search)



3.2. ábra. *a) Irányított gráf szélességi bejárása; b) irányítatlan gráf szélességi bejárása*

Stratégia: Meglátogatjuk a kiindulási pontot (s), majd ennek szomszédait (azonosítóik növekvő sorrendjében), azután a szomszédok szomszédait és így tovább. A gráf azon éleit, amelyeken keresztül elérjük az egyes pontokat, *faél*eknek nevezzük (lásd lennebb). Minden faél egy apa–fiú kapcsolatot képvisel a gráf csomópontjai között. Egy csomópont azoknak a pontoknak apja, amelyek az ő szomszédaiként érhetők el. Ebből a szempontból tekintve a dolgokat, úgy is mondhatnánk, hogy a szélességi bejárás generációról generációra halad: meglátogatja s-t, majd s fiait, azután a fiak fiait és így tovább. A faélek alkotta s gyökerű fát *szélességi fá*nak nevezzük.

A szélességi bejárás csak azokat a pontokat éri el, amelyek ahhoz a komponenshez tartoznak, amelynek s is része.

Algoritmus: Egy sorszerkezetet (Q) használunk. Kezdetben a sor egyedül a kiindulási pontot (s) tartalmazza. Minden lépésben először a sorelső pont (MÁSOL_SORELSŐ függvény) még meg nem látogatott szomszédait (a FEHÉR színűeket) betesszük a sor végére (BETESZ_SORVÉGÉRE eljárás), majd töröljük az illető pontot a sorból (TÖRÖL_SORELEJÉRŐL eljárás).

A bejárás alatt nyilvántartjuk, hogy minden egyes (fiú)csomópontot mely (apa)csomópont szomszédjaként értünk el (azaz melyik sorelső pont szomszédjaként került be az illető pont a Q sor végére). Ezt a nyilvántartást az apa-tömb segítségével végezzük.

A szín-tömböt arra használjuk, hogy nyilvántartsuk a pontok szélességi bejárás alatti státusát: érintetlen pont (FEHÉR, még nem került be a Q sorba), elért pont (SZÜRKE, bent van a Q sorban), elhagyott pont (FEKETE, eltávolítottuk a Q sorból). A szélességi sorrendet a pontok elérési (szürkévé válási) sorrendje jelenti. A 3.2. példagráfokra ez a sorrend az alábbi (amennyiben az 1-es pontból indulunk):

- az irányított gráf esetén: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 16, 17, 18, 19, 21, 22;
- az irányítatlan gráf esetén: 1, 2, 3, 4, 8, 5, 6, 10, 11, 12, 7, 9, 13, 14, 20, 15, 19, 18, 16, 22, 17, 21.

A 3.2.a, b ábra a szélességi bejárás azon momentumát ábrázolja, amikor a 11-es pont lett a sorelső. A Q sor alábbi ábrázolásaiban a fekete cellák a sorból már eltávolított elemeket jelentik, a fehér cellák pedig a jövőbeni értéküket tartalmazzák. A Q sor aktuális tartalmát a szürke cellák képviselik.

- az irányított gráf esetén:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	20	16	17	18	19	21	22

- az irányítatlan gráf esetén:

1	2	3	4	8	5	6	10	11	12	7	9	13	14	20	15	19	18	16	22	17	21

A d tömbben azt tároljuk, hogy milyen távolságra vannak az egyes pontok a kiindulási ponttól (az út hossza alatt az utat alkotó élek számát értjük). Minden (fiú)pont egy éllel távolabb esik s-től, mint az (apa)pontja.

További jelölések:
s – kiinduló pont
d[u] – az u csúcs távolsága s-től (élszámban kifejezve)
Q – sorszerkezet

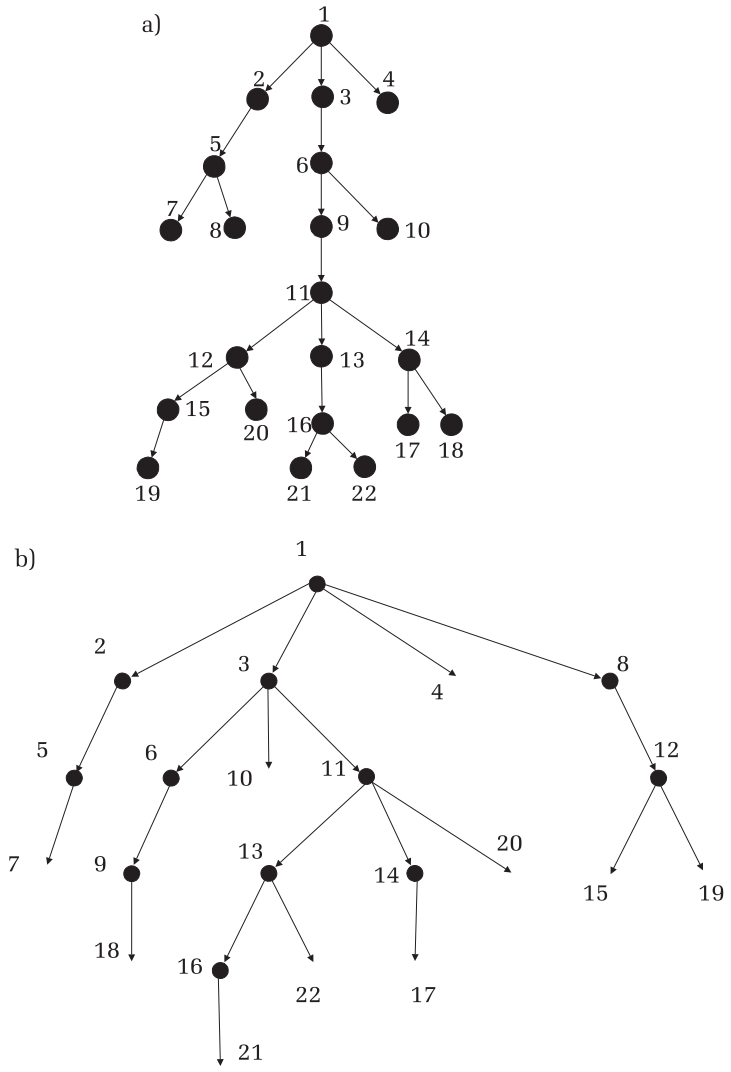
```
eljárás SZÉLESSÉGI_BEJÁRÁS(G,s)
   minden u ∈ V(G)-{s} végezd
      szín[u] ← FEHÉR
      apa[u] ← 0
      d[u] ← ∞
   vége minden
   szín[s] ← SZÜRKE
   apa[s] ← 0
   d[s] ← 0
   Q ← {s}
   amíg Q ≠ ∅ végezd
      u ← MÁSOL_SORELSŐ(Q)
      kiír: u
      minden v ∈ Szomszéd(u) végezd
         ha szín[v] = FEHÉR akkor
            szín[v] ← SZÜRKE
            d[v] ← d[u] + 1
            apa[v] ← u
            BETESZ_SORVÉGÉRE(Q,v)
         vége ha
      vége minden
      TÖRÖL_SORELEJÉRŐL(Q)
      szín[u] ← FEKETE
   vége amíg
vége SZÉLESSÉGI_BEJÁRÁS
```

Az alábbi táblázatok a 3.2.a,b példagráfokra tartalmazzák a d és *apa* tömböket.

3.1. táblázat.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
d	0	1	1	1	2	2	3	3	3	3	4	5	5	5	6	6	6	6	7	6	7	7
apa	0	1	1	1	2	3	5	5	6	6	9	11	11	11	12	13	14	14	15	12	16	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
d	0	1	1	1	2	2	3	1	3	2	2	2	3	3	3	4	4	4	3	3	5	4
apa	0	1	1	1	2	3	5	1	6	3	3	8	11	11	12	13	14	9	12	11	16	13



3.3. ábra. *a) Irányított gráf szélességi fája; b) irányítatlan gráf szélességi fája*

A 3.3.a, b ábrák a két példagráf szélességi fáit ábrázolják, amelyeket az algoritmus az *apa*-tömbben kódol.

Megjegyzés: Úgy is fogalmazhatnánk, hogy szélességi bejárás esetén mindig abból a szürke pontból lépünk tovább, amelyik legkorábban vált szürkévé. Mivel erre az elvre épül a sorszerkezet is, ezért logikus, hogy ezt használjuk, mint adatszerkezetet, a szélességi bejárás implementálásakor. Mivel csak fehér pontok irányába lépünk tovább, nyilvánvaló, hogy a bejárt élek (feszítő)fát alkotnak. E fát nagyon „gazdaságosan" tárolja az apa-tömb. Ennek az „ára" az, hogy a fa élei ellentétes irányítással kerülnek eltárolásra. Ezért van szükség rekurzív eljárásra, ha a gyökértől egy adott levélhez vezető utat az irányításának megfelelően szeretnénk kiírni.

Bonyolultság: A **SZÉLESSÉGI_BEJÁRÁS** algoritmus bonyolultsága[1] – amennyiben szomszédsági lista tárolja a gráfot $-O(n+m)$ ([2], 409. oldal).

Legrövidebb utak

3.1. tétel. Az algoritmus végén a d tömb az s csúcsból az egyes csúcsokhoz vezető legrövidebb utak *hosszát* (az út menti élek száma) tartalmazza. (Bizonyítás végett lásd a [2], 410–412. oldalt.)

Megjegyzés: Az *apa*-tömb alapján – egy rekurzív eljárással (`Kiír_LRU`) – meghatározható bármely csúcsra az s-ből hozzá vezető legrövidebb *út*.

```
eljárás Kiír_LRU(i,apa[])
   ha apa[i]≠0 akkor
      Kiír_LRU(apa[i],apa)
   vége ha
   kíir: i
vége Kiír_LRU
```

1 Az algoritmusok egyik legfontosabb jellemzője az (idő)bonyolultság, azaz, hogy a feladat megoldásához szükséges elemi műveletek száma (feltételezzük, hogy minden elemi műveletnek azonos az időigénye) milyen módon függ a feladat bemenetének (input) méretétől. Gráfokkal kapcsolatos feladatok esetén a bemenet méretét általában a csúcsok száma (n), az élek száma (m), vagy ezek összege $(n+m)$ képviseli. Például, ha a bemenet mérete n és a végrehajtandó elemi műveletek száma $g(n) = a \cdot n + b$, $(a > 0)$, akkor azt mondjuk, hogy az algoritmus lineáris bonyolultságú, azaz bonyolultsága $O(n)$ (ejtsd: nagy ordó n). Általánosabban, a $g(n)$ függvény nagyságrendje $O(f(n))$, ha létezik olyan c pozitív konstans, amelyre $g(n) \geqslant c \cdot f(n)$ fennáll majdnem minden nem negatív n értékre. ([2], 21. oldal)

Elkerülhetetlen pontok

Legyen egy körmentes irányított gráf, amelynek van egy forrása és egy nyelője, és minden út elvezet a forrásból a nyelőbe. Egy pontot elkerülhetetlennek nevezünk, ha minden forrásból nyelőbe vezető út áthalad rajta.

Szélességi bejárással meghatározhatók egy – az előbbi feltételeknek eleget tevő – gráf elkerülhetetlen pontjai. Nem nehéz átlátni, hogy ha a szélességi bejárás nyomán mindig egy olyan szürke pontból lépünk tovább, amelyiknek már minden be-szomszédja fekete, akkor azok az elkerülhetetlen pontok, amelyek egy adott pillanatban egymaguk maradnak a Q sorban.

Az élek osztályozása (a szélességi bejárás nyomán)

Egy gráf (u, v) élei az alábbi módon osztályozhatók (ha a gráf nem összefüggő, akkor a bejárt komponens éleire vonatkoztatjuk):

- *Faél*: ha v-t először az (u, v) él vizsgálata nyomán értük el.
- *Visszamutató él*: ha v őse u-nak a szélességi fában (és ha (v, u) nem minősült már faélnek).
- *Előre mutató él*: ha v utóda u-nak a szélességi fában (és ha (u, v) nem minősült már faélnek).
- *Keresztél*: az összes többi él. Azokat az éleket kötik össze, amelyeknek végpontjai között nincs ős–utód, vagy utód–ős kapcsolat a szélességi fában.

Egy irányított gráf szélességi bejárása nyomán a következőket észleljük:

- Egyetlen él sem minősül előre mutatónak.
- Minden (u, v) faélre `d[v] = d[u] + 1.`
- Minden (u, v) keresztélre `d[v]` $\leqslant$ `d[u] + 1.`
- Minden visszamutató élre `0` $\leqslant$ `d[v]` $\leqslant$ `d[u].`

Egy irányítatlan gráf szélességi bejárása nyomán belátható, hogy:

- Egyetlen él sem minősül sem visszamutató, sem előre mutató élnek.
- Minden (u, v) faélre `d[v] = d[u] + 1.`
- Minden (u, v) keresztélre `d[v] = d[u] vagy d[v] = d[u] + 1.`

Megjegyzés: Minden él akkor kap besorolást, amikor a szélességi bejárás *először* érzékeli a létezését.

A 3.2.a irányított gráfon megvastagítottuk a faéleket, szaggatott vonal jelöli a keresztéleket, és pontvonal ábrázolja a visszamutató éleket.

A 3.2.b irányítatlan gráfon megvastagítottuk a faéleket, és szaggatott vonal jelöli a keresztéleket.

Megjegyzés: Ha egy irányított gráfban töröljük az éleinek irányítását, akkor az ugyanabból a pontból indított szélességi bejárás is átminősíti az éleket.

Megjegyzés: Nagyon kifejezők az alábbi megnevezések a bejárt gráf éleit illetően:

Faél – olyan él, amely részévé vált a szélességi (vagy mélységi; lásd később) *fá*nak

Visszamutató él – visszamutat az aktuális pont valamelyik szürke ősére (az ősök *mögöttünk* maradtak)

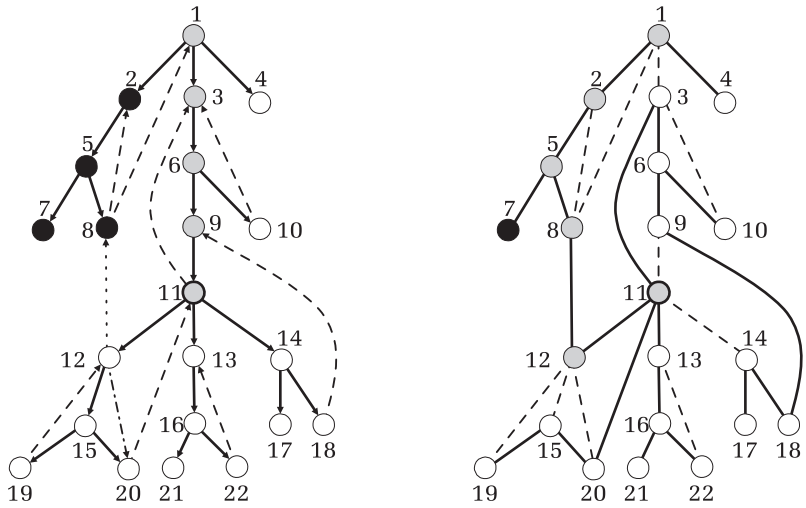
Előremutató él – előremutat az aktuális pont valamelyik, már feketévé vált utódjára (az utódok *előttünk* vannak)

Keresztél – keresztbe mutat a szélességi fa (vagy mélységi; lásd később) egy *másik ágának* valamelyik, már feketévé vált pontjára

3.2. Mélységi bejárás (DFS – Depth First Search)

Stratégia: Ahhoz, hogy megértsük a mélységi bejárást, képzeljük el, hogy a gráf egy ország úthálózatát ábrázolja, és „valaki", aki „külföldről" érkezik az „országba", meg akarja látogatni (autóval) ennek minden „városát" (azaz, mintha szó szerint végigjárnánk a gráfot annak élei mentén). Kezdetben minden pontot fehérnek tekintünk (érintetlen). Ha megérkeztünk egy ponthoz, szürkére változtatjuk a színét. Amikor „kénytelenek vagyunk" *visszat*érni egy pontból, akkor feketén hagyjuk magunk mögött.

„Utazásunkat" az s pontból indítjuk (a fenti példák esetén az indulási pont az 1-es). Tehát s színét szürkére változtatjuk. Merre menjünk? Az érintetlen (fehér) szomszédok közül a legkisebb azonosítójú ponthoz! Ha megérkezünk az illető szomszéd ponthoz (szürkére változtatjuk a színét), ott hasonlóan járunk el: a legkisebb azonosítójú fehér szomszédja felé haladunk tovább. Mi van akkor, ha az aktuális pontnak nincs (vagy már nincs több) fehér szomszédja? Ilyenkor vissza-„rükvercezünk" (ekkor változtatjuk feketére a színét) ahhoz a ponthoz, ahonnan idejöttünk. Itt folytatjuk, amit abbahagytunk, azaz megpróbálunk egy következő fehér szomszéd (ha létezik) irányába továbbmenni. Miután kimerítettünk



3.4. ábra. *a) Irányított gráf mélységi bejárása; b) irányítatlan gráf mélységi bejárása*

minden „fehér irányt" (sorra minden fehér szomszéd irányába elmentünk és rükvercben visszajöttünk), akkor innen is vissza-rükvercezünk (a színe feketére vált) ahhoz a ponthoz, ahonnan annak idején idejöttünk. A mélységi bejárás akkor ér véget, amikor az indulási pontból rükverceznénk vissza (elhagyjuk az országot).

A mélységi sorrendet hagyományosan a pontok elérési (szürkévé válási) sorrendje jelenti. A 3.4. példagráfokra ez a következő:

- irányított gráf: 1, 2, 5, 7, 8, 3, 6, 9, 11, 12, 15, 19, 20, 13, 16, 21, 22, 14, 17, 18, 10, 4;
- irányítatlan gráf: 1, 2, 5, 7, 8, 12, 11, 3, 6, 9, 18, 14, 17, 10, 13, 16, 21, 22, 20, 15, 19, 4.

Vegyük észre, hogy egy pont szürkévé válik, amint előrehaladó „autónkkal" elértük, és mindaddig szürke marad, míg rükvercben el nem hagyjuk. Ettől a pillanattól fogva színe feketére változik.

A gráf azon éleit, amelyeken áthaladtunk, faéleknek nevezzük (lásd lennebb). Minden faél egy apa–fiú kapcsolatot képvisel a gráf csomópontjai között. Egy csomópont azoknak a pontoknak apja, amelyek az ő fehér szomszédaiként érhetők el. Az apa-ponttól előre haladva érkezünk egy adott fiú-ponthoz, a fiú-pontoktól viszont rükvercben térünk

vissza az apa-pontjukhoz. A faélek alkotta s gyökerű fát *mélységi fá*nak nevezzük.

Algoritmus: Akik jártasak a programozásban, azokat „autónk útja" valószínűleg a „rekurzió útjára" emlékeztette. A MÉLYSÉGI_MENET rekurzív eljárás egyetlen összefüggő komponens mélységi bejárását valósítja meg. A MÉLYSÉGI_BEJÁRÁS eljárás második **minden** ciklusa minden egyes komponensre meghívja a MÉLYSÉGI_MENET eljárást.

Tehát a mélységi bejárás (a szélességitől eltérően) minden komponens bejárását biztosítja.

Az algoritmus meghatározza minden csomópont esetén az elérési (amikor szürkére változik) és elhagyási időpontokat (amikor fekete színt kap). Ennek érdekében egy globális időváltozót használunk. Minden csomópont-színváltás alkalmával az időváltozó lép egyet. Az elér és elhagy tömbök lehetővé teszik a csomópontok elérési és elhagyási sorrendjeinek felállítását.

További jelölések:

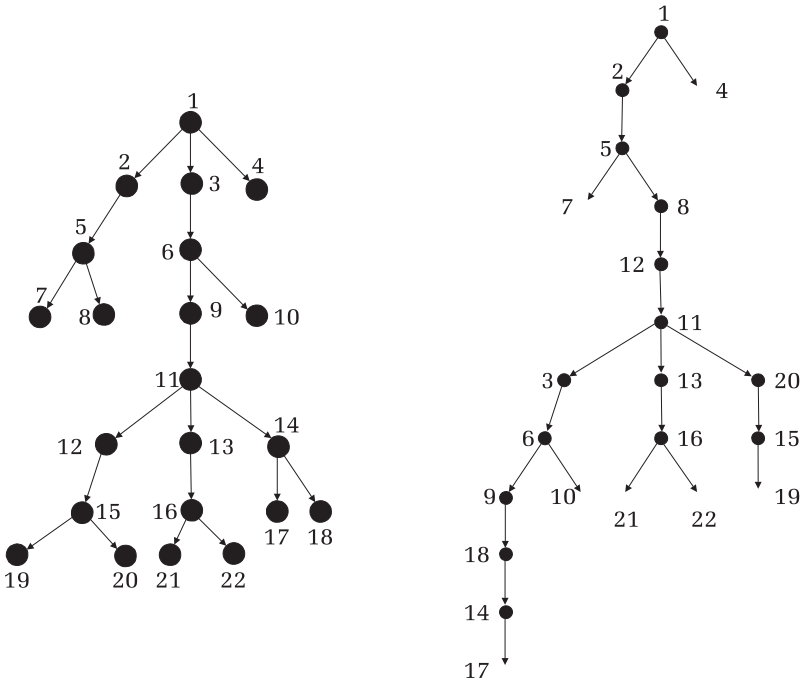
elér[u] – az u csúcs elérési időpontja

elhagy[u] – az u csúcs elhagyási időpontja

```
eljárás MÉLYSÉGI_BEJÁRÁS(G)
   minden u ∈ V(G) végezd
      szín[u] ← FEHÉR
      apa[u] ← 0
   vége minden
   idő ← 0
   minden u ∈ V(G) végezd
      ha szín[u] = FEHÉR akkor
         MÉLYSÉGI_MENET(G,u)
      vége ha
   vége minden
vége MÉLYSÉGI_BEJÁRÁS

eljárás MÉLYSÉGI_MENET(G,u)
   kiír: u
   szín[u] ← SZÜRKE
   idő ← idő + 1
   elér[u] ← idő
   minden v ∈ Szomszéd(u) végezd
      ha szín[v] = FEHÉR akkor
         apa[v] ← u
         MÉLYSÉGI_MENET(v)
      vége ha
```

```
        vége minden
        szín[u] ← FEKETE
        idő ← idő + 1
        elhagy[u] ← idő
vége MÉLYSÉGI_MENET
```

A 3.5.a, b ábrák a két példagráf mélységi fáit ábrázolják, amelyeket az algoritmus az *apa*-tömbben kódol. Megfigyelhető, hogy míg a szélességi fák általában szélesek és „alacsonyak", addig a mélységi fák inkább mélyek és „karcsúak". Például a K_n irányítatlan teljes gráf szélességi fája két szint mélységű és $(n-1)$ pont „széles". Másfelől K_n mélységi fája egy n szintes „egyenes" „fa-ág".



3.5. ábra. *a) Irányított gráf mélységi fája; b) irányítatlan gráf mélységi fája*

Az alábbi táblázatok a 3.4.b irányítatlan példagráfra tartalmazzák az elérési és elhagyási időpontokat, illetve a mélységi fát kódoló apa-tömböt.

3.2. táblázat.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Elér	1	2	9	42	3	10	4	6	11	19	8	7	23	13	32	24	14	12	33	31	25	27
El-hagy	44	41	22	43	40	21	5	39	18	20	37	38	30	16	35	29	15	17	34	36	26	18

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Apa	0	1	11	1	2	3	5	5	6	6	12	8	11	18	20	13	14	9	15	11	16	16

Megjegyzés: Úgy is fogalmazhatnánk, hogy mélységi bejárás esetén mindig abból a szürke pontból lépünk tovább, amelyik legkésőbb vált szürkévé. Mivel erre az elvre épül a veremszerkezet is, ezért logikus, hogy ezt használjuk mint adatszerkezetet a mélységi bejárás implementálásánál. Mivel csak fehér pontok irányába lépünk tovább, nyilvánvaló, hogy a bejárt élek (feszítő)fát alkotnak.

Bonyolultság: A **MÉLYSÉGI_BEJÁRÁS** algoritmus bonyolultsága – amennyiben szomszédsági lista tárolja a gráfot $-O(n+m)$. (Bizonyítás végett lásd a [2], 415. oldalt.)

Az élek osztályozása (a mélységi bejárás nyomán)

Egy gráf (u, v) élei az alábbi módon osztályozhatók a mélységi bejárás nyomán:

- *Faél*: ha v-t először az (u, v) él vizsgálata nyomán értük el.
- *Visszamutató él*: ha v őse u-nak a mélységi fában (és ha (v, u) nem minősült már faélnek).
- *Előre mutató él*: ha v utóda u-nak a mélységi fában (és ha (u, v) nem minősült már faélnek).
- *Keresztél*: az összes többi él. Azokat az éleket kötik össze, amelyeknek végpontjai között nincs ős–utód, vagy utód–ős kapcsolat a mélységi fában (illetve mélységi erdőben).

Az (u, v) él osztálya meghatározható a v csúcs színe alapján:

- FEHÉR esetén *fa*él,
- SZÜRKE esetén *visszamutató* él,
- FEKETE esetén vagy *előre mutató él* (`ha elér[u]<elér[v]`), vagy *keresztél* (ha `elér[u]>elér[v]`).

Megjegyzés: Minden él akkor kap besorolást, amikor a mélységi bejárás *először* érzékeli a létezését.

A 3.4.a irányított gráfon megvastagítottuk a faéleket, szaggatott vonal jelöli a visszamutató éleket, pontvonal ábrázolja az egyetlen keresztélet (a (12, 8) él), és pontozott szaggatott vonal jelzi, hogy melyik az előre mutató él (ebből is egy van: (12, 20)). A mélységi bejárás a faéleken halad előre és ezeken is rükvercel vissza. A visszamutató éleket akkor érzékeli az algoritmus, amikor az aktuális pont (u) szomszédait (v) pásztázva valamelyiket szürkének találja.

3.2. tétel. Irányítatlan gráf mélységi keresésekor bármely él vagy faél, vagy visszamutató él. (Bizonyítás végett lásd a [2], 419. oldalt.)

A 3.4.b irányítatlan gráfon megvastagítottuk a faéleket, és szaggatott vonal jelöli a visszamutató éleket. A visszamutató éleket akkor érzékeli az algoritmus, amikor az aktuális pont (u) szomszédait (v) pásztázva valamelyiket szürkének találja. Kivételt képez az aktuális pont apa-pontja (`v = apa[u]`), amely bár szürke, az `(apa[u],u)` él már faélnek minősült.

Megjegyzés: Ha egy irányított gráfban töröljük az élek irányítását, akkor az ugyanabból a pontból indított mélységi bejárás is átminősíti az éleket. A keresztélek rendszerint faéllé alakulnak át, az előre mutatók pedig visszamutatóvá.

4. FEJEZET

A MÉLYSÉGI BEJÁRÁS ALKALMAZÁSAI

Mivel e fejezetben bemutatott algoritmusok mind a mélységi bejárás közvetlen alkalmazásainak tekinthetők, ezért mindegyiknek a bonyolultsága $O(n+m)$.

4.1. Zárójelezés

Egy gráf mélységi bejárása alkalmával a csúcsok elérési és elhagyási időpontjait zárójeles szerkezettel ábrázolhatjuk. Más szóval, ha minden szürkére festéskor nyitunk egy zárójelt és minden feketére váltáskor zárunk egyet, akkor n csomópont esetén egy $2n$ elemű helyesen záruló zárójelsort kapunk. (Bizonyítás végett lásd a [2], 417. oldalt.)

A 3.4.a ábrán szemléltetett mélységi bejáráshoz kacsolódó zárójelsor:
(((( ) ( ))) (((((( ) ( ))) ((( ) ( ))) (( ) ( )))) ( ))) ( ))

A 3.4.b ábrán szemléltetett mélységi bejáráshoz kacsolódó zárójelsor:
(((( ) ((((((( )))) ( ))) ((( ) ( ))) ((( )))))) ( ))

4.2. Körmentesség

4.1. lemma. Egy *irányított* gráf akkor és csakis akkor körmentes, ha mélységi bejárása során nem találunk visszamutató élt. (A lemma igaz irányítatlan gráfok esetén is.)

A lemma belátása: Egy (u, v) visszamutató él definíció szerint visszamutat u-nak egy v ősére, tehát körön van. A mélységi bejárás akkor érzékel visszamutató élt, ha az aktuális u pont v szomszédját szürkének találja. A v pont szürke színe arra utal, hogy v őse u-nak a mélységi fában. Ez azt jelenti, hogy létezik egy $(v \to u)$ „szürke faél út" v-től u-hoz. Tehát az (u, v) visszamutató él a $(v \to u, v)$ kör létezéséről tanúskodik. Irányítatlan gráfok esetén az (u, v) él csak akkor visszamutató, ha v az

apa[u] pontnál magasabban lévő őse u-nak (az (u,apa[u]) él már faélnek minősült az (apa[u],u) irányból).

Megjegyzés: A körmentesség ellenőrzése végett szükséges a MÉLYSÉGI_MENET eljárás minden ciklusának ha utasítását egy különben ággal kiegészíteni.

Irányított gráfok esetén:

```
...
különben
   ha szín[v] = SZÜRKE akkor
      körmentes ← HAMIS
   vége ha
...
```

Irányítatlan gráfok esetén:

```
...
   különben
   ha (szín[v] = SZÜRKE) ÉS (apa[v] ≠ u) akkor
      körmentes ← HAMIS
   vége ha
...
```

A fenti algoritmusok jól használhatók egy gráf mélységi fájához tartozó alapkörrendszerének meghatározására. Minden (u, v) visszamutató él kapcsán kiírjuk a $(v \to u, v)$ irányított/irányítatlan kört. (A $v \to u$ szakaszt az apa-tömbből olvashatjuk ki rekurzív eljárással.)

4.3. Topologikus rendezés

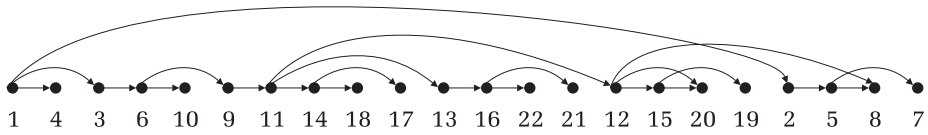
4.2. definíció. Topologikus sorrenden egy *irányított körmentes* gráf pontjainak olyan sorrendjét értjük, amelyben minden pont megelőzi a ki-szomszédait.

Ha egy irányított körmentes gráf mélységi bejárásakor az egyes csúcsokat az elhagyásukkor beszúrjuk egy lista elejére, akkor a lista topologikusan rendezve fogja tartalmazni a csúcsokat. Tehát a topologikusan rendezett csúcsok az elhagyási időpontok (elhagy tömb) fordított sorrendjében szerepelnek. (Az elsőnek elhagyott pont lesz az utolsó a topologikus listában.) (Bizonyítás végett lásd a [2], 422. oldalt.)

Ha egy gráf csomópontjait egy vízszintes mentén topologikus sorrendben helyezzük el, akkor a gráf összes éle „előre" fog mutatni. Ha a

gráf élei a pontok közötti alárendeltségi kapcsolatot ábrázolják, akkor a topologikus sorrend egyfajta hierarchikus sorrendet jelent.

Miért a pontok „fordított elhagyási sorrendje" jelenti a topologikus sorrendet? Egy pont elhagyásakor annak minden ki-szomszédja már fekete (ha lenne fehér ki-szomszédja, akkor nem hagynánk még el az illető pontot; ha lenne szürke ki-szomszédja, akkor a gráf tartalmazna kört). Más szóval, az elhagyás pillanatában minden ki-szomszédot már azt megelőzően elhagytunk, azaz már bekerült az épülő topologikus listába. Tehát, ha a szóban forgó pontot a lista elejére szúrjuk be, akkor ez minden ki-szomszédja elé kerül. (Minden (u, v) élnek a végpontja (v) hamarabb lesz fekete, mint a kezdőpontja (u).)



4.1. ábra. *A 3.4.a gráf csomópontjainak topologikus sorrendje (amennyiben eltekintünk a szaggatott vonalakkal jelölt visszamutató élektől)*

4.4. Irányítatlan gráfok összefüggősége

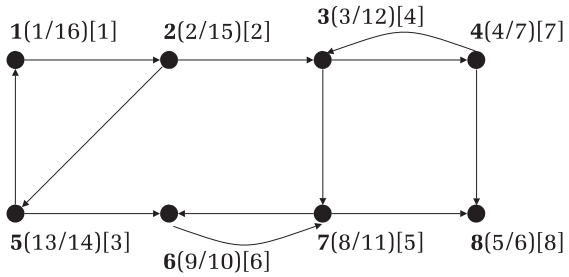
Egy irányítatlan gráf összefüggő komponenseinek számát az adja meg, hogy hányszor kerül meghívásra a `MÉLYSÉGI_BEJÁRÁS` eljárás keretén belül az `MÉLYSÉGI_MENET` eljárás. A `MÉLYSÉGI_MENET` eljárás minden meghívása „letapogat" egy komponenst.

4.5. Irányított gráfok erősen összefüggősége

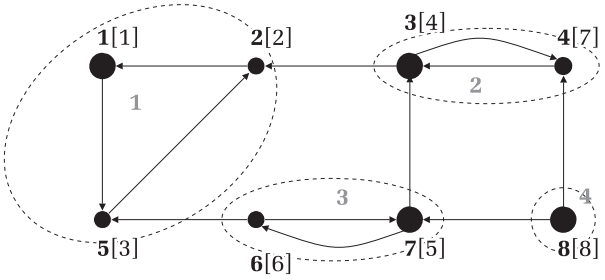
Legyen G^T G transzponált gráfja, amelyet úgy kapunk, hogy G-ben minden él irányítását megfordítjuk. G és G^T erősen összefüggő komponensei megegyeznek.

4.3. tétel. Egy irányított gráf erősen összefüggő komponensei az alábbi algoritmus segítségével határozhatók meg: (bizonyítás végett lásd a [2], 427. oldalt)

– A `MÉLYSÉGI_BEJÁRÁS(G)` hívás által minden u csúcsra meghatározzuk az `elhagy[u]` elhagyási időpontot.
– G^T meghatározása: felépítjük G szomszédsági listájából G^T szomszédsági listáját.
– `MÉLYSÉGI_BEJÁRÁS(G^T)` hívása úgy, hogy az eljárás fő ciklusában a csúcsokat az `elhagy[u]` szerint csökkenő sorrendben vizsgáljuk. Ezen bejárás nyomán kapott mélységi erdő fáinak a csúcsai éppen az erősen összefüggő komponensek lesznek.



4.2. ábra. *Irányított gráf 1-es pontjából induló mélységi bejárása nyomán az elérési/elhagyási időpontok (kerek zárójelben), valamint a topologikus sorrend (szögletes zárójelben)*



4.3. ábra. *A 4.2. gráf transzponált gráfja. Minden pont mellett az eredeti gráf mélységi bejárása szerinti topologikus sorrend. A pontozott foltok a transzponált gráf mélységi bejárása topologikus sorrend szerint meghívott mélységi menetei által letapogatott részgráfokat jelölik. Szürke számok jelölik a letapogatás sorrendjét. Megnagyítottuk azokat a pontokat, amelyekből sor kerül a `MÉLYSÉGI_MENET` eljárás meghívására. A pontozott foltokkal jelölt részgráfok egyben az eredeti gráf erősen összefüggő komponensei.*

Megjegyzés: Lássuk át az előbbi algoritmus mögötti gondolatmenetet. Egy gráf bármely pontjára igaz, hogy azok a pontok vannak vele

egy erősen összefüggő komponensben, amelyek vele egy körön vannak. Ebből következően, azok a pontok, amelyek nincsenek egyetlen körön sem, külön-külön erősen összefüggő komponensek lesznek. A fenti algoritmus az alábbi észrevételeken alapszik:

- Ha egy körmentes gráf pontjait topologikus sorrendbe állítjuk, akkor a gráf élei mind „előre" mutatnak. Ha megfordítjuk az élek irányítását, és a mélységi bejárás `MÉLYSÉGI_MENET` eljárását a pontok topologikus sorrendje szerint hívogatjuk meg, akkor minden menetben egypontú komponenseket fogunk „letapogatni". Mindez úgy tekinthető, hogy meghatároztuk a körmentes gráf erősen összefüggő komponenseit (mint n darab egypontú fából álló erdőt).
- A körök egyik jellegzetessége, hogy bármelyik pontjukból induljunk is ki, el tudunk jutni az összes többi pontjukba. Továbbá, ha megfordítjuk egy kör éleinek irányítását, akkor egy másik körhöz jutunk, amelyre ugyanúgy érvényes az előbbi tulajdonság.
- Emlékezzünk, hogy a topologikus sorrend nem más, mint a pontok mélységi bejárás nyomán nyert feketévé válási (elhagyási) sorrendjének a fordítottja. Bár általános esetben (a gráf tartalmazhat kört is) nem beszélhetünk a pontok topologikus sorrendjéről, mélységi bejárással minden gráfban felállítható a pontoknak egy úgynevezett „fordított elhagyási sorrendje".

Ezek után nem nehéz átlátni, hogy ha megfordítjuk egy gráf éleinek irányítását, és a mélységi bejárás `MÉLYSÉGI_MENET` eljárását a fentebb értelmezett „fordított elhagyási sorrendben" hívogatjuk, akkor algoritmusunk minden menetben a gráf egy erősen összefüggő komponensét tapogatja le. Valahányszor a `MÉLYSÉGI_MENET` eljárás kezdőpontja körön van, az eljárás a megfordított éleken keresztül is eléri az illető ponttal egy körön levő összes pontot (más szóval, letapogatja a szóban forgó pontot tartalmazó erősen összefüggő komponenst). Másfelől, azon kezdő pontok esetében, amelyek nincsenek körön – a „fordított elhagyási sorrend" szerinti hívások miatt (ami esetükben topologikus sorrendet jelent) –, minden pont külön komponensként lesz azonosítva.

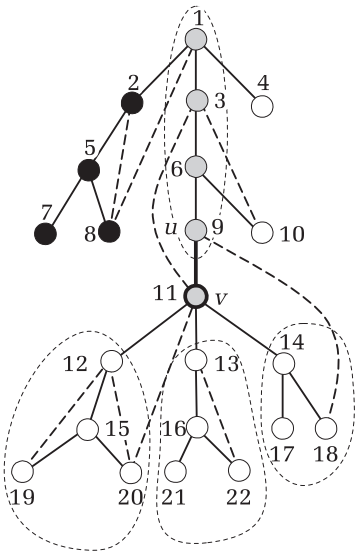
4.6. Elvágó élek (hidak) meghatározása

4.4. definíció. Elvágó él egy *irányítatlan* gráf azon éle, amelynek eltávolításával nő az összefüggő komponenseinek a száma.

Stratégia: Egy (u, v) él akkor elvágó, ha a mélységi bejárás faéle (visszamutató él nem lehet elvágó, mert cikluson van) és a mélységi fában a mélyebben levő végpontjához (v) tartozó részfában (amelynek v a gyökere) nincs olyan csúcs, amelyből lenne visszamutató él a szóban forgó él magasabban levő végpontjához (u) vagy ennek valamely őséhez (egy ilyen él körre helyezné (u, v)-t). Ez azt jelenti, hogy v-ből nem mutat vissza él v-nek u-nál magasabb őséhez, és v-nek egyetlen fiú-részfájából sem mutat vissza él v bármely őséhez. Ezen információ v elhagyási pillanatában áll leghamarabb rendelkezésre (miután v minden szomszédját megvizsgáltuk és minden fiú-részfájából visszajöttünk), ezért ekkor dönthető el, hogy (u, v) elvágó él vagy sem. (A v pont ősei a mélységi fában a nála magasabb szinteken szürkén hagyott csomópontok.)

A fenti stratégia „miért?"-jének átlátása végett gondoljuk végig az alábbiakat (lásd a 4.4. ábrát): Az (u, v) él eltávolítása után v fiú-részfái továbbra is kapcsolatban maradnak egymással a közös apa-pontjuk (v) révén. Ezért csak annak lehetősége áll fenn, hogy az (u, v) él eltávolításával v ősei (mint összefüggő részgráf: (1, 3, 6, *9*) plusz egyéb leszármazottaik (2, 5, 7, 8, 4, 10)) „elszakadnak" a v és leszármazottjai alkotta összefüggő részgráftól (**11**, (*12*, 15, 19, 20), (*13*, 16, 21, 22), (*14*, 17, 18)). Ez csak akkor nem történik meg, ha v-ből, vagy bármelyik fiú-részfájából, létezik legalább egy visszamutató él, amely úgymond megduplázza az (u, v) „kapocs-élet". Ilyenek a (11, 3) és a (18, 9) visszamutató élek. (Irányítatlan gráfoknak nincs előremutató vagy keresztélük.)

Algoritmus: Az algoritmus egy módosított mélységi bejárás, amelyet függvényre változtattunk. Az `ELVÁGÓ_ÉL` függvény paraméterként az aktuális pontot (v) kapja meg és annak szintjét (*sz*) a mélységi fában (a kiindulási pont, azaz a gyökér szintje nulla). A függvény visszatéríti v legmagasabb ősének szintjét, ahova létezik a v gyökerű mélységi részfa valamelyik pontjából visszamutató él (RFminVM – a v gyökerű **R**ész**F**a **min**imum szintre **V**issza**M**utató éle). Észrevesszük, hogy (u, v) faél, tehát (v, u) nem lehet visszamutató. A függvény visszatéríti v szürke szomszédainak (kivéve u-t) szintjei, illetve a v fiaira vonatkozó rekurzív hívások által visszatérített értékek közötti minimumot. Ha a v gyökerű részfából nincs v-nél magasabbra visszamutató él, akkor a függvény végtelent (∞) térít vissza. Ez egyúttal azt is jelenti, hogy az (u, v) él híd (kivéve, ha v a mélységi bejárás kiindulópontja; a kiindulópont nem lehet egy (u, v) él v, azaz mélyebben lévő pontja).



4.4. ábra. *Elvágó élek meghatározása*

```
függvény ELVÁGÓ_ÉL(v,sz)
                    // v: aktuális pont; sz: aktuális szint
   szín[v] ← SZÜRKE
   szint[v] ← sz
   RFminVM ← ∞
   minden w ∈ Szomszéd(v) végezd
      ha szín[w] = FEHÉR akkor       // w utódja (fia) v-nek
      apa[w] ← v
      UminVM ← ELVÁGÓ_ÉL(w,sz+1)
      ha UminVM < sz akkor
      //      v-nek w gyökerű fiú-részfájának visszamutató éle
      // (ha egyáltalán létezik) v-re nézve is visszamutató
         ha UminVM < RFminVM akkor
            RFminVM ← UminVM
         vége ha
      vége ha
      különben
         ha szín[w] = SZÜRKE akkor           // w őse v-nek
            ha szint[w] < sz-1 akkor
                              // w u-nál magasabb őse v-nek
               ha szint[w] < RFminVM akkor
                  RFminVM ← szint[w]
```

```
                vége ha
             vége ha
          vége ha
       vége ha
    vége minden
    szín[v] ← FEKETE
    ha apa[v] ≠ 0 akkor                  // v nem a kiinduló pont
       ha RFminVM = ∞ akkor
          kiír: apa[v],v
       vége ha
    vége ha
    vissza RFminVM
vége ELVÁGÓ_ÉL
```

4.7. Elvágó pontok meghatározása

4.5. definíció. Elvágó pont egy *irányítatlan* gráf azon csúcsa, amelynek eltávolításával (nyilván a hozzá tartozó élekkel együtt) nő a gráf összefüggő komponenseinek száma.

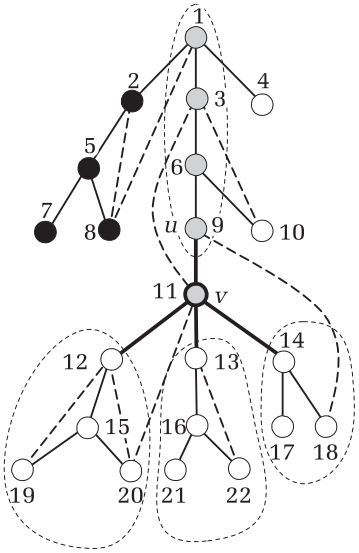
Stratégia (algoritmus):

- Egy csúcs akkor elvágó, ha a mélységi fában létezik olyan fiú-részfája, amelyből nem mutat vissza él az illető csomópont valamelyik őséhez (kivéve, ha a pont éppen a gyökér és így nincs őse). Ez esetben a pont eltávolításakor a szóban forgó fiú-részfa „leszakad" a gráfról.
- A mélységi bejárás kiindulópontja (illetve pontjai, amennyiben a gráf nem összefüggő) akkor elvágó, ha van legalább két fiú-részfája, ugyanis a pont (gyökér) eltávolításával ezek „szétszakadnak".

A fenti stratégia „miért?"-jének átlátása végett gondoljuk végig az alábbiakat (lásd a 4.5. ábrát): A v pontnak és a rá illeszkedő éleknek az eltávolítása után v fiú-részfái között megszűnik a v ponton keresztüli kapcsolat. Az egyetlen „remény" arra, hogy a gráf (bejárás alatt levő komponense) mégis összefüggő maradjon, hogy v minden fiú-részfájából mutasson vissza él v őseihez. Ily módon e fiú-részfák egymással is kapcsolatban maradnak az őseiken keresztül. Ha valamelyik fiú-részfából nincs visszamutató él v őseihez, akkor v eltávolításakor ez leszakad a

gráfról. A (*12*, 15, 19, 20) és (*13*, 16, 21, 22) fiú-részfák ebben a helyzetben vannak.

Különösen oda kell figyelnünk a (20, 11) típusú visszamutató élekre. Bár ez az él „valódi" visszamutató éle a (*12*, 15, 19, 20) fiú-részfának (az `ELVÁGÓ_PONT(12,5)` függvényhívás 4-et térít vissza, azaz a 11-es pont szintjét), nem „valódi" visszamutató él „v-re nézve" is. A fiú-részfára nézve azért valódi, mert annak gyökerénél (12) magasabbra mutat vissza. A v gyökerű részfára nézve viszont nem valódi, mert nem mutat v-nél is magasabbra vissza. Ezért v (11) eltávolításakor a (20, 11) „ál" visszamutató él is eltávolítódik. Ál, mert a valóságban nem duplázza meg a (*9*, **11**, *12*) ős–utód kapcsolatot.



4.5. ábra. *Elvágó pontok meghatározása*

```
függvény ELVÁGÓ_PONT(v,sz)
                    // v: aktuális pont; sz: aktuális szint
   szín[v] ← SZÜRKE
   szint[v] ← sz
   RFminVM ← ∞
   leszakadt_fiúk = 0                                        *
   fiúk = 0                                                  *
   minden w ∈ Szomszéd(v) végezd
      ha szín[w] = FEHÉR akkor       // w utódja (fia) v-nek
         fiúk = fiúk + 1                                     *
```

```
            apa[w] ← v
            UminVM ← ELVÁGÓ_PONT(w,sz+1)
            ha UminVM < sz akkor
            // v-nek w gyökerű fiú-részfájának visszamutató éle (ha
                                                              //
    egyáltalán létezik) v-re nézve is visszamutató
               ha UminVM < RFminVM akkor
                  RFminVM ← UminVM
               vége ha
            különben
             // a szóban forgó fiú-részfa elszakad v őseitől *
              leszakadt_fiúk = leszakadt_fiúk + 1            *
            vége ha
         különben
            ha szín[w] = SZÜRKE akkor            // w őse v-nek
               ha szint[w] < sz-1 akkor
                                 // w u-nál magasabb őse v-nek
                  ha szint[w] < RFminVM akkor
                     RFminVM ← szint[w]
                  vége ha
               vége ha
            vége ha
         vége ha
      vége minden
      szín[v] ← FEKETE
      ha apa[v] ≠ 0 akkor             // v nem a kiindulópont *
         ha leszakadt_fiúk > 0 akkor                          *
            kiír: v, leszakadt_fiúk                           *
         vége ha                                              *
      különben                  // v a kiindulópont (a gyökér) *
         ha fiúk > 1 akkor                                    *
            kiír: v, fiúk                                     *
         vége ha                                              *
      vége ha                                                 *
      vissza RFminVM
   vége ELVÁGÓ_PONT
```

Megjelöltük '*' karakterrel azokat a sorokat, amelyekben az ELVÁGÓ_ÉL és ELVÁGÓ_PONT algoritmusok különböznek egymástól.

Megjegyzés: Megfigyelhető, hogy a szélességi és mélységi bejárások alkalmazásai három kategóriába sorolhatók:

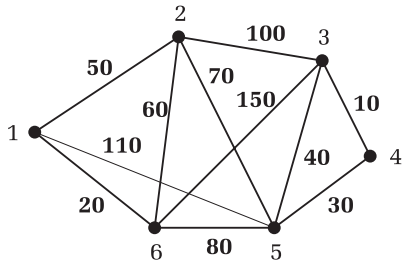
– Egyes esetekben nem kell egyebet tenni, mint egy az egyben alkalmazni valamelyik bejárást (irányítatlan gráfok összefüggősége).
– Néha arra van szükség, hogy hasznosítsuk a bejárások nyomán szerezhető többletinformációt (topologikus sorrend előállítása).
– Vannak olyan esetek is, amikor módosítanunk kell valamelyik bejárást ahhoz, hogy megoldjuk a feladatot (elvágó élek és pontok).

5. FEJEZET

MINIMÁLIS SÚLYÚ FESZÍTŐFÁK (IRÁNYÍTATLAN ÖSSZEFÜGGŐ SÚLYOZOTT GRÁFOKBAN)

Gyakorlati probléma: Minden olyan feladat megoldása, amelyben egy bizonyos szolgáltatási központnak adott fogyasztókat kell legolcsóbban ellátnia, és amelyben az ellátás nem történhet körforgalomban, a feladat megoldása az azt képviselő gráf minimális költségű feszítőfájának meghatározása. Ilyen feladat például: vízvezeték-rendszerek, telefonhálózatok, gázvezeték-hálózatok kiépítése a leggazdaságosabban, vagyis kiküszöbölve minden fölösleges vezeték lefektetését.

Példagráf a fejezetben található algoritmusok szemléltetésére:



5.1. ábra. *Súlyozott irányítatlan gráf*

Legyen $G = (V, E)$ egy összefüggő irányítatlan gráf, amelyhez a *súly*: $E \to \mathbb{R}$ súlyfüggvényt rendeljük. Határozzuk meg G minimális súlyú feszítőfáját. Két algoritmust adunk erre a feladatra, amelyek Kruskal és Prim matematikusok nevéhez fűződnek.

5.1. Kruskal algoritmusa

Stratégia: Gondolatban eltávolítjuk a gráf összes élét, majd súlyuk szerint növekvő sorrendben visszatesszük őket, átugorva azokat,

amelyek kört hoznának létre. (Tipikusan mohó algoritmus; lásd a B. függeléket.)

```
függvény KRUSKAL(G)
   A = ∅
   minden u ∈ V(G) végezd
      HALMAZT_KÉSZÍT(u)
              // minden pontot külön halmazokba helyezünk el
   vége minden
   RENDEZÉS_SZÚLY_SZERINT(E(G))
   minden (u,v) ∈ E(G) végezd}
      ha HOLVAN(u) ≠ HOLVAN(v) akkor
                          // u és v más-más halmazban vannak
         A = A ∪ {\{}(u,v){\}}
         UNIÓ(u,v)           // egyesítjük u és v halmazait
      vége ha
   vége minden
   vissza A
vége KRUSKAL
```

A minimális súlyú feszítőfát az A halmaz fogja tartalmazni mint éllistát.

Mivel gondolatban minden élet eltávolítottunk, kezdetben a gráf csúcsai úgy foghatók fel mint különálló fák. Úgy is mondhatnánk, hogy van egy n darab egycsúcsú fából álló erdőnk. Ezen erdő egyes fáihoz tartozó csúcsokat diszjunkt halmazokban tároljuk. Kezdetben természetesen minden csúcs külön halmazba kerül (`HALMAZT_KÉSZÍT eljárás`). A soron következő él akkor nem alkot kört a már visszahelyezettekkel, ha végpontjai két külön fához tartoznak, azaz más-más halmazban vannak (ezt a `HOLVAN` függvény segítségével ellenőrizzük). Az él visszahelyezésével viszont egyesül az illető két fa. Ezt úgy valósítjuk meg, hogy egyesítjük a végpontok halmazait (`UNIÓ` eljárás). Végül az erdő fái egyetlen fává egyesülnek, melyet a visszahelyezett élek alkotnak. Ez lesz a keresett minimális súlyú feszítőfa.

Bonyolultság: Az `UNIÓ-HOLVAN` adatszerkezet alkalmazása „úttömörítés" heurisztikával és gyorsrendezéssel együtt $O(m \lg m)$ komplexitást biztosít a Kruskal algoritmusnak ([2], 439. oldal).

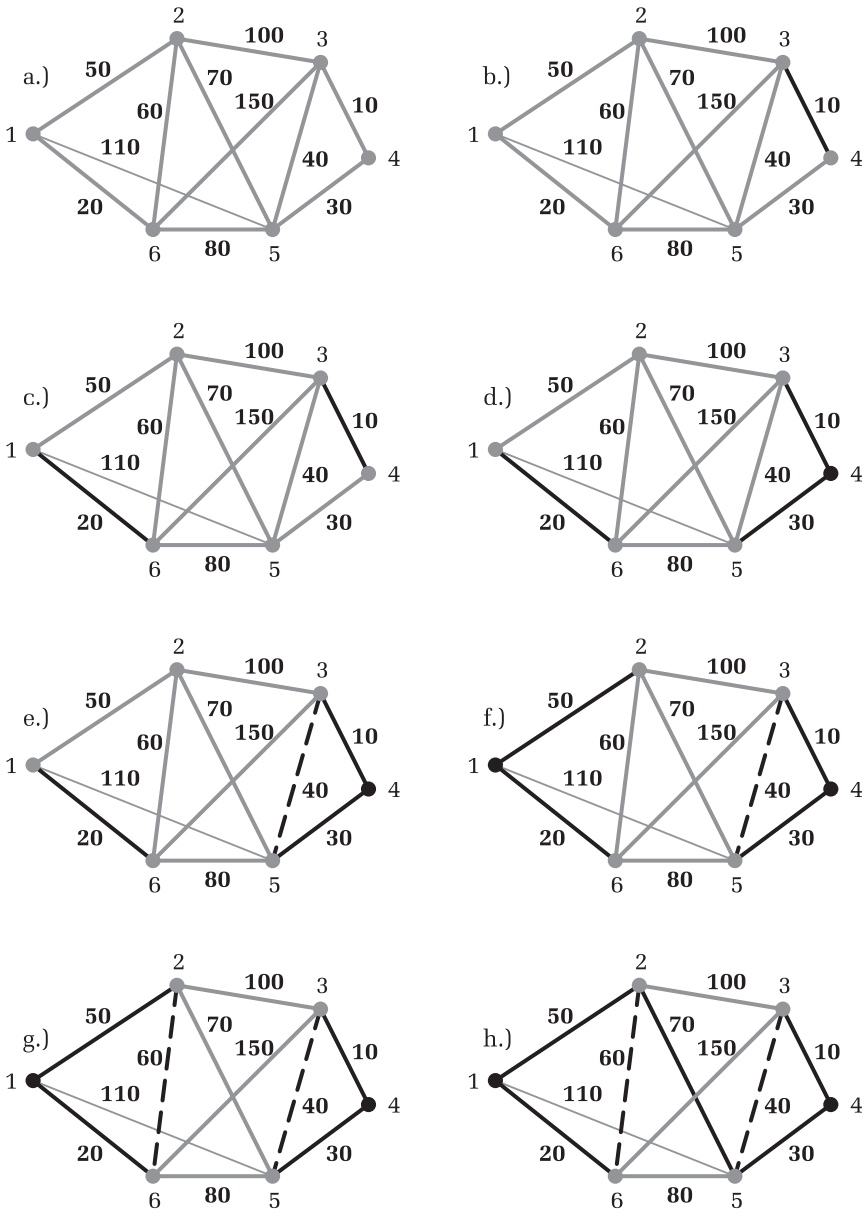
Az alábbiakban bemutatunk egy egyszerűbb implementációt. A `fa[1..n]` tömb fogja nyilvántartani, hogy az egyes csomópontok az erdőnek éppen melyik fájához tartoznak (az i-edik csomópont a `fa[i]` fához). Kezdetben minden csomópont külön fa (az i-edik pont az i-edik fa: `fa[i] = i`). Két fa egyesítését az egyesít eljárás valósítja meg. Az éleket, mint

éllistát, az élek[1..m] bejegyzés-tömb tárolja. Minden élről eltároljuk a kezdőpontját, a végpontját és a súlyát. Az alábbi eljárás kiírja a G gráf minimális súlyú feszítőfáját alkotó éleket, ezek súlyát, illetve a feszítőfa összsúlyát.

```
eljárás KRUSKAL(G)
   minden i ← 1,n végezd
      fa[i] ← i
   vége minden
   RENDEZÉS_SZÚLY_SZERINT(élek,m)
   össz_súly ← 0
   minden i ← 1,m végezd
      egyik = fa[élek[i].u]
      másik = fa[élek[i].v]
      ha egyik ≠ másik akkor
         össz_súly ← össz_súly + élek[i].súly
         kiír: élek[i].u, élek[i].v, élek[i].súly
         EGYESÍT(fa,n,egyik,másik)
      vége ha
   vége minden
   kiír: össz_súly
vége KRUSKAL

eljárás EGYESÍT(fa,n,egyik,másik)
   minden i ← 1,n végezd
      ha fa[i] = másik akkor
         fa[i] ← egyik
      vége ha
   vége minden
vége EGYESÍT
```

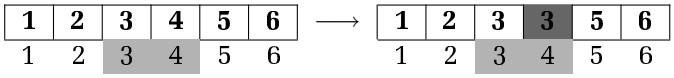
Az 5.2.a–h ábrasorozathoz tartozó fa-tömbök:
Az 5.2.a ábrához: (minden pont külön fa; n fájú erdő)

1	**2**	**3**	**4**	**5**	**6**
1	2	3	4	5	6

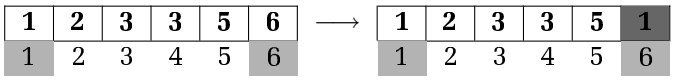


5.2. ábra. *A Kruskal-algoritmus lépéssorozata. A minimális súlyú feszítőfa éleit megvastagítottuk. A „kihagyott" éleket (a körök elkerülése végett) szaggatott vonallal jelöltük.*

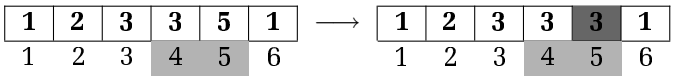
Az 5.2.b ábrához: (a 3-as és 4-es pontok fáinak egyesítése)



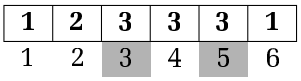
Az 5.2.c ábrához: (az 1-es és 6-os pontok fáinak egyesítése)



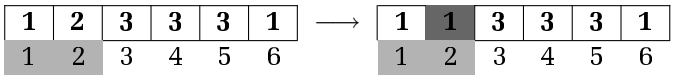
Az 5.2.d ábrához: (a 4-es és 5-ös pontok fáinak (a 3-as és 5-ös fa) egyesítése)



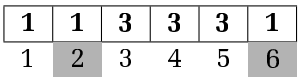
Az 5.2.e ábrához: (a 3-as és 5-ös pontok ugyanahhoz a fához (3-as) tartoznak)



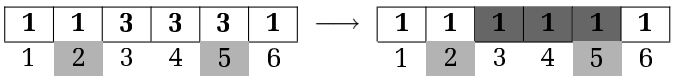
Az 5.2.f ábrához: (az 1-es és 2-es pontok fáinak egyesítése)



Az 5.2.g ábrához: (a 2-es és 6-os pontok ugyanahhoz a fához (1-es) tartoznak)



Az 5.2.h ábrához: (a 2-es és 5-ös pontok fáinak (az 1-es és 3-as fa) egyesítése)



5.2. Prim algoritmusa

Stratégia: Egy adott r csúcsból kiindulva, r gyökerű faként építjük fel a minimális súlyú feszítőfát, minden lépésben a fához kapcsolódó élek közül a minimális súlyúval növelve azt. (Tipikusan mohó algoritmus; lásd a B. függeléket.)

Definíció: Legyen $A \subset V(G)$. A $(V - A, A)$ halmazpárt a G gráf egy *vágás*ának nevezzük, amely a gráf azon $(u, v) \in E(G)$ éleit tartalmazza, amelyekre $u \in V - A$ és $v \in A$. Tekintsük a $V - A$ halmazt a vágás „bal oldalának", az A halmaz pedig legyen a vágás „jobb oldala".

```
függvény PRIM(G,r)
   Q ← V(G)
   minden v ∈ Q végezd
      táv[v] ← ∞
   vége minden
   táv[r] ← 0
   apa[r] ← 0
   amíg Q ≠ ∅ végezd
      u ← KIVESZ_MIN(Q)
      minden v ∈ szomszéd(u) végezd
         ha (v ∈ Q ÉS (súly(u,v) < táv[v])) akkor
            apa[v] ← u
            táv[v] ← súly(u,v)
         vége ha
      vége minden
   vége amíg
   vissza apa
vége PRIM
```

Q – a csúcsoknak egy elsőbbségi sora a táv tömbbeli értékeik alapján. Egy adott pillanatban azokat a csúcsokat tartalmazza, amelyeket még nem kapcsoltunk a fához (tehát kezdetben az összes csúcsot). Így a $V - Q$ halmaz fogja tartalmazni a már a fához csatolt csúcsokat (kezdetben üres halmaz). A $(V - Q, Q)$ vágás bal oldalán van a „növekvő" fa, a jobb oldalán találhatók pedig a csatolandó pontok.

`táv[v]` – a Q-beli v csúcs fához kapcsolódó élei közül (amelyek a $(V - Q, Q)$ vágáshoz tartoznak) a legkisebb súlyú él súlyát tárolja. Ha nincs ilyen él, akkor `táv[v]` = ∞. Kezdetben úgy tekintjük, mintha minden csúcs ∞ távolságra lenne a még nem létező fától, kivéve az r kiindulópontot, melynek távolságát 0-nak vesszük

(ez garantálja, hogy elsőként az r csúcsot tesszük be a fába). Azért választottuk a táv nevet, mert a Q-beli csúcsok esetén táv[v] a v csúcsnak a fától mért „távolságát" jelenti.

Újabb csúcsot csatolni a kialakulóban lévő fához – a „legközelebbit" (mohó stratégia szerint) – azt jelenti, hogy az illető csúcsot töröljük Q-ból, és ezzel ez a csúcs implicite átkerül a $(V - Q)$ fába. Ez nyilván maga után vonja azt is, hogy az illető csúcsot a Q-ban maradt szomszédaihoz kapcsoló élek a vágás részévé válnak (azok az élek pedig, amelyek a fához kötötték, automatikusan kikerülnek a vágásból). Mindez szükségessé teszi a táv tömb frissítését, ugyanis megtörténhet, hogy az újonnan vágásba került éleken keresztül egyes Q-beli csúcsok „közelebb" kerülnek a fához.

`KIVESZ_MIN(Q)` – meghatározza a táv tömb alapján a fához „legközelebb" lévő Q-beli csúcsot (azt a csúcsot, amelyet az aktuális *vágás* legkisebb súlyú éle köt a fához), és a fához kapcsolja (törli Q-ból). Az apa-tömbben regisztráljuk, hogy a fának melyik csúcsához csatoltuk az új csúcsot.

`apa[v]` – amíg a `v` csúcs még Q-ban van, azon fabeli szomszédját tárolja, amelyikhez az aktuális vágás legkisebb súlyú éle köti. Valahányszor a `v` csúcsnak egy hozzá még „közelebb" lévő szomszédját a fához csatoljuk, frissítenünk kell az `apa[v]` értékét ezzel a szomszéddal. Amikor `v` végül bekerül a fába, `apa[v]` a fabeli „apját" fogja tárolni.

Az algoritmus végén az apa-tömb minden csúcsnak a minimális súlyú feszítőfabeli apját fogja tárolni, tehát a keresett fa apa-mutatók általi ábrázolása lesz. Ezért téríti vissza a függvény eredményként az apa-tömböt.

Megfigyelhető, hogy az algoritmus alatt a vágás úgymond végigvonul a gráfon, minden pillanatban határt képezve a növekvő fa és a gráf többi csúcsa között.

Bonyolultság: Ha a Q elsőbbségi sort bináris kupacként implementáljuk, akkor a Prim algoritmus $O(m \lg n)$ bonyolultságú lesz ([2], 440. oldal).

Az alábbiakban a `Prim` algoritmus egy lehetséges implementációját mutatjuk be. A `fa[1..n]` tömb nyilvántartja, hogy mely pontok tartoznak a „növekvő" fához, illetve melyek várnak csatolásra. Ha az `i` pont már része a fának, akkor `fa[i]` értéke 1, különben 0. A gráf éleit az `élek[1..m]` bejegyzéstömb tárolja mint éllistát. Minden élről eltároljuk a kezdőpontját (u mező), a végpontját (v mező), a súlyát (*súly* mező), illetve azt, hogy

része-e az aktuális vágásnak vagy nem (*vágás* mező). Az élek tömböt az élek súlya szerint növekvő sorrendbe rendezzük.

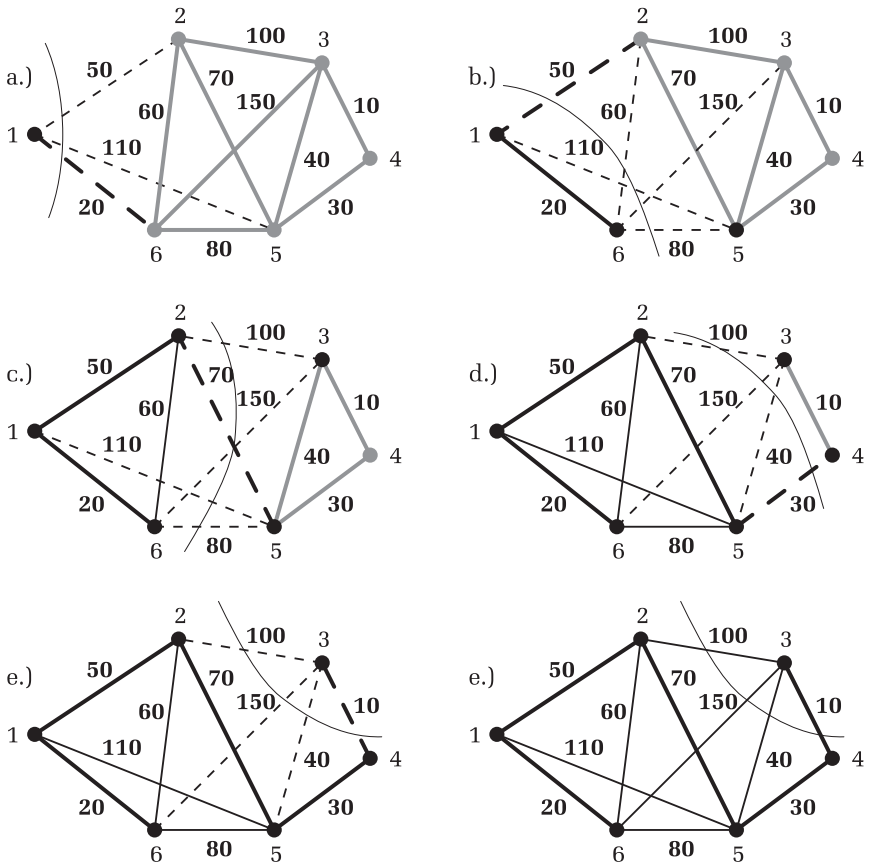
Az algoritmus hatékonyságának növelése érdekében a gráfot szomszédsági listaként is ábrázoljuk, és felépítünk egy szomszédsági mátrixot is (`SZ_M[1..n][1..n]`). Az `SZ_M[i][j]` és `SZ_M[j][i]` tömbelemek az (i, j) él éllistabeli sorszámát tárolják. A szomszédsági listát az `Sz_L[1..n]` bejegyzés-tömb tárolja. Az `Sz_L[i].fokszám` mező az `i`-edik pont fokszámát tartalmazza, az `Sz_L[i].szomszéd[]` tömb-mező pedig az `i`-edik pont szomszédainak listáját.

Kezdetben a fa az `r` pontból áll, az aktuális vágás pedig az `r`-re illeszkedő éleket tartalmazza. Minden lépésben (összesen `n-1`) három műveletet hajt végre az algoritmus:

1. *Kiválasztjuk a vágás minimális súlyú élét.* Mivel az élek tömb rendezett, a minimális súlyú vágásbeli él az élek tömb vágáshoz tartozó elemei közül az első lesz (`KIVESZ_VÁGÁSBÓL_MIN` eljárás).
2. *A fához csatolja a kiválasztott élet.* Ez úgy is felfogható, hogy csatoljuk (az illető élen keresztül) a fához legközelebb eső csomópontot (`csatolt_pont`). Az algoritmus szempontjából ez annyit jelent, hogy a fa tömb megfelelő elemét 1-re állítjuk.
3. *Aktualizálja a vágást.* Az előbbi művelet úgy tekinthető, mintha a fához legközelebb eső pontot (a csatolása révén) áthúznánk a vágás bal oldalára. Következésképpen: egyfelől kikerülnek a vágásból azok az élek, amelyek az illető pontot a fában található szomszédaihoz kötik, másfelől viszont bekerülnek a vágásba a pontnak a vágás jobb oldalán maradt szomszédaihoz vezető élei. Az `sz_pont` változóba sorra bekerülnek a csatolt pont szomszédai, az `sz_él` változóba pedig az ezekhez vezető élek (pontosabban az illető éleknek az élek tömbbeli sorszámai).

A `PRIM` eljárás kiírja a minimális súlyú feszítőfa éleit, illetve összsúlyát. (Mellesleg az `apa`-tömböt is felépíti.)

Az 5.1. táblázatok a `Prim` algoritmust követik nyomon az 5.1. példagráfra. A táblázatok az 5.3. ábrákkal összehangoltan mutatják be a `d` és `apa`-tömbökben lépésről lépésre végbemenő változásokat (6 lépés).



5.3. ábra. *A* `Prím` *algoritmus. Jelölések: görbe vonal – a vágás; folytonos fekete vonalak – a vágásból kikerült („bal oldali") élek; vastagított folytonos vonalak – a fa élei; szaggatott vonalak – vágásbeli élek; vastagított szaggatott vonal – a vágás minimális súlyú éle; szürke élek – amelyek még nem kerültek be a vágásba („jobb oldali" élek).*

5.1. táblázat. *Fekete háttér: az illető pont már a minimális súlyú feszítőfához tartozik. Fehér/szürke háttér: az illető pont még nem része a minimális súlyú feszítőfának. Szürke háttér: a minimális súlyú feszítőfához legközelebb eső pont. Félkövér értékek: a csatolt ponton keresztül frissített távolságértékek.*

	1	2	3	4	5	6		1	2	3	4	5	6
táv	0	∞	∞	∞	∞	∞	⟶	0	**50**	∞	∞	**110**	**20**
apa	0	0	0	0	0	0		0	**1**	0	0	**1**	**1**

	1	2	3	4	5	6		1	2	3	4	5	6
táv	0	50	∞	∞	110	20	⟶	0	50	**150**	∞	**80**	0
apa	0	1	0	0	1	1		0	1	**6**	0	**6**	1

	1	2	3	4	5	6		1	2	3	4	5	6
táv	0	50	150	∞	80	0	⟶	0	0	**100**	∞	**70**	0
apa	0	1	6	0	6	1		0	1	**2**	0	**2**	1

	1	2	3	4	5	6		1	2	3	4	5	6
táv	0	50	100	∞	70	0	⟶	0	0	**40**	**30**	0	0
apa	0	1	2	0	2	1		0	1	**5**	**5**	2	1

	1	2	3	4	5	6		1	2	3	4	5	6
táv	0	50	40	30	0	0	⟶	0	0	**10**	0	0	0
apa	0	1	5	5	2	1		0	1	**4**	5	2	1

	1	2	3	4	5	6		1	2	3	4	5	6
táv	0	50	10	0	0	0	⟶	0	0	0	0	0	0
apa	0	1	4	5	2	1		0	1	4	5	2	1

```
eljárás PRIM(G,r)
                                        // a fa tömb inicializálása
   minden v∈Q végezd
      fa[v] ← 0
   vége minden
   fa[r] ← 1
   apa[r] ← 0
   RENDEZÉS_SZÚLY_SZERINT(élek,m)
       // az SZ_M mátrix felépítése; a vágás inicializálása
   minden i = 1,m végezd
      SZ_M[élek[i].u][élek[i].v] ← i
      SZ_M[élek[i].v][élek[i].u] ← i
      ha (élek[i].u = r) VAGY (élek[i].v = r) akkor
         élek[i].vágás ← 1
      különben
         élek[i].vágás ← 0
      vége ha
   vége minden
                          // Az algoritmus magva (n-1) lépésben
   össz_súly = 0
   minden lépés = 1,n-1 végezd
                                                    // 1. művelet
      i ← KIVESZ_VÁGÁSBÓL_MIN(élek,m)
      kiír: élek[i].u, élek[i].v, élek[i].súly
      össze_súly = össz_súly + élek[i].súly
                                                    // 2. művelet
      ha fa[élek[i].u] = 1 akkor
         csatolt_pont ← élek[i].v
         apa[csatolt_pont] ← élek[i].u
      különben
         csatolt_pont ← élek[i].u
         apa[csatolt_pont] ← élek[i].v
      vége ha
      fa[csatolt_pont] ← 1
                                                    // 3. művelet
      minden j ← 1,Sz_L[csatolt_pont].fokszám végezd
         sz_pont ← Sz_L[csatolt_pont].szomszéd[j]
         sz_él ← SZ_M[csatolt_pont][sz_pont]
         ha (fa[sz_pont] = 1) akkor
            élek[sz_él].vágás ← 0
         különben
            élek[sz_él].vágás ← 1
```

```
            vége ha
          vége minden
        vége minden
        kiír: össz_súly
    vége PRIM
```

Megjegyzés: Ha a `Prim` algoritmus klasszikus változatában (lásd a `PRIM` függvényt) a táv tömbben a pontoknak a kiindulóponttól (r – a fa gyökere) mért távolságát tároljuk, akkor ugyanezzel a gondolatmenettel meghatározhatók egy adott pontból az összes többi ponthoz vezető legrövidebb utak. Ezt valósítja meg *Dijkstra* algoritmusa, amelyet a következő fejezetben mutatunk be. Az így kialakuló legrövidebb utak fája természetesen különbözhet a `Prim` algoritmus minimális súlyú feszítőfájától (bár a gyökerük azonos lesz).

6. FEJEZET

EGY CSÚCSBÓL INDULÓ LEGRÖVIDEBB UTAK
(súlyozott *irányított* gráfokban)

Gyakorlati probléma: Nagy városokban a közszállítás javítását célzó matematikai módszerek egyik előkészítő fázisában az utasforgalom statisztikai felmérése után, a közszállítás szempontjából alkalmasnak ítélt úthálózatnak megfelelő gráfban érdekelnek a központi buszállomástól az összes többi ponthoz vezető legrövidebb utak.

Legyen $G = (V, E)$ egy irányított gráf, amelyhez a *súly*: $E \to \mathbb{R}$ súlyfüggvényt rendeljük. Egy út súlyát úgy definiáljuk mint az út menti élek súlyainak összegét. Az `s` és `v` csúcsok között a legrövidebb út súlyát `lru(s,v)` jelöli. Ha `s` és `v` között nincs út, akkor definíció szerint `lru(s,v)` $= \infty$. Ha az `s`-től `v`-hez vezető úton negatív összsúlyú kör található, akkor definíció szerint `lru(s,v)` $= -\infty$. (A súly és hossz fogalmakat egymással párhuzamosan használjuk.)

A bemutatásra kerülő mindhárom algoritmus a fokozatos közelítés technikáját alkalmazza, amelyet az alábbiakban mutatunk be:

Fokozatos közelítés

Ez a technika sajátosan használja ki az *optimalitás alapelvét*, amelyre különben a mohó és dinamikus programozás-algoritmusok is épülnek. Íme az optimalitás alapelve legrövidebb út problémák esetén: *Egy legrövidebb út részútjai is legrövidebb utak* (lásd a [2], 448. oldalt).

Következmény_1: A kevesebb élből álló legkisebb súlyú utakból felépíthetők a több élű legkisebb súlyú utak. Mindhárom algoritmus ezt a megközelítést alkalmazza.

Következmény_2: Az s pontból induló legrövidebb utak egy s gyökerű fát alkotnak (a G gráf egy feszítőfáját). Mindhárom algoritmus úgy határozza meg e legrövidebb utak fáját, mint amely az s pontból nő ki. Ez azt jelenti, hogy a már meghatározott legrövidebb utak folyamatosan fát alkotnak. A fa fokozatosan épül fel élről élre. Minden él egy újabb csomóponttal bővíti a fát.

Következmény_3: Ha s-ből v-be tartó legrövidebb úton u az utolsó előtti állomás, akkor

```
lru(s,v) = lru(s,u) + súly(u,v)
```

Következmény_4: Ha s a kezdőcsúcs, akkor bármely (u,v) élre fennáll, hogy

```
lru(s,v) ⩽ lru(s,u) + súly(u,v)
```

A negyedik következmény belátása: az s pontból a v ponthoz vezető legrövidebb út hossza (lru(s,v)) értelemszerűen kisebb vagy egyenlő bármelyik másik s-ből v-be vezető út hosszánál, illetvet hosszával. Tehát kisebb vagy egyenlő az egyenlőtlenség jobb oldalán lévő útnál, illetve úttal is.

Fokozatos közelítés technikája: Minden v csúcsra nyilvántartunk egy d[v] értéket, amely felső korlátja az s kezdőcsúcsból a v-be vezető legrövidebb út súlyának. Kezdetben mindenik csúcs „távolságát" s-től ∞-re becsüljük, kivéve s „távolságát" s-től, amely nyilván 0. Egy (u,v) éllel való közelítés a következőképpen történik:

```
ha d[v] > d[u] + súly(u,v) akkor
   d[v] = d[u] + súly(u,v)
   apa[v] = u
vége ha
```

Az alábbi algoritmusok addig végeznek az élekkel egymás után közelítéseket, míg a d tömb az s kezdőcsúcsból induló legrövidebb utak súlyait, az apa-tömb pedig a legrövidebb utak fájában az egyes csúcsok apa-csúcsait fogja tartalmazni.

Az algoritmusok csak abban különböznek, hogy hányszor és milyen sorrendben végeznek közelítéseket az egyes élekkel. A topologikus sorrenden alapuló (körmentes irányított gráfokban működő) algoritmus és a Dijkstra-algoritmus minden éllel legtöbb egyszer közelít. A Bellman–Ford-algoritmus az egyes élekkel többször is közelít.

Nyilvánvaló, hogy egy adott v ponthoz vezető legrövidebb úton utolsó előtti pont csakis v valamelyik be-szomszédja lehet. Más szóval az lru(s,v) érték felépíthető a v be-szomszédjaihoz vezető legrövidebb utak hosszaiból, a v pont megfelelő be-élein való közelítések révén. Ezért a legrövidebb utak meghatározásához megfelelő lenne egy olyan pontsorrend, amelyben a v pont lru(s,v) értékének kiszámítását megelőzi az összes be-szomszédjához vezető legrövidebb utak meghatározása. A

pontok topologikus sorrendje (amennyiben létezik) éppen ezt a sorrendet jelenti.

Ha a gráf minden éle pozitív súlyú, akkor az is természetes, hogy az `lru(s,v)` érték kiszámításánál a `v` pontnak csak azokat a be-szomszédjait (`u`) kell figyelembe venni, amelyeknek `lru(s,u)` értéke kisebb, mint `lru(s,v)`. Ez esetben hosszúságaik (súlyuk) szerint növekvő sorrendben fogjuk meghatározni a legrövidebb utakat. Ezen a megfigyelésen alapszik a Dijkstra-algoritmus.

6.1. Topologikus sorrenden alapuló legrövidebb út algoritmus

Feltétel: A gráf legyen körmentes.

Az irányított körmentes gráfok esetén beszélhetünk a csomópontok topologikus sorrendjéről. Ez a sorrend meghatározható a gráf mélységi bejárása „nyomán" (lásd a 4.3. alfejezetet). A topologikus sorrendben minden pont megelőzi mindenik ki-szomszédját. Más szóval: minden pontot megelőznek a be-szomszédai. Emlékezzünk, hogy a fokozatos közelítés technikája a következő feltételes finomító műveletre épül:

```
ha lru(s,u) + súly(u,v) < lru(s,v) akkor
   lru(s,v) = lru(s,u) + súly(u,v)
vége ha
```

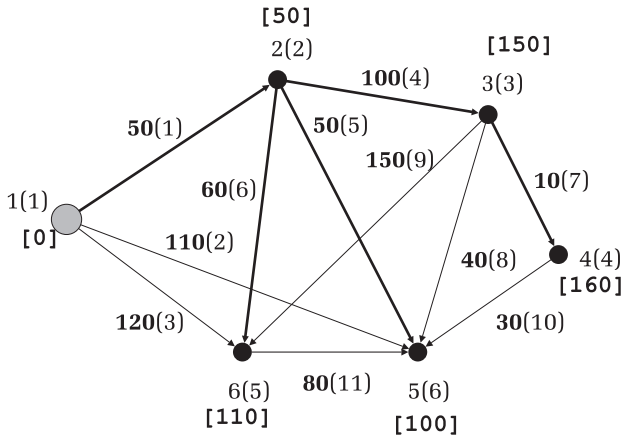
Az előbbi művelet a `v` pont szemszögéből azt jelenti, hogy megpróbálunk a hozzá vezető legrövidebb út hosszán javítani a be-szomszédjaihoz vezető legrövidebb utak hosszai alapján. Ugyanez az `u` pont nézőpontjából: megpróbálunk a ki-szomszédaihoz vezető legrövidebb utak hosszán javítani a hozzá vezető legrövidebb út hossza alapján.

Stratégia, algoritmus: Kezdetben `d[i]` $= \infty$ bármely $i = 1, \ldots, n$ pontra, kivéve az `s` kiindulási pontot, amelyre `d[s] = 0`. Azok a pontok, amelyek megelőzik `s`-t a topologikus sorrendben, nyilván nem érhetők el `s`-ből. Ezek `d` tömbbeli értéke végtelen marad. Az `s` pontot úgy tekinthetjük, mint amelyikhez megvan a legrövidebb út. Kezdve `s`-sel, végigjárjuk a többi csomópontot topologikus sorrendben. Minden csomópont összes ki-éle segítségével megpróbálunk javítani a megfelelő ki-szomszédok `d` tömbbeli értékén. Amikor „megérkezünk" egy adott ponthoz, akkor már minden be-élén keresztül javítottuk (ha esedékes volt) a hozzá vezető

utat. Ez azt jelenti, hogy amikor megérkezünk egy `v` ponthoz, a `d[v]` tömbelem már az `lru(s,v)` értéket tartalmazza.

Milyen sorrendben végez az alábbi algoritmus közelítéseket az egyes élekkel? A kezdőpontjaik topologikus sorrendje szerint! Ez az él-sorrend biztosítja, hogy a legrövidebb utakat alkotó élekkel olyan sorrendben történjenek a közelítések, amilyen sorrendben ezek az illető utakon előfordulnak (lásd a 6.1. ábrát). Ez magyarázza meg, miért elég minden `(u,v)` éllel egyszer javítani a végpontjához (`v`) vezető út hosszán (`d[v]`). A 6.3. alfejezetben látni fogjuk, hogy amennyiben nem biztosítható egy ilyen él-sorrend, az egyes élekkel talán többször is kell közelíteni.

Konkrétabban: Tegyük fel, hogy az `s`-ből `v`-be vezető legrövidebb úton `u` az utolsó előtti állomás. Amikor az algoritmus az `(u,v)` éllel közelít, a `d[u]` tömbelem már az `lru(s,u)` értéket tartalmazza. Ez lehetővé teszi, hogy az `(u,v)` éllel való közelítés révén a `d[v]` tömbelemben az `lru(s,v)` értéket állítsa be.



6.1. ábra. *Körmentes súlyozott irányított gráf. A pontok azonosítói mellett zárójelben feltüntettük a topologikus sorrendet. Az élek súlyai mellett feltüntettük, hogy milyen sorrendben közelítünk velük. Megvastagítottuk a legrövidebb utak fáját. Minden `v` pont mellett szögletes zárójelben az `lru(s,v)` értéket láthatjuk.*

A **TOPO_LRU** eljárás a `topo[1..n]` tömbben a csomópontokat topologikus sorrendben kapja meg. A cím szerint átadott `d[1..n]` és `apa[1..n]` tömbökben az eljárás kiszámítja a legrövidebb utak hosszát, illetve minden csomópontnak az apa-csomópontját (a legrövidebb utak fájában). Az apa-tömbből előállíthatók a legrövidebb utak.

A topologikus sorrend a példagráfra : 1, 2, 3, 4, 6, 5.

A 6.1. táblázatok a topologikus sorrenden alapuló legrövidebb út algoritmust követik nyomon a 6.1. példagráfra. A táblázatok a d és apa-tömbökben lépésről lépésre végbemenő változásokat mutatják be (6 lépés).

6.1. táblázat. *Fekete háttér: az illető ponthoz már rendelkezésre áll a legrövidebb út. Fehér/szürke háttér: az illető ponthoz még nincs meg a legrövidebb út. Szürke háttér: topologikus sorrendben soron következő (aktuális) pont. Félkövér értékek: az aktuális ponton keresztül frissített távolságértékek.*

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	∞	∞	∞	∞	∞	→	0	**50**	∞	∞	**110**	**120**
apa	0	0	0	0	0	0		0	**1**	0	0	**1**	**1**

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	∞	∞	110	120	→	0	50	**150**	∞	**100**	**110**
apa	0	1	0	0	1	1		0	1	**2**	0	**2**	**2**

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	1	150	∞	100	110	→	0	50	150	**160**	100	110
apa	0	1	2	0	2	2		0	1	2	**3**	2	2

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	150	160	100	110	→	0	50	150	160	100	110
apa	0	1	2	3	2	2		0	1	2	3	2	2

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	150	160	100	110	→	0	50	150	160	100	110
apa	0	1	2	3	2	2		0	1	2	3	2	2

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	150	160	100	110	→	0	50	150	160	100	110
apa	0	1	2	3	2	2		0	1	2	3	2	2

```
eljárás TOPO_LRU
             (G(V,E,súly),s,topo[1..n],d[1..n],apa[1..n],n)
   minden v∈V(G) végezd
      d[v] ← ∞
   vége minden
   d[s] ← 0
   apa[s] ← 0
   iₛ ← 1            // ráállítjuk iₛ-t az s pont topologikus
                    //          sorrend szerinti pozíciójára
   amíg (topo[iₛ] ≠ s) végezd
```

```
      is ← is + 1
   vége amíg
               // kezdve s-sel, végigjárjuk a csomópontokat
               //                    topologikus sorrendben
   minden i ← is,n-1 végezd
      u ← topo[i]
      minden v∈szomszéd(u) végezd
         ha d[u] + súly(u,v) < d[v] akkor
            apa[v] ← u
            d[v] ← d[u] + súly(u,v)
         vége ha
      vége minden
   vége minden
vége TOPO_LRU
```

Bonyolultság: A topologikus sorrenden alapuló legrövidebb út algoritmus bonyolultsága (amennyiben a gráf szomszédsági listaként van tárolva) $O(n+m)$ ([2], 464. oldal).

Mivel a fent leírt algoritmus ki-élek által végzi a közelítési műveleteket, mondhatnánk, hogy előretekintő stratégiát alkalmaz. Elképzelhető egy hátratekintő stratégia is. Ez azt jelentené, hogy akkor próbálunk finomítani egy pont `d` tömbbeli értékén, amikor megérkeztünk hozzá. Természetesen ez esetben az illető pont be-élein keresztül fogunk megpróbálni közelíteni a megfelelő be-szomszédok már kiszámított `lru` értékei alapján.

Megjegyzés: Az első algoritmus jelenje a jövő építéséről, a másodiké pedig a múltra építésről szól.

6.2. Dijkstra algoritmusa

Feltétel: A gráf tartalmazhat kört, de ne tartalmazzon negatív súlyú élt.

Stratégia: Emlékezzünk rá, hogy az optimalitás alapelve kimondja, hogy a legrövidebb utak részútjai is legrövidebb utak. Ha nincs negatív súlyú éle a gráfnak, akkor ez azt is jelenti, hogy a hosszabb (nagyobb súlyú) legrövidebb utak felépíthetők a rövidebb (kisebb súlyú) legrövidebb utakból. Ezzel összhangban Dijkstra algoritmusa súlyuk (hosszúságuk) szerint növekvő sorrendben határozza meg az egyes pontokhoz vezető

legrövidebb utakat. Tegyük fel, hogy a `v` pont lesz az, amelyikhez a következő hosszúságú legrövidebb út vezet. Ha `u` az utolsó előtti állomás ezen az úton, akkor `u`-hoz már meghatároztuk a legrövidebb utat. Következésképpen `u` már része a legrövidebb utak fájának. Tehát a következő legrövidebb út egy sajátossága, hogy az (`s`→`u`) szakasz már része a fának, az utolsó éle (az `(u,v)` él) pedig a fa valamelyik ki-éle lesz.

A fentiekkel összhangban a következő pont, amelyikhez a legrövidebb út meghatározásra kerül, a legrövidebb utak fájának valamelyik ki-szomszédja lesz. Egészen pontosan a fa gyökeréhez legközelebb eső ki-szomszédja. E mohó választást a következő érvelés indokolja: mivel a fa többi ki-szomszédja mind távolabb esik a fa gyökerétől, mint e legközelebbi, ezért azok nem képezhetik részét a legközelebbihez vezető legrövidebb útnak (fordítva viszont megtörténhet). Úgy is mondhatnánk, hogy a legrövidebb utak fájának a ki-szomszédjai a „következő legközelebbi pont" helyezésre kandidálnak, és a „pálmát" a fa gyökeréhez legközelebb eső kandidátus viszi el. A díj: a „nyertes pont" „beléphet" a legrövidebb utak fájába.

Milyen sorrendben közelít a Dijkstra-algoritmus az `(u,v)` élekkel? A kezdőpontjaik `lru(s,u)` értéke szerinti növekvő sorrendben! Ez azt jelenti, hogy a Dijkstra-algoritmus is tudja biztosítani (akárcsak a topologikus sorrenden alapuló legrövidebb út algoritmus), hogy a legrövidebb utakat alkotó élekkel olyan sorrendben történjenek a közelítések, amilyen sorrendben ezek az illető utakon előfordulnak (lásd a 6.2.g ábrát). Ez magyarázza meg, miért elég minden `(u,v)` éllel legtöbb egyszer javítani a végpontjához (`v`) vezető út hosszán (`d[v]`). Egészen pontosan akkor, amikor a kezdőpontja a legrövidebb utak fájának részévé vált. A 6.3. alfejezetben látni fogjuk, hogy amennyiben nem biztosítható egy ilyen él-sorrend, az egyes élekkel talán többször is kell közelíteni.

Dijkstra algoritmusa nagyon hasonlít Prim algoritmusához. Dijkstra algoritmusában a Q elsőbbségi sor azokat a csúcsokat tartalmazza, amelyekhez még nem határoztuk meg a legrövidebb utat (tehát kezdetben Q az összes csúcsot tartalmazza). Tehát itt is létezik a $(V - Q, Q)$ vágás, amelynek bal oldalán a „növekvő" legrövidebb utak fája (azokkal a pontokkal, amelyekhez már megvan a legrövidebb út), a jobb oldalán pedig a Q halmaznak pontjai találhatók. A két algoritmus között az elvi különbség az, hogy Dijkstra algoritmusa mindig *a fa gyökeréhez* (az s kiinduló ponthoz) *legközelebbi* (nem pedig a fához legközelebbi) pontot csatolja a fához. A `táv` tömb helyett a `d` tömböt használja, a `Q` prioritássor pedig

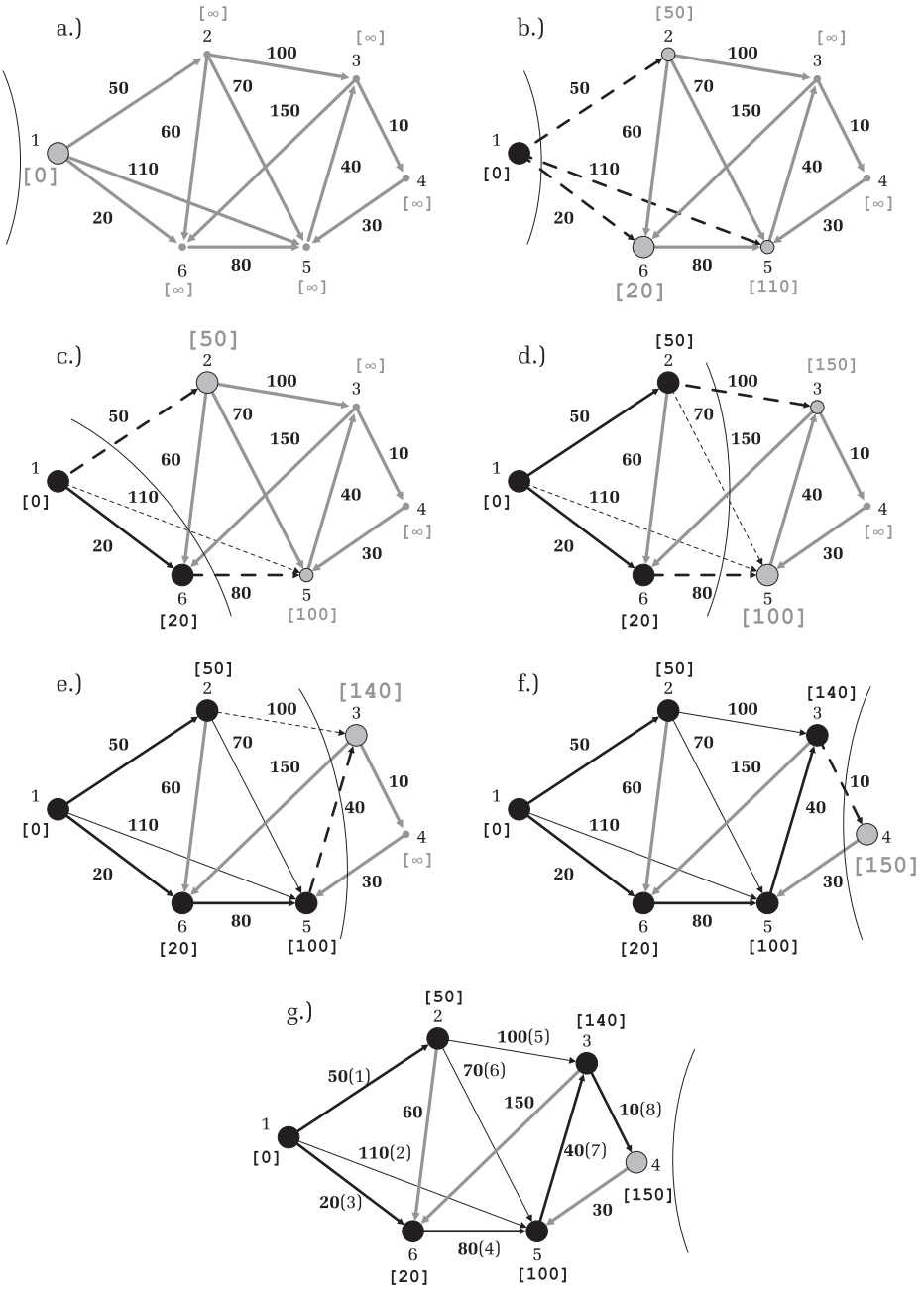
a csúcsok e `d` tömbbeli értékeire épül. A `d[1..n]` tömb elemei a pontoknak a fa gyökerétől mért – kizárólag fa-pontokon keresztüli – legrövidebb távolságát tárolják. Továbbá Dijkstra algoritmusa a `d` és `apa`-tömbök frissítését a fokozatos közelítés alapelve szerint valósítja meg (valahányszor az éppen csatolt csúcs révén új élek kerülnek a vágásba). Bár Dijkstra algoritmusát hagyományosan mohó stratégiának tartják, fellelhetők benne dinamikus programozásra utaló elemek is.

A 6.2. ábrasorozat nyomon követi Dijkstra algoritmusát. Nagy fekete pontok és vastagított folytonos fekete vonalak jelölik a kialakulóban lévő „legrövidebb utak fáját" (a vágás bal oldala). Szürke pontokkal ábrázoltuk a vágás jobb oldalán lévő pontokat (a Q halmaz). Ezek közül megnagyobbítottuk azokat, amelyek vágásbeli élek által kapcsolódnak a fához. Ezek a pontok kandidálnak arra, hogy a következő lépésben valamelyikük a fához csatolódjon.

A vágáshoz tartozó élek esetén szaggatott fekete vonalakat használtunk. Megvastagítottuk mindenik „kandidátus pontnak" azt a vágásbeli be-élét, amelyen keresztül a legelőnyösebben kapcsolódik a legrövidebb utak fájához (amelyen keresztül az eddigi legrövidebb út köti `s`-hez). Például a 6.2.c ábrán bemutatott állapotban az 5-ös pont két vágásbeli élen keresztül is kapcsolódik a fához, ((1, 5) és (6, 5)), amelyek közül a (6, 5)-ös él az, amelyiken a pillanatnyilag legrövidebb út áthalad. Az apa-tömb tárolja az egyes pontokhoz vezető ezen legrövidebb utak utolsó előtti állomásait (`apa[5]=6`). Minden kandidátus pont (`v`) esetén a megvastagított szaggatott vonal az `(apa[v],v)` élet jelöli.

Minden pont mellett szögletes zárójelben feltüntettük az aktuális helyzet szerinti legrövidebb út hosszát (a `d` tömb aktuális állása szerint). A fa-pontok esetén ez az érték végleges (fekete színt használtunk). Ha egy pont mellett ∞ áll, ez azt jelenti, hogy az illető pont még nem került egyél távolságra a legrövidebb utak fájától (még nem elérhető). A kandidátus pontoknál szürkével írtuk a jelenlegi állapot szerinti legrövidebb utak hosszát (kisebbek, mint ∞ és nagyobbak bármely „fekete értéknél"). A „nyertes kandidátus" távolságértékét megnagyítottuk.

Az algoritmus: A Dijkstra-algoritmus minden lépésben azt a kandidátust csatolja a fához, amelyhez a következő legkisebb értékű legrövidebb út vezet (első lépésben az `s` pontot). A `KIVESZ_MIN(Q)` függvény azt a kandidátust téríti vissza, amelyiknek a legkisebb a `d` tömbbeli értéke (egyben törli is ezt a pontot `Q`-ból). Ha már nincs kandidátus `Q`-ban (a fának nincs ki-szomszédja; nincs él a vágásban; minden `Q`-beli pontnak a `d` tömbbeli értéke végtelen), akkor a függvény nullát térít vissza. Ez azt



6.2. ábra. *A Dijkstra-algoritmus működésének szemléltetése*

jelenti, hogy a Q-ban maradt pontok nem elérhetőek s-ből. A „nyertes kandidátust" (u) a legnagyobb szürke pont ábrázolja.

Az u pont csatolása mintegy nyugtázza, hogy az aktuális d[u] érték finomítása befejeződött (azaz elérte az lru(s,u) értéket), és hogy az aktuális apa[u] pont az s-ből u-ba vezető véglegesített legrövidebb út utolsó előtti állomás. Minden csatolás után frissíteni kell a vágást és ezzel együtt a kandidátus pontok halmazát, illetve a d és apa-tömböket. A vágásból kikerülnek a csatolt pont „fa-pontoktól" induló be-élei, és bekerülnek a vágás jobb oldalán maradt ki-szomszédaihoz vezető ki-élei. A csatolt ponton (u) keresztül, ennek v ki-szomszédai d[u]+súly(u,v) távolságra kerültek s-től. Ha ez a távolság kisebb, mint a jelenlegi d[v] érték, akkor szükséges d[v] felülírása a d[u]+súly(u,v) értékkel, illetve az apa[v] értéknek u-val való frissítése.

A 6.2. táblázatok a Dijkstra-algoritmust követik nyomon a 6.2. példagráfra. A táblázatok a 6.2. ábrákkal összehangoltan mutatják be a d és *apa*-tömbökben lépésről lépésre végbemenő változásokat (6 lépés).

```
függvény DIJKSTRA}(G,s)
   Q ← V(G)
   minden v∈Q végezd
      d[v] ← ∞
   vége minden
   d[s] ← 0
   apa[s] ← 0
   amíg Q ≠ ∅ végezd
      u ← KIVESZ_MIN(Q)
      ha u = 0 akkor
         ugorj
      vége ha
      minden v ∈ szomszéd(u) végezd
         ha (v∈Q ÉS (d[u] + súly(u,v) < d[v])) akkor
            apa[v] ← u
            d[v] ← d[u] + súly(u,v)
         vége ha
      vége minden
   vége amíg
   vissza apa
vége DIJKSTRA
```

Az algoritmus végén minden v $\in V$ pontra a d[v] tömbelem az lru(s,v) értéket tárolja, az apa[v] tömbelem pedig a v pont apa-csomópontját a legrövidebb utak fájában.

6.2. táblázat. *Fekete háttér: az illető pont már a legrövidebb utak fájához tartozik. Fehér/szürke háttér: az illető pont még nem része a legrövidebb utak fájának. Szürke háttér: a legrövidebb utak fája gyökeréhez legközelebb eső pont. Félkövér értékek: a csatolt ponton keresztül frissített távolságértékek.*

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	∞	∞	∞	∞	∞	→	0	**50**	∞	∞	**110**	**20**
apa	0	0	0	0	0	0		0	**1**	0	0	**1**	**1**

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	∞	∞	110	20	→	0	50	∞	∞	**100**	20
apa	0	1	0	0	1	1		0	1	0	0	**6**	1

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	∞	∞	100	20	→	0	50	**150**	∞	100	20
apa	0	1	0	0	6	1		0	1	**2**	0	6	1

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	150	∞	100	20	→	0	50	**140**	∞	10	20
apa	0	1	2	0	6	1		0	1	**5**	0	6	1

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	140	∞	100	20	→	0	50	140	**150**	100	20
apa	0	1	5	0	6	1		0	1	5	**3**	6	1

	1	2	3	4	5	6		1	2	3	4	5	6
d	0	50	140	150	100	20	→	0	50	140	150	100	20
apa	0	1	5	3	6	1		0	1	5	3	6	1

Megjegyzések:

– A Dijkstra-stratégia a következőképpen összegezhető: Tegyük fel, hogy az egyes pontokhoz vezető legrövidebb utak az $u_1 = s, u_2, u_3, \ldots, u_n$ sorrendben kerülnek meghatározásra ($lru(s, u_1) \leqslant lru(s, u_2) \leqslant lru(s, u_3) \leqslant \ldots \leqslant lru(s, u_n)$). Először közelítettünk az $u_1 = s$ pontnak az $\{u_2, u_3, \ldots, u_n\}$ halmaz fele mutató ki-élein keresztül majd az u_2 pontnak az $\{u_3, u_4, \ldots, u_n\}$ halmaz fele mutató ki-élein keresztül és így tovább. Általánosan: a k-adik lépésben közelítettünk az u_k pontnak az $\{u_{k+1}, \ldots, u_n\}$ halmaz fele mutató ki-élein keresztül. Másfelől: amikor a k-adik lépésben az u_k ponthoz vezető legrövidebb utat véglegesnek nyilvánítottuk, akkorra már a $d[u_k]$ érték az u_k pontnak az $\{u_1, u_2, \ldots, u_{k-1}\}$ halmazból

induló minden be-élén keresztül finomítva volt. Dijkstra algoritmusa előretekintő algoritmus, mivel ki-éleken keresztül végzi a közelítő műveleteket.

– A Dijkstra-eljárásban megtestesülő stratégia az alábbi módon is összefoglalható: A legrövidebb utak fáját úgy építjük fel, mint egy virtuális gyökerű fát. Az s pont nulla távolságra, az összes többi pont végtelen távolságra van e virtuális gyökértől. Kezdetben egyetlen pont sincs a fához kapcsolva (minden pont eleme Q-nak). A d tömb kezdeti értékei úgy tekinthetők, mint a virtuális gyökértől az egyes pontokhoz vezető azon legrövidebb utak hosszai, amelyek nem tartalmaznak közbeeső állomásokat (közvetlen utak). A legkisebb d-értékű Q-beli pont esetén ez nyilvánvalóan a hozzá vezető végleges legrövidebb út hosszát jelenti. Az első lépésben természetesen az $u_1 = s$ pont lesz ebben a helyzetben. Töröljük s-et Q-ból, és ezzel az s pont automatikusan a fa részévé válik (rátevődve a virtuális gyökérre). Az $u_1 = s$ pont csatolását követő frissítések (s ki-élei révén) után a d tömb Q-beli pontokat képviselő elemei az illető pontokhoz vezető azon legrövidebb utak hosszait tartalmazzák, amelyeken legfennebb az $u_1 = s$ pont szerepel közbeeső állomásként. Újból elmondható, hogy a jelenlegi állás szerinti legkisebb d-értékű Q-beli pont esetén ez a hozzá vezető végleges legrövidebb út hosszát jelenti. Legyen ez az u_2 pont ($d[u_2] = lru(s, u_2)$). Töröljük u_2-t Q-ból. Miután elvégeztük a szükséges frissítéseket (u_2 pontnak a Q-ban maradt szomszédaihoz vezető ki-élein keresztül), a d tömb Q-beli pontokat képviselő elemei az illető pontokhoz vezető azon legrövidebb utak hosszait tartalmazzák, amelyeken legfennebb az u_1 és/vagy u_2 pontok szerepelnek közbeeső állomásként. Általánosan: a k-adik lépés után a d tömb Q-beli pontokat képviselő elemei az illető pontokhoz vezető azon legrövidebb utak hosszait tartalmazzák, amelyeken legfennebb az $\{u_1, u_2, \ldots, u_k\}$ halmazból szerepelnek pontok közbeeső állomásként. Hasonló gondolatmenetet alkalmaz a 7. fejezetben tárgyalt Floyd-algoritmus.
– Dijkstra algoritmusa a szélességi bejárásra emlékeztet, abban az értelemben, hogy a legrövidebb utak fája is, akár a szélességi fa, a legrövidebb úton éri el a csúcsokat, sőt a legrövidebb út szerinti sorrendben.
– A 6.2.g ábrán szürkén maradt éleket a Dijkstra-algoritmus „visszamutató éleinek" nevezhetnénk. Ezekkel az élekkel nem történik

közelítés (nem kerülnek be a vágásba). Az (u, v) él akkor visszamutató, ha u később került be (vagy be se került, mert nem elérhető s-ből) a legrövidebb utak fájába mint v. Mivel a vágást minden lépésben csak a csatolt pont Q fele mutató ki-éleivel bővítjük, ezért a pont fa fele mutató ki-élei (ezek a visszamutató élek) nem kerülnek be a vágásba. A fekete élek „előremutató éleknek" tekinthetők. Ezek közül kerülnek ki a faélek, amelyeket vastagított vonalak ábrázolnak.

– *Miért nem jelentenek gondot a Dijkstra-algoritmusnak a körök?* Legyen $\{u_1, u_2, \ldots, u_k = u_1\}$ egy kör a gráfban. Feltételezzük továbbá, hogy az algoritmus a kör u_i $(1 < i \leqslant k)$ pontját csatolja elsőkét a legrövidebb utak fájához. Ez azt jelenti, hogy a kört záró (u_{i-1}, u_i) él „visszamutató élnek" fog bizonyulni. Mivel a Dijkstra-algoritmus tudomást sem vesz a visszamutató élekről, ezért nem jelentenek neki gondot a körök. A példagráfunk két kört tartalmaz: (3, **6**, 5, 3) és (3, 4, **5**, 3). Az algoritmus az első körnek a 6-os csúcsát csatolja elsőként a fához, a második kör esetében pedig az 5-ös csúcsot. A fent leírtakkal összhangban a (3, 6) és (4, 5) élek szürkék maradtak.
– *Miért nem működik helyesen Dijkstra algoritmusa negatív súlyú élek esetén?* Emlékezzünk, hogy egy adott pillanatban valamely `u`$\in$`Q` csúcs esetén `d[u]` azon legrövidebb út súlyát tárolja, amely olyan csúcsokon keresztül vezet hozzá, amelyekhez már meghatároztuk a legrövidebb utat. A mohó választás alapötlete az, hogy ha jelen pillanatban a fa ki-szomszédai közül `u` az `s`-hez legközelebb eső pont, akkor biztosan `d[u]` lesz az ide vezető legrövidebb út hossza (ha a többi `Q`-beli csúcs távolabb esik `s`-től, mint `u`, akkor rajtuk keresztül nem vezethet rövidebb út `u`-hoz, mint `d[u]`). Ilyenkor az algoritmus `u`-t a fához csatolja azáltal, hogy törli `Q`-ból. Az előbbi gondolatmenet viszont nem mindig igaz, amennyiben a gráfnak vannak negatív súlyú élei is. Megtörténhet ugyanis, hogy a fa `u` és `v` ki-szomszédai esetén bár `d[u]<d[v]`, létezik egy `(v,u)` negatív súlyú él úgy, hogy `d[v]+súly(v,u)<d[u]`. Ez viszont azt jelenti, hogy a távolabb eső `v` csúcson keresztül lesz `d[u]`-nál rövidebb kerülő út `u`-hoz. A hiba ott csúszik be, hogy amikor a Dijkstra-algoritmus odajut, hogy a `(v,u)` éllel kellene közelítsen (mert éppen a fához csatolta `v`-t), ezt az élet visszamutató élnek fogja találni és átugorja (`v` már nem eleme `Q`-nak). Mivel az `u` pontot már korábban a fához csatoltuk, szóba sem jön a `d[u]`

értéknek a `(v,u)` élen keresztüli tovább finomítása. Az alábbiakban bemutatásra kerülő Bellman–Ford-algoritmusnak sikerül bizonyos esetekben ezt a helyzetet is kezelnie.

– Dijkstra algoritmusa működik irányítatlan gráfok esetén is. Minden élet úgy kezel, mint az elsőször elért végpontjának a „ki-élét". Mivel ez esetben nem lesznek vissza-élek, minden éllel megpróbál közelíteni.

Bonyolultság: A Dijkstra-algoritmus bonyolultsága (amennyiben a Q elsőbbségi sort bináris kupacként implementáljuk) $O(m \lg n)$ ([2], 459. oldal).

6.3. Bellman–Ford-algoritmus

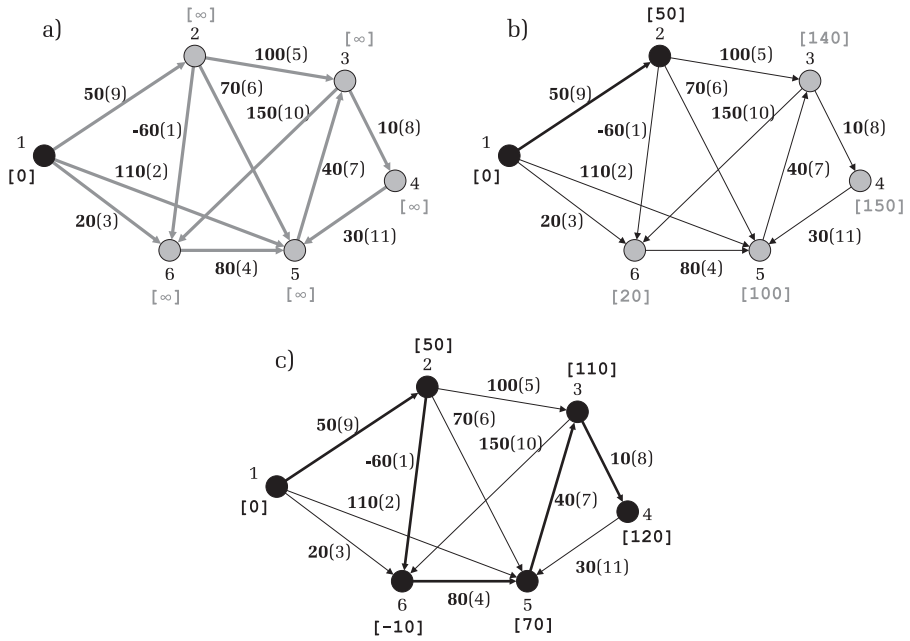
Feltétel: A gráf ne tartalmazzon a kezdőpontból elérhető negatív összsúlyú kört. Ha van ilyen kör, az algoritmus jelzi, hogy nincs megoldás.

Az előbbi algoritmusoknak sikerült úgy meghatározni a legrövidebb utakat, hogy minden éllel legfennebb egyszer közelítettek (ha esedékes volt). Ez azért volt lehetséges, mert biztosítani tudtak egy olyan él-sorrendet, hogy a legrövidebb utakat alkotó élekkel az illető utakon való előfordulási sorrendjükben történjen a közelítés. Az első esetben ezt az él-sorrendet a kezdőpontjaik szerinti topologikus sorrend biztosította. Dijkstra algoritmusának esetében a kezdőpontok *lru* értékein alapuló él-sorrend volt a megfelelő.

Stratégia, algoritmus: Jelen esetben a körök miatt a topologikus sorrend nem értelmezhető, a negatív élek jelenléte pedig kizárja az *lru* értékek szerinti sorrendet mint nem megfelelőt. Mi a megoldás? Mivel ez esetben nem ismert olyan él-sorrend, amely garantálná a legrövidebb utak egyetlen menetből való előállítását, szükségessé válhat az élek többszöri átpásztázása, újra és újra megpróbálva közelíteni velük. Mindezt addig ismételjük, amíg sikerül anélkül végigjárni az éleket, hogy sor került volna közelítésre. Ezt figyeli a `volt_közelítés` változó.

A Bellman–Ford-algoritmus az első menetben csak azokat a legrövidebb utakat határozza meg, amelyeknek esetében a választott él-sorrend biztosítja, hogy az út menti élekkel az illető utakon való előfordulásuk sorrendjében történjen a közelítés. Ugyanakkor biztos az, hogy az első menetben meghatározásra kerülnek az egy él hosszú legrövidebb utak, a másodikban a két élből állók, és így tovább. Mivel egy legrövidebb

út legfennebb $n-1$ élből áll, így legfennebb $n-1$ menetre lehet szükség. Ha a Bellman–Ford-algoritmus az n-edik menetben is végez közelítő műveletet, akkor ez negatív összsúlyú kör jelenlétéről tanúskodik.



6.3. ábra. *Bellman–Ford-stratégia*

A 6.3.a–c ábrasorozat a Bellman–Ford-algoritmus működését szemlélteti. Az élek súlyértékei mellett feltüntettük, hogy milyen sorrendben (tetszőleges) fog az algoritmus közelíteni az élekkel egy-egy menetben. Kezdetben csak az s kiindulóponthoz van meg a legrövidebb út hossza. Ezért a 6.3.a ábrán csak az s pontot és a mellette lévő `d[s]=0` értéket (szögletes zárójelben) rajzoltuk/írtuk feketével. Első menetben csak a 2-es ponthoz épül meg a legrövidebb út (6.3.b ábra). A második menetben az algoritmusnak sikerül meghatározni az összes többi ponthoz vezető legrövidebb utakat (6.3.c ábra). A harmadik menetben a `volt_közelítés` változó `HAMIS` marad, ami a **végezd-amíg** ciklusból való kilépést eredményezi. A **BELLMAN_FORD** függvény a példagráf esetében `IGAZ` értékkel tér vissza, ami azt jelenti, hogy a gráfban értelmezettek a legrövidebb utak (nem tartalmaz s-ből elérhető negatív összsúlyú kört), és a d, illetve

apa-tömbökben a hívó eljárás rendelkezésére állnak e legrövidebb utak hosszai, illetve utolsó előtti állomásai.

```
függvény BELLMAN_FORD(G(V,E),s,d[1..n],apa[1..n],n)
   minden v∈V(G) végezd
      d[v] ← ∞
   vége minden
   d[s] ← 0
   apa[s] ← 0
   i ← 0
   végezd
      volt_közelítés ← HAMIS
      minden (u,v)∈E(G) végezd
         ha d[u] + súly(u,v) < d[v] akkor
            apa[v] ← u
            d[v] ← d[u] + súly(u,v)
            volt_közelítés ← IGAZ
         vége ha
      vége minden
      i ← i + 1
   amíg (volt_közelítés = IGAZ) ÉS (i < n)
   ha (volt_közelítés = IGAZ) ÉS (i = n) akkor
      vissza HAMIS
   különben
      vissza IGAZ
   vége ha
vége BELLMAN_FORD
```

A 6.3. táblázat nyomon követi a d tömb finomítási folyamatát. Az élek oszlop azt a tetszőleges sorrendet tartalmazza, amelyben a közelítések történnek. Megvastagítottuk azokat az éleket, amelyekkel az illető menetben való finomítás beállítja végpontjuk *lru* értékét. Valahányszor finomításra került sor a d tömbben, a megváltozott értéket ugyancsak kiemeltük.

Bonyolultság: A Bellman–Ford-algoritmus bonyolultsága az egymásba ágyazott `minden` ciklusokból adódóan $O(nm)$ ([2], 462. oldal).

Megjegyzések:

– *Milyen esetben van szüksége a Bellman–Ford-algoritmusnak $n-1$ menetre?* Tegyük fel, hogy létezik egy $n-1$ hosszú legrövidebb út. Legyen ez a $(v_1 = s, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n)$ út. Természetesen a v_i $(i = 2, \ldots, n)$ csúcshoz vezető legrövidebb út csak azután határozható meg, ha a v_{i-1} csúcshoz vezető út már rendelkezésre

6.3. táblázat.

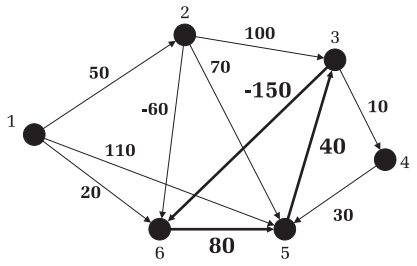
	Élek	1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10
	6:(2,5) [70]	0	50	140	150	70	–10
	7:(5,3) [40]	0	50	**110**	150	70	–10
	8:(3,4) [10]	0	50	110	**120**	70	–10
	9:(1,2) [50]	0	50	110	120	70	–10
	10:(3,6) [150]	0	50	110	120	70	–10
	11:(4,5) [30]	*0*	*50*	*110*	*120*	*70*	*–10*

áll, ugyanis $lru(s, v_i) = lru(s, v_{i-1}) + súly(v_{i-1}, v_i)$. Mi történik akkor, ha az egyes menetekben a legrövidebb utat alkotó élekkel történetesen a $(v_{n-1}, v_n), (v_{n-2}, v_{n-1}), \ldots, (v_1, v_2)$ sorrendben (az úton való előfordulásuk fordított sorrendjében) próbálunk közelíteni? Mivel a (v_i, v_{i-1}) éllel való végső közelítésre csak a (v_{i-1}, v_{i-2}) éllel való végső közelítés után kerülhet sor, ezért az első menetben csak v_2-höz tudjuk meghatározni a legrövidebb utat, majd a másodikban v_3-hoz, és végül az $(n-1)$-edik menetben v_n-hez. Amennyiben olyan sorrendet választunk, amelyik a $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ éleket ebben a sorrendben tartalmazza, a legrövidebb út meghatározásához elég az élek egyetlen végigjárása. Ezt a fajta sorrendet tudta biztosítani a topologikus sorrenden alapuló legrövidebb út algoritmus (**TOPO_LRU** eljárás) és Dijkstra algoritmusa (**DIJKSTRA** eljárás).

– *Egészen pontosan hány menetre van szüksége a Bellman–Ford-algoritmusnak?* Tekintsük azt az irányított gráfot, amelynek a csomópontjai a legrövidebb utak fájának élei (a faélek). E gráf két pontja között akkor van irányított él, ha a kezdőpont-él az eredeti gráf valamelyik legrövidebb útján megelőzi a végpont-élet. Világos, hogy az újonnan definiált gráf körmentes lesz (mert a legrövidebb utak fája körmentes). Ily módon beszélhetünk a csomópontjai topologikus sorrendjéről. Ez viszont az eredeti gráf éleire vonatkoztatva egy olyan sorrendet jelent, amelyben minden él azok előtt szerepel, amelyeket megelőz a legrövidebb utakon. Bármely él-sorrend, amelyik megfelel az előbbi feltételnek, biztosítani tudja a legrövidebb utaknak egy menetből való meghatározását. Mivel a fenti feltétel mellett több helyes topologikus él-sorrend is létezhet, ezért megtörténhet, hogy a **TOPO_LRU** és **DIJKSTRA** eljárások különböző él-sorrendet generálnak. A Bellman–Ford-algoritmusnak annyi menetre lesz szüksége, amennyire „távol van" az általa választott él-sorrend egy helyes topologikus él-sorrendtől. A „távolság"-on azt értjük, hogy hány él nincs a helyén. Pontosabban: hány lépésből lehetne a választott sorrendből egy helyes topologikus sorrendet előállítani úgy, hogy minden lépésben egy élet teszünk a helyére. Nevezzük ezeket *gát-él*eknek. A Bellman–Ford-algoritmus első menete előállítja azokat a legrövidebb utakat, amelyek nem tartalmaznak gát-éleket, illetve azt, amelyik utolsó élként tartalmazza az első gát-élet. A következő menet előállítja azokat a legrövidebb utakat,

amelyeket csak az első gát-él „gátolt", beleértve azt is, amelyiken a második gát-él az utolsó él. Mivel minden menetben (kivéve az utolsót) egy gát-él „kerül a helyére", ezért eggyel több menetre lesz szükség, mint ahány gát-él van. Bár ennyi menetből összeáll a legrövidebb utak fája, szükség van egy további menetre, hogy az algoritmus „tudomást szerezzen" erről a tényről. A példagráfunkban egy gát-él van (az (1, 2) él), ezért három menetet hajt végre ez esetben a Bellman–Ford-algoritmus. Az (1, 2) él azért gát-él, mert bár elsőként szerepel mindenik legrövidebb úton, kilencedik a közelítési él-sorozatban. A többi fa-él ugyanolyan sorrendben szerepel a legrövidebb utakon is, mint amilyen sorrendben a közelítési él-sorozatban is követik egymást.

– Érdekes megfigyelni, hogy a 6.3. ábra példagráfjára a Dijkstra-algoritmus más (hibás) eredményt adna. Ez a negatív súlyú (2, 6) él miatt van. A Dijkstra-algoritmus első lépésben (az s pont csatolása után) a 6-os pontot csatolná a fához, mint azt, amelyikhez a legrövidebb közvetlen él vezet s-től. Ennek az lesz a következménye, hogy a 2-es pont csatolásakor a (2, 6)-os élet visszamutató élként érzékeli és átugorja. Ez azért végzetes hiba, mert a 6-os ponthoz vezető legrövidebb út (az (1, 2, 6) út) tartalmazza a (2, 6)-os élet. Bármelyik legrövidebb útnak a meghatározása feltételezi, hogy legalább egyszer megpróbálunk közelíteni mindenik rajta lévő éllel. Hogyan történhetett meg, hogy a távolabb eső 2-es ponton keresztül rövidebb út vezessen a kisebb távolságra lévő 6-os ponthoz? A magyarázat a (2, 6)-os él negatív súlyában rejlik. Ez az, amivel nem számol a Dijkstra-algoritmus. Mivel a Dijkstra-stratégia szempontjából a „visszamutató" élek kihagyása teljesen logikus, ezért a probléma gyökere nem az él-átugrás, hanem az, hogy ez az algoritmus elvileg nem felel meg negatív súlyú éleket tartalmazó gráfok esetében.
– Ha a (3, 6) él súlyát (-150)-re változtatjuk, akkor negatív összsúlyú kör alakul ki a gráfban (6.4. ábra): a (3, 6, 5, 3) kör súlya (-30) lesz. Az alábbi táblázat azt mutatja be, hogy azon pontok esetében, amelyekhez vezető legrövidebb út tartalmazná a negatív összsúlyú kört (a 3, 4, 5 és 6 pontok), miként csökken minden menetben a d tömbbeli értékük a kör súlyának abszolút értékével. Ezért jelentettük ki a fejezet elején, hogy e pontok esetében a legrövidebb út hossza $-\infty$.



6.4. ábra. *Negatív összsúlyú kört tartalmazó gráf*

6.4. táblázat.

	Élek	1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1: (2, 6) [–60]	0	∞	∞	∞	∞	∞
	2: (1, 5) [110]	0	∞	∞	∞	**110**	∞
	3: (1, 6) [20]	0	∞	∞	∞	110	**20**
	4: (6, 5) [80]	0	∞	∞	∞	**100**	20
	5: (2, 3) [100]	0	∞	∞	∞	100	20
	6: (2, 5) [70]	0	∞	∞	∞	100	20
	7: (5, 3) [40]	0	∞	**140**	∞	100	20
	8: (3, 4) [10]	0	∞	140	**150**	100	20
	9: (1, 2) [50]	0	***50***	140	150	100	20
	10: (3, 6) [–150]	0	50	140	150	100	**–10**
	11:(4, 5) [30]	*0*	*50*	*140*	*150*	*100*	*–10*
2. menet	1: (2, 6) [–60]	0	50	140	150	100	–10
	2: (1, 5) [110]	0	50	140	150	100	–10
	3: (1, 6) [20]	0	50	140	150	100	–10
	4: (6, 5) [80]	0	50	140	150	**70**	–10

2. menet	5: (2, 3) [100]	0	50	140	150	70	–10
	6: (2, 5) [70]	0	50	140	150	70	–10
	7: (5, 3) [40]	0	50	**110**	150	70	–10
	8: (3, 4) [10]	0	50	110	**120**	70	–10
	9: (1, 2) [50]	0	50	110	120	70	–10
	10: (3, 6) [–150]	0	50	110	120	70	**–40**
	11: (4, 5) [30]	*0*	*50*	*110*	*120*	*70*	*–40*
3. menet	1: (2, 6) [–60]	0	50	110	120	70	–40
	2: (1, 5) [110]	0	50	110	120	70	–40
	3: (1, 6) [20]	0	50	110	120	70	–40
	4: (6, 5) [80]	0	50	110	120	**40**	–40
	5: (2, 3) [100]	0	50	110	120	40	–40
	6: (2, 5) [70]	0	50	110	120	40	–40
	7: (5, 3) [40]	0	50	**80**	120	40	–40
	8: (3, 4) [10]	0	50	80	**90**	40	–40
	9: (1, 2) [50]	0	50	80	90	40	–40
	10: (3, 6) [–150]	0	50	80	90	40	**–70**
	11: (4, 5) [30]	*0*	*50*	*80*	*90*	*40*	*–70*

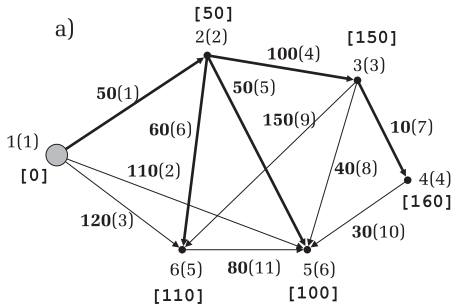
6.4. Legrövidebb út algoritmusok és dinamikus programozás

A dinamikus programozás az optimalitás alapelve implementációjának tekinthető. E stratégia lényege, hogy az egyszerűbb részfeladatok optimális megoldásaiból (indulva a triviálisan egyszerűktől) felépíti az egyre bonyolultabb részfeladatok optimális megoldásait (végül az eredeti feladatét). Egy másik jellegzetessége a dinamikus programozásnak, hogy a már megoldott részfeladatok optimális megoldásait (gyakran az ezt

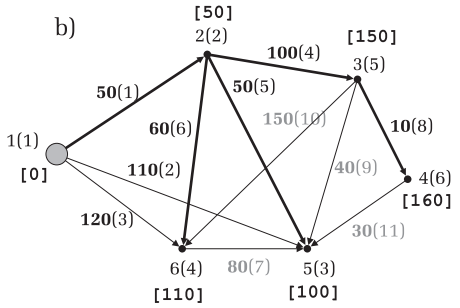
képviselő optimumértéket) eltárolja, hogy kéznél legyenek (amennyiben szükség lesz rájuk) az „építkezés" későbbi fázisaiban.

A fentiekben bemutatott mindhárom algoritmus alapvetően dinamikus programozást alkalmaz. (A fokozatos közelítés módszere a dinamikus programozás egyik változatának tekinthető.) Mindhárom esetben igaz, hogy az élszámban rövidebb legkisebb súlyú utakból építettük fel a hosszabb legkisebb súlyú utakat. A soron következő optimális út (amely meghatározásra került) mindig egy olyan út volt, amelyiknek az utolsó előtti állomásához vezető optimális út (és annak súlya) már rendelkezésre állt. Az algoritmusok ezt a lépést (a következő optimális út `lru` értékének kiszámítását) az illető út utolsó élével való közelítés révén valósították meg. Úgy is mondhatnánk, hogy a dinamikus programozásos stratégia implementálása nem jelent mást, mint megfelelő sorrendben közelíteni az optimális utak utolsó éleivel. A 6.3. alfejezet második megjegyzésében ezt a sorrendet a legkisebb súlyú utak fáját alkotó *élek* topologikus sorrendjeként azonosítottuk. A leghatékonyabb az lenne, ha csak ezzel az él-sorozattal közelítenénk (a többi éllel való közelítések szükségtelenek). Ennek lehetősége azért kizárt, mert feltételezi, hogy előre ismerjük az optimális utakat. Mivel a fölösleges közelítések, vagy közelítési próbálkozások, „nem ártanak", ezért a megoldást a gráf összes élének az átfésülése adja. Az első két algoritmus (a topologikus sorrenden alapuló és a Dijkstra-algoritmus) egyszer járja végig a gráf éleit. Mindkettőnek sikerül olyan él-sorrendben végezni a közelítéseket, hogy e közelítések részsorozataként megvalósuljon a dinamikus programozás szerinti építkezés. Arra a helyzetre, amikor a gráf tartalmaz kört és negatív élet, nem ismert olyan algoritmus, amelyik generálni tudna egyetlen megfelelő él-sorrendet. Ezért a Bellman–Ford-algoritmus a gráf összes élének többszöri átfésülése által tudja csak biztosítani a dinamikus programozás igényelte közelítéssorozatot. Mindhárom algoritmus a d tömböt használja a már meghatározott optimális utak súlyainak eltárolására.

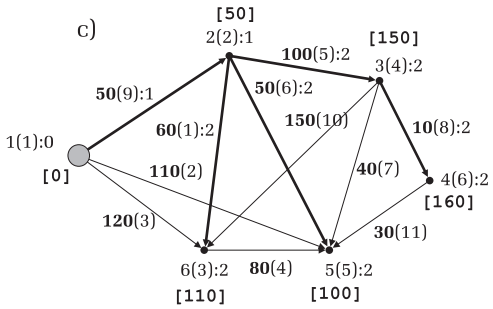
Az alábbi ábrák ugyanazon a gráfon (körmentes és nem tartalmaz negatív élet) mutatják be, hogy miként biztosítják a megvizsgált algoritmusok a dinamikus programozás előírta közelítéssorozatot. Az élek súlyai mellett az általuk végzett közelítési sorrend látható. Az első algoritmus esetében ez a sorrend a kezdőpontok topologikus sorrendjén alapszik. Dijkstra algoritmusa a kezdőpontok *lru* értéke szerinti sorrendet használja. Szürkével jelöltük az átugrott vissza-éleket. A Bellman–Ford-algoritmus bemutatásánál egy tetszőleges él-sorrendet választottunk. A dinamikus programozást megvalósító közelítésekhez tartozó élek esetén



(1,2), (1,5), (1,6), **(2,3), (2,5), (2,6), (3,4),** (3,5), (3,6), (4,5), (6,5)



(1,2), (1,5), (1,6), **(2,3), (2,5), (2,6),** (6,5), **(3,4),** (3,5), (3,6), (4,5)



1: (2,6), (1,5), (1,6), (6,5), (2,3), (2,5), (3,5), (3,4), **(1,2),** (3,6), (4,5)

2: **(2,6),** (1,5), (1,6), (6,5), **(2,3), (2,5),** (3,5), **(3,4),** (1,2), (3,6), (4,5)

6.5. ábra. *a) Topologikus sorrenden alapuló optimális út algoritmus; b) Dijkstra algoritmusa; c) Bellman–Ford-algoritmus*

félkövér karakterekkel írtuk az illető művelet sorszámát. A Bellman–Ford-algoritmus kapcsán azt is megjelöltük, hogy hányadik menetben kerül sor az illető közelítésre. A csomópontok mellett zárójelben az látható, hogy milyen sorrendben kerülnek meghatározásra az illető pontokhoz vezető legrövidebb utak. Mindenik ábra alá leírtuk az ábrázolt közelítéssorozatot (kiemelve az optimális utakat felépítő közelítéseket). Az ábrák jól érzékeltetik, hogy

- az első két algoritmus generálta élsorozatok különbözhetnek;
- algoritmusonként különbözhet az optimális utakat kiépítő közelítések topologikus él-sorrendje (és ezzel együtt az optimális utak meghatározásának sorrendje).

7. FEJEZET

LEGRÖVIDEBB UTAK MINDEN PONTPÁR KÖZÖTT

Gyakorlati probléma: Egy nagy kiterjedésű városban egy építővállalatnak sok építőtelepet kell építőanyaggal ellátnia. Ismerve az anyagraktárak és a munkatelepek közötti közlekedési lehetőségeket, elkészíthető a megfelelő úthálózati gráf, melyben minden egyes él hosszúságát az átfutási idő adja. A feladat tehát: meghatározni az így felépített gráf két-két megfelelő pontja között a legrövidebb utat.

7.1. Floyd algoritmusa

Feltétel: A gráf ne tartalmazzon negatív összsúlyú kört.

A **TOPO_LRU** és **DIJKSTRA** algoritmusok esetében a d tömb egy adott pillanatban az egyes pontokhoz vezető azon legrövidebb utak súlyát tartalmazta, amelyek közbeeső állomásként legfennebb fa-pontokat (a legrövidebb utak fájához tartozó pontok) tartalmaztak, azaz olyan pontokat, amelyekhez már korábban kiépült az optimális út. Hasonló gondolatmenetet használ a Floyd-algoritmus is. Ez az algoritmus meghatározza egy irányított gráf *minden pontpárja között* a legrövidebb utat.

Stratégia: A Floyd-algoritmus a `súlyM[1..n][1..n]` súlymátrixból indul ki.

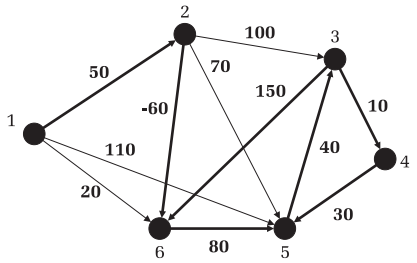
```
súlyM[i][j] = 0, ha i=j
            = ∞, ha (i,j) ∉ E(G)
            = súly(i,j), (i,j) ∈ E(G)
```

Az algoritmus úgy tekinti a `súlyM` mátrix elemeit, mint bármely pontpár között azon legrövidebb utak súlyértékét, amelyek nem tartalmaznak közbeeső állomásokat. Kiindulva ebből a mátrixból, előállítjuk a d_0 = `súlyM`, d_1, ..., d_n mátrixsorozatot. A d_k (`k = 1, 2, ..., n`) mátrix bármely pontpár esetén annak a legrövidebb útnak a súlyát tartalmazza, amely *legfennebb* az `{1, 2, ..., k}` halmazból tartalmaz közbeeső

állomásokat. A d_n mátrix a valódi legrövidebb utak súlyait fogja tartalmazni. A k-adik lépésben a Floyd-algoritmus d_{k-1}-ből előállítja d_k-t. Ez dinamikus programozást jelent.

Algoritmus: A d_k[i][j] érték kiszámításakor két esetet kell megvizsgálnunk: rajta van-e a k pont vagy nincs az i-ből j-be vezető, legfennebb {1, 2, ..., k} halmazbeli közbeeső állomásokat tartalmazó legrövidebb úton? Ha nincs rajta, akkor: d_k[i][j]←d_{k-1}[i][j], különben: d_k[i][j]←d_{k-1}[i][k]+d_{k-1}[k][j] (az optimális út részútjai is optimális utak). Azért vehetjük az (i→k) és (k→j) szakaszok optimumértékét a d_{k-1} tömbből, mert ezek legfennebb az {1, 2, ..., k–1} halmazból tartalmaznak közbeeső állomásokat (e szakaszoknak vég-, illetve kezdőpontja a k pont). Tehát a d_k[i][j] érték vagy d_{k-1}[i][j]-vel vagy a (d_{k-1}[i][k]+d_{k-1}[k][j]) összeggel lesz egyenlő. Mivel a legkisebb súlyú utakat keressük, ezért nyilván mindig a kisebbik értéket kell választanunk.

Figyeljük meg, hogy a d_k tömb előállításakor a k-adik sor és a k-adik oszlop elemeinél mindig a d_k[i][j]←d_{k-1}[i][j] variánsra kerül sor (ahol i=k vagy j=k). A többi d_k[i][j] elem (i≠k és j≠k) esetén viszont – valahányszor d_k[i][j]←d_{k-1}[i][k]+d_{k-1}[k][j] változatot kell alkalmaznunk – a d_{k-1} mátrixnak kizárólag a k-adik sorára és oszlopára támaszkodunk. Ebből kifolyólag a Floyd-algoritmus egyetlen d tömböt használ, amely kezdetben a súlyM mátrix elemeit tartalmazza. Az algoritmus k-adik lépésében a d_{k-1} mátrixot felülírhatjuk a d_k mátrixszal (ezt az előbbi észrevétel teszi lehetővé).



7.1. ábra. *Legrövidebb utak minden pontpár között*

A 7.1. ábra példagráfján kiemeltük a legrövidebb utak részgráfját. A 7.1. táblázat végigviszi a Floyd-algoritmust a példagráfon. Minden

lépésben kiemeltük szürke háttérrel, hogy a d_{k-1} mátrix mely elemeiből épülnek fel a d_k mátrix egyes elemei. Félkövér betűket használtunk, valahányszor a d_k[i][j]←d_{k-1}[i][k]+d_{k-1}[k][j] képlet alkalmazása javított a megfelelő elem értékén.

Ha szeretnénk magukat a legrövidebb utakat is, akkor el kell tárolnunk ezeknek utolsó előtti állomásait. Az ue[1..n][1..n] tömb az előbbi fejezetek algoritmusaiban szereplő apa-tömb szerepét tölti be. Kezdetben az ue tömb nulla értéket tartalmaz ott, ahol a megfelelő pontpár között nincs közvetlen él (nem értelmezett az utolsó előtti állomás), illetve kezdőpontot ott, ahol van (a közvetlen úton az utolsó előtti állomás a kezdőpont). Valahányszor frissül a d tömb (d[i][j]←d[i][k]+d[k][j]), frissítenünk kell az ue tömböt is (ue[i][j]←ue[k][j]). Ha frissítjük a d[i][j] értéket, merthogy a k pontot is tartalmazó (i→k→j) „kerülő út" rövidebb, mint az i és j pontok között eddig talált legrövidebb út (amely legfennebb az 1, 2, ..., k-1 pontokat tartalmazza), akkor az új út utolsó előtti állomása nyilván a (k→j) szakasz utolsó előtti állomása lesz. A 7.2. a és b táblázata a d és ue tömbök végső állapotát mutatja be. A **FELÉPÍT_OPT_ÚT** eljárás az ue tömb alapján felépíti bármely pontpár esetén a legrövidebb utat.

A 7.2. ábra szemléletesen mutatja be a Floyd-algoritmus nyomán történő optimális út-építkezést.

Az alábbiakban bemutatjuk a Floyd-algoritmust:

```
eljárás FLOYD
          (súlyM[1..n][1..n],n,d[1..n][1..n],ue[1..n][1..n])
   minden i ← 1,n végezd
      minden j ← 1,n végezd
         ha súlyM[i][j] = ∞ akkor
            ue[i][j] ← 0
         különben
            ue[i][j] ← i
         vége ha
      vége minden
   vége minden
   d ← súlyM
   minden k ← 1,n végezd
      minden i ← 1,n végezd
         minden j ← 1,n végezd
            ha i≠k ÉS j≠k akkor
               ha d[i][k] + d[k][j] < d[i][j] akkor
                  d[i][j] ← d[i][k]+ d[k][j]
```

7.1. táblázat. *A d_0=súlyM, $d_1, \ldots, d_6$ mátrixsorozat*

$\mathbf{d}_0$	1	2	3	4	5	6
1	0	50	∞	∞	110	20
2	∞	0	100	∞	70	–60
3	∞	∞	0	10	∞	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

→

$\mathbf{d}_1$	1	2	3	4	5	6
1	0	50	∞	∞	110	20
2	∞	0	100	∞	70	–60
3	∞	∞	0	10	∞	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

$\mathbf{d}_1$	1	2	3	4	5	6
1	0	50	∞	∞	110	20
2	∞	0	100	∞	70	–60
3	∞	∞	0	10	∞	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

→

$\mathbf{d}_2$	1	2	3	4	5	6
1	0	50	**150**	∞	110	**–10**
2	∞	0	100	∞	70	–60
3	∞	∞	0	10	∞	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

$\mathbf{d}_2$	1	2	3	4	5	6
1	0	50	150	∞	110	–10
2	∞	0	100	∞	70	–60
3	∞	∞	0	10	∞	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

→

$\mathbf{d}_3$	1	2	3	4	5	6
1	0	50	150	**160**	110	–10
2	∞	0	100	**110**	70	–60
3	∞	∞	0	10	∞	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	**50**	0	**190**
6	∞	∞	∞	∞	80	0

$\mathbf{d}_3$	1	2	3	4	5	6
1	0	50	150	160	110	–10
2	∞	0	100	110	70	–60
3	∞	∞	0	10	∞	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	50	0	190
6	∞	∞	∞	∞	80	0

→

$\mathbf{d}_4$	1	2	3	4	5	6
1	0	50	150	160	110	–10
2	∞	0	100	110	70	–60
3	∞	∞	0	10	**40**	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	50	0	190
6	∞	∞	∞	∞	80	0

$\mathbf{d}_4$	1	2	3	4	5	6
1	0	50	150	160	110	–10
2	∞	0	100	110	70	–60
3	∞	∞	0	10	40	150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	50	0	190
6	∞	∞	∞	∞	80	0

→

$\mathbf{d}_5$	1	2	3	4	5	6
1	0	50	150	160	110	–10
2	∞	0	100	110	70	–60
3	∞	∞	0	10	40	150
4	∞	∞	**70**	0	30	**220**
5	∞	∞	40	50	0	190
6	∞	∞	**120**	**130**	80	0

$\mathbf{d}_5$	1	2	3	4	5	6
1	0	50	150	160	110	–10
2	∞	0	100	110	70	–60
3	∞	∞	0	10	40	150
4	∞	∞	70	0	30	220
5	∞	∞	40	50	0	190
6	∞	∞	120	130	80	0

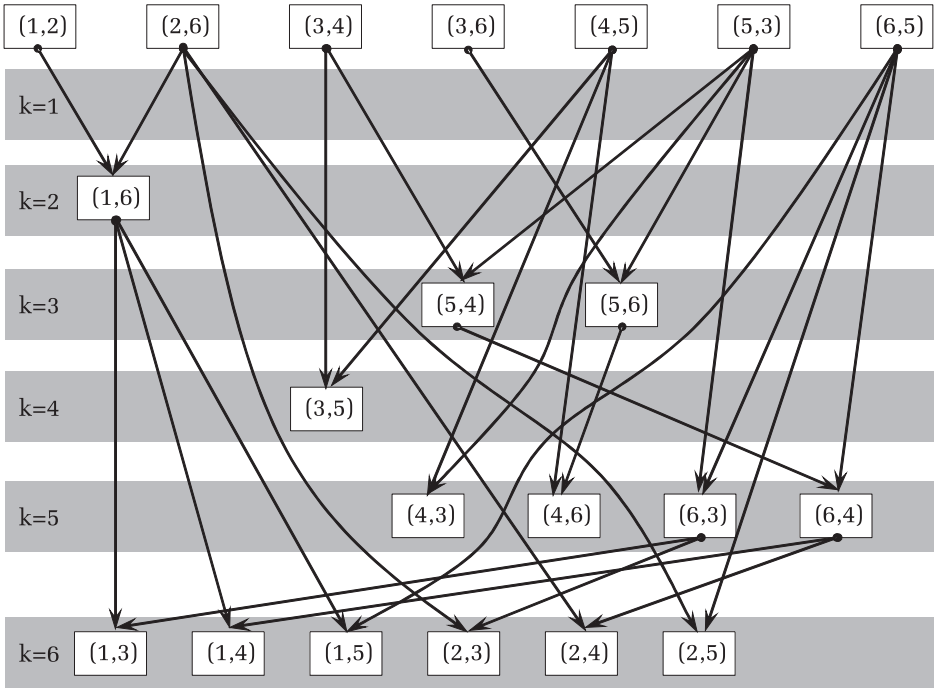
→

$\mathbf{d}_6$	1	2	3	4	5	6
1	0	50	**110**	**120**	**70**	–10
2	∞	0	**60**	**70**	**20**	–60
3	∞	∞	0	10	40	150
4	∞	∞	70	0	30	220
5	∞	∞	40	50	0	190
6	∞	∞	120	130	80	0

7.2. táblázat. *A* d *és* ue *tömbök végső állapota a fejezet példagráfjára*

d	1	2	3	4	5	6
1	**0**	**50**	**110**	**120**	**70**	**-10**
2	∞	**0**	**60**	**70**	**20**	**-60**
3	∞	∞	**0**	**10**	**40**	**150**
4	∞	∞	**70**	**0**	**30**	**220**
5	∞	∞	**40**	**50**	**0**	**190**
6	∞	∞	**120**	**130**	**80**	**0**

ue	1	2	3	4	5	6
1	**0**	**1**	**0**	**0**	**0**	**0**
2	**0**	**0**	**5**	**3**	**6**	**2**
3	**0**	**0**	**0**	**3**	**4**	**3**
4	**0**	**0**	**5**	**0**	**4**	**3**
5	**0**	**0**	**5**	**3**	**0**	**3**
6	**0**	**0**	**5**	**3**	**6**	**0**



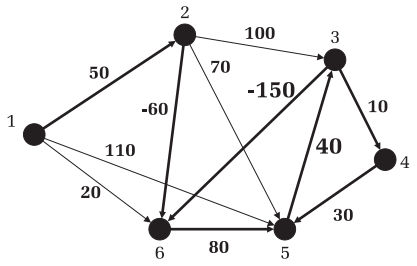
7.2. ábra. *A Floyd-algoritmus optimális út kiépítési stratégiája a példagráfon*

```
                    ue[i][j] ← ue[k][j]
                vége ha
            vége ha
        vége minden
    vége minden
vége minden
vége FLOYD
```

Bonyolultság: A Floyd-algoritmus bonyolultsága (a háromszorosan egymásba ágyazott `minden` ciklusok miatt) $O(n^3)$ ([2], 486. oldal).

Megjegyzések:

- Mivel példaként ugyanazt a gráfot használtuk, mint a 6.3. alfejezetben, ezért megfigyelhető, hogy a $\mathtt{d}_6$ mátrix első sora megegyezik a Bellman–Ford-algoritmus `d` tömbjének végső állapotával.
- Ha a gráf tartalmaz negatív összsúlyú köröket, akkor a Floyd-algoritmus kétféleképpen is jelezheti ezt. A 7.3. táblázatok ezt szemléltetik a 7.3. példagráfra vonatkoztatva.
  1. A `d` tömb főátlóján a negatív kört alkotó elemeknek megfelelő pozíciókban negatív értékek jelennek meg. E pontok esetén létezik rövidebb út önmagukhoz, mint a helyben maradás nulla hosszú útja.
  2. Ha folytatnánk az algoritmust, az $(n+1)$-edik menetben is történnének finomítások.



7.3. ábra. *Példa arra az esetre, amikor a gráf tartalmaz negatív összsúlyú kört. A (3, 6, 5, 3) kör összsúlya –30*

7.3. táblázat. *A $d_0, d_1, \ldots, d_6, d_7$ mátrixsorozat*

$\mathbf{d}_0$	1	2	3	4	5	6
1	0	50	∞	∞	110	20
2	∞	0	100	∞	70	-60
3	∞	∞	0	10	∞	-150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

$\mathbf{d}_1$	1	2	3	4	5	6
1	0	50	∞	∞	110	20
2	∞	0	100	∞	70	-60
3	∞	∞	0	10	∞	-150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

$\mathbf{d}_2$	1	2	3	4	5	6
1	0	50	150	∞	110	-10
2	∞	0	100	∞	70	-60
3	∞	∞	0	10	∞	-150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	∞	0	∞
6	∞	∞	∞	∞	80	0

$\mathbf{d}_3$	1	2	3	4	5	6
1	0	50	150	160	110	-10
2	∞	0	100	110	70	-60
3	∞	∞	0	10	∞	-150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	50	0	-110
6	∞	∞	∞	∞	80	0

$\mathbf{d}_4$	1	2	3	4	5	6
1	0	50	150	160	110	-10
2	∞	0	100	110	70	-60
3	∞	∞	0	10	40	-150
4	∞	∞	∞	0	30	∞
5	∞	∞	40	50	0	-110
6	∞	∞	∞	∞	80	0

$\mathbf{d}_5$	1	2	3	4	5	6
1	0	50	150	160	110	-10
2	∞	0	100	110	70	-60
3	∞	∞	0	10	40	-150
4	∞	∞	70	0	30	-80
5	∞	∞	40	50	0	-110
6	∞	∞	120	130	80	**-30**

$\mathbf{d}_6$	1	2	3	4	5	6
1	0	50	110	120	70	-40
2	∞	0	60	70	20	-90
3	∞	∞	**-30**	-20	-70	-180
4	∞	∞	40	0	0	-110
5	∞	∞	10	20	**-30**	-140
6	∞	∞	90	100	50	**-60**

$\mathbf{d}_7$	1	2	3	4	5	6
1	0	50	**80**	**60**	**10**	**-100**
2	∞	0	**30**	**10**	**-40**	**-150**
3	∞	∞	**-60**	**-80**	**-130**	**-240**
4	∞	∞	**-20**	**-100**	**-150**	**-260**
5	∞	∞	**-50**	**-130**	**-180**	**-290**
6	∞	∞	**30**	**-50**	**-100**	**-210**

8. FEJEZET

A KRITIKUS ÚT MÓDSZERE

Gyakorlati probléma: Ha egy építkezési vállalat megbízást kap egy bizonyos munkálat elvégzésére, akkor az egész munkálatot munkaszakaszokra bontja, a tapasztalat alapján megállapítja az egyes munkaszakaszok elvégzéséhez szükséges időt. Feltevődik a kérdés, hogy mekkora a legrövidebb idő, ami alatt az egész munkálat befejezhető. Ha az egyes munkaszakaszok kezdetét és végét pontok, míg a munkálat hosszát a nekik megfelelő hosszúságú élek jelentik, akkor az egész munkálatra felépíthető egy gráf. A feladat a gráfelmélet nyelvén: a munkálat kezdetét jelentő és a munkálat végét jelentő két csomópont között melyik a leghosszabb út?

A 6. fejezetben bemutatott algoritmusok lehetővé teszik különböző típusú irányított gráfok valamely pontjától az összes többi ponthoz vezető legrövidebb utak meghatározását polinomiális algoritmussal[1]. A topologikus sorrenden alapuló algoritmus, amely körmentes gráfok esetén működik, áthangolható leghosszabb utak meghatározására. Ennek egy gyakorlati alkalmazását tárgyalja ez a fejezet.

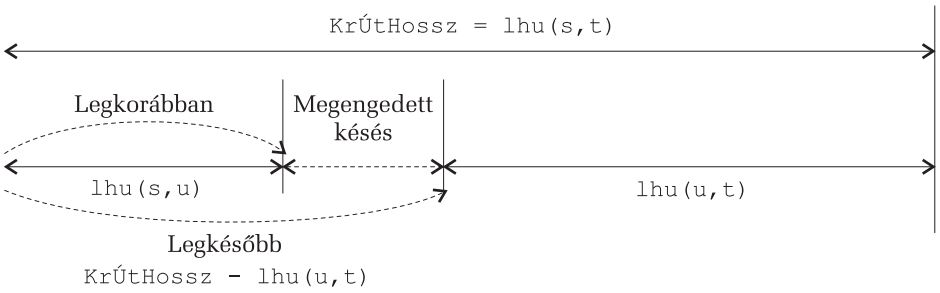
Tegyük fel, hogy G egy összefüggő, körmentes, súlyozott, irányított gráf, amelynek van egy forráspontja (s) és egy nyelőpontja (t). Egy ilyen gráfot *tevékenység-gráf*nak nevezünk, amennyiben az élei tevékenységeket ábrázolnak. Az élek súlyértékei a megfelelő tevékenységek időigényét képviselik. A gráf pontjaihoz események rendelhetők. Egy adott pont képviselte esemény akkor következik be, ha minden be-éle ábrázolta tevékenység befejeződött, és kezdődhetnek a ki-pontokhoz rendelt tevékenységek. Az $s = 1$ pont a „munkálatok" elkezdését, a $t = n$ pont pedig a befejezés eseményét képviseli. Feltételezzük, hogy az s eseményre a 0. időpontban kerül sor.

Az s ponttól a t ponthoz vezető leghosszabb utat (vagy utakat) a tevékenység-gráf *kritikus út*jának (vagy útjainak) nevezzük. A kritikus út menti élek kritikus tevékenységeket, a pontok pedig kritikus eseményeket ábrázolnak. A kritikus út hossza a munkálatok összidejét adja

1 A polinomiális algoritmus mint fogalom definíciója végett lásd az A függeléket.

meg. Azért találó a kritikus megnevezés, mert a kritikus tevékenységek időbeni csúszása megnövelné a teljes munka időigényét. Szerződés alapú munkavállalások esetén ez penalizációkat vonhat maga után. Ezért a munkavezetőknek különösen a kritikus tevékenységekre kell odafigyelniük, hogy azokat időben végezzék el.

A feladat értelmében meg kell határoznunk egy tevékenység-gráf kritikus útjait. Ezek hosszát `KrÚtHossz`-szal jelöljük. Továbbá, minden eseményre kiszámítjuk a legkorábbi időpontot, amikor ez bekövetkezhet, illetve azt a legkésőbbi időpontot, ameddig késhet, anélkül hogy ez befolyásolná a munkálatok összidőigényét. A kritikus események esetén ez a két érték természetesen egybe fog esni. Egy adott `u` ponthoz rendelt legkorábbi időpont nyilván az `s` ponttól hozzá vezető leghosszabb út súlya lesz. Jelölje ezt: `lhu(s,u)`. Az `u` által ábrázolt eseményre azután kerül sor, miután az `s`-ből `u`-ba vezető összes út élei által képviselt tevékenységek lejártak. Konkrétabban, az `u` esemény pontosan akkor következik be, amikor a hozzá vezető leghosszabb út által ábrázolt egymás utáni események már mind befejeződtek. Ami az `u` ponthoz tartozó legkésőbbi időpontot illeti, ez értelemszerűen a `(KrÚtHossz - lhu(u,t))` érték lesz. Az `u` esemény bekövetkezte után még `lhu(u,t)` időnek kell eltelnie (az `u`-t követő leghosszabb tevékenységlánc időigénye), amíg a `t` esemény bekövetkezhet. Lásd a 8.1. ábrát.



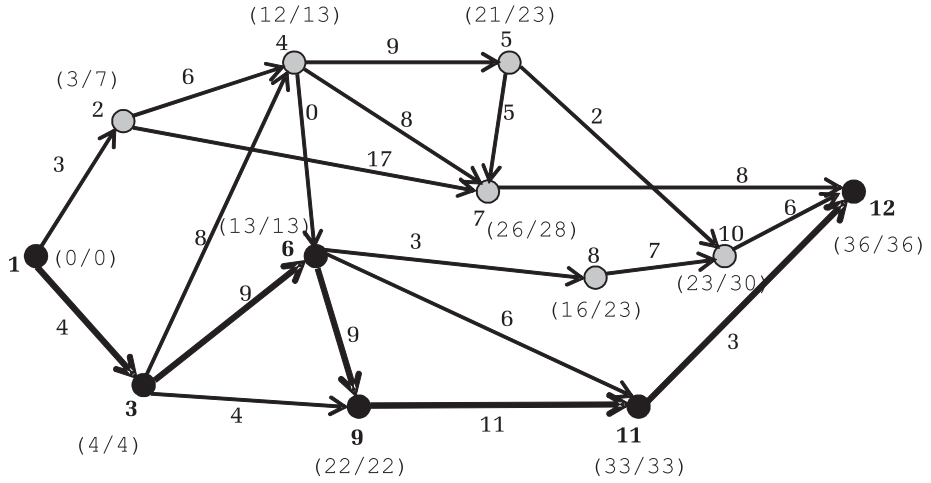
8.1. ábra. *Az u ponthoz tartozó legkorábbi és legkésőbbi időpontok, illetve a megengedett késés*

Stratégia: Ha lefuttatunk a `G` gráfra egy `s` pontból induló mélységi bejáráshoz hasonló eljárást, akkor minden `u` pont elhagyásakor (fordított topologikus sorrend) meghatározható ennek az `lhu(u,t)` értéke. Ez az érték `maximum(súly(u,v)+lhu(v,t))` lesz, ahol `v`∈`kiszomszédok(u)`. A fordított topologikus sorrendből adódóan az `lhu(v,t)` értékek már

rendelkezésre állanak az `lhu(u,t)` meghatározásakor. Az `lhu(t,t)` érték nyilván nulla.

Képezzük ezután `G` transzponált gráfját (G^T), és hajtsunk végre erre egy `t` pontból induló mélységi bejáráshoz hasonló eljárást. Ez lehetővé teszi, hogy minden `u` pont elhagyásakor (az eredeti gráf szerinti topologikus sorrend) beállítsuk ennek `lhu(s,u)` értékét.

A 8.2. ábrán látható példagráfon megvastagítottuk a kritikus utat, a kritikus tevékenységeket (éleket) és a kritikus eseményeket (pontokat). Minden pont mellett feltüntettük a legkorábbi/legkésőbbi időpontpárost. A két érték különbsége a megengedett késés. A számítások végett lásd a fejezet végén a megjegyzést.



8.2. ábra. *Tevékenység-gráf ($s = 1$, $t = 12$). Kritikus út: 1, 3, 6, 9, 11, 12. A kritikus út hossza: 36*

Algoritmus: Az alábbiakban bemutatásra kerülő `KRITIKUS_ÚT` eljárás meghívja kétszer az `LHU` függvényt. A `lhu_t[1..n]` és `lhu_s[1..n]` tömbök minden `u` pont esetén az `lhu(u,t)`, illetve `lhu(s,u)` értékeket tárolják:

```
lhu_t[u] = lhu(u,t)
lhu_s[u] = lhu(s,u)
```

Minden olyan `u` pontra, amelyre még nem számítottuk ki az `lhu(u,t)` és `lhu(s,u)` értékeket, a megfelelő tömbelemek (-1)-et tartalmaznak. A triviális eseteknek megfelelő tömbelemek értékei természetesen nullák. Az `LHU(G,s,t,lhu_t,KövKrPont)` függvényhívás feltölti az `lhu_t` tömböt,

és visszatéríti a kritikus út hosszát. A KövKrPont[1..n] tömbben azt is eltároljuk, hogy minden u pont esetén melyik ki-szomszéd szolgáltatja az optimum (maximum) értéket. E tömb alapján utólag rekonstruálható maga a kritikus út, hiszen a KövKrPont[u] tömbelem azt tárolja, hogy melyik pont követi u-t a kritikus úton (KiírKrÚt eljárás).

Hasonlóképpen az LHU(G^T,t,s,lhu_s,ElőKrPont) függvényhívás feltölti az lhu_s tömböt, és eredményként ugyanazt a kritikus út hosszértéket adja vissza. Az ElőKrPont[1..n] tömbben minden u pontnak az optimális be-szomszédja (a transzponált gráfban ki-szomszéd) tárolódik el. Az s forrás és t nyelő esetén ElőKrPont[s] = 0 és KövKrPont[t] = 0. A TRANSZPONÁL(G,G^T) eljárás G-ből (G szomszédsági listájából) előállítja G^T-t (G^T szomszédsági listáját). A KiírKrIdőpontok eljárás kiírja minden pont esetén a megfelelő esemény bekövetkezhetőségének legkorábbi időpillanatát, illetve azt a legkésőbbi időpontot, ameddig ha bekövetkezik, akkor a késés nem növeli a teljes munkálat időigényét.

Az LHU függvény rekurzívan implementált, dinamikus programozásos stratégiát alkalmaz. Az ismétlődő részfeladatok többszöri megoldásának elkerülése végett a kiszámított optimumértékeket eltároltuk (az lhu_t és lhu_s tömbökben), és a rekurzív hívásokat csak akkor ejtettük meg, amikor nem állt már rendelkezésre az illető ki-szomszéd megfelelő lhu értéke.

```
eljárás KRITIKUS_ÚT (G(V,E,súly),s,t)
   minden u $\in $ V(G) végezd
      lhu_t[u] ← -1
      lhu_s[u] ← -1
   vége minden
   lhu_t[t] ← 0
   lhu_s[s] ← 0
   KövKrPont[t] ← 0
   ElőKrPont[s] ← 0
   KrÚtHossz ← LHU(G,s,t,lhu_t,KövKrPont)
   TRANSZPONÁL(G,G^T)
   KrÚtHossz ← LHU(G^T,t,s,lhu_s,ElőKrPont)
   Kiír: KrÚtHossz
   KiírKrÚt(KövKrPont,s)
   KiírKrIdőpontok(KrÚtHossz,lhu_t,lhu_s,n)
vége KRITIKUS_ÚT

függvény LHU(G(V,E,súly),u,t,lhu_t[1..n],KövKrPont[1..n])
   minden v ∈ ki_szomszéd(u) végezd
      ha lhu_t[v] = -1 akkor
```

```
            lhu_t[v] ← LRU(v)
         vége ha
         ha lhu_t[v] + súly(u,v) > lhu_t[u] akkor
            lhu1_t[u] ← lhu_t[v] + súly(u,v)
            KövKrPont[u] ← v
         vége ha
      vége minden
      vissza lhu_t[u]
vége LHU

eljárás KiírKrIdőpontok
                        (KrÚtHossz,lhu_t[1..n],lhu_s[1..n],n)
      minden u ← 1,n végezd
         Kiír: "Legkorábbi időpont: ", lhu_s[u]
         Kiír: "Legkorábbi időpont: ", KrÚtHossz - lhu_t[u]
         Kiír: "Maximális megengedett késés: ",
               KrÚtHossz - lhu_s[u] - lhu_t[u]
      vége minden
vége KiírKrIdőpontok

eljárás KiírKrÚt(KövKrPont[1..n],s)
      u ← s
      végezd
         Kiír: u
         u ← KövKrPont[u]
      amíg u ≠ 0
vége KiírKrÚt
```

Bonyolultság: A kritikus utakat meghatározó algoritmus bonyolultsága (akárcsak a topologikus sorrenden alapuló legrövidebb út algoritmus) $O(n+m)$ ([2], 464. oldal).

Megjegyzés: A körmentes gráfok szintekre bonthatók. A 8.3. ábrán a 8.2. példagráf szintekre bontását láthatjuk. Az `lhu(s,u)` értékek kiszámíthatók szintről szintre haladva balról jobbra irányban. Hasonló módon határozhatók meg az `lhu(u,t)` értékek is: szintről szintre haladva jobbról balra irányban. E két ellentétes pontsorrend megfelel egy topologikus, illetve egy fordított topologikus sorrendnek, ahogy a fenti algoritmus is ilyen sorrendekre épül. A számoláshoz a rekurzív képletek:

$$lhu(s,u) = \max\{lhu(s,v) + \textit{súly}(v,u) \mid v \in \textit{be-szomszédok}(u)\},$$
$$\text{ha } u \in V(G) - \{s\}$$
$$= 0, \text{ ha } u = s$$

$$lhu(u,t) = \max\{lhu(v,t) + \textit{súly}(u,v) \mid v \in \textit{ki-szomszédok}(u)\},$$
$$\text{ha } u \in V(G) - \{t\}$$
$$= 0, \text{ ha } u = t$$

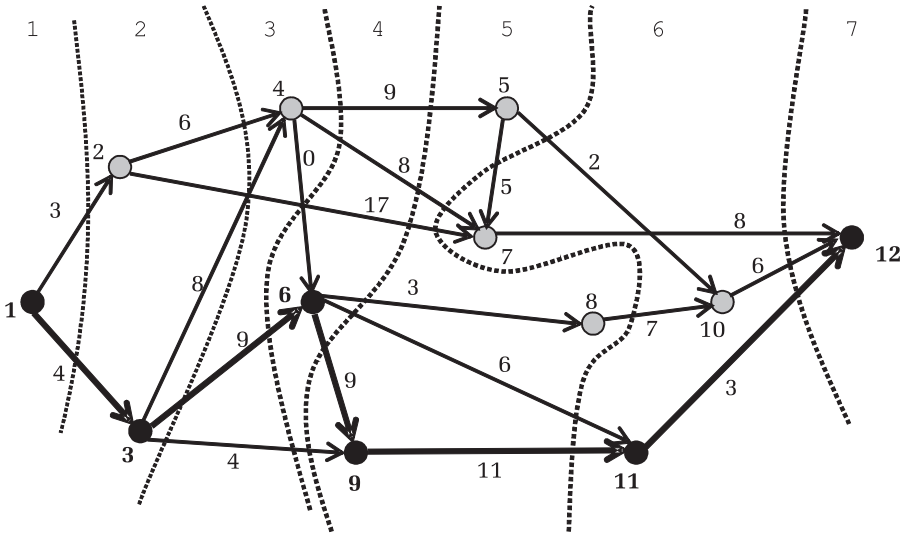
Az alábbiakban az `lhu(s,u)` értékek kiszámítását láthatjuk. Kiemeltük a kritikus pontokat és az ezekhez tartozó `lhu(s,u)` értékeket. A kritikus úton minden kritikus pont esetében megjelöltük az őt megelőző pontot is.

1. szint:

$$\boldsymbol{lhu(1,1) = 0}$$

2. szint:

$$lhu(1,2) = \max\{lhu(1,1) + \textit{súly}(1,2)\} = \max\{3\} = 3$$
$$\boldsymbol{lhu(1,3)} = \max\{lhu(1,\mathbf{1}) + \textit{súly}(1,3)\} = \max\{\mathbf{4}\} = \mathbf{4}$$



8.3. ábra. *Szintekre bontott tevékenység-gráf*

3. szint:

$$\begin{aligned} lhu(1,4) &= \max\{lhu(1,2)+súly(2,4), lhu(1,3)+súly(3,4)\} = \\ &= \max\{9,12\} = 12 \end{aligned}$$

4. szint:

$$\begin{aligned} \boldsymbol{lhu(1,6)} &= \max\{lhu(1,4)+súly(4,6), lhu(1,\mathbf{3})+súly(3,6)\} = \\ &= \max\{12,\mathbf{13}\} = \mathbf{13} \end{aligned}$$

5. szint:

$$\begin{aligned} lhu(1,5) &= \max\{lhu(1,4)+súly(4,5)\} = \max\{21\} = 21 \\ lhu(1,8) &= \max\{lhu(1,6)+súly(6,8)\} = \max\{16\} = 16 \\ \boldsymbol{lhu(1,9)} &= \max\{lhu(1,\mathbf{6})+súly(6,9), lhu(1,3)+súly(3,9)\} = \\ &= \max\{\mathbf{22},8\} = \mathbf{22} \end{aligned}$$

6. szint:

$$\begin{aligned} lhu(1,7) &= \max\{lhu(1,2)+súly(2,7), lhu(1,4)+súly(4,7), \\ &\quad lhu(1,5)+súly(5,7)\} = \\ &= \max\{20,20,26\} = 26 \\ lhu(1,10) &= \max\{lhu(1,5)+súly(5,10), lhu(1,8)+súly(8,10)\} = \\ &= \max\{23,23\} = 23 \\ lhu\mathbf{(1,11)} &= \max\{lhu(1,6)+súly(6,11), lhu(1,\mathbf{9})+súly(9,11)\} = \\ &= \max\{19,\mathbf{33}\} = \mathbf{33} \end{aligned}$$

7. szint:

$$\begin{aligned} lhu\mathbf{(1,12)} &= \max\{lhu(1,7)+súly(7,12), lhu(1,10)+súly(10,12), \\ &\quad lhu(1,\mathbf{11})+súly(11,12)\} = \\ &= \max\{34,29,\mathbf{36}\} = \mathbf{36} \end{aligned}$$

```
lhu_s[1..12] = (0, 3, 4, 12, 21, 13, 26, 16, 22, 23, 33, 36)
ElőKrPont[1..12] = (-1, 1, 1, 3, 4, 3, 5, 6, 6, 5, 9, 11)
```

Az alábbiakban az $lhu(u,t)$ értékek kiszámítását láthatjuk. Kiemeltük a kritikus pontokat és az ezekhez tartozó $lhu(u,t)$ értékeket. A kritikus úton minden kritikus pont esetében megjelöltük az őt követő pontot is.

7. szint:

$$\boldsymbol{lhu(12,12)=0}$$

6. szint:

$$\begin{aligned} lhu(7,12) &= \max\{lhu(12,12)+súly(7,12)\} = \max\{8\} = 8 \\ lhu(10,12) &= \max\{lhu(12,12)+súly(10,12)\} = \max\{6\} = 6 \\ \boldsymbol{lhu(11,12)} &= \max\{lhu(\mathbf{12},12)+súly(11,12)\} = \\ &= \max\{3\} = \mathbf{3} \end{aligned}$$

5. szint:

$$\begin{aligned} lhu(5,12) &= \max\{lhu(7,12)+súly(5,7), lhu(10,12)+súly(5,10)\} = \\ &= \max\{13,10\} = 13 \\ lhu(8,12) &= \max\{lhu(10,12)+súly(8,10)\} = \max\{13\} = 13 \\ \boldsymbol{lhu(9,12)} &= \max\{lhu(\mathbf{11},12)+súly(9,11)\} = \\ &= \max\{14\} = \mathbf{14} \end{aligned}$$

4. szint:

$$\begin{aligned} lhu(\mathbf{6,12}) &= \max\{lhu(8,12)+súly(6,8), lhu(11,12)+súly(6,11), \\ &\quad lhu(\mathbf{9},12)+súly(6,9)\} = \\ &= \max\{16,9,23\} = \mathbf{23} \end{aligned}$$

3. szint:

$$\begin{aligned} lhu(4,12) &= \max\{lhu(5,12)+súly(4,5), lhu(7,12)+súly(4,7), \\ &\quad lhu(6,12)+súly(4,6)\} = \\ &= \max\{22,16,23\} = 23 \end{aligned}$$

2. szint:

$$\begin{aligned} lhu(2,12) &= \max\{lhu(4,12)+súly(2,4), lhu(7,12)+súly(2,7)\} = \\ &= \max\{29,25\} = 29 \\ lhu(\mathbf{3,12}) &= \max\{lhu(4,12)+súly(3,4), lhu(\mathbf{6},12)+súly(3,6), \\ &\quad lhu(9,12)+súly(3,9)\} = \\ &= \max\{21,32,18\} = \mathbf{32} \end{aligned}$$

1. szint:

$$lhu(\mathbf{1,12}) = \max\{lhu(2,12) + súly(1,2), lhu(\mathbf{3},12) + súly(1,3)\} = \\ = \max\{32,36\} = \mathbf{36}$$

```
lhu_t[1..12] = (36, 29, 32, 23, 13, 23, 8, 13, 14, 6, 3, 0)
KövKrPont[1..12] = (3, 4, 6, 6, 7, 9, 12, 10, 11, 12, 12, -1)
```

A következőkben minden egyes `u` pont (esemény) esetén a legkorábbi (amikor bekövetkezhet: `lhu_s[u]`) és legkésőbbi (amikor be kell következnie: `KrÚtHossz-lhu_t[u]`) időpontokat láthatjuk, illetve a megengedett késéseket (`KrÚtHossz-lhu_s[u]-lhu_t[u]`):

pont	„legkorábbi időpont"	„legkésőbbi időpont"	megengedett késés
1	**0**	**0**	**0**
2	3	7	4
3	**4**	**4**	**0**
4	12	13	1
5	21	23	2
6	**13**	**13**	**0**
7	26	28	2
8	16	23	7
9	**22**	**22**	**0**
10	23	30	7
11	**33**	**33**	**0**
12	**36**	**36**	**0**

9. FEJEZET

HÁLÓZATI FOLYAMOK

Legyen $G(V, E)$ egy összefüggő súlyozott irányított gráf, amelynek van egy forráspontja (s) és egy nyelőpontja (t). Az élek súlyértékeit ez esetben kapacitásnak nevezzük, és feltételezzük, hogy nem negatív valós értékek: $c : E \to \mathbb{R}_+$. A (G, s, t, c) négyest hálózatnak nevezzük.

Minden hálózat esetén beszélhetünk a hálózaton átfolyó folyamról. Ez úgy képzelhető el, mintha a hálózati gráf egy vízvezetékrendszer lenne, amelyen átfolyik egy bizonyos mennyiségű víz. A kapacitásértékek a csövek vastagságát jelölik. A víz a forráspontból „buzog fel" (itt termelődik), és a nyelőpont „nyeli el". A hálózaton átfolyó folyamot definiálni nem jelent mást, mint megmondani, hogy a hálózati gráf minden egyes élén mekkora az átfolyó folyammennyiség. Konkrétabban, az $f : E \to \mathbb{R}_+$ függvényt a (G, s, t, c) hálózaton átfolyó folyamnak nevezzük, ha teljesül az alábbi két feltétel:

1. $f((u, v)) \leqslant c((u, v))$ bármely $(u, v) \in E(G)$. Egyetlen élen sem nagyobb a folyamérték, mint az illető él kapacitása.
2. Bármely $u \in V(G) - \{s, t\}$ pontok esetén a be-éleken befolyó folyamértékek összege egyenlő a ki-éleken kifolyó folyamértékek összegével (folyammegmaradási feltétel). Nincs folyamelnyelés vagy -termelés a hálózat „csatlakozási" pontjaiban. Ami kifolyt a forrásból, pontosan az jut el a nyelőbe.

A hálózaton átfolyó folyamérték egyenlő a forrásból a forráspont ki-élein kifolyó folyammennyiséggel, azaz e ki-élek folyamértékeinek összegével. Ez az érték természetesen azonos a nyelőpont be-élei folyamértékeinek összegével. Ami minket érdekel, az egy adott hálózaton átfolyható maximális folyamérték. Ezt határozza meg *Ford–Fulkerson* algoritmusa.

Normális körülmények között, ha egy s-ből t-be vezető útról ($s \to t$) beszélünk, akkor ezt úgy értjük, hogy az útmenti élek azonos – az út irányításával megegyező – irányításúak. Tekintsünk el ez alkalommal e feltételezéstől. Tehát egy hálózati gráf $s \to t$ útján egy olyan s-ből t-be vezető utat fogunk érteni, amelyen az élek tetszőleges irányításúak lehetnek. Természetesen megmaradnak azok a követelmények, hogy egy

út nem tartalmazhatja kétszer ugyanazt a pontot, illetve hogy az útmenti szomszédos éleknek illeszkedniük kell egymáshoz.

Haladjunk végig egy adott $s \to t$ úton ebben az irányban (s-től t felé). Azokat az éleket, amelyeken irányításuknak megfelelően haladunk át, *az illető útra nézve előremutató*, a többieket pedig *az illető útra nézve visszamutató* éleknek nevezzük. A hálózati gráf egy adott $s \to t$ útjának valamely élét *az illető útra nézve telített*nek nevezzük, ha előremutató és a kapacitásával megegyező értékű folyam halad át rajta, vagy visszamutató és nulla mennyiségű folyam folyik vissza rajta. Másfelől: a hálózati gráf egy adott $s \to t$ útjának valamely élét *az illető útra nézve telítetlen*nek nevezzük, ha előremutató és a kapacitásánál kisebb értékű folyam halad át rajta, vagy visszamutató és nullánál nagyobb mennyiségű folyam folyik vissza rajta. *Javítóút*nak nevezünk egy olyan $s \to t$ utat, amelynek minden éle telítetlen.

Minden javítóút mentén növelhető a hálózaton átfolyó folyamérték. Hogyan? Ha egy javítóút minden előremutató élén növeljük, és minden visszamutató élén csökkentjük a folyamértéket f_0-val, akkor a hálózaton áthaladó folyammennyiség nő f_0-val. Egy ilyen művelet csak akkor megengedett, ha a végrehajtása után az éleken átfolyó folyamértékek továbbra is kielégítik a folyam definíciójának két alapfeltételét. Ami a második feltételt illeti (megmaradási feltétel), nyilvánvaló, hogy csak a szóban forgó javítóút közbeeső pontjaira kell ellenőrizni, hogy teljesül-e továbbra is. Legyen u egy ilyen pont. Az u pontot a javítóúton közvetlenül megelőző és követő élek irányítását tekintve, négy eset állhat fenn:

- Ha u-nak mindkét éle be-él, akkor az első előremutató, a második pedig visszamutató. Mivel az előremutató élek folyamértékét növeljük f_0-val, a visszamutatókét pedig csökkentjük ugyanennyivel, u be-éleinek összfolyamértéke változatlan marad, azaz továbbra is egyenlő lesz az u ki-élein kifolyó folyamértékek összegével.
- Ha mindkét él ki-éle u-nak, akkor az első visszamutató, a második pedig előremutató. Mivel az előremutató élek folyamértékét növeljük f_0-val, a visszamutatókét pedig csökkentjük ugyanannyival, u ki-éleinek összfolyamértéke változatlan marad, azaz továbbra is egyenlő lesz az u be-élein befolyó folyamértékek összegével.
- Ha az első él be-éle, a második pedig ki-éle az u pontnak, akkor mindkettő előremutató, és ily módon mindkettő folyamértéke nő

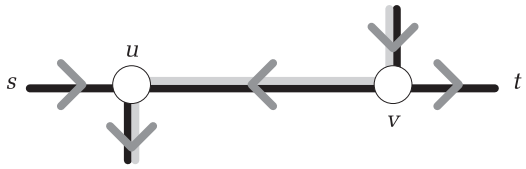
az f_0 értékkel. Ez azt jelenti, hogy u be-élein is és ki-élein is az összfolyamérték ugyanannyival nő, azaz egyenlő marad.

– Ha az első él ki-éle, a második pedig be-éle az u pontnak, akkor mindkettő visszamutató, és ily módon mindkettő folyamértéke csökken ugyanazzal az f_0 értékkel. Ez azt jelenti, hogy u ki-élein is és be-élein is az összfolyamérték ugyanannyival csökken, azaz egyenlő marad.

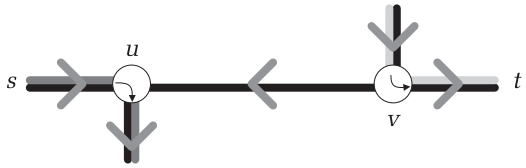
Mennyi a legnagyobb érték, amivel egy javítóút mentén növelhető a folyam? Ennek meghatározásában a folyamdefiníció első követelményét kell figyelembe vennünk. Egyértelmű, hogy ha (u, v) előremutató éle az illető útnak, akkor legtöbb $c((u, v)) - f((u, v))$ értékkel növelhető a rajta áthaladó folyam. Másfelől, ha (u, v) visszamutató él, akkor legtöbb $f((u, v))$-vel csökkenthető a folyam értéke. Nyilvánvalóan e maximumok minimuma lesz az a *legnagyobb közös érték*, amellyel az út minden élén „javítható" a folyamérték (az előremutatókon növelhető, a visszamutatókon csökkenthető). Mivel minden javítóútnak az első éle előremutató ki-éle s-nek, ezért a rajta átfolyó folyammennyiség növelése a forráspont termelésének növelését feltételezi. Más szóval, a fent leírt javítási művelet valóban növeli a hálózaton átfolyó folyammennyiséget.

Magától értetődő, hogy az előremutató élek folyamértékeinek növelésével nő a hálózaton átfolyó folyammennyiség. Nehezebben átlátható azonban az, hogy hogyan járul ehhez hozzá a visszamutató élek folyamértékeinek csökkentése. Intuitíve: ha kevesebb folyam folyik vissza, akkor több folyik előre. Lássuk konkrétabban, hogy mi is történik a „színfalak mögött". Tegyük fel, hogy az $s \to t$ javítóút u és v pontjai között (u megelőzi v-t) visszamutató élek találhatók. Ez esetben az u pontra illeszkedő mindkét útmenti él be-éle u-nak, és a v pontot közvetlenül megelőző, illetve követő útmenti élek ki-élei ennek. Mivel s-nek csak ki-élei vannak és t-nek csak be-élei, ezért u nem lehet s, és v nem lehet t. Ha az $u \to v$ visszaél-szakaszon csökkentjük a folyamértékeket f_0-val, akkor az u pontba f_0-val több folyam folyhat be u javítóút menti előremutató be-élén. E többletfolyam az $u \to v$ visszaél-szakaszról visszavont folyammennyiség helyén fog továbbhaladni az u pont ki-élein. Mi történik a v pontban? Az az f_0 mennyiségű be-folyam, ami eddig a v pont visszamutató élén folyt tovább, ezután v javítóút menti előremutató ki-élén fog továbbhaladni. Ezért nőtt f_0-val v javítóút menti előremutató ki-élének a folyamértéke (9.1., 9.3. ábrák). Mivel az u pont ki-élein, illetve a v pont be-élein a ki-, illetve be-folyamok értékei változatlanok maradnak, ezért

egy kicsit olyan ez, mintha az u pontba a javítóúton s irányából befolyó többletfolyam eltűnne u-ban, majd v-ben előbukkanna, és folytatná útját a javítóúton előre irányban t felé. Ez az észrevétel azt is megmagyarázza, hogy a javítóút menti korrekciók miért nem befolyásolják a folyamértékeket a hálózat többi élein.



9.1. ábra. *Folyamértékek javítás előtt. Világosszürke: az $s \to t$ javítóút $u \to v$ visszamutató szakaszán visszafolyó f_0 folyammennyiség*



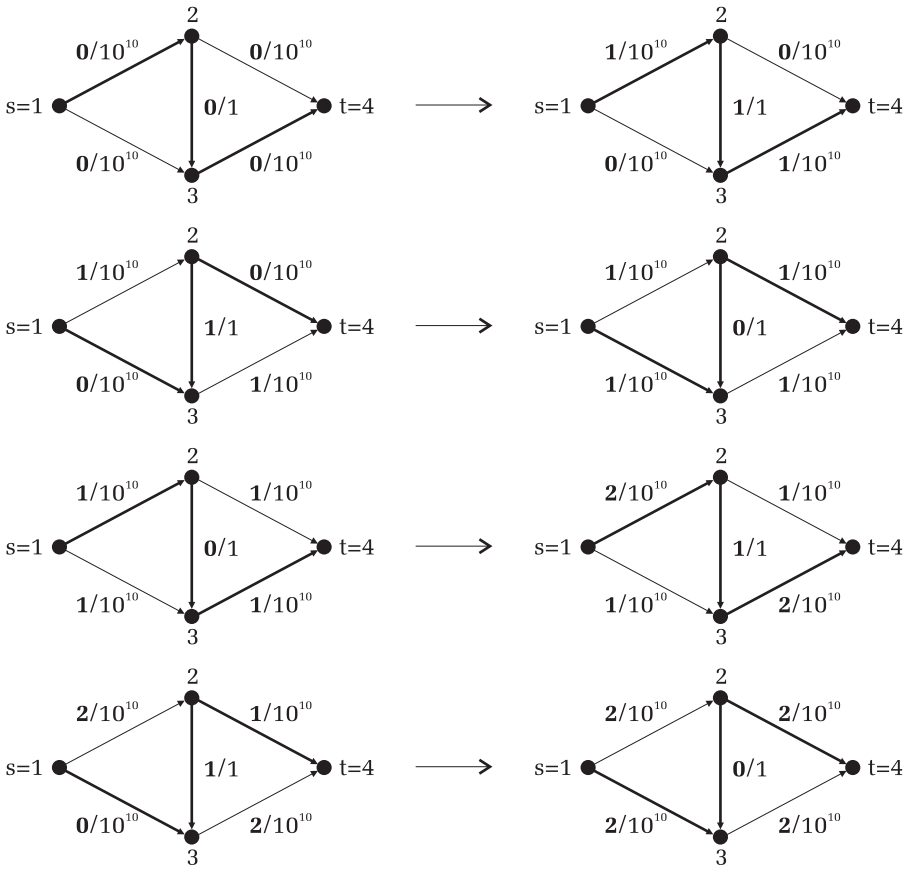
9.2. ábra. *Folyamértékek javítás után. Sötétszürke: az u pontban elterelt f_0 plusz folyammennyiség. Világosszürke: a v pontban átirányított f_0 folyammennyiség. Fekete: változatlanul maradt folyammennyiségek*

Stratégia, algoritmus: A Ford–Fulkerson-algoritmus (`FORD_FULKERSON` függvény; visszatéríti a maximális folyamértéket: fe) az alábbi gondolatmenetet követi.

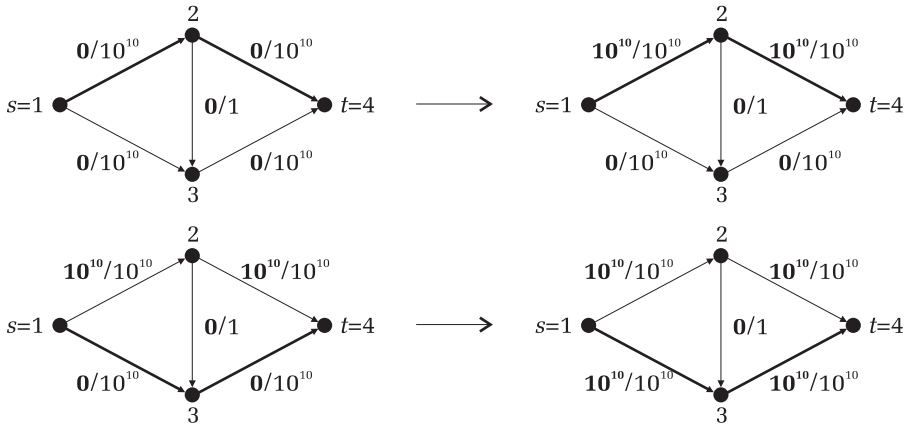
- Kezdetben minden élen a folyamérték nulla.
- Az algoritmus addig ismétli az alábbi három műveletet, amíg létezik $s \to t$ javítóút:
  1. Keres egy javítóutat (`KERES_JAVÍTÓ_ÚT` eljárás).
  2. Meghatározza a maximumot, növelhető az illető út mentén a folyamérték (`JAVÍTÁS` eljárás).
  3. Elvégzi az út élein a javítás által feltételezett folyamértékkorrekciókat (`JAVÍTÁS` eljárás).
- A maximális folyamérték a javítóértékek összege lesz.

9.1. tétel (Az algoritmus alapjául szolgál). Egy hálózaton átfolyó folyammennyiség akkor és csakis akkor maximális, ha a hálózatban nem létezik javítóút. (Bizonyítás végett lásd a [6], 65. oldalt.)

Nem mindegy, hogy milyen sorrendben találjuk meg a javítóutakat. Ezt szemlélteti az alábbi példa (9.2., 9.4. ábrák). Az élek mentén az első szám a folyamértéket jelenti, a második pedig a kapacitást. Ha először az $(1 = s, 2, 3, 4 = t)$ út mentén javítunk 1-gyel, azután az $(1 = s, 3, 2, 4 = t)$ út mentén 1-gyel, azután újra az $(1 = s, 2, 3, 4 = t)$ út mentén 1-gyel, majd megint az $(1 = s, 3, 2, 4 = t)$ út mentén 1-gyel, és így tovább, akkor $2 \cdot 10^{10}$ lépés után kapjuk meg a maximális folyamot (9.2. ábra). Ellenben, ha először az $(1 = s, 2, 4 = t)$ utat választanánk, majd pedig az $(1 = s, 3, 4 = t)$ utat, akkor két lépés után rendelkezésre állna a $2 \cdot 10^{10}$ értékű maximális folyam (9.4. ábra).



9.3. ábra. *Maximális folyam* $2 \cdot 10^{10}$ *lépésből. Az ábrákon az első négy lépést látjuk*



9.4. ábra. *Maximális folyam 2 lépésből*

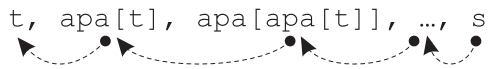
9.2. tétel (Edmond–Karps). Ha minden lépésben a legkevesebb élet tartalmazó javítóutat választjuk, akkor a Ford–Fulkerson-algoritmus polinomiális időben megtalálja a maximális folyamot. (Bizonyítás végett lásd a [2], 515–518. oldalt.)

A hálózati gráfot eltároljuk szomszédsági mátrix által is (mint irányított gráfot) és szomszédsági lista segítségével is (mint irányítatlan gráfot). A szomszédsági mátrixunknak (`Sz_M[1..n][1..n]`) elemei bejegyzés típusúak. Ha létezik az (u, v) él, akkor az `Sz_M[u][v].c` mező az illető él kapacitását, az `Sz_M[u][v].f` mező pedig a folyam értékét tárolja. Ha nem létezik az u pontból a v pontba él, akkor az `Sz_M` mátrix megfelelő elemének kapacitásmezője -1 értékű. A szomszédsági listánk (`Sz_L[1..n]`) elemei is bejegyzés típusúak. Az `Sz_L[u].fokszám` mező az u pont ki- és be-szomszédainak az összszámát (ki-fokszám + be-fokszám) tartalmazza, az `Sz_L[u].szomszédok[]` tömbmező pedig az u pont szomszédait (ki- és be-szomszédait) tárolja.

A legrövidebb (élszámban) javítóutat szélességi bejárással keressük meg (`KERES_JAVÍTÓ_ÚT` eljárás). A szélességi bejárás az irányítatlan szomszédsági listát használja, hogy az éleken mindkét irányban végig tudjon menni. Továbbá, egy olyan szélességi bejárásra van szükségünk, amelyik csak telítetlen éleken halad át. Ez biztosítja, hogy valóban *javító*utat találjon. Ennek érdekében használja a `KERES_JAVÍTÓ_ÚT` eljárás a hálózati gráf `Sz_M` szomszédsági mátrixát is. Az eljárás az `Sz_M` mátrixból „nézi ki", hogy az aktuális u pontnak a v pont be- vagy ki-szomszédja. Ha v

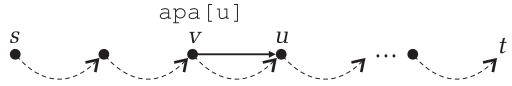
ki-szomszéd (`Sz_M[u][v].c` $\neq -1$), akkor létezik az (u, v) él, amely előremutatóként kerül a keresett javítóútra. Ez esetben a telítetlenség feltétele: `Sz_M[u][v].f < Sz_M[u][v].c`. Másfelől, ha v be-szomszédja u-nak (`Sz_M[v][u].c` $\neq -1$), akkor arról van szó, hogy a (v, u) élen fordított irányban szeretnénk áthaladni (a (v, u) él visszamutató éle lesz a keresett javítóútnak). Ilyen helyzetben a telítetlenségi feltétel: `Sz_M[v][u].f` > 0. Ha az u és v pontok között léteznek mind az (u, v), mind a (v, u) élek, akkor az eljárás mindkettőt figyelembe veszi. Előbb az (u, v) élet ellenőrzi (mint előremutatót), és ha ez már telített, akkor próbálkozik a (v, u) éllel (mint visszamutatóval). Figyeljünk fel arra is, hogy ha egy él telített előremutatóként, akkor telítetlen visszamutatóként (és fordítva). (Kivételt képeznek a nulla kapacitású élek.)

A `KERES_JAVÍTÓ_ÚT` eljárás a szélességi fát az `apa[1..n]` tömbben kódolja. Ha az eljárás talál $s \to t$ javítóutat, akkor fordított irányban (t-től s fele) ez a következő lesz:

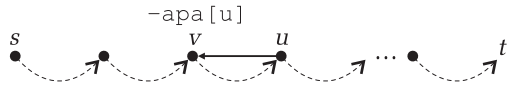


9.5. ábra.

Az `apa[u]` tömbelem attól függően pozitív vagy negatív előjelű, hogy a megfelelő `apa[u]` pont ki- vagy be-szomszédja az u pontnak. Ha v „pozitív apja" u-nak (`v=apa[u]`), akkor a (v, u) él szerepel a javítóúton (mint előremutató él) (9.6. ábra). Ellenben, ha a v pont „negatív apja" az u pontnak (`v=-apa[u]`), akkor az (u, v) visszamutató él áll a javítóút v és u pontjai között (9.7. ábra).



9.6. ábra. *Az `(apa[u],u)` él előremutató éle az $s \to t$ javítóútnak*



9.7. ábra. *Az `(u,apa[u])` él visszamutató éle az $s \to t$ javítóútnak*

Ha a `KERES_JAVÍTÓ_ÚT` eljárás eljutott a t pontba, akkor meghívja a `JAVÍTÁS` eljárást. Ebben a pillanatban az `apa`-tömbből kiolvasható a megtalált javítóút pontjainak fordított sorrendje. A `JAVÍTÁS` rekurzív eljárás

a t pontból indul, és apáról apára haladva meghatározza, hogy melyik az a legnagyobb érték, amellyel a szóban forgó javítóút mentén növelhető a folyamérték. Minden szomszédos pontpár kapcsán (u az aktuális pont, v pedig az `apa` pont)

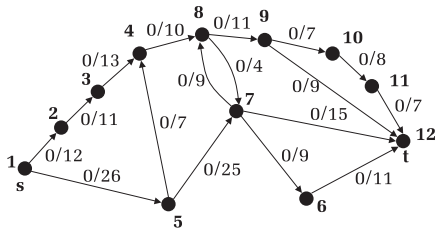
- ellenőrzi, hogy előremutató (`apa[u] > 0`) vagy visszamutató (`apa[u] < 0`) él került-e közéjük (9.6–9.7. ábrák),
- kiszámítja, hogy mennyivel növelhető (`Sz_M[v][u].c - Sz_M[v][u].f`), illetve csökkenthető (`Sz_M[u][v].f`) az illető élen a folyamérték.

Ezen értékek közül a legkisebb a keresett javítóérték. Ehhez a minimum kereséséhez az eljárás a `min` nevű, *cím szerint* átadott paramétert használja. A `JAVÍTÁS` eljárás a „rekurzió visszaútján" el is végzi a korrekciókat:

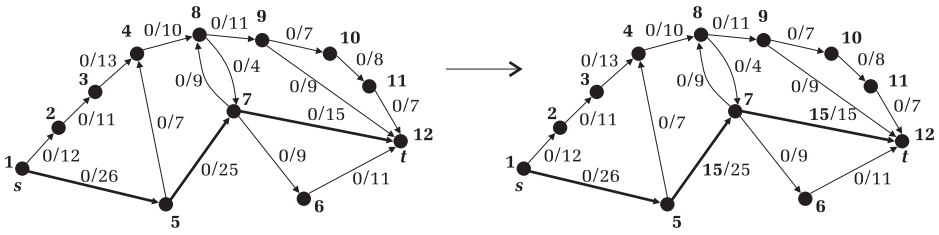
- Minden előremutató élen növeli a folyamértéket `min`-nel.
- Minden visszamutató élen csökkenti a folyamértéket `min`-nel.

Ha a `KERES_JAVÍTÓ_ÚT` eljárás nem éri el a t pontot, akkor a `min` paraméterben nullát térít vissza a `FORD_FULKERSON` eljárásnak. Ez utóbbi ebből ismeri fel, hogy nincs több javítóút a hálózati gráfban, és ily módon az fe összegváltozó a maximális folyam értékét tartalmazza.

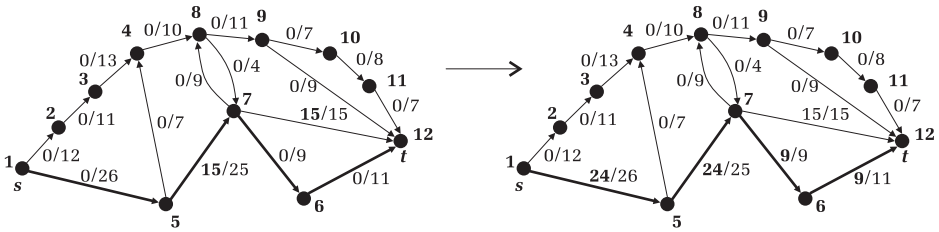
A 9.8–9.15. ábrák a Ford–Fulkerson-algoritmust szemléltetik.



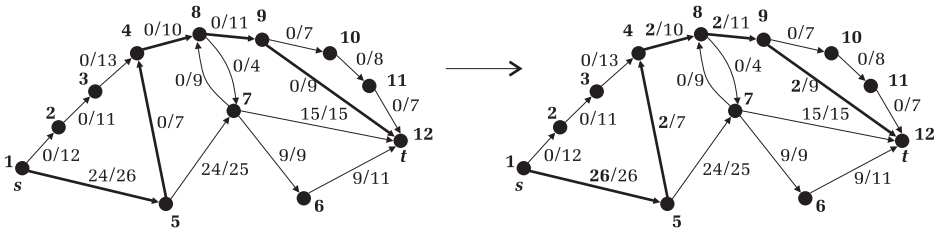
9.8. ábra. *Példagráf a Ford–Fulkerson-algoritmus szemléltetéséhez. Az élek mellett a folyam/kapacitás értékpárost láthatjuk*



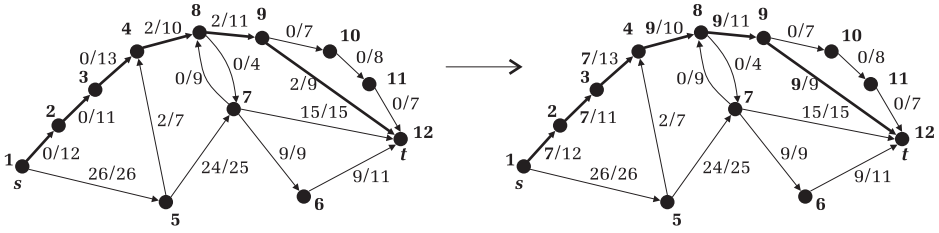
9.9. ábra. *Az (1, 5, 7, 12) javítóút mentén növeljük a folyamértékeket 15-tel*



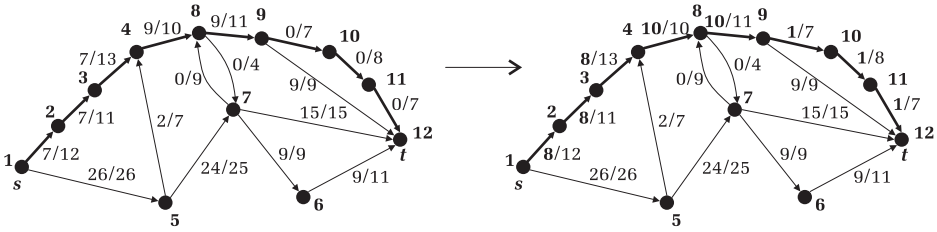
9.10. ábra. *Az (1, 5, 7, 6, 12) javítóút mentén növeljük a folyamértékeket 9-cel*



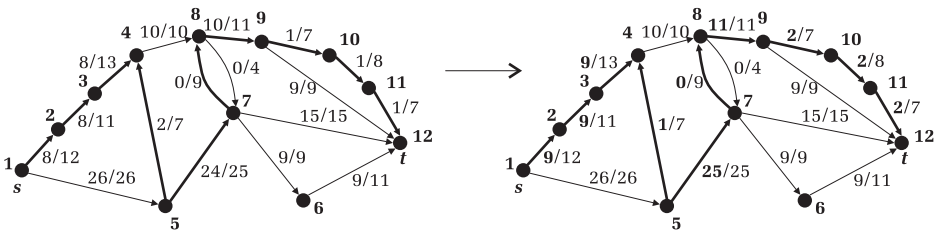
9.11. ábra. *Az (1, 5, 4, 8, 9, 12) javítóút mentén növeljük a folyamértékeket 2-vel*



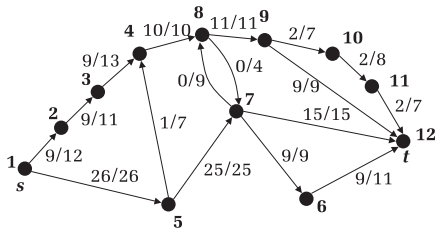
9.12. ábra. *Az (1, 2, 3, 4, 8, 9, 12) javítóút mentén növeljük a folyamértékeket 7-tel*



9.13. ábra. *Az (1, 2, 3, 4, 8, 9, 10, 11, 12) javítóút mentén növeljük a folyamértékeket 1-gyel*



9.14. ábra. *Az (1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12) javítóút mentén növeljük a folyamértékeket 1-gyel. Az (5, 4) él visszamutató élként szerepel az úton*



9.15. ábra. *A példagráf állapota a Ford–Fulkerson-algoritmus lefutása után. A maximális folyamértéke 35*

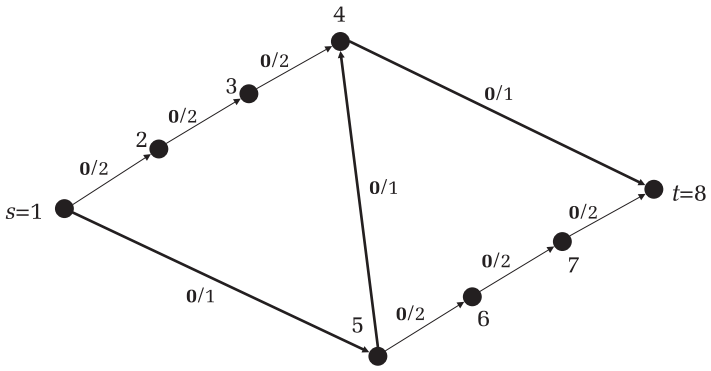
```
eljárás JAVÍTÁS(Sz_M[1..n][1..n],u,apa[1..n],min)
   ha apa[u] < 0 akkor
      v ← -apa[u]
      ha min > Sz_M[u][v].f akkor
         min ← Sz_M[u][v]].f
      vége ha
      JAVÍTÁS(Sz_M,v,apa,min)
      Sz_M[u][v].f ← Sz_M[u][v].f - min
   különben
      ha apa[u] > 0 akkor
         v ← apa[u]
         ha min > - Sz_M[v][u].f akkor
            min ← Sz_M[v][u].c - Sz_M[v][u].f
         vége ha
         JAVÍTÁS(Sz_M,v,apa,min)
         Sz_M[v][u].f ← Sz_M[v][u].f + min
      vége ha
   vége ha
vége JAVÍTÁS

eljárás KERES_JAVÍTÓ_ÚT
                    (Sz_M[1..n][1..n],Sz_L[1..n],s,t,min)
   minden u ∈ V(G)-{s} végezd
      szín[u] ← FEHÉR
      apa[u] ← 0
   vége minden
   szín[s] ← SZÜRKE
   apa[s] ← 0
   Q ← {s}
   amíg Q ≠ ∅ végezd
      u ← MÁSOL_SORELSŐ(Q)
      minden i ← 1, Sz_L[u].fokszám végezd
         v ← Sz_L[u].[i]
         ha szín[v] = FEHÉR akkor
            ha Sz_M[u][v].c ≠ -1 ÉS
                        Sz_M[u][v].f < Sz_M[u][v].c akkor
               szín[v] ← SZÜRKE
               apa[v] ← u
               ha v = t akkor
                  JAVÍTÁS(Sz_M,t,apa,min)
                  vissza
               vége ha
               BETESZ_SORVÉGÉRE(Q,v)
```

```
                különben
                   ha Sz_M[v][u].c ≠ -1 ÉS
                                        Sz_M[v][u].f > 0 akkor
                      szín[v] ← SZÜRKE
                      apa[v] ← -u
                      ha v = t akkor
                         JAVÍTÁS(Sz_M,t,apa,min)
                         vissza
                      vége ha
                      BETESZ_SORVÉGÉRE(Q,v)
                   vége ha
                vége ha
             vége ha
          vége minden
          TÖRÖL_SORELEJÉRŐL(Q)
          szín[u] ← FEKETE
       vége amíg
       min ← 0
    vége KERES_JAVÍTÓ_ÚT

    függvény FORD_FULKERSON(Sz_M[1..n][1..n],Sz_L[1..n],s,t)
       fe ← 0
       végezd
          min ← ∞
          KERES_JAVÍTÓ_ÚT(Sz_M,Sz_L,s,t,min)
          fe ← fe + min
       amíg min ≠ 0
       vissza fe
    vége FORD_FULKERSON
```

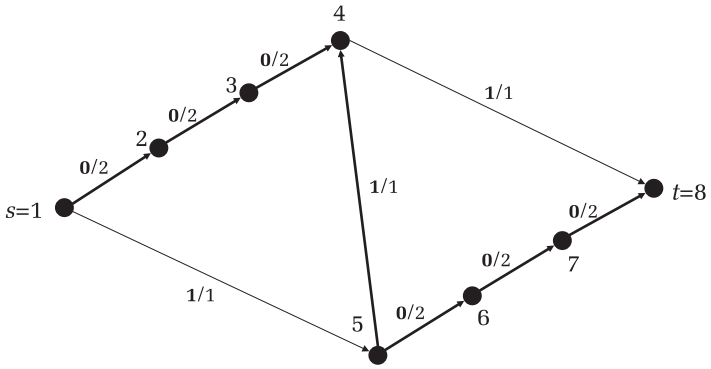
Bonyolultság: A Ford–Fulkerson-algoritmus bonyolultsága (amennyiben az Edmonds–Karp-tétellel összhangban szélességi bejárással mindig megkeressük az élszámban legrövidebb javítóutat) $O(nm^2)$. (Bizonyítás végett lásd a [2], 515. oldalt.)

Megjegyzések:

- Miért nem elég csak az irányított javítóutakat (amelyeken minden él előremutató) megvizsgálni a maximális folyam meghatározásához? A választ a 9.16–9.17. ábrák illusztrálják.
- A legrövidebb javítóút (élszámban) az $(1 = s, 5, 4, 8 = t)$ út lesz, amelyen 1-gyel javítható a folyamérték. E javítást követően nincs több irányított javítóút, pedig még javítható lenne a hálózaton



9.16. ábra. *Növeljük a folyamértéket a megvastagított javítóút mentén. (A legrövidebb javítóút három előremutató élet tartalmaz.)*



9.17. ábra. *Növeljük a folyamértéket a megvastagított javítóút mentén. (Ez az egyetlen javítóút és tartalmaz visszamutató élet.)*

átfolyó folyamérték 1-gyel. E javítás az $(1 = s, 2, 3, 4, 5, 6, 7, 8 = t)$ úton valósítható meg, amelyen az (5, 4) visszamutató él.

– A Ford–Fulkerson-algoritmus elméleti alapjául szolgáló 9.1. tétel az alábbi változatban vált híressé:

9.3. tétel. A $(G(V, E), s, t, c)$ hálózatban a maximális folyamérték egyenlő a minimális kapacitású vágás kapacitásával.

A vágás fogalmán itt a következőt értjük: Legyen (X, Y) a hálózati gráf pontjainak egy olyan partíciója ($X \cup Y = V$, $X \cap Y = \emptyset$), amelyre $s \in X$ és $t \in Y$. Az (X, Y) vágást a gráf azon élei alkotják, amelyeknek egyik végpontja X-ben, a másik pedig Y-ban van. Az (X, Y) vágás kapacitása a definíció szerint egyenlő az X-től Y fele mutató élek (előremutató élek) kapacitásainak összegével. A visszamutató éleket nem vesszük figyelembe a vágás kapacitásának értelmezésében.

A tétel belátása: Egy vágáson természetesen akkor folyik át (előre irányban) a legnagyobb mennyiségű folyam, ha a vágás minden előreéle telített, és a vissza-éleken nem folyik vissza semmi (azaz, az illető vágásra nézve ezek is telítettek). Ez az érték éppen a vágás kapacitásával egyenlő. Mivel a hálózaton átfolyó folyamnak a gráf minden vágásán át kell haladnia, ezért a maximális folyamérték a „minimális vágás" kapacitásával lesz egyenlő (további részletek végett lásd a [6], 66. oldalt).

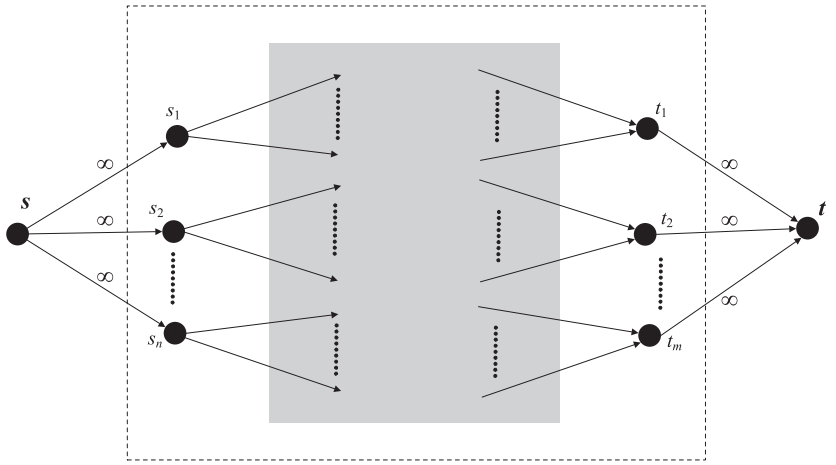
9.4. tétel. Ha egy hálózati gráf kapacitásfüggvénye az $\mathbb{N}^*$-ból vesz értékeket, akkor létezik olyan maximális folyam, amelyikhez a gráf minden élén természetes szám értékű ($\in \mathbb{N}$) folyam tartozik.

A tétel belátása: Az állítás nyilvánvalóan adódik abból, hogy kezdeti állapotban minden élen a folyamérték nulla, és az algoritmus során a folyamértékek minden élen csak egész számmal változhatnak.

Az előbbi tétel direkt következménye: ha a gráf minden élének kapacitása 1, akkor létezik olyan maximális folyam, amelyik minden élen vagy 0-t vagy 1-et vesz fel.

9.1. A folyam-probléma általánosításai

A következőkben három olyan helyzetet vizsgálunk meg, amikor a feladat könnyen visszavezethető a fentiekben tárgyalt hagyományos folyam-problémára.



9.18. ábra. *A „több forrás – több nyelő" probléma visszavezetése „egy forrás – egy nyelő" problémára*

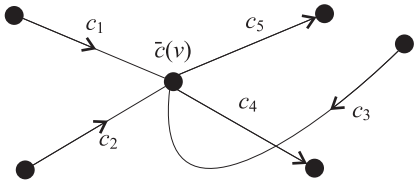
Több forrás – több nyelő

Mi a helyzet egy olyan hálózati gráf esetén, amelyik több forráspontot és több nyelőpontot tartalmaz? Egy ilyen feladat könnyen visszavezethető a klasszikus „egy forrás – egy nyelő" problémára. A megoldás abban áll, hogy felveszünk egy virtuális forrást, amelyet végtelen kapacitású ki-élekkel hozzákötünk minden valódi forráshoz. Hasonlóképpen járunk el a nyelők esetében is. Minden valódi nyelőt összekötünk, végtelen kapacitású ki-élekken keresztül, egy virtuális nyelőponttal. Az így nyert hálózati gráfban meghatározzuk a virtuális forrásból a virtuális nyelőbe átfolyó maximális folyamot. A kiterjesztett gráfra kapott ezen eredmény e gráf valódi forrásoktól valódi nyelőkig levő szakaszán éppen az eredeti feladat megoldását jelenti. A 9.6. ábra szemléletesen mutatja be mindezt.

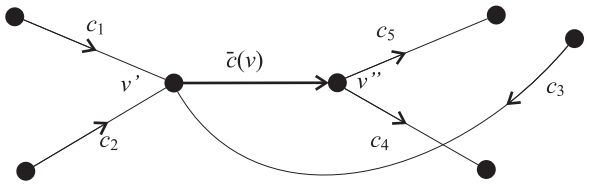
Amikor a csomópontoknak is van kapacitásuk

Ha a csomópontok is rendelkeznek kapacitásértékekkel, akkor megtehetjük, hogy minden csomópontot helyettesítünk a pont kapacitásával azonos kapacitású éllel. Ezzel a feladatot visszavezetjük a klasszikus feladatra, ahol a pontoknak nincs kapacitásértékük. A helyettesített pont be-élei az új él kezdőpontjához, a ki-élei pedig a helyettesítő él végpontjához fognak illeszkedni. Ezt illusztrálják a 9.19–9.20. ábrák. A v pontot

helyettesítettük a (v', v'') éllel. A helyettesítés után a v pont be-élei (a c_1, c_2 és c_3 kapacitású élek) a v' pont be-éleivé váltak. A v pont ki-élei (a c_4 és c_5 kapacitású élek) a (v', v'') helyettesítő él végpontjának (v'') ki-éleivé váltak.



9.19. ábra. *Részlet egy olyan hálózati gráfból, amelyben a csomópontok is rendelkeznek kapacitásértékekkel*



9.20. ábra. *A szóban forgó csomópontot helyettesítettük egy azonos kapacitású éllel*

Ha a hálózati gráf irányítatlan

Irányítatlan gráfok esetében egyszerűen helyettesítünk minden irányítatlan élet irányított oda-vissza-élekkel. Bár ez azt fogja eredményezni, hogy a forrásnak is lesznek be-élei és a nyelőnek is ki-élei, mindez nem fogja „megzavarni" az algoritmus működését. Ezek az élek csak visszamutató élként kerülhetnének bármely javítóútra. Ez viszont nem fog soha bekövetkezni, hiszen visszamutató élként alapállásból telítettek (a kezdeti folyamértékük nulla). A következő fejezet többszörös összefüggéssel foglalkozó része tartalmaz példát erre vonatkozóan.

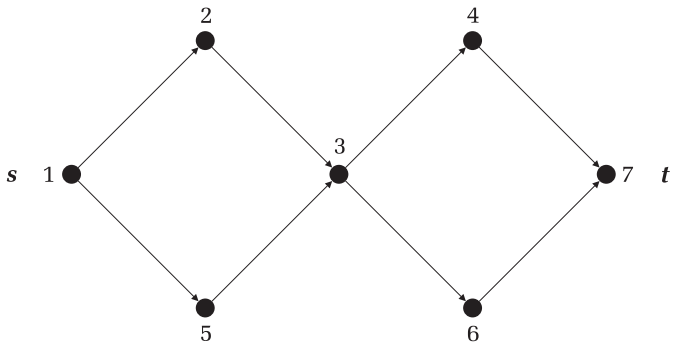
10. FEJEZET

TÖBBSZÖRÖS ÖSSZEFÜGGŐSÉG

Gyakorlati probléma: Egy gráf pontjai az USA vietnámi katonai bázisait ábrázolják, a gráf élei pedig a köztük lévő kommunikációs kapcsolatokat. Az amerikai hadügyminisztérum arról értesül, hogy az ellenség felszámolt k bázist. Nyugodtak maradhatnak-e a tábornokok abban a tekintetben, hogy a megmaradt támaszpontok továbbra is kapcsolatban vannak egymással? Ez a kérdés visszavezethető arra, hogy a gráf hányszorosan volt összefüggő.

Az előbbi fejezetben bebizonyítottuk, hogy ha egy hálózati gráf minden élének kapacitása 1, akkor létezik olyan maximális folyam, amelyik minden élen vagy 0-t vagy 1-et vesz fel. Ha elhagyjuk azokat az éleket, amelyeken a folyamérték nulla, akkor a megmaradt hálózat irányított s-ből t-be vezető *él-idegen* utakból fog állni (diszjunkt utak, nincs közös élük). Az él-idegen utak *lefogásá*hoz legalább annyi él kell, ahány út van. (Egy él akkor fog le egy utat, ha részét képezi annak.)

Továbbá, ha két s-ből t-be vezető útnak nincs s-től és t-től különböző közös pontja, akkor *pont-idegen* utakról beszélünk. A pont-idegen utak nyilván él-idegenek is, de ez fordítva nem igaz. A pont-idegen utak lefogásához legalább annyi s-től és t-től különböző pont kell, ahány út van. (Egy pont akkor fog le egy utat, ha részét képezi annak.)



10.1. ábra. *Él-idegen (de nem pont-idegen) $s \to t$ utak:* $(s = 1, 2, 3, 6, 7 = t)$; $(s = 1, 5, 3, 4, 7 = t)$

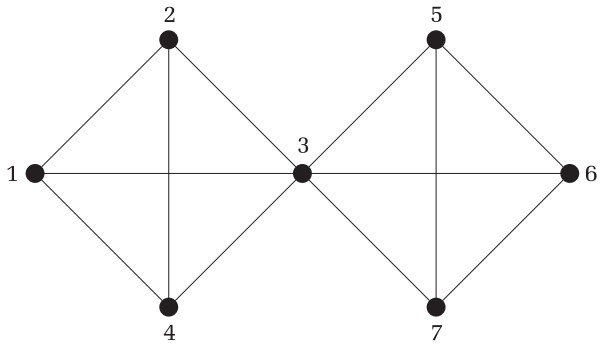
10.1. tétel (Menger). (Bizonyítás végett lásd a[6], 69–70. oldalt.)

1. Ha G irányított gráf és s, $t \in V(G)$, akkor az s-ből t-be vezető páronként él-idegen irányított utak maximális száma megegyezik az összes irányított $s \to t$ utat lefogó élek minimális számával.
2. Ha G irányított gráf és $s, t \in V(G)$ úgy, hogy s nem szomszédos t-vel, akkor az s-ből t-be vezető páronként pont-idegen irányított utak maximális száma megegyezik az összes irányított, $s \to t$ utat lefogó, s-től és t-től különböző pontok minimális számával.
3. Ha G irányítatlan gráf és $s, t \in V(G)$, akkor az s-ből t-be vezető páronként él-idegen irányítatlan utak maximális száma megegyezik az összes irányítatlan, $s \to t$ utat lefogó élek minimális számával.
4. Ha G irányítatlan gráf és $s, t \in V(G)$ úgy, hogy s nem szomszédos t-vel, akkor az s-ből t-be vezető páronként pont-idegen irányítatlan utak maximális száma megegyezik az összes irányítatlan, $s \to t$ utat lefogó, s-től és t-től különböző pontok minimális számával.

10.2. definíció. Egy G gráfot *k-szorosan pont-összefüggő*nek (vagy egyszerűen *k-szorosan összefüggő*nek) nevezünk, ha legalább $(k+1)$ pontja van, és akárhogy hagyunk el belőle k-nál kevesebb pontot, a maradék gráf összefüggő marad.

10.3. definíció. Egy G gráfot *k-szorosan él-összefüggő*nek nevezünk, ha akárhogy hagyunk el belőle k-nál kevesebb élet, a maradék gráf összefüggő marad.

A k-szoros pont-összefüggőség ($k \geqslant 2$) „erősebb", mint a k-szoros él-összefüggőség (10.2. ábra).



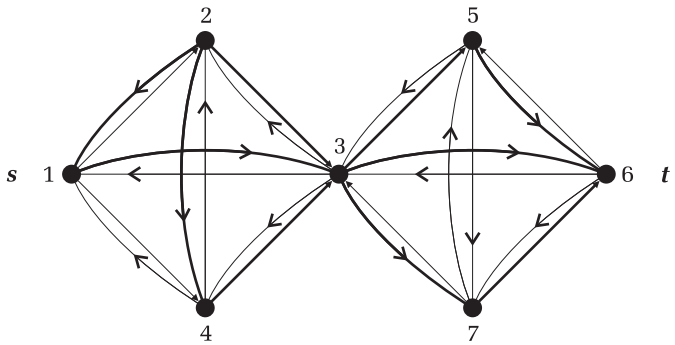
10.2. ábra. *1-szeresen pont-összefüggő, 3-szorosan él-összefüggő gráf*

10.4. tétel. Egy gráf akkor és csakis akkor k-szorosan pont-összefüggő, ha legalább $(k+1)$ pontja van, és bármely két pontja között létezik k pont-idegen út. Egy gráf akkor és csakis akkor k-szorosan él-összefüggő, ha bármely két pontja között létezik k él-idegen út. (Bizonyítás végett lásd a [6], 71. oldalt.)

10.5. tétel (Menger). A legalább 3 pontú gráf akkor és csakis akkor 2-szeresen pont-összefüggő, ha két tetszőleges pontján át vezet kör. Továbbá, bármely legalább 3 pontú gráf akkor és csakis akkor 2-szeres pont-összefüggő, ha bármely két élén át vezet kör. (Bizonyítás végett lásd a [6], 71. oldalt.)

10.6. tétel (Dirac). Ha $k \geqslant 2$ és a G gráf k-szorosan pont-összefüggő, akkor bármely $x_1, x_2, \ldots, x_k$ pontján át vezet kör. (Bizonyítás végett lásd a [6], 72. oldalt.)

Algoritmus: Származtassunk a G gráfból egy olyan G' hálózati gráfot, amelynek minden élén a kapacitásérték 1. (Mivel a hálózati gráfok irányítottak, minden irányítatlan élet helyettesítenünk kell egy irányított oda-vissza élpárral; 10.3. ábra.) Ha lefuttatjuk a Ford–Fulkerson-algoritmust az így kapott G' gráfhoz tartozó (i, j) pontpárra (i forrás, j nyelő), akkor a maximális folyamérték megadja az eredeti G gráfban az (i, j) pontpár közötti páronként különböző él-idegen utak számát. A



10.3. ábra. *A 10.2. gráf (G) hálózati gráffá való konvertálása (G'). G 3-szorosan él-összefüggő; G' minden pontpárja között a maximális folyam 3. Kiemeltünk az (1, 6) pontpár között egy maximális számú (3) él-idegen úthalmazt*

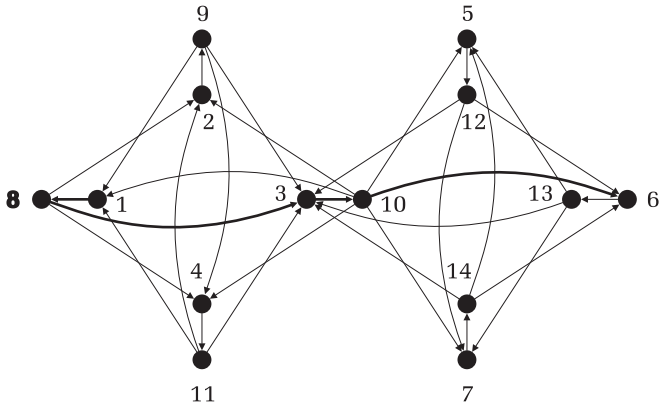
K_ÉLÖSSZEFÜGGŐ eljárás meghívja a Ford–Fulkerson-algoritmust a G' hálózati gráfhoz tartozó minden (i, j) pontpárra ($i < j$). A paraméterként kapott szomszédsági mátrix és szomszédsági lista G'-re vonatkoznak. A fent bemutatott tétel értelmében a kapott maximális folyamértékek minimuma a gráf többszörös él-összefüggőségének a fokát (a k értékét) fogja jelenteni.

```
eljárás K_ÉLÖSSZEFÜGGŐ(Sz_M[1..n][1..n],Sz_L[1..n],n)
   k ← ∞
   minden i ← 1,n-1 végezd
      minden j ← j+1,n végezd
         fe ← FORD_FULKERSON(Sz_M,Sz_L,i,j)
         ha k > fe akkor
            k ← fe
         vége ha
      vége minden
   vége minden
   kiír: k
vége K_ÉLÖSSZEFÜGGŐ
```

Rendeljünk G' minden pontjához kapacitásértékeket is. Az s forrás és a t nyelő kapacitása legyen végtelen, a többi ponté legyen 1. Az így kapott gráf bármely (i, j) pontpárja között a maximális folyamérték az illető pontok közötti pont-idegen utak számával lesz egyenlő (az $s = i$ forrás és a $t = j$ nyelő kapacitása legyen végtelen, a többi ponté pedig legyen 1). A már említett tétel értelmében e maximumok minimuma a gráf többszörös pont-összefüggőségének a fokát (a k értékét) fogja megadni.

Algoritmus: Mivel a Ford–Fulkerson-algoritmus kézenfekvő az él-összefüggőség meghatározásához, vezessük vissza a pont-összefüggőségi problémát él-összefüggőségi feladatra. Vegyünk fel a fentiekben kialakított G' hálózati gráf minden i pontja mellé egy $(n+i)$ azonosítójú pontot. Minden $(i, n+i)$ rendezett pontpárt kössön össze egy $(i, n+i)$ irányított él. Az i pont ki-éleinek kezdőpontjait helyezzük át az $(n + i)$ pontba. Az így kapott G'' kiterjesztett hálózati gráf $2 \cdot n$ pontú és $(2 \cdot m + n)$ élű lesz (10.4. ábra). Úgy is fogalmazhatnánk, hogy a G' gráf minden pontját helyettesítettük egy-egy irányított éllel. Ez eszünkbe juttathatja azt, ahogy az előző fejezetben a pontkapacitás okozta „kényelmetlenséget" feloldottuk (lásd a 9.1. alfejezetet). A Ford–Fulkerson-algoritmusnak a kiterjesztett hálózati gráf $(i = s, j = t)$ pontpárjára való lefuttatásakor ($1 \leqslant i < j \leqslant n$; szükségtelen a kiegészítő pontokat is bevenni) a forrás $(i, n + i)$ ki-élének kapacitása végtelen, az összes többi él kapacitása

pedig 1. Az eredeti G gráf annyiszorosan pont-összefüggő, ahányszorosan él-összefüggő a kiterjesztett gráf. Ezt a gondolatmenetet követi a K_PONTÖSSZEFÜGGŐ eljárás. A KITERJESZTGRAF eljárás G' szomszédsági mátrixából (Sz_M) és szomszédsági listájából (Sz_L) felépíti G'' szomszédsági mátrixát (k_Sz_M) és szomszédsági listáját (k_Sz_L).



10.4. ábra. *A 10.3. hálózati gráfból (G') származtatott kiterjesztett hálózati gráf (G''). G 1-szeresen pont-összefüggő; G'' minden pontpárja között a maximális folyam 1. Kiemeltünk az (1, 6) pontpár között egy maximális számú (egyetlen elemű) él-idegen úthalmazt*

```
eljárás K_PONTÖSSZEFÜGGŐ(Sz_M[1..n][1..n],Sz_L[1..n],n)
   k ← ∞
   KITERJESZTGRAF(k_Sz_M, k_Sz_L, Sz_M, Sz_L, n)
   k_Sz_M[1..2*n][1..2*n].c ←1   // minden él kapacitása 1
   minden i ← 1,n-1 végezd
      minden j ← j+1,n végezd
         k_Sz_M[1..2*n][1..2*n].f ← 0
                                   // minden élen a folyam 0
         k_Sz_M[i][i+n].c ← ∞
         fe ← FORD_FULKERSON(k_Sz_M,k_Sz_L,i,j)
         k_Sz_M[i][i+n].c ← 1
         ha k > fe akkor
            k ← fe
         vége ha
      vége minden
   vége minden
   kiír: k
vége K_PONTÖSSZEFÜGGŐ
```

Bonyolultság: A többszörös összefüggőséget vizsgáló algoritmusok bonyolultsága (mivel egymásba ágyazott `minden` ciklusok belsejében hívják meg a Ford–Fulkerson-algoritmust) $O(n^3m^2)$. (Bizonyítás végett lásd a [2], 515. oldalt.)

11. FEJEZET

PÁROSÍTÁSOK GRÁFOKBAN

Gyakorlati probléma: Be kell osztanunk n szakképesített munkást n különböző szerszámgépre. Mindegyik munkást felkészültsége, ügyessége és gyakorlata alapján a normához viszonyított százalékban kifejezett, különböző termelékenységi mutatóval oszthatunk be az egyes szerszámgépekhez. Határozzuk meg az n munkás leggazdaságosabb beosztását az egyes gépekhez. Ha a munkásokat és a gépeket pontok, a beosztási lehetőségeket pedig a megfelelő élek ábrázolják, akkor a feladat a gráfelmélet nyelvén a következő: határozzuk meg a gráf maximális súlyú n elemű párosítását.

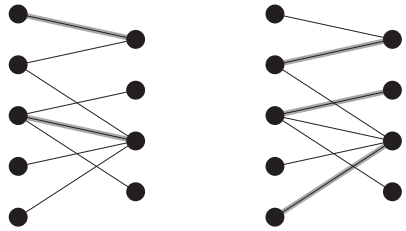
11.1. Párosítás páros gráfokban

11.1. definíció. Egy G irányítatlan gráfot *páros gráfnak* nevezünk, ha G pontjainak $V(G)$ halmaza két részre, egy A és egy B halmazra osztható úgy, hogy G minden élének egyik végpontja A-ban, másik végpontja B-ben van. Ennek jelölése: $G = (A, B)$. A $K_{a,b}$-vel jelölt *teljes páros gráf* olyan $G = (A, B)$ páros gráf, ahol $|A| = a$ és $|B| = b$, és amelyben minden A-beli pont össze van kötve minden B-beli ponttal. ($|X|$ az X halmaz elemeinek számát jelöli.)

11.2. tétel. Egy G gráf akkor és csakis akkor páros gráf, ha minden körének hossza páros szám. (Bizonyítás végett lásd a [6], 56. oldalt.)

11.3. definíció. *Párosítás*nak nevezünk egy M élhalmazt, ha semelyik két élnek nincs közös pontja (független élek). Azt mondjuk, hogy a párosítás lefedi éleinek végpontjait. Egy párosítást *teljes párosítás*nak nevezünk, ha a gráf minden pontját lefedi. Különben *részleges párosítás*ról van szó. Beszélhetünk továbbá maximális élszámú párosításról is.

11.4. tétel (Hall). Egy $G = (A, B)$ páros gráfban akkor és csakis akkor van A-t lefedő párosítás, ha minden $X \subseteq A$ részhalmazra $|N(X)| \geqslant$



11.1. ábra. *Egy példagráf egy-egy részleges párosítását láthatjuk. A második párosítás maximális élszámú. A gráf nem tartalmaz teljes párosítást*

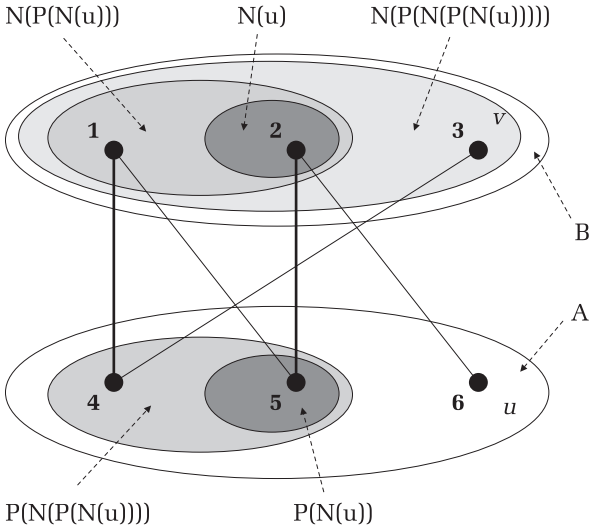
$|X|$. Ezt a feltételt Hall feltételének is nevezik. ($N(X)$-szel jelöltük az X halmaz pontjai szomszédainak halmazát; ha $X \subseteq A$, akkor $N(X) \subseteq B$.)

A tétel belátása: Ha van A-t lefedő párosítás, akkor nyilvánvaló, hogy tejesül a Hall-feltétel, mert ekkor maga a párosítás minden A-beli ponthoz hozzárendel egy B-beli pontot: tehát az A halmaz minden X részhalmaza esetén az $|N(X)|$ érték legalább $|X|$.

Tegyük fel, hogy teljesül a Hall-feltétel, és lássuk be, hogy létezik A-t lefedő párosítás G-ben. Fésüljük végig G éleit egy tetszőleges sorrendben, és válasszunk ki közülük annyi független élet, amennyit csak tudunk. Ha az így kapott M párosítás az A halmaz minden pontját lefedi, akkor bebizonyítottuk a tételt. Ha az A halmaznak maradt lefedetlen pontja, akkor legyen ezek egyike u. A Hall-tételből következik, hogy u-nak van legalább egy B-beli szomszédja. Ha ezek egyike (például v) lefedetlen pont, akkor az (u, v) éllel bővíthető az M párosítás.

Mi van akkor, ha az M párosítás u minden szomszédját lefedi? Be fogjuk bizonyítani, hogy a Hall-feltételből adódóan ez esetben is növelhető az M párosítás.

Keressünk egy olyan alternáló javítóutat, amelyik u-ból indul és valamelyik le nem fedett B-beli ponttal fejeződik be (jelöljük most ezt a pontot v-vel). Alternáló abban az értelemben, hogy a páratlan sorszámú élei (1., 3., ...) nem elemei az M párosításnak, a páros sorszámúak (2., ...) viszont igen. Egy ilyen út páratlan számú élből áll, legalább három éle van, és a kezdő- és végpontját kivéve minden pontja lefedett. Azért nevezzük javítóútnak is, mert ha páros sorszámú éleit kivesszük M-ből, és helyükbe betesszük a páratlan sorszámúakat, akkor az M párosítás elemszáma nő eggyel (páratlan hosszú úton eggyel több páratlan sorszámú él van, mint páros sorszámú). Ezt minden további nélkül megtehetjük, mert bármely út páratlan sorszámú élei független élek, és



11.2. ábra. *A Hall-tétel bizonyítása. Javítás előtt:* $M = \{(1,4),(2,5)\}$. *Alternáló javítóút:* $u = 6,2,5,1,4,3 = v$. *Maximális párosítás:* $M = \{(2,6),(1,5),(3,4)\}$

ugyanazokat a pontokat fedik le, mint a páros sorszámúak, plusz az eddig lefedetlen kezdő- és végpontokat. A következőkben belátjuk, hogy ameddig A-nak van lefedetlen u pontja, addig mindig található u-ból induló alternáló javítóút, amely révén növelhető M. Ez viszont nem jelent mást, mint hogy létezik A-t feledő párosítás.

Induljunk el u-ból párhuzamosan ennek minden szomszédja felé. Jelölje $N(u)$ az u pont szomszédainak halmazát. Úgy is mondhatnánk, hogy az „N-függvénnyel" átutazunk[1] A-ból B-be. Természetesen $N(u) \in B$, és minden eleme M által lefedett (azt az esetet, amikor létezik u-nak lefedetlen szomszédja, már letárgyaltuk). Jelölje továbbá $P(N(u))$ az $N(u)$ elemei párosításbeli párjainak halmazát. Nyilvánvalóan $P(N(u)) \subseteq A$, $|P(N(u))| = |N(u)|$ és $u \notin P(N(u))$. Olyan ez, mintha a „P-függvénnyel" visszajönnénk B-ből A-ba. Képezzük most az $Y = P(N(u)) \cup \{u\}$ halmazt. Vegyük észre, hogy $|Y| = |P(N(u))| + |\{u\}| = |N(u)| + 1$. Mivel a Hall-tétel értelmében $|N(Y)| \geqslant |Y|$, következik, hogy $N(Y)$-nak $N(u)$ valódi részhalmaza kell hogy legyen. Más szóval az $N(Y) \subseteq B$ halmaznak $N(u)$-hoz képest plusz pontokat kell tartalmaznia. Mivel e plusz

1 Mivel egy $f : X \to Y$ függvény minden X-beli pontnak megfeleltet egy Y-beli pontot, úgy is fogalmazhatunk, hogy f révén átjutunk X-ből Y-ba (vagy hogy f átvisz X-ből Y-ba).

pontok nem szomszédai u-nak (nem elemei $N(u)$-nak), ezért a $P(N(u))$ halmaz elemeinek a szomszédai kell hogy legyenek. Ez viszont azt jelenti, hogy amikor újra átkelünk az „N függvény" révén A-ból B-be, akkor $|N(P(N(u)))| > |P(N(u))|$. Ha a szóban forgó plusz pontok valamelyike lefedetlen, akkor legyen ez a v pont, és megvan a keresett alternáló javítóút. Ha az $N(P(N(u)))$ halmaz minden pontja lefedett, akkor folytatjuk az eljárást: újra „visszajövünk P-vel" B-ből A-ba, majd ismét „viszamegyünk N-nel" A-ból B-be, és így tovább. Az eljárás finalitását, a fentiekkel összhangban, az alábbi összefüggés biztosítja (a kulcsrelációk a szigorú egyenlőtlenségek: „$<$"):

$$1 = |\{u\}| \leqslant |N(u)| = |P(N(u))| < |N(P(N(u)))| = \\ = |P(N(P(N(u))))| < |N(P(N(P(N(u)))))|\ldots$$

Mivel a gráf véges, és így benne az M párosítás is, ezért előbb-utóbb abba a helyzetbe kell jutnunk, hogy valamelyik, N függvény általi, A-ból B-be való átkeléskor megjelenő „plusz pontok" egyike lefedetlen lesz. Ez a pont fog a keresett alternáló javítóút végpontjául szolgálni.

A felvázolt „bizonyítási algoritmust" a következőképpen lehetne összefoglalni:

- Indulunk u-ból.
- Minden páratlan lépésben az „N-függvény átvisz" A-ból B-be. Az első lépésben az „N-átvitelt" a „$\leqslant$", a harmadik lépéstől viszont a „$<$" reláció jellemzi.
- Minden páros lépésben a „P-függvény visszahoz" B-ből A-ba. Minden alkalommal a „P-átvitelt" a „=" reláció jellemzi.
- Megállunk, ha lefedetlen B-beli ponthoz jutunk.

11.5. tétel (Frobenius). Egy $G = (A, B)$ páros gráfban akkor és csakis akkor van teljes párosítás, ha $|A| = |B|$ és $|N(X)| \geqslant |X|$ minden $X \subseteq A$-ra. (Bizonyítás végett lásd a [6], 59. oldalt.)

A Hall-tétel bizonyítása alapján, alternáló javítóutak keresésével hatékony algoritmust kapunk a maximális élszámú párosítás megtalálására. Ez az algoritmus *magyar módszer* néven vált ismertté. (Az elnevezés Harold W. Kuhntól származik, aki Kőnig és Egervári magyar matematikusok eredményeire támaszkodva először adott polinomiális algoritmust maximális súlyú teljes párosítás meghatározására páros gráfban.)

11.1.1. Maximális élszámú párosítás

Magyar módszer

Az algoritmus egyszerűsítése érdekébe a gráfot háromféleképpen tároljuk el: éllista segítségével (`élek[1..m]`), szomszédsági listával (`Sz_L[1..n]`) és szomszédsági mátrixszal (`Sz_M[1..n][1..n]`). Az élek tömbben minden élről eltároljuk a kezdőpontját (`u` mező), a végpontját (`v` mező), illetve azt, hogy része-e a maximális párosításnak vagy nem (`p` mező). Ha létezik az `(u,v)` él, akkor a szomszédsági mátrix `Sz_M[u][v]` és `Sz_M[v][u]` elemei az illető él élek tömbbeli pozíciójának indexét (az él egyfajta címkéjét) tartalmazza. Az `Sz_L` szomszédsági lista elemei is bejegyzés típusúak. Az `Sz_L[u].fokszám` mező az u pont szomszédainak a számát tartalmazza, az `Sz_L[u].szomszédok[]` tömbmező pedig az u pont szomszédait tárolja.

További adatszerkezetek a szélességi bejáráshoz szükséges `szín[1..n]` tömb, illetve a szélességi fát kódoló `apa[1..n]` tömb. Az algoritmus használja a `fest[1..n]` tömböt is a gráf pontjainak két színnel való kiszínezésének kódolásához.

Algoritmus: Mindenekelőtt a gráfot kiszínezzük két színnel (az 1-es és 2-es színekkel). A `SZÍNEZ_2_színnel` függvény a szélességi bejárás közvetlen alkalmazása: ha a Q sor első eleme (u pont) 1-es színű, akkor u-nak a sor végére bekerülő fehér szomszédai (a v pontok) a 2-es színt kapják meg (és fordítva). E színezési eljárás úgy tekinthető, mint a G páros gráf A és B halmazainak a meghatározása. A `SZÍNEZ_2_színnel` függvény kiszámolja az A és B halmazok számosságát (az `f1` és `f2` változókban) és visszatéríti a kevesebb elemszámú halmaz színét (illetve a fest cím szerint átadott tömbben a festés kódját). Ezt az értéket a központi `MAXIMÁLIS_PÁROSÍTÁS` eljárás a `min_festék` változóban tárolja el. A `max_festék` változó a másik színt tartalmazza. Legyen A a `min_festék` színű halmaz és B a `max_festék` színű.

A színezést követően a `MAXIMÁLIS_PÁROSÍTÁS` eljárás végigpásztázza az éllistát és kiválaszt annyi független élet, amennyit csak tud. Az így kapott M párosítás még nem föltétlenül maximális. Minden kiválasztott él esetén az élek tömb megfelelő elemének p mezőjét 1-re állítja, illetve az él végpontjainak színét a fest tömbben negatívra változtatja (így jelzi, hogy az illető él részévé vált az M párosításnak, és hogy a végpontjai az M lefedte ponthalmazhoz tartoznak).

Ezután a központi eljárás a *min_festék* színű halmazban (A) még le nem fedett pontokat keres, és minden ilyen i pontra meghívja a KERES_JAVÍTÓ_ÚT függvényt. Ha a KERES_JAVÍTÓ_ÚT függvény talál i-ből induló javítóutat, akkor visszatéríti annak végpontját (ezt a j változó tárolja). E j pont biztosan max_festék színű lesz, azaz egy B halmazbeli le nem fedett pont. A FORDÍT_ALTERNÁLÁS rekurzív eljárás végigmegy a javítóúton (j-től i fele, apáról apára haladva), és minden élnek megváltoztatja a státusát: ha eleme volt a párosításnak, akkor kiveszi belőle, ha pedig nem volt eleme, akkor beleteszi. Ez a művelet eggyel növeli M elemeinek számát. Az i és j pontok lefedését a központi eljárás a színeik invertálásával oldja meg. A MAXIMÁLIS_PÁROSÍTÁS eljárás végül kiírja a maximális párosítás éleit.

A KERES_JAVÍTÓ_ÚT függvény is a szélességi bejárás alkalmazásának tekinthető. Indul az A-beli i pontból, meglátogatja i összes B-beli $b_1, b_2, \ldots, b_d$ szomszédját, majd ezeknek kizárólag a párosításbeli párjait (az A halmaz $a_1, a_2, \ldots, a_d$ pontjait), és így tovább. Tehát egy olyan módosított szélességi bejárásról van szó, amelyik az A halmaztól a B felé bármilyen élen haladhat (+/-max_festék), de B-től A felé csak párosításbelieken (-min_festék). Ha a függvénynek sikerül elérni egy még le nem fedett B-beli pontot (max_festék színűt), akkor ez azt jelenti, hogy talált egy alternáló javítóutat, és az illető pontot téríti vissza, különben nullát térít vissza.

```
függvény SZÍNEZ_2_színnel(Sz_L[1..n],s,fest[1..n])
   minden i ← 1, n végezd
      szín[i] ← FEHÉR
   vége minden
   szín[s] ← SZÜRKE
   fest[s] ← 1
   f1 ← 1
   f2 ← 0
   Q ← {s}
   amíg Q ≠ ∅ végezd
      u ← MÁSOL_SORELSŐ(Q)
      minden i ← 1, Sz_L[u].fokszám végezd
         v ← Sz_L[u].[i]
         ha szín[v] = FEHÉR akkor
            szín[v] ← SZÜRKE
            ha fest[u] = 1 akkor
               fest[v] ← 2
               f2 ← f2 +1
```

```
            különben
               fest[v] ← 1
               f1 ← f1 +1
            vége ha
            BETESZ_SORVÉGÉRE(Q,v)
         vége ha
      vége minden
      TÖRÖL_SORELEJÉRŐL(Q)
      szín[u] ← FEKETE
   vége amíg
   ha f1 < f2 akkor
      vissza 1
   különben
      vissza 2
   vége ha
vége SZÍNEZ_2_színnel

függvény KERES_JAVÍTÓ_ÚT
            (Sz_L[1..n],s,fest[1..n],max_festék,apa[1..n])
   minden i ← 1, n végezd
      szín[i] ← FEHÉR
      apa[i] ← 0
   vége minden
   szín[s] ← SZÜRKE
   apa[s] ← 0
   Q ← {s}
   amíg Q ≠ ∅ végezd
      u ← MÁSOL_SORELSŐ(Q)
      minden i ← 1, Sz_L[u].fokszám végezd
         v ← Sz_L[u].[i]
         ha szín[v] = FEHÉR ÉS
         fest[v]∈{max_festék,-max_festék,-min_festék} akkor
            szín[v] ← SZÜRKE
            apa[v] ← u
            BETESZ_SORVÉGÉRE(Q,v)
            ha fest[v] = max_festék akkor
               vissza v
            vége ha
         vége ha
      vége minden
      TÖRÖL_SORELEJÉRŐL(Q)
      szín[u] ← FEKETE
   vége amíg
```

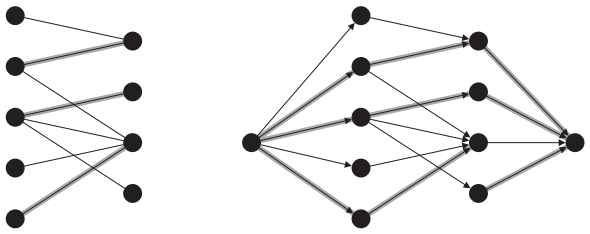
```
   vissza 0
vége KERES_JAVÍTÓ_ÚT

eljárás FORDÍT_ALTERNÁLÁS
                (i,j,élek[1..m],Sz_M[1..n][1..n],apa[1..n])
   akt_él ← Sz_M[j][apa[j]]
   ha élek[akt_él].p = 1 akkor
      élek[akt_él].p ← 0
   különben
      élek[akt_él].p ← 1
   vége ha
   ha apa[j]≠ i akkor
      FORDÍT_ALTERNÁLÁS(i,apa[j],élek,Sz_M,apa)
   vége ha
vége FORDÍT_ALTERNÁLÁS

eljárás MAXIMÁLIS_PÁROSÍTÁS
                    (Sz_L[1..n],Sz_M[1..n][1..n],élek[1..m])
   min_festék ← SZÍNEZ_2_színnel(Sz_L,1,fest)
   ha min_festék = 1 akkor
      max_festék ← 2
   különben
      max_festék ← 1
   vége ha
   minden i ← 1, m végezd
      ha (fest[élek[i].u] > 0) ÉS (fest[élek[i].v] > 0)
      akkor
         fest[élek[i].u] ← -fest[élek[i].u]
         fest[élek[i].v] ← -fest[élek[i].v]
         élek[i].p ← 1
      vége ha
   vége minden
   minden i ← 1, n végezd
      ha fest[i] = min_festék akkor
         j ← SZÉLESSÉGI_keresés(Sz_L,i,fest,max_festék,apa)
         ha j = 0 akkor
            ugorj
         különben
            FORDÍT_ALTERNÁLÁS(i,j,élek,Sz_M,apa)
            fest[i] ← -fest[i]
            fest[j] ← -fest[j]
         vége ha
      vége ha
```

```
        vége minden
        minden i ← 1, m végezd
           ha élek[i].p = 1 akkor
              kiír: élek[i].u, élek[i].v
           vége ha
        vége minden
    vége MAXIMÁLIS_PÁROSÍTÁS
```

Bonyolultság: A magyar módszert alkalmazó algoritmus bonyolultsága (a kisebb halmaz minden le nem fedett pontjára meghív egy szélességi bejárás szerű eljárást) $O(n(n+m))$.

Maximális párosítás a Ford–Fulkerson-algoritmussal

A maximális párosítás megtalálásának problémája felfogható a maximális folyam feladat egy nyilvánvaló alkalmazásaként is. Adjunk irányítást a $G = (A, B)$ páros gráfnak. Legyenek az élek kezdőpontjai az A-beli pontok, a végpontokat pedig vegyük a B halmazból. Vegyünk fel egy virtuális forrást, amelyből induljon irányított él minden A halmazbeli ponthoz. Hasonlóképpen, legyen egy virtuális nyelő is, amelyhez minden B-beli ponttól érkezik egy-egy irányított él. Továbbá e hálózati gráf minden élén tekintsük a kapacitás értékét 1-nek. Ha meghatározzuk e hálózatban a maximális folyam értékét, ez egyenlő lesz a G gráf maximális párosításának élszámával. Másfelől a maximális folyamhoz tartozó él-idegen utak A és B halmazok közötti élei éppen egy maximális párosítást adnak meg. Ezt szemléltetik az alábbi ábrák.



11.3. ábra. *Példagráfunk és a neki megfelelő hálózati gráf. Bejelöltük a maximális párosítást, illetve az ennek megfelelő maximális folyamot*

11.2. Párosítás tetszőleges gráfokban

11.6. tétel (Tuttle). Egy G gráfban akkor és csakis akkor létezik teljes párosítás, ha akárhogy hagyunk el a gráfból néhány pontot, a maradék részgráfban a páratlan sok pontot tartalmazó komponensek száma az elhagyott pontok számánál kisebb vagy egyenlő. (A feltétel szükségességének bizonyítása végett lásd a [6], 61. oldalt.)

11.3. Kőnig és Gallai tételei

Bevezetjük az alábbi jelöléseket:
- $\nu(G)$: független élek maximális száma (nincs közös végpontjuk)
- $\tau(G)$: lefogó pontok minimális száma (lefogják a gráf összes élét)
- $\alpha(G)$: független pontok maximális száma (nem szomszédosak)
- $\rho(G)$: lefogó élek minimális száma (lefogják a gráf összes pontját)

11.7. tétel. Minden G gráfra $\nu(G) \leqslant \tau(G)$.

A tétel belátása: Nyilvánvaló, hogy egy maximális méretű független élhalmaz lefogásához *legalább annyi* pont kell, ahány élből áll az illető halmaz. Továbbá könnyű találni olyan példákat, amelyeknek esetében *ugyanannyi*, illetve *több* pont szükséges az összes él lefogásához, mint a független élek maximális száma. (Példák: $\nu(K_2) = \tau(K_2)$, $\nu(K_3) < \tau(K_3)$.)

11.8. tétel. Minden G gráfra $\alpha(G) \leqslant \rho(G)$.

A tétel belátása: Egyértelmű, hogy egy maximális méretű független ponthalmaz lefogásához *legalább annyi* él kell, ahány pontból áll az illető halmaz. Továbbá könnyű találni olyan példákat, amelyeknek esetében *ugyanannyi*, illetve *több* él szükséges az összes pont lefogásához, mint a független pontok maximális száma. (Példák: $\alpha(K_2) = \rho(K_2)$, $\alpha(K_3) < \rho(K_3)$.)

11.9. tétel (Gallai). Minden hurokmentes G gráfra $\alpha(G) + \tau(G) = |V(G)|$.

A tétel belátása: Be fogjuk bizonyítani, hogy $|V(G)| \geqslant \alpha(G) + \tau(G)$ is, és $|V(G)| \leqslant \alpha(G) + \tau(G)$ is, azaz $|V(G)| = \alpha(G) + \tau(G)$.

Először lássuk be, hogy bármely X független ponthalmazra igaz, hogy $V(G)-X$ lefogó ponthalmaz, és fordítva, bármely Y lefogó ponthalmazra igaz, hogy $V(G)-Y$ független ponthalmaz. Világos, hogy $V(G)-X$ lefog minden olyan élet, amely nem illeszkedik X-beli pontra. Mivel X független ponthalmaz, ezért – amennyiben nincs hurokél – a ráilleszkedő élek másik végpontja $(V(G)-X)$-ben van, tehát a $(V(G)-X)$-beli pontok ezeket is lefogják. Másfelől az is világos, hogy mivel Y lefogó ponthalmaz, ezért a $(V(G)-Y)$-beli pontok között nem lehetnek élek, ugyanis ha lennének, akkor ezeket Y nem fogná le. Tehát $(V(G)-Y)$ független ponthalmaz.

Legyen most X_{max} egy maximális méretű független ponthalmaz ($|X_{max}| = \alpha(G)$). A fentiekkel összhangban nyilván X_{max}-ról is elmondható, hogy a $V(G)-X_{max}$ halmaz lefogó. Ebből triviálisan következik, hogy a lefogó pontok minimális száma nem lehet nagyobb $|V(G)-X_{max}|$-nál. Azaz: $\tau(G) \leqslant |V(G)-X_{max}|$. Tehát

$$|V(G)| = |X_{max}| + |V(G)-X_{max}| \geqslant \alpha(G)+\tau(G).$$

Másfelől legyen Y_{min} a G gráf egy minimális méretű lefogó ponthalmaza ($|Y_{min}| = \tau(G)$). Ugyancsak a fentiekkel összhangban Y_{min}-re is igaz, hogy $V(G)-Y_{min}$ független ponthalmaz. Ebből újból triviális, hogy a független pontok maximális száma nem lehet kisebb, mint $|V(G)-Y_{min}|$. Azaz: $\alpha(G) \geqslant |V(G)-Y_{min}|$. Tehát

$$|V(G)| = |Y_{min}| + |V(G)-Y_{min}| \leqslant \tau(G)+\alpha(G).$$

11.10. tétel (Gallai). Minden olyan G gráf esetében, amely nem tartalmaz izolált pontot, $\nu(G)+\rho(G) = |V(G)|$.

A tétel belátása: Be fogjuk bizonyítani, hogy $\nu(G)+\rho(G) \leqslant |V(G)|$ is, és $\nu(G)+\rho(G) \geqslant |V(G)|$ is, azaz $\nu(G)+\rho(G) = |V(G)|$.

Legyen X_{max} most egy maximális méretű független *él*halmaz ($|X_{max}| = \nu(G)$). Ez azt jelenti, hogy X_{max}-nak $\nu(G)$ darab éle lefogja G-nek $2\nu(G)$ pontját. A fenmaradt $|V(G)|-2\nu(G)$ pont lefogásához pontosan annyi él kell, ahány pont van. Ez azért igaz, mert

1. ha bármely kettő között ezek közül lenne él (ami azt jelentené, hogy két pont lefogható egy éllel), akkor X_{max} nem lenne maximális, hiszen bővíthető lenne az illető éllel;
2. nincsenek izolált pontok.

Tehát az összes pont lefogható $\nu(G)+|V(G)|-2\nu(G) = |V(G)|-\nu(G)$ darab éllel. Ez viszont azt jelenti, hogy a lefogó élek minimális száma

nem lehet nagyobb ennél a számnál. Azaz $\rho(G) \leqslant |V(G)| - \nu(G)$, vagyis $\nu(G) + \rho(G) \leqslant |V(G)|$.

Másfelől megfigyelhető, hogy ha Y_{min} egy minimális lefogó *él*halmaz ($|Y_{min}| = \rho(G)$), akkor Y_{min}, mint részgráf, diszjunkt csillagok egyesítése. Ez azért igaz, mert

1. ha Y_{min}-ben lenne kör, akkor nem lenne minimális, hiszen a kör bármelyik élét elhagyva a megmaradt élhalmaz továbbra is lefogó maradna;
2. ha Y_{min}-ben lenne három él hosszú út, akkor szintén nem lenne minimális, hiszen az illető út középső élét elhagyva a megmaradt élhalmaz ez esetben is lefogó maradna.

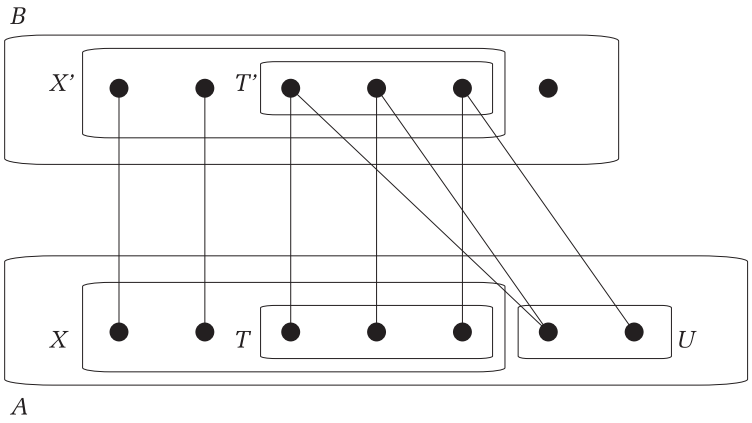
Legyen c az Y_{min}-beli élek alkotta csillagerdő komponenseinek száma. Mivel G-ben nincsenek izolált pontok és Y_{min} lefogó, ezért a csillagerdő pontjainak száma azonos G összes pontjainak számával ($|V(G)|$). Mivel minden csillagban 1-gyel több pont van, mint él, ezért $\rho(G) + c = |V(G)|$, azaz $c = |V(G)| - \rho(G)$. Ha minden csillagból kiválasztunk egy élet, akkor az így kapott élhalmaz nyilván független lesz. Ez viszont azt jelenti, hogy a független élek maximális száma nem lehet kisebb c-nél. Azaz: $\nu(G) \geqslant c$. Behelyettesítve c-t, kapjuk, hogy $\nu(G) \geqslant |V(G)| - \rho(G)$, vagyis $\nu(G) + \rho(G) \geqslant |V(G)|$.

11.11. tétel (Kőnig). Ha G *páros* gráf, akkor $\nu(G) = \tau(G)$. Ha G-nek nincs izolált pontja, akkor $\alpha(G) = \rho(G)$ is igaz.

A tétel belátása: A 11.5. tétel kimondta, hogy bármely G gráfban $\nu(G) \leqslant \tau(G)$. Be fogjuk bizonyítani, hogy páros gráfok esetén ez fordítva is igaz ($\nu(G) \geqslant \tau(G)$), azaz $\nu(G) = \tau(G)$. Ha ez igaz, akkor ebből és a Gallai-tételekből (amelyeknek értelmében $\alpha(G) + \tau(G) = \nu(G) + \rho(G)$) triviálisan következik az $\alpha(G) = \rho(G)$ állítás is.

Legyen M egy olyan párosítás, amely a javítóutak módszerével már nem bővíthető. Mivel egy párosítás élei függetlenek, a független élek maximális száma nyilván nem lehet kisebb $|M|$-nél, azaz $\nu(G) \geqslant |M|$. Legyenek továbbá a következő jelölések (lásd a 11.4. ábrát):

- X az M párosítás éleinek A halmazbeli végpontjai,
- X' az M párosítás éleinek B halmazbeli végpontjai,
- $U = A - X$,
- T' azon B-beli pontok halmaza, amelyek elérhetők U-ból alternáló úton,
- T a T' pontjainak párosításbeli párjainak halmaza,
- $Y = T' \cup (X - T)$.



11.4. ábra. *A Kőnig-tétel bizonyítása*

Figyeljük meg, hogy az Y halmaznak pontosan $|M|$ pontja van, és ezek lefogják a gráf összes élét. Ezért az is kijelenthető, hogy a lefogó élek minimális száma nem lehet nagyobb, mint $|M|$, azaz $\tau(G) \leqslant |M|$. Tehát $\tau(G) \leqslant |M| \leqslant \nu(G)$, vagyis $\tau(G) \leqslant \nu(G)$.

Megjegyzés: A magyar módszer hatékony algoritmus a $\nu(G)$ és $\tau(G)$ értékek meghatározására, illetve egy maximális független élhalmaz és egy minimális lefogó ponthalmaz megtalálására bármely páros gráfban.

12. FEJEZET

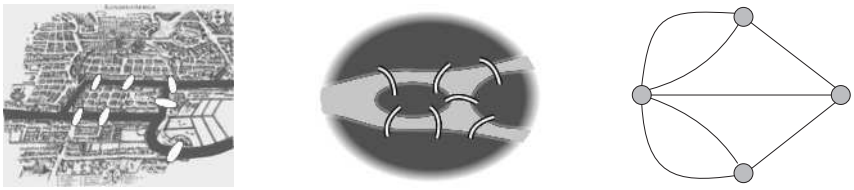
EULER- ÉS HAMILTON-GRÁFOK

Gyakorlati probléma: Utcai öntözőkocsi gazdaságos útvonalának meghatározása a zárt Euler-vonal megállapításával történhet meg, ha a bejárandó utak hálózata olyan irányított gráffal ábrázolható, melyben minden pont ki-fokszáma egyenlő a be-fokszámával. Odafigyelünk arra is, hogy egyirányú utcákon egyszer, kétirányú utcákon kétszer kell áthaladnia az öntözőkocsinak.

A königsbergi hidak problémája egy híres matematikai probléma, amelyet *Leonhard Euler* oldott meg. A probléma története, hogy a poroszországi Königsberg (most Kalinyingrád, Oroszország) városban hét híd ívelt át a várost átszelő Prégel folyón úgy, hogy ezek a folyó két szigetét is érintették. A königsbergiek azzal a kérdéssel fordultak Eulerhez, vajon végig lehet-e menni az összes hídon úgy, hogy mindegyiken csak egyszer haladjanak át, és egyúttal visszaérjenek a kiindulópontba. 1736-ban Euler bebizonyította, hogy ez lehetetlen. (A történethez hozzátartozik az a legenda is, hogy 1750 körül állítólag a königsbergi elit tagjai rendszeresen sétálgattak vasárnaponként a hidakon, egy olyan útvonalat keresve, amely megfelel a fenti feltételeknek.)

12.1. ábra. *Leonhard Euler, a „matematika Mozartja" (1707–1783)*

A bizonyítás során Euler a problémát a gráfelmélet nyelvén fogalmazta meg, azaz leegyszerűsítette azt: a földeket, azaz a folyó partjait, beleértve a szigeteket is, a mai megfogalmazás szerint csomópontoknak, a hidakat pedig éleknek tekintette. Az így létrehozott csomópontok és élek egy gráfot határoznak meg.



12.2. ábra. *A königsbergi hidak modellezése gráfokkal*

Euler észrevette, hogy a problémát az így létrehozott gráf csomópontjainak fokszámaira lehet visszavezetni. A csomópont fokszámán az adott csomóponthoz csatlakozó élek számát értjük. A konkrét esetben a hidak elhelyezkedése alapján megalkotott gráfban három pontnak 3 a fokszáma, egynek pedig 5. Euler bebizonyította, hogy akkor és csak akkor létezik ebben a gráfban a hidakon pontosan egyszer végighaladó séta, ha minden csomópont fokszáma páros. A fenti feltételnek eleget tevő összefüggő gráfokat ma Euler-gráfoknak (zárt Euler-vonal) nevezzük. Mivel a königsbergi hidak gráfjában több páratlan fokszámú csúcspont is található, Euler eredményéből következik, hogy a königsbergi hidakat nem lehet bejárni a fent megkövetelt módon.

Ha a kiindulópontnak és a célpontnak nem kell azonosnak lennie, akkor *nyílt Euler-vonal*ról, illetve *nyílt Euler-séta*ról beszélünk. Ahhoz, hogy egy gráfban nyílt Euler-vonal legyen, összefüggőnek kell lennie, és pontosan két darab páratlan fokszámú csomóponttal kell rendelkeznie. Ebben az esetben a nyílt Euler-séta kiinduló- és végpontja pontosan a két páratlan fokszámú pontja a gráfnak.

A matematika történetében a königsbergi hidak problémáját, illetve ennek Euler-féle megoldását tartják az első gráfelméleti problémának. Azóta a gráfelmélet a kombinatorika egy önálló területévé vált.

Ezen túlmenően az, hogy Euler felismerte, hogy a probléma megoldásának a kulcsa a hidak, illetve pontosabban az egy partszakaszhoz kapcsolódó hidak számában, nem pedig ezek konkrét elhelyezkedésében keresendő, a topológiai szemlélet legkorábbi megjelenésének is tekinthető.

12.1. tétel. A G gráf akkor és csak akkor Euler-gráf, ha összefüggő és bármely csúcsának a foka páros.

A tétel belátása: Egyértelmű, hogy egy Euler-gráf szükségképpen összefüggő, és minden csúcspontjának a foka páros. A feltétel elégséges voltához tekintsük a G gráf valamely zárt vonalát. G-nek van zárt vonala, mert bármely pontjából kiindulva, szomszédos éleken lépegetve (ügyelve arra, hogy ne haladjunk kétszer át ugyanazon az élen), előbb-utóbb visszaérkezünk a kezdeti pontba. Ehhez az a biztosíték, hogy G véges, összefüggő (ezért tudunk biztosan elindulni a kiválasztott pontból), és minden pontjának fokszáma nullánál nagyobb páros szám (ha be tudtunk lépni egy pontba, akkor ki is tudunk lépni belőle; az első pont esetében ez azt jelenti, hogy végül belépünk oda, ahonnan kezdetben kiléptünk). Ha zárt vonalunkra (L_1) rákerült minden él, akkor megvan a zárt Euler-vonal. Ellenkező esetben, G összefüggősége miatt, biztos van olyan e kimaradt él, amelyik legalább egyik végpontjával L_1 valamelyik v pontjára illeszkedik. Az előbbi gondolatmenet értelmében, ha elindulunk e v pontból az e él mentén (természetesen elkerülve L_1 éleit, illetve azokat az éleket, amelyeket e második vonal mentén már érintettünk), akkor végül visszaérkezünk v-be (legyen ez az L_2 zárt vonal). Ehhez megint csak az a biztosíték, hogy ha G minden pontjának fokszáma páros, akkor G $\setminus$ L_1 minden pontjának fokszáma is páros (egy zárt vonal összes élének „törlésével" minden vonal menti pont fokszáma páros értékkel csökken). Tehát L_2 mentén is igaz, hogy ha egy pontba be tudtunk lépni, akkor ki is tudunk lépni belőle. Ha v-ből kiindulva először bejárjuk L_1-et, majd L_2-t, akkor a két zárt vonalat egyetlen zárt vonalnak tekinthetjük. Ha ezen összevont zárt vonalon sincs még rajta az összes él, akkor folytatjuk az „algoritmust" az előbbi módon. (A sikeres zárt-vonalkereséshez az indulást G összefüggősége, a továbbhaladást a fokszámok párossága, a célba jutást pedig a gráf véges volta garantálja.)

Az elmondott bizonyítás lényegében algoritmust ad a G gráf Euler-vonalának meghatározására. Ennek implementálását az olvasóra hagyjuk.

12.2. tétel. Adott G összefüggő gráfra a következő állítások az ekvivalensek:

1. G Euler-gráf.
2. G minden csúcsának a foka páros.
3. G él-idegen körök uniója.

A tétel belátása: A bizonyítást az 1→2→3→1 séma alapján érdemes elvégezni.

1→2: Ahhoz, hogy az első állításból következik a második, elegendő azt észrevenni, hogy a tetszőleges L zárt vonal tetszőleges u csúcspontjára igaz, hogy L bejárása során pontosan annyiszor léptünk be u-ba, ahányszor kiléptünk belőle. Ezért u foka páros. Azaz G bármely csúcspontjának a fokszáma páros.

2→3: A G gráf összefüggőségéből és csúcsai fokszámának páros voltából adódik, hogy minden pont fokszáma nagyobb vagy egyenlő kettővel. Ezért kiindulva a G gráf tetszőleges v pontjából, megkapjuk G-nek egy L zárt vonalát. Zárt vonal mindig tartalmaz legalább egy kört. Tetszőleges kör bármely pontjának a fokszáma páros. Ha a G gráfunk valamely C körének töröljük az éleit, akkor G bármely csúcspontjának a foka továbbra is páros maradt. Mindaddig találunk újabb él-idegen köröket, amíg az élek törlése után megmaradó gráfnak van olyan v csúcspontja, amelynek foka nagyobb, mint nulla. Az eljárás miatt a körök éleinek a halmazai diszjunktak.

3→1: Valóban, ha a G gráf összefüggő és él-idegen körök uniója, akkor be lehet járni a gráf éleit oly módon, hogy minden élen csak egyszer megyünk végig. Bizonyítsunk a körök száma szerinti teljes indukcióval. Ha a gráf csak egyetlen él-idegen körből áll, akkor az az egy kör önmagában egy zárt Euler-vonal lesz. Ha már bejártunk $(k-1)$ kört és a k-adik körrel a zárt vonalunknak az u pontja közös, akkor járjuk be a $(k-1)$ kört alkotó zárt vonalat u-ból indulva, majd ha már visszatértünk u-ba, folytassuk a bejárást a k-adik kör éleinek a bejárásával.

12.3. tétel. Ha a G egyszerű, összefüggő gráfnak $2k$ darab páratlan fokú csúcspontja van, akkor élei lefedhetők k darab nyílt vonallal.

A tétel belátása: Egészítsük ki a G gráfot k darab éllel G'-vé, oly módon, hogy G' minden csúcsának a foka páros legyen. Ez természetesen megtehető, ha ügyelünk arra, hogy az új élekkel mindig páratlan fokú csúcsokat kössünk össze. G'-re teljesedik a 12.1. tétel feltétele, ezért van egy zárt Euler-vonala, mely nyilván tartalmazza a betett k darab élt is. Ha a k darab új élt töröljük, k darab nyílt vonalat kapunk.

Irányított Euler-gráfok

A fenti gondolatmenet alapján belátható, hogy egy irányított gráfban pontosan akkor van zárt Euler-vonal, ha minden csúcs be- és ki-fokszáma

megegyezik. Egy irányított gráfban pontosan akkor létezik nyílt Euler-vonal, ha minden csúcs esetében a bemenő és kimenő élek száma azonos, és pontosan egy csúcsnál eggyel kevesebb a kimenő élek száma, mint a bemenőké, illetve pontosan egy csúcsnál eggyel nagyobb.

A kínai postás probléma

A kínai postás probléma a gráfelmélet egyik érdekes kérdése: Hány élismétléssel lehet bejárni egy gráfot úgy, hogy minden élen áthaladjunk legalább egyszer?

A problémával akkor kezdtek el foglalkozni a kínaiak, amikor megpróbáltak a postások számára minél rövidebb útvonalat kijelölni úgy, hogy mindenkinek ki tudják kézbesíteni a levelét, vagyis olyan útvonalakat kerestek, ahol a postás minden úton áthalad legalább egyszer, de a lehető legkevesebbszer halad át olyanon, amelyen már végigment egyszer.

Észrevétel: Ha létezik a gráfban Euler-kör, akkor az egyben a kínai postás optimális útvonala. A postás tényleges útvonalán a többször bejárt éleket megfelelő multiplicitású többszörös éleknek tekintve olyan gráfot kapunk, amelynek van Euler-köre. Egyik élen sem érdemes több mint kétszer átmenni. A feladat azzal ekvivalens, hogy legkevesebb hány él megduplázásával tehető olyanná a gráf, hogy legyen benne Euler-kör, azaz hogy minden csúcs fokszáma páros legyen. (Az összefüggőséget eleve feltételezzük.) Ennek a feladatnak megoldására van jól működő algoritmus. Az olvasóra hagyjuk ennek megtalálását.

12.1. Hamilton-gráfok

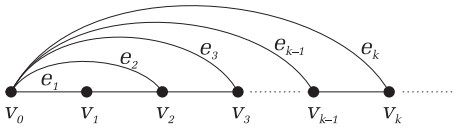
Gyakorlati probléma: Egy villamosvezetékeket szerelő vállalat darukat, traktorokat és egyéb gépi berendezéseket használ. Ugyanabban az időben több helyen is dolgozik. A gépek mozgatása egyik helyről a másikra költséges. Feltevődik a kérdés, hogy milyen sorrendben kell a vállalatnak elvégeznie a szereléseket, hogy a lehető legkevesebbet költsön a gépek ide-odaszállítására. Amennyiben az egyes vonalakat egy gráf pontjaival, az egyik vonal szereléséről a másikra való áttérési költséget pedig irányított élekkel ábrázoljuk, akkor a feladat megoldását a gráf legrövidebb irányított Hamilton-útja adja meg.

Sir William Rowan Hamilton (1805–1865) 1859-ben egy olyan játékot hozott forgalomba, amelynek a lényege az volt, hogy egy előre megadott gráf csúcspontjait kellett bejárni oly módon, hogy bármely csúcsban pontosan egyszer kellett járni. Állítólag a játéknak nem volt átütő sikere Hamilton kortársai között.

12.3. ábra. *Sir William Rowan Hamilton (1805–1865)*

Úgy tűnhet, hogy ez a probléma hasonló ahhoz, amelyben a gráfnak az éleit kell bejárnunk pontosan egyszer. Előrebocsátjuk, hogy ha a gráf csúcspontjait kell pontosan egyszer bejárni, a feladat jóval nehezebb. Az általános esetben Hamilton-utak, illetve Hamilton-körök keresésére ma sem ismert igazán jó algoritmus. Az operációkutatás területéhez tartozik az utazó ügynök problémája. Az utazó ügynök problémában a kereskedelmi utazónak adott városokat kell bejárnia, oly módon, hogy minden városba csak egyszer megy el, és végül visszatér a cégének a székhelyére. Ez esetben a gráf csúcspontjai az utazó által meglátogatandó városok, az élek pedig a városokat összekötő útvonalak. Mivel egy-egy útnak jól meghatározott útiköltsége lehet, több út esetén célszerű azt az utat választani, amelynek a költsége minimális. E feladat a kombinatorikus optimalizálás tárgykörébe tartozik. A következő tétel megfogalmazása előtt említjük meg, hogy egy kör, illetve út hosszán a bennük szereplő élek számát értjük.

12.4. tétel. Ha a G egyszerű gráfban bármely csúcspont foka legalább k ($k \geq 2$), akkor a gráfban van egy legalább $(k+1)$ hosszúságú kör.



12.4. ábra.

A tétel belátása: A fokszámra vonatkozó követelményből adódóan a G gráf leghosszabb útja biztosan legalább k élből áll. (Mivel minden pont fokszáma legalább k, ezért kiindulva egy tetszőleges pontból, ezen első k állomás esetén biztos mindig van egy még érintetlen szomszéd, amelyik irányba továbbhaladhatunk.) Jelölje ezen leghosszabb út (L) út csúcspontjait a kezdőponttól indulva rendre $v_0, v_1, \ldots, v_k, v_{k+1}, \ldots, v_p$. Mivel v_0 foka legalább k, a v_0-t a v_1-gyel összekötő e_1 élen kívül még legalább $(k-1)$ él indul ki v_0-ból. Ezeknek az éleknek a másik végpontja szükségszerűen mind ott található L csúcspontjai között, hiszen ellenkező esetben összeütközésbe kerülnénk azzal a feltevéssel, hogy L a leghosszabb út. (Ha v_0-nak lenne olyan szomszédja, amelyik nem eleme L-nek, akkor L kezdődhetne vele, és így 1-gyel hosszabb lenne.) A „legrosszabb esetben" az $e_2, e_3, \ldots, e_k$ élek másik végpontjai éppen a $v_2, v_3, \ldots, v_k$ pontok. Ekkor az e_k él az L útnak a v_0-tól v_k-ig tartó részútjának két végpontját köti össze, tehát egy olyan kört kapunk, amelyben van legalább $(k+1)$ él. (Ha v_0 valamelyik szomszédja v_k-n túli pontja L-nek, akkor a gráfnak $(k+1)$-nél hosszabb köre is van.)

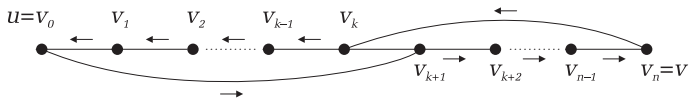
12.5. tétel. Ha a G egyszerű gráf bármely v csúcsának fokára teljesül, hogy $d(v) \geqslant n/2$, akkor G összefüggő.

A tétel belátása: Legyen u és v a G gráf két különböző csúcsa. A fokszámra vonatkozó feltétel szerint u-val és v-vel is egyenként legalább $n/2$ pont van összekötve az u-ból, illetve a v-ből induló élek által. Ezen u-val, illetve v-vel közvetlenül összekötött pontok között van olyan, amely u-val és v-vel is össze van kötve (ha nem létezne ilyen pont, akkor G csúcsainak a száma nagyobb vagy egyenlő volna, mint $(n/2 + n/2 + 2)$). Következik, hogy u és v között vezet út.

Adott G gráf csúcsainak számát ($|V| = n$) nevezhetjük még G *helyettesítési rendjé*nek is, éleinek ($|E| = m$) számát pedig a G gráf *méreté*nek. Az u csúcsponttal szomszédos csúcsok halmazát $N(u)$-val jelöljük.

12.6. tétel (O. Ore, 1961). Ha a G gráfra teljesül, hogy rendje $n \geqslant 3$ és bármely két nem szomszédos u, v csúcspont fokának az összege nagyobb vagy egyenlő G rendjénél ($d(u) + d(v) \geqslant n$), akkor G-nek van Hamilton-köre.

A tétel belátása: Indirekt bizonyítást alkalmazunk. Azon gráfok közül, amelyekre teljesülnek a tétel feltételei, de maga az állítás nem, tekintsünk egy olyat (G')-t, amelyben az élek száma maximális. Maximális abban az értelemben, hogy ha G'-hez hozzáveszünk egy olyan e élet, amely a nem szomszédos u és v éleket köti össze, akkor az így kapott G gráf már fog tartalmazni Hamilton-kört. G minden Hamilton-köre tartalmazza az e élet, tehát G'-nek van olyan L Hamilton-útja, amely u-t és v-t köti össze. Legyen ez az út: $u = v_0, v_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_n = v$. Vegyük észre, hogy ha v_{k+1} szomszédos u-val, azaz v_{k+1} eleme $N(u)$-nak, akkor v_k nem eleme $N(v)$-nek.



12.5. ábra.

Ellenkező esetben a $v_0, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_0$ kör Hamilton-köre volna G'-nek. Tehát a $V - \{v\}$ pontok közül az u-val szomszédos pontok nem szomszédosak v-vel, ezért $d(u) \leqslant (n-1) - d(v)$. Ez utóbbi egyenlőtlenség ellentmond a tétel feltételeinek.

Ore tételének speciális esete Dirac tétele.

12.7. tétel (G. A. Dirac, 1952). Ha az $n = 2k$ csúcspontú egyszerű G gráf bármely pontjának a foka legalább k, akkor G-nek van Hamilton-köre.

Az időrendben való jobb tájékozódás végett (egységes jelölést alkalmazva) felsoroljuk a Hamilton-körökre vonatkozó érdekesebb eredményeket. Jelölje a $G(V, E)$ gráf csúcspontjainak fokszámait rendre $d_1 \leqslant d_2 \leqslant \ldots \leqslant d_n$ ($|V| = n$).

12.8. tétel. Ha a $G(V, E)$ egyszerű gráfra ($2 < n$) teljesül a következő feltételek valamelyike, akkor G-nek van Hamilton-köre: (Bizonyítás végett lásd [12].)

- G. A. Dirac (1952): $1 \leqslant k \leqslant n \Rightarrow d_k \geqslant n/2$,
- O. Ore (1961): $u, v \in V, (u, v) \notin E \Rightarrow d(u) + d(v) \geqslant n$,
- Pósa Lajos (1962): $1 \leqslant k \leqslant n/2 \Rightarrow d_k > k$,

– J. A. Bondy (1969): $j < k$ és $d_j, d_k \leqslant k-1 \Rightarrow d_j + d_k \geqslant n$,
– V. Chvátal (1972): $d_k \leqslant k < n/2 \Rightarrow d_{n-k} \geqslant n-k$.

12.9. tétel. Az n ($n > 3$) csúcsú teljes gráfnak $((n-1)!)/2$ különböző Hamilton-köre van.

A tétel belátása: A csúcsok lehetséges sorrendjeinek száma: $n!$. Ezeknek mindegyike meghatároz egy Hamilton-kört a teljes gráfban, de vannak olyanok, amelyek ugyanazt a kört. El szeretnénk érni, hogy minden egyes Hamilton-kört pontosan egyszer számoljunk. A gráfunk bármely Hamilton-köre pontosan $2n$-szer áll így elő, hiszen a csúcsok felsorolását egy Hamilton-kör bármely pontjából kiindulva elkezdhetjük, és ezt két különböző irányban tehetjük meg. Így a különböző Hamilton-körök száma: $(n!)/(2n) = ((n-1)!)/2$.

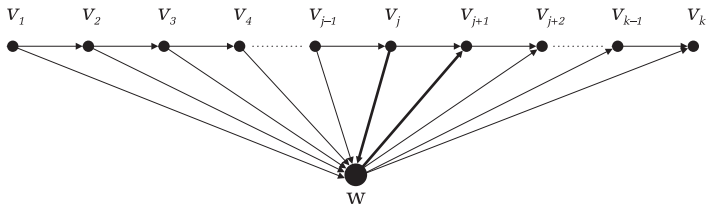
12.10. tétel. A $K_{n,n}$ teljes páros gráf ($n \geqslant 2$ esetén) különböző Hamilton-köreinek száma: $(n!)^2/(2n)$.

A tétel belátása: A gráf egy-egy pontosztályába tartozó csúcsokat nevezzük „alsó", illetve „felső" csúcsoknak. Soroljuk fel a csúcsoknak az összes olyan permutációját, mely alsó csúccsal kezdődik, és felváltva tartalmaz alsó, illetve felső csúcsokat. Az ilyen permutációk száma $(n!)^2$, hisz külön-külön a két halmazban levő pontok $n!$-féleképpen permutálhatók, de a permutációkat össze kell fésülni. Ezen permutációk mindegyike egy Hamilton-kört határoz meg, de vannak olyanok, amelyek ugyanazt. Minden Hamilton-kör pontosan $2n$-szer áll így elő, hisz a Hamilton-kör n darab „alsó" csúcsának mindegyikéből két különböző irányban sorolhatjuk fel a csúcsokat. Így a különböző Hamilton-körök száma: $(n!)^2/(2n)$.

12.11. tétel (Rédei László, 1934). Minden *turné* gráfban (olyan irányított gráf, amelyben bármely pontpár között vagy oda, vagy vissza él van) van irányított Hamilton-út.

A tétel belátása: Legyen G egy turné gráf és ebben legyen P egy maximális hosszúságú irányított út, amelyben az irányítás mentén a $v_1, v_2, \ldots, v_k$ pontok követik egymást. Indirekt módon tegyük fel, hogy P nem Hamilton-út, azaz van olyan w pont, amelyik nincs rajta P-n. Mivel G turné gráf, w és P összes pontja között van él. P maximalitásából adódóan a w és v_1 pontpár esetén v_1-től w felé halad az él, a w és v_k pontpár esetén pedig w-től v_k felé. Ebből következik, hogy ha a $\{v_1, w\}, \{v_2, w\}, \ldots, \{v_k, w\}$ pontpárok közötti éleket átfésüljük ebben

a sorrendben, akkor páratlan sokszor (tehát legalább egyszer) irányításváltás történik. Ezzel összhangban tegyük fel, hogy a w és v_j pontpár esetén v_j-től w felé halad az él, a w és v_{j+1} pontpár esetén pedig w-től v_{j+1} felé. Ez esetben a $(v_1, v_2, \ldots, v_j, w, v_{j+1}, \ldots, v_k)$ irányitott út P-nél hosszabb, ami ellentmondás. Tehát G leghosszabb irányított útja szükségszerűen Hamilton-út. (Rédei bebizonyította azt az erősebb állítást is, hogy minden turné gráfban páratlan sok Hamilton-út van.)



12.6. ábra. *A w pont beékelése a P útra a v_j és v_{j+1} pontok közé*

A tétel bizonyításának gondolatmenetét követve könnyen alkotható hatékony algoritmus a szóban forgó Hamilton-út megtalálására:

- Kiindulva egy tetszőleges pontból, addig haladunk előre (körmentesen), amíg lehetséges.
- Ezután az indulási pontból addig hátrálunk (körmentesen), amíg lehetséges. (Az éleken az irányításukkal ellentétesen haladva át.)
- Ha az így kapott P útra nem került rá az összes pont, akkor a bizonyítás ötletét használva minden kimaradt pont beékelhető P-be (lásd a 12.6. ábrát).

12.12. tétel. Egy turné gráfban pontosan akkor van irányított Hamilton-kör, ha a turné gráf erősen összefüggő.

12.1.1. Az utazó ügynök problémája

Nem negatív élsúlyozott K_n teljes gráfban keresünk minimális súlyú C_H Hamilton-kört.

A „legközelebbi szomszéd" algoritmus (mohó algoritmus, lásd a B függeléket)

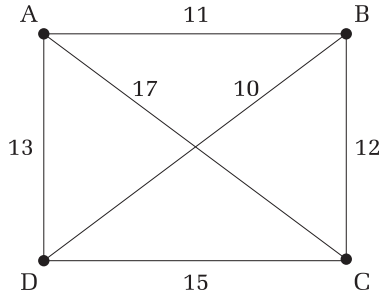
Kiindulva egy tetszőleges pontból, mindig a legközelebbi még érintetlen szomszéd irányába megyünk tovább. Az n-edik lépésben visszatérünk a kiindulópontba. E mohó algoritmus nem garantálja az optimális Hamilton-kört. Ez abból is látszik, hogy más-más eredményhez jutunk aszerint, hogy melyik pontból indulunk. Ami kijelenthető, az az, hogy a kapott körhossz *felső korlát* lesz az utazó ügynök problémára. Ha a gráfunk nem teljes, a fenti algoritmus azt sem biztosítja, hogy egyáltalán találunk (amennyiben létezik ilyen) Hamilton-kört.

A rendezett élek algoritmusa (mohó algoritmus, lásd a B függeléket)

Feltesszük, hogy a K_n élsúlyozott teljes gráf élei súlyuk növekvő sorrendje szerint rendezve vannak. Az éleket súlyuk szerint növekvő sorrendben választjuk ki, ügyelve arra, hogy az aktuális él egyik végpontja se illeszkedjen olyan pontra, amelyre már korábban kiválasztott élek közül kettő illeszkedik, és hogy a kiválasztott élek ne alkossanak n csúcspontnál kevesebb pontból álló kört. Azokat az éleket, amelyek nem felelnek meg az előbbi feltételeknek, átugorjuk. Ha a kiválasztott élek száma n, akkor megkaptunk K_n-ben egy súlyozott C_H Hamilton-kört.

Alsó korlátot oly módon nyerhetünk az utazó ügynök problémára, ha észrevesszük, hogy K_n egy minimális súlyú C_H Hamilton-körének tetszőleges v pontját törölve, a $K_n - \{v\}$ gráfnak egy súlyozott feszítőfáját kapjuk. Keressünk a $K_n - \{v\}$ gráfban egy minimális súlyú T feszítőfát (például Kruskal algoritmusával). Jelölje $S(T)$ a T-t alkotó $(n-2)$ él súlyainak az összegét. Legyen e_1 és e_2 a v-re illeszkedő élek közül a két legkisebb súlyú. Az $S(T) + \textit{súly}(e_1) + \textit{súly}(e_2)$ érték *alsó korlát* az utazó ügynök problémára.

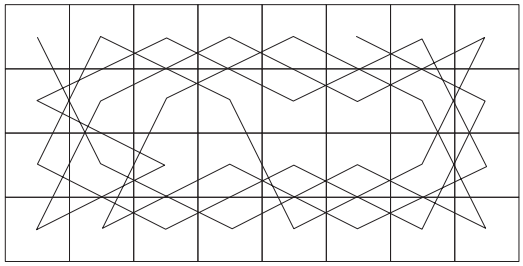
A 12.7. ábrán a $G - \{D\}$ gráfban a minimális súlyú feszítőfát az AB és BC élek alkotják. A D pontra illeszkedő két legkisebb súlyú él a BD és AD. Tehát az utazó ügynök probléma alsó korlátja e példára $k = 11 + 12 + 10 + 13 = 46$. A gráf B csúcsából indulva a legközelebbi szomszéd algoritmus rendre a BD, AD, AC, BC éleket adja, és így nyerjük a $k = 10 + 13 + 17 + 12 = 52$ felső korlátot.



12.7. ábra. *Optimális Hamilton-kör súlyozott K_4 gráfban: A, B, C, D, A. Összsúlya: 51*

Egy kézenfekvő megoldás az optimális Hamilton-kör meghatározására az lenne, hogy generáljuk az összes Hamilton-kört, amelyek közül kiválasztjuk a minimális súlyút. Ez a feladat az exponenciális bonyolultságú backtracking algoritmussal oldható meg (lásd a B függeléket).

A fejezet befejezéséül íme egy „látványos" Hamilton-út:



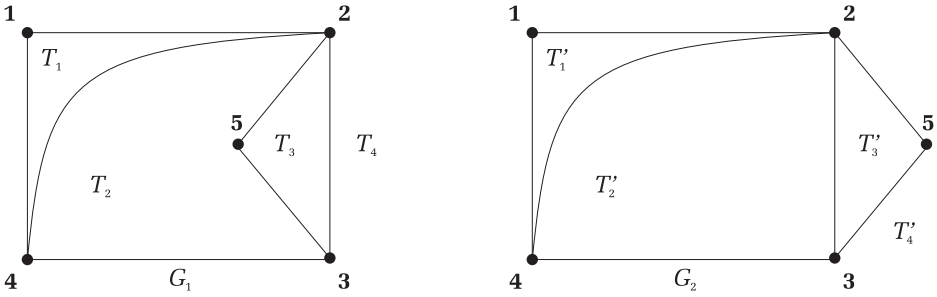
12.8. ábra. *A huszár Hamilton-útja a 4×8-as sakktáblán*

13. FEJEZET

SÍKBA RAJZOLHATÓ GRÁFOK

Gyakorlati probléma: Korszerű út/vonat-hálózatot szeretnénk kiépíteni n város között. Ismert, hogy mely városok között kell léteznie direkt útnak, illetve direkt vonatsín-kapcsolatnak. A kérdés az, hogy megépíthető-e a közlekedési hálózat felül-, illetve aluljárók nélkül. A gráfelmélet nyelvén: síkba rajzolható-e a városok közlekedési hálózatának gráfja?

Egy G gráfot *síkba rajzolható*nak mondunk, ha a gráf éleit olyan vonalakkal lehet ábrázolni a síkban, hogy bármely két élnek csak a G gráf csúcspontjaiban lévő végpontjai legyenek közösek.



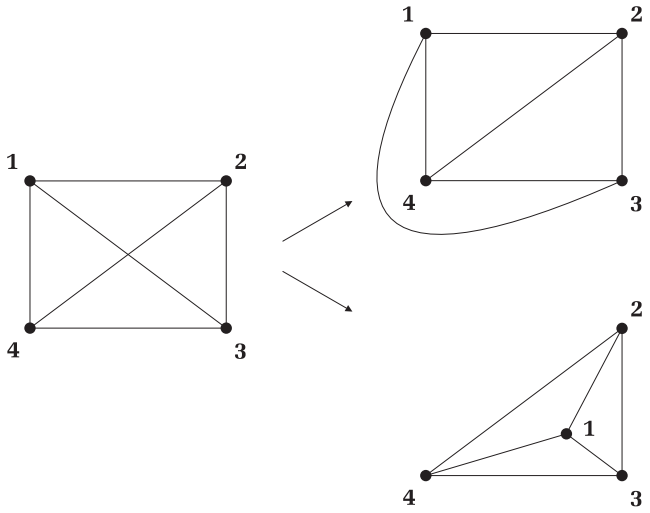
13.1. ábra. *Izomorf gráfok különböző módokon síkba rajzolva*

A 13.1. ábra azt mutatja, hogy a G_1 és G_2 gráfok izomorfak, de a síkba rajzolásuk lényegesen különbözik. A G_2 külső végtelen tartományt (lásd lennebb) 5 él határolja, és G_1-nek nincs 5 él által határolt tartománya. A G_1 gráf T_2 tartományát 4 él határolja, G_2-nek viszont csak 3, illetve 5 éllel határolt tartományai vannak.

A 13.2. ábra azt mutatja be, hogy egy gráf attól még síkba rajzolható lehet, hogy az aktuális lerajzolásából ez nem nyilvánvaló.

13.1. tétel. Bármely G véges gráf megvalósítható a háromdimenziós euklideszi térben.

A tétel belátása: Helyezzük el a G gráf n pontját egy t egyenes mentén. Húzzuk be a gráf m élét m darab t-re illeszkedő, kettőnként

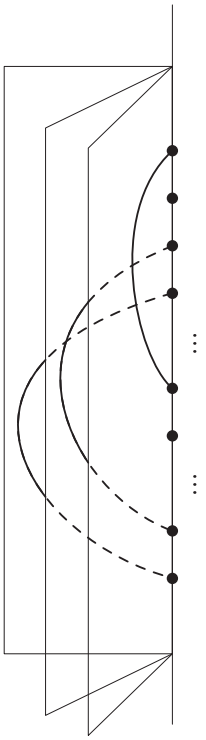


13.2. ábra.

különböző síkban úgy, hogy csak a végpontjaikban metsszék a t egyenest. Lásd a 13.3. ábrát.

Gyakran olyan egyszerű összefüggő síkbeli gráfokkal van dolgunk, melyek úgynevezett sokszöghálót alkotnak abban az értelemben, hogy felbonthatók olyan minimális körökre, melyek a belsejükben nem tartalmazzák a gráfnak egyetlen pontját sem. E köröket nevezzük *tartomány*oknak. Egy ilyen gráf minden csúcsának foka nagyobb, mint kettő, és bármely éle pontosan két tartományt határol. Vegyük például Afrika térképét (eltekintünk Madagaszkártól és Lesothótól). Ha élekként az országhatárokat és a tengerpartokat értelmezzük, és csúcspontokként azokat a pontokat, ahol legalább három él találkozik, akkor egy sokszöghálót kapunk. Az országoknak megfelelő minimális körök a tartományok. Az Afrikát körülvevő tartományt *végtelen tartomány*nak nevezzük.

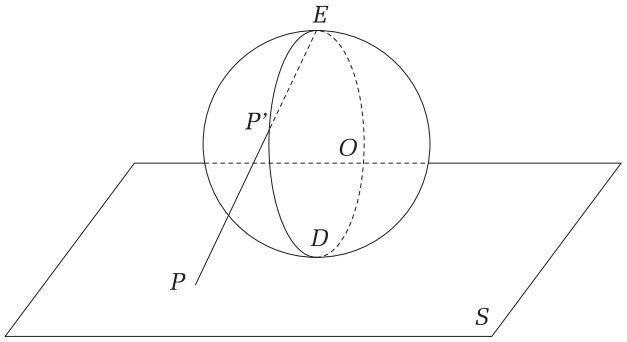
Térképészetben gyakran használt eljárás a sztereografikus projekció, mikor is egy G gömb és egy S sík pontjai között létesítünk megfeleltetést. Tételezzük fel, hogy a G gömbünk egy D pontban érinti a síkot. Legyen E a D pont átellenes pontja (gondoljunk a déli, illetve az északi sarokra). A sík valamely P pontjának gömbi megfelelőjét (P'-t) megkapjuk, ha vesszük a P-t az E-vel összekötő egyenesnek a metszéspontját a G gömbbel. A sztereografikus projekcióval ábrázolhatunk a síkon egy gömbre rajzolt gráfot, és fordítva, egy síkba rajzolt gráfot izomorf módon ábrázolhatunk a gömbön. (Gömbről síkra vetítés esetén, ha a gömbi



13.3. ábra. *Tetszőleges gráf a háromdimenziós euklideszi „térbe rajzolva". A gráf éleit megvastagítottuk*

13.4. ábra. *Afrika térképe*

gráfnak egyik pontja éppen az északi sarokra esne, akkor fordítsuk el a gömböt úgy, hogy az E pont ne essen egybe a gráf egyetlen pontjával se.)



13.5. ábra. *Sztereografikus projekció*

Legyen a G síkgráf éleinek a száma m, csúcsainak száma n, tartományainak száma t, és a komponenseinek a számát jelölje c.

13.2. tétel (Euler-formula). Ha G összefüggő síkgráf, akkor: $n - m + t = 2$.

A tétel belátása: Ha G fagráf (1 tartománya és $(n-1)$ éle van), akkor nyilvánvaló a formula helyessége: $n - (n-1) + 1 = 2$. Legyen most egy tetszőleges G összefüggő síkgráf. Gondolatban távolítsuk el annyi élét, hogy fa maradjon. Az így kapott fagráfról már beláttuk, hogy kielégíti az Euler-formulát. Most tegyük vissza egyenként az eltávolított éleket. Minden betett él növeli 1-gyel az m értékét is és a tartományok számát is. Mivel m és t ellentétes előjellel szerepel az $(n-m+t)$ kifejezésben, az Euler-formula bal oldala invariáns az élvisszahelyezési műveletre nézve.

13.3. tétel (Euler-formula). Ha G síkgráf, akkor: $n - m + t = 1 + c$.

A tétel belátása: Ha G erdő (1 tartománya és $(n-c)$ éle van), akkor nyilvánvaló a formula helyessége: $n - (n-c) + 1 = 1 + c$. Legyen most egy tetszőleges G síkgráf. Gondolatban távolítsuk el annyi élét, hogy G minden komponense fává alakuljon. Az így kapott erdőről már beláttuk, hogy kielégíti az Euler-formulát. Most tegyük vissza egyenként az eltávolított éleket. Minden betett él növeli 1-gyel az m értékét is és a tartományok számát is. Mivel m és t ellentétes előjellel szerepel az $(n-m+t)$ kifejezésben, az Euler-formula bal oldala invariáns az élvisszahelyezési műveletre nézve.

Az alábbi egyenlőtlenségek az Euler-formula következményei, és azt hangsúlyozzák, hogy a síkba rajzolható gráfoknak nem lehet „sok" élük.

13.4. tétel. Ha G egy legalább 3 pontú egyszerű összefüggő síkgráf, akkor: $m \leqslant 3n - 6$.

A tétel belátása: Egyfelől az összes tartományt határoló élek száma kisebb vagy egyenlő, mint $2m$. Ez azért van így, mert minden él legfennebb két tartományt határol (az elvágó élek csak egyet). Másfelől az összes tartományt határoló élek száma nagyobb vagy egyenlő, mint $3t$. Ez azért igaz, mert minden tartományt legalább 3 él határol. Mindebből következik, hogy $3t \leqslant 2m$. Behelyettesítve t értékét az Euler-formula alapján $(t = 2 + m - n)$, a keresett egyenlőtlenséghez jutunk: $m \leqslant 3n - 6$.

13.5. tétel. Ha G egy legalább 4 pontú egyszerű síkgráf, és minden körének hossza legalább 4, akkor: $m \leqslant 2n - 4$.

A tétel belátása: Ez esetben is kijelenthető, hogy az összes tartományt határoló élek száma kisebb vagy egyenlő, mint $2m$. Mivel minden kör hossza legalább 4, ez alkalommal az összes tartományt határoló élek száma nagyobb vagy egyenlő, mint $4t$. Mindebből következik, hogy $4t \leqslant 2m$. Behelyettesítve ebbe az egyenlőtlenségbe a t helyett a $(2 + m - n)$ kifejezést, a keresett egyenlőtlenséghez jutunk: $m \leqslant 2n - 4$.

13.6. tétel. Minden egyszerű síkba rajzolható gráfban létezik legfeljebb 5-ödfokú pont. (A síkgráfokban nem lehet mindenik pont „nagy" fokszámú.)

A tétel belátása: Feltételezzük indirekt módon, hogy minden pont fokszáma legalább hat. Ez esetben kijelenthető, hogy $2m \geqslant 6n$, azaz hogy $m \geqslant 3n$. Másfelől viszont $m \leqslant 3n - 6$ (13.4. tétel). A két egyenlőtlenség azonban ellentmond egymásnak.

Gráfok síkba rajzolhatóságát nem befolyásolja, ha valamely élükön egy új másodfokú csúcspontot veszünk fel, vagy ha valamely két élét a gráfnak, amelyek ugyanarra a v másodfokú csúcsra illeszkedtek, egybeolvasztjuk, és v-t töröljük. Az előbbi két transzformációt nevezzük *topologikus bővítésnek,* illetve *szűkítésnek.*

13.7. definíció. A G gráfot a G' gráffal *topologikusan izomorfnak* mondjuk, ha G-ből véges sok topologikus szűkítéssel, illetve bővítéssel előállítható egy G'-vel izomorf gráf.

13.8. tétel (G. Kuratowski, 1930). A G gráf akkor és csak akkor síkba rajzolható, ha nincs olyan részgráfja, amely topologikusan izomorf a K_5 teljes gráffal, vagy a $K_{3,3}$ gráffal. Ez utóbbit szokás *háromház-háromkút* gráfnak is nevezni.

A K_5 és a $K_{3,3}$ gráfokat a fenti tétel kapcsán Kuratowski-féle gráfoknak is szokták nevezni. A tételt nem bizonyítjuk. Az Euler-formula segítségével azonban könnyen bizonyítható, hogy a Kuratowski-féle gráfok nem síkba rajzolhatók.

13.9. tétel. A Kuratowski-féle gráfok nem rajzolhatók síkba.

A tétel belátása: Azonnal belátható, hogy a K_5 gráfra nem igaz az $m \leqslant 3n-6$ egyenlőtlenség (13.4. tétel). Bár a $K_{3,3}$ gráfra igaz, hogy $m \leqslant 3n-6$, nem elégíti ki az $m \leqslant 2n-4$ egyenlőtlenséget (13.5. tétel).

13.6. ábra. *A $K_{3,3}$ és K_5 Kuratowski-féle gráfok*

13.10. tétel (Fáry-Wagner). Minden síkgráf lerajzolható a síkban úgy, hogy élei egyenes szakaszok legyenek. A tételt nem bizonyítjuk.

13.1. Síkgráfok duálisai

13.11. definíció. Síkba rajzolható G sokszöggráfokhoz sok esetben hasznos hozzárendelni egy G^* *duális gráf*ot a következő módon. G minden T_i tartományában felveszünk egy u_i pontot. Legyen e egy olyan éle G-nek, mely a T_i és T_j tartományok határán fekszik. Ekkor vezessen az e^* él u_i-ból u_j-be. A 13.7. ábrán szaggatott vonallal rajzoltuk meg a G_1, illetve a G_2 gráf duálisának az éleit. A duális gráfok csúcspontjait a tartományokban vettük fel és szürkével ábrázoltuk (lásd még a 13.8. ábrát).

13.12. tétel. Bármely síkba rajzolható G gráf valamely körét/vágását alkotó éleknek megfelelő G^*-beli élek vágást/kört alkotnak. (A duális képzés kört vágásba, vágást pedig körbe visz át.)

Egy gráf duálisának „definíciója" matematikailag problematikus. Két izomorf gráfot matematikailag identikusnak tekintünk. (Bár izomorf gráfok különböző módokon is lerajzolhatók, a rajz csak szemléltetés.) A dualitás fogalmával ott van a gond, hogy izomorf gráfok duálisai nem föltétlenül izomorfak egymással. Ez a helyzet a 13.7. ábra G_1 és G_2 egymás közt izomorf gráfok esetében is. Az is furcsa, hogy egy gráf duálisának duálisa nem föltétlen izomorf az eredeti gráffal.

13.13. definíció. Két gráfot *gyengén izomorf*nak nevezünk, ha élei között kölcsönösen egyértelmű és körtartó leképezés hozható létre (körtartás helyett vágástartás is mondható).

13.14. tétel (Whitney). Ha G_1 síkba rajzolható és G_2 gyengén izomorf vele, akkor

- G_2 is síkba rajzolható,
- G_1^* és G_2^* szintén gyengén izomorfak egymással,
- $(G_1^*)^*$ gyengén izomorf G_1-gyel, és $(G_2^*)^*$ gyengén izomorf G_2-vel.

Természetes ötletnek tűnik, hogy a dualitást is definiálhatnánk kört vágásba (és fordítva) átvivő megfeleltetésnek.

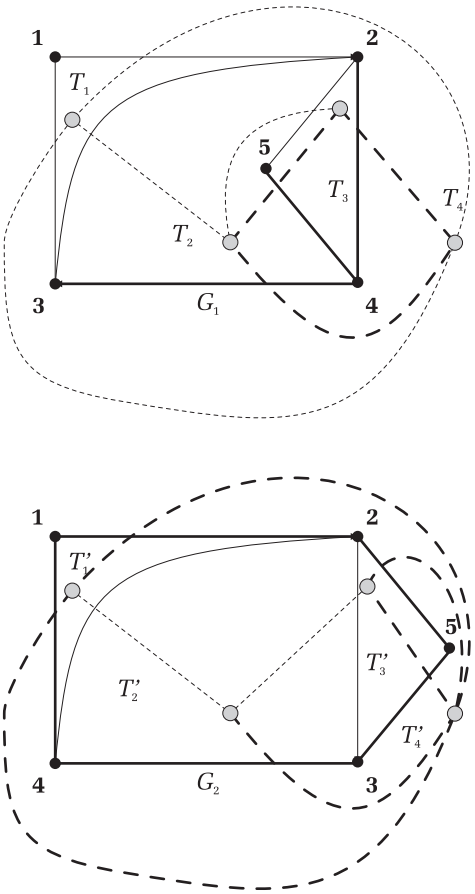
13.15. definíció. A G és G^* gráfokat egymás *absztrakt duális*ainak tekintjük, ha az éleik között létesíthető olyan, kölcsönösen egyértelmű leképezés, amely kört vágásba, vágást pedig körbe visz át.

13.16. tétel (Whitney). Egy gráfnak akkor és csakis akkor létezik absztrakt duálisa, ha síkba rajzolható.

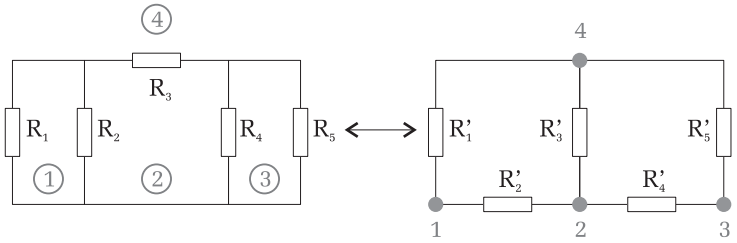
13.1.1. Egy villamosmérnöki alkalmazása

A 2.5. és 2.6. alfejezetekben kifejtettük, hogy a kétpólusú alkatrészekből álló áramkörökre vonatkozó Kirchhoff-féle hurok- és csomóponttörvények $C\boldsymbol{u} = \mathbf{0}$, illetve $Q\boldsymbol{i} = \mathbf{0}$ alakba írhatók (az $\boldsymbol{u}$ és $\boldsymbol{i}$ vektorok elemei az egyes alkatrészek feszültség-, illetve áramértékei).

A 13.8.a ábrán látható áramkörnek, mint síkgráfnak, a 13.8.b áramkör a duálisa, és fordítva. Figyeljük meg, hogy az első hálózat (a) alkatrészeinek *feszültségei* között ugyanolyan kapcsolatok érvényesek, mint a



13.7. ábra. *G_1 és G_2 izomorf síkgráfok (folytonos vonalak) és duálisaik (szaggatott vonalak). G_1 vastagított folytonos vonalakkal jelölt vágásának a duális gráfban a vastagított szaggatott vonalakkal jelölt kör felel meg. G_2 vastagított folytonos vonalakkal jelölt körének a duális gráfban a vastagított szaggatott vonalakkal jelölt vágás felel meg. Bár a G_1 és G_2 gráfok izomorfak egymással, a duálisaikra ez már nem igaz.*



13.8. ábra. *„Duális" áramkörök. Az első ábrán megjelöltük (szürkével) a tartományokat, a másodikon pedig az ezeknek megfelelő csomópontokat. Ha az R_k ellenállás az i és j tartományokat határoló élen van, akkor az R'_k ellenállás az i és j pontokat összekötő élre került.*

második hálózat (b) alkatrészeinek *áramai* között, és fordítva. Például (a megfelelő mérőirányok rögzítése után):

az (a) hálózatban	*a (b) hálózatban*
$u_1 = u_2$	$i_1 = i_2$
$i_3 = i_4 + i_5$	$u_3 = u_4 + u_5$

Ahhoz, hogy a két hálózatot leíró egyenletrendszert matematikailag teljesen azonosnak tekinthessük, szükséges még az is, hogy az egyes *alkatrészek* esetén is az $u_k = R_k i_k$ egyenletek (a. hálózatban) helyett formálisan $i_k = R'_k u_k$-t (b. hálózatban) írhassunk. Ennek feltétele azonban az, hogy $R'_k = R_k^{-1}$. Ez viszont nem más, mint a villamos hálózatok elméletéből jól ismert dualitási elv:

> Ha a G és G^* gráfok egymás duálisai, és a megfelelő élek helyére olyan ellenállásokat helyezünk, amelyeknek értékei egymásnak reciprokai, akkor az első hálózat k-adik alkatrészének feszültségértéke egyenlő lesz a második hálózat k-adik alkatrészének áramértékével, és fordítva.

Természetesen nem szükséges ellenállás-hálózatokra szorítkoznunk, hiszen az elv bármely kétpólusú alkatrészekből összerakott hálózatra érvényes, csak a feszültség és áram szerepét kell felcserélnünk. Például az $u = L \cdot du/dt$ egyenletű induktivitás „duálisa" az $i = C \cdot di/dt$ egyenletű kapacitás.

Néhány további villamos áramköri „duális" fogalom a 13.1.a táblázatban látható. A 13.1.b táblázat a síkba rajzolható gráfok köréből tartalmaz „duális" fogalompárokat.

13.1. táblázat. *„Duális” fogalmak*

a) Villamos hálózatokban	
Feszültség	Áram
Feszültségforrás	Áramforrás
Rövidzár	Szakadás
Párhuzamos kapcsolás	Soros kapcsolás
Ellenállás	Vezetés (reciprok ellenállás)
Induktivitás	Kapacitás

b) Síkgráfok esetén	
Él	Él
Pont	Tartomány
Kör	Vágás
Fa	Fa komplementere
Párhuzamos élek	Soros élek
Hurokélek	Elvágó élek

14. FEJEZET

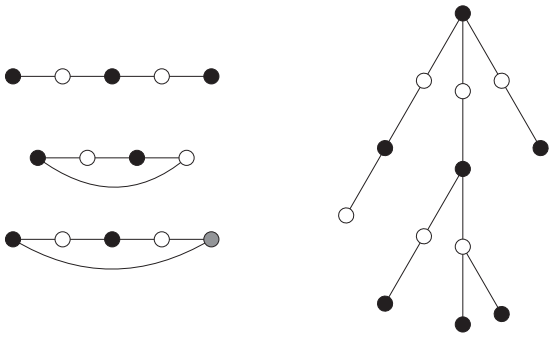
GRÁFOK SZÍNEZÉSE

Gyakorlati probléma: Színezzük ki a világtérképet négy színel. Tegyünk több ajánlatot is, hogy a megrendelő a „legesztétikusabbat" választhassa.

Gráfok síkba rajzolhatóságáról beszélve definiáltuk a tartomány fogalmát. Térképek színezésében szokásos eljárás az, hogy a szomszédos országokat (tartományokat) különböző színekkel színezik ki. Két országot szomszédosnak szokás nevezni, ha a közös határuk hossza 0-nál nagyobb szám, azaz van közös élük. A G síkbeli gráfot k *színezhető*nek mondjuk, ha G tartományait ki lehet festeni k színnel oly módon, hogy bármely két szomszédos ország különböző színű legyen. Ha az adott síkba rajzolható G gráf duálisa G^*, akkor a G gráf k színnel való színezhetősége G^*-ra vonatkozólag azt jelenti, hogy G' csúcsait ki lehet színezni k színnel oly módon, hogy a szomszédos csúcsok színei különbözőek legyenek. Csúcsok színezésekor mindig feltételezzük, hogy a színezendő gráfunk egyszerű. A színezési probléma duális gráfra való átfogalmazása lehetőséget ad arra, hogy tetszőleges gráf színezhetőségéről beszéljünk.

14.1. definíció. A G gráf kromatikus száma $\chi(G)$, ha G csúcsai $\chi(G)$ színnel kiszínezhetők, de kevesebbel, $(\chi(G)-1)$-gyel már nem.

A P_n hosszú út ($n > 1$) egymást követő csúcsait felváltva színezhetjük fehérre és feketére, ezért $\chi(P_n) = 2$. A páros sok csúcsot tartalmazó C_{2n} ($n \geqslant 2$) kör csúcsai hasonlóan jól színezhetők két színnel, ezért $\chi(C_{2n}) = 2$. A C_{2n+1} ($n \geqslant 1$) páratlan sok éllel rendelkező kör csúcsai azonban pontosan három színnel színezhetők jól, azaz $\chi(C_{2n+1}) = 3$. Az is könnyen belátható, hogy bármely T fagráf (tree = fa) csúcsai két színnel jól színezhetők. Legyen u egy tetszőleges pontja a T fagráfnak, és válasszuk u színének a feketét. Ezek után a T valamely v csúcspontját attól függően színezzük fehérre vagy feketére, hogy az u-ból v-be páratlan sok élen vagy páros sok élen keresztül érkezünk meg. Így $\chi(T) = 2$, ha $|V(T)| \geqslant 2$.



14.1. ábra. *A P_5 (fehér/fekete), C_4 (fehér/fekete), C_5 (fehér/fekete/szürke) gráfok és egy fagráf (fehér/fekete) kiszínezése*

A G gráf $V(G)$ csúcsainak színezését oly módon is definiálhatjuk, hogy G csúcsainak a halmazát leképezzük a természetes számok halmazára valamely f függvény segítségével. Értelemszerűen ekkor egy-egy természetes számot tekintünk egy-egy színnek. Egy színezést jónak (vagy valódinak) fogunk mondani, ha a G gráf szomszédos u, v csúcsainak színei mindig különbözők. Vegyük észre, hogy a G gráf $V(G)$ csúcsainak bármely színezése $V(G)$-nek olyan osztályozását definiálja, ahol egy osztályba nem kerülhetnek szomszédos csúcsok. Az $\chi(G)$ kromatikus szám az a legkisebb szám, amelyre teljesül, hogy a G gráf $V(G)$ csúcsait olyan módon lehet $\chi(G)$ számú diszjunkt részhalmazra bontani, hogy nincs a részhalmazok egyikében sem két olyan u, v pont, amelyek a G gráfban éllel volnának összekötve. Egy gráfnak megtalálni a legjobb színezését általában nem könnyű feladat. Megengedett valódi színezést azonban könnyen kaphatunk a mohó színezési algoritmussal.

Az algoritmus a következőképpen működik. Rögzítsük le egy tetszőleges sorrendjét a G gráf $V(G)$ csúcsainak. A soron következő pontot a már kiszínezett szomszédainál nem használt legkisebb színnel jelöljük meg. Az algoritmus a színek „gyors" kiválasztásában mohó. Mohó olyan értelemben, hogy mindig az „első" „megengedett" színt választja. Előfordulhat, hogy más szín választása, összességében, az egész gráf színezéséhez gazdaságosabb volna, de az algoritmus éppen azért gyors, mert elkerüli ennek a vizsgálatát.

14.2. tétel. A G gráf $\chi(G)$ kromatikus száma akkor és csak akkor egyenlő 2-vel, ha G páros gráf és van legalább egy éle.

A tétel belátása: Ha G-t kiszíneztük jól fehérrel és feketével, akkor G bármely élének két végpontja különböző színű lesz. A fehér pontokat V_1-be, a fekete színű pontokat V_2-be gyűjthetjük. Másrészt, ha G páros gráf, akkor csúcsainak halmaza $V(G) = V_1 \cup V_2$ felbontható két nem üres diszjunkt részhalmazra oly módon, hogy G egyetlen élének a végpontjai sem lehetnek csupán V_1-ben vagy csak V_2-ben. Ezért a V_1 pontjait színezhetjük fehérre és V_2 pontjait feketére. A feladat által garantált létező él miatt $\chi(G) = 2$.

14.3. tétel. Ha a G gráf egyszerű, akkor $\chi(G) \leqslant \Delta(G) + 1$. ($\Delta(G)$ a gráf csúcspontjai fokszámának maximuma.)

A tétel belátása: A csúcspontok száma szerinti indukcióval bizonyítunk. Két csúcspont esetén nincs mit bizonyítani, mivel a gráf egyszerűsége miatt a fokszám maximuma nulla vagy egy lehet. Legyen igaz az állítás az n csúcspontú gráfokra, s igazoljuk az állítást az $n + 1$ csúcsponttal rendelkező gráfokra. Ha a fokszámok maximuma $\Delta(G)$, akkor az $(n + 1)$-edik csúcs legfeljebb $\Delta(G)$ darab csúccsal van összekötve. Színezzük ezt a megmaradt plusz egy színnel.

Megjegyzések:

- Figyeljünk fel arra, hogy a 14.3. tétel állítása bizonyos esetekben nem javítható. Például az n csúcspontú teljes gráf csúcspontjai fokszámainak a maximuma $n-1$, és a teljes gráf kromatikus száma pontosan n. Hasonló módon egy páratlan hosszú kör maximális fokszáma 2, a kromatikus száma pedig 3.
- Más esetekben azonban nagyon távol van a felső becslés a kromatikus szám valódi értékétől. Például az n pontú S_n csillaggráfnak $n-1$ a maximális fokszáma (a központi $(n-1)$-ed fokú ponthoz $n-1$ levél kapcsolódik), de a kromatikus száma mindössze kettő. Hasonló a helyzet a $K_{n,n}$ gráf esetén is: maximális fokszáma n, a kromatikus száma viszont csak 2. Ezek a példák arról szólnak, hogy egyes gráfok maximális fokszámértéke bármeddig növelhető, miközben a kromatikus számuk állandó marad (például 2). Tehát a kromatikus számnak a maximális fokszámmal való felülről becslése igen gyenge becslés.

14.4. definíció. A G gráf valamely teljes részgráfját *klikk*nek nevezzük. Egy G gráf klikkszáma a legnagyobb klikk csomópontjainak száma és $\omega(G)$-vel jelöljük.

14.5. tétel. A G gráf kromatikus száma mindig nagyobb vagy egyenlő mint a klikkszáma: $\chi(G) \geqslant \omega(G)$.

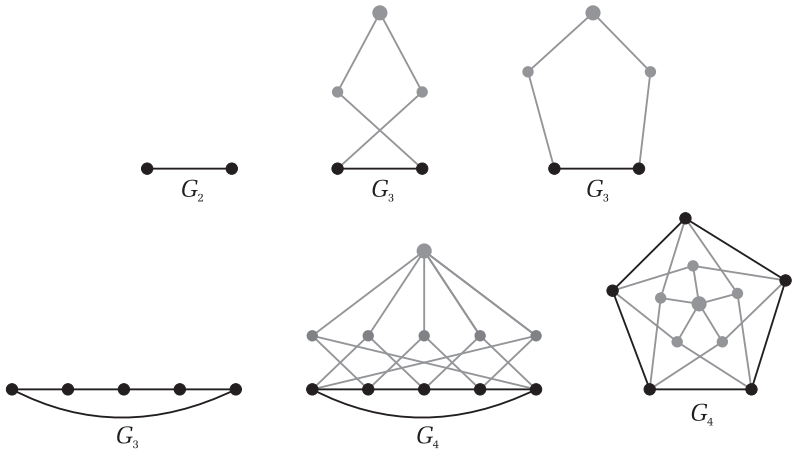
Megjegyzések:

- A bizonyítás magától értetődő. Az egyenlőség például a teljes gráfokra áll fenn.
- Az alábbi konstrukcióval kimutatjuk, hogy a kromatikus számnak a klikkszámmal való alulról való becslése is nagyon gyenge becslés. Egy olyan gráfot építünk, amelynek kromatikus száma akármeddig növelhető, miközben a klikkszáma 2 marad (nem tartalmaz háromszöget).

Mycielski konstrukciója

- Kiindulunk a G_2 gráfból, a kétpontú egyszerű teljes gráfból.
- G_k-ból úgy építjük fel G_{k+1}-et, hogy
  - G_k minden v_i pontja mellé felveszünk egy-egy u_i pontot.
  - Minden u_i pontot összekötünk a megfelelő v_i pont G_k-beli szomszédjaival.
  - Felveszünk még egy w pontot is, amelyet összekötünk minden u_i ponttal.

Megjegyzés: Ha G_k-nak n_k pontja volt, akkor $n_{k+1} = 2n_k + 1$ (ahol $n_2 = 2$).



14.2. ábra. *G_2-ből megépül G_3, illetve G_3-ból G_4. Szürkével jelöltük a v_i pontok párjaiként felvett u_i pontokat, illetve nagyobbnak rajzoltuk a w pontot*

14.6. tétel. Bármely $k \geqslant 2$ esetén a Mycielski szerint konstruált G_k gráf klikkszáma 2, a kromatikus száma viszont k.

A tétel belátása: Indukciót alkalmazunk k szerint. Nyilvánvaló, hogy G_2 klikkszáma is és kromatikus száma is 2. Először belátjuk, hogy ha G_k nem tartalmazott háromszöget, akkor G_{k+1} sem tartalmaz (azaz klikkszáma 2). Egyértelmű, hogy G_{k+1}-ben nem lehetnek w csúcsú háromszögek, mert w csak az u_i pontokkal van összekötve, ezek pedig egymással nincsenek összekötve. Másfelől olyan háromszög sem lehet, amelyiknek egyik csúcsa u címkéjű és másik két csúcsa v címkéjű, hiszen minden u_i csúcs pontosan azokkal a v pontokkal van összekötve, amelyekkel a párja, a v_i pont. Tehát, ha lenne G_{k+1}-ben egy u és két v csúcsú háromszög, akkor lett volna G_k-ban is csupa v csúcsú háromszög, ami viszont ellentmondás. Az, hogy más típusú háromszög sem lehet G_{k+1}-ben, triviálisan belátható.

Most lássuk be, hogy amennyiben G_k kiszínezéséhez kell k szín, akkor G_{k+1} kiszínezéséhez kell $(k+1)$. Indirekt módon tegyük fel, hogy G_{k+1} kiszínezhető k színnel. Legyen egy ilyen jó színezés, amelyben w a k-adik színt kapja. Mivel az u pontok szomszédai w-nek, ezért biztosan az $\{1, 2, \ldots, k-1\}$ halmazból kaptak színeket. Most fessük át mindenik v_i pont színét a párjaként felvett u_i pont színére. Mivel minden v_i pontnak ugyanazok a szomszédai, mint a megfelelő u_i pontnak, ezért kijelenthető, hogy egy másik k színű jó színezést kaptunk G_{k+1} részére. Ebben a színezésben viszont a v pontok (a G_k gráf pontjai) mindenike az $\{1, 2, \ldots, k-1\}$ halmazból van kiszínezve, ami ellentmond annak, hogy G_k kromatikus száma k.

14.1. Perfekt gráfok

14.7. definíció. A G gráfot *perfekt*nek nevezzük, ha kromatikus száma egyenlő a klikkszámával, és minden feszített részgráfjára is igaz ugyanez.

Megjegyzések:

- Kifejtettük, hogy bármely gráfban a klikkszám alsó becslés a kromatikus számra, hiszen egy jól-színezésnél a legnagyobb klikk minden csúcsa különböző színt kap. A perfekt gráfok azok, amelyekre ez a becslés éles, nemcsak magában a gráfban, hanem minden feszített részgráfjában is. Vannak nem perfekt gráfok is, mint például a legalább 5 hosszú páratlan körök, amelyeknek színezéséhez legalább 3 szín kell, de a legnagyobb klikk pontjainak száma csak 2.
- A perfekt gráf fogalma Berge-től származik az 1960-as évek elejéről, és azóta a gráfelmélet egyik legfontosabb és legsikeresebb fogalmává vált.

14.8. tétel. Minden páros gráf perfekt.

A tétel belátása: Mivel minden G páros gráfnak minden feszített részgráfja is páros gráf, ezért elég belátni, hogy minden páros gráfra $\omega(G) = \chi(G)$. Ez triviálisan igaz, ugyanis egy páros gráf háromszögmentes, és ha legalább egy éle van, akkor $\omega(G) = 2$ és $\chi(G) = 2$. Ha nincs éle a gráfnak, akkor pedig $\omega(G) = \chi(G) = 1$.

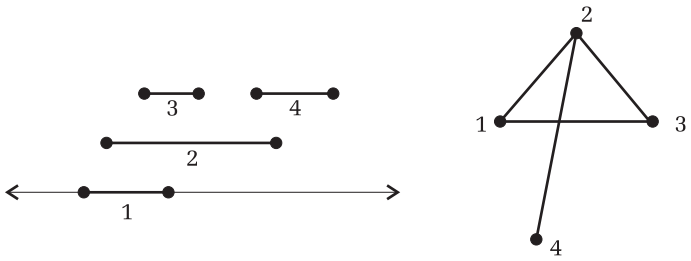
14.9. definíció. Az *intervallumgráf* olyan gráf, amelynek a pontjai megfeleltethetőek a valós számok egy-egy intervallumának, és két pontja között pontosan akkor van él, ha a megfelelő intervallumok metszete nem üres.

Megjegyzések:

Az operációkutatásban az intervallumgráfokat erőforráskiosztási problémák modellezésére használják. Az intervallumok jelzik az egyes kérelmek időtartamát; az optimális kiosztásnak a legnagyobb súlyú független ponthalmaz felel meg. Egy másik alkalmazásuk a folytonos szakaszok megkeresése a DNS-térképek készítésekor.

14.10. tétel. Minden intervallumgráf perfekt.

A tétel belátása: Intervallumgráfoknak feszített részgráfjai is intervallumgráfok, tehát elég belátni, hogy minden intervallumgráfra $\chi(G) = \omega(G)$. Tudjuk, hogy $\chi(G) \geqslant \omega(G)$, ezért elég belátni, hogy $\chi(G) \leqslant \omega(G)$. Legyen $\omega(G) = k$. Színezzük az intervallumokat bal végpontjuk szerint, balról jobbra, a legelső színnel, ami nem mond ellent a korábbi intervallumok színezésének (mohó stratégia). Ha valamelyik intervallum színezéséhez a $(k+1)$-edik színt kellene használnunk, az azt jelentené,



14.3. ábra. *Intervallumgráf.* $\omega(I) = \chi(I) = 3$

hogy ennek az intervallumnak bal oldali végpontja benne van k másik intervallumban. Ez azt jelentené, hogy a gráfban van $(k+1)$ méretű klikk, ami ellentmondás.

14.11. tétel (perfekt gráf tétel – Lovász László, 1972). Perfekt gráf komplementere is perfekt.

Megjegyzések:

- Eredetileg ez volt a *(gyenge) perfekt gráf sejtés* (Fulkerson, 1971).
- Később Lovász igazolta, hogy egy G gráf pontosan akkor perfekt, ha minden H feszített részgráfjára igaz, hogy $\omega(H)\omega(\underline{H}) \geqslant |V(H)|$, ahol $\underline{H}$-val H komplementerét jelöltük.
- Mivel a fenti feltétel szimmetrikus G-re és G komplementerére, az eredmény azonnal adja a perfekt gráf tételt.

14.12. tétel (erős perfekt gráf tétel – Chudnovsky, Robertson, Seymour és Thomas, 2002). Egy gráf akkor és csakis akkor perfekt, ha nem tartalmaz feszített részgráfként legalább öt hosszú páratlan kört, vagy annak komplementerét.

Megjegyzések:

- Ezt már Berge megsejtette 1960/1961-ben, és *erős perfekt gráf sejtés* néven volt ismert.
- Nem sokkal ezután Chudnovsky, Cornuéjols, Liu, Seymour és Vušković polinomiális algoritmust adott annak eldöntésére, hogy egy adott gráf perfekt-e.

14.2. Él-kromatikus szám

14.13. definíció. A G gráf kromatikus indexe $\chi_e(G)$ (él-kromatikus száma), ha G élei $\chi_e(G)$ színnel kiszínezhetők, de kevesebbel $(\chi_e(G)-1)$-gyel már nem.

Megjegyzés: Triviális, hogy bármely hurokélmentes G gráfra $\chi_e(G) \geqslant \Delta(G)$.

14.14. tétel (V. G. Vizing, 1964). Ha a G gráf egyszerű gráf, akkor $\chi_e(G) = \Delta(G)$ vagy $\chi_e(G) = \Delta(G) + 1$. (Bizonyítás végett lásd a [6], 85. oldalt.)

Megjegyzések:

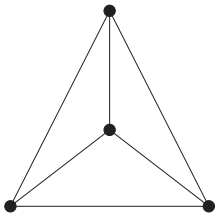
- Az, hogy $\chi_e(G) \leqslant \Delta(G) + 1$, belátható, követve a mohó stratégiát.
- Csak exponenciális időben dönthető el, hogy egy általános G gráf esetében $\chi_e(G) = \Delta(G)$ vagy $\chi_e(G) = \Delta(G) + 1$.

14.3. Az öt-/négyszín-tételek

1840-ben Möbius egy előadásában megfogalmazta a négyszínsejtést, mely azt állítja, hogy bármely síkbeli térkép kiszínezhető 4 színnel. A sejtést De Morgan tette ismertté. Ő a problémát Franci Guthrietól vette át 1850 körül. Cayley 1879-ben jelentet meg egy dolgozatot a négyszínsejtésről a Proc. Royal Geographical Society első kötetében. 1890-ben Heawood egy Kempetől származó hibás bizonyítást vizsgál, és sikerül bebizonyítania, hogy síkgráfoknál öt szín elegendő a színezéshez. A négyszínsejtésre K. Appel és W. Haken 1976-ban számítógép felhasználásával adtak bizonyítást. Bizonyításukban azonban azóta többen és többször is találtak hibát, de azok javíthatónak bizonyultak. A matematikusok egy része vitatja azt, hogy egy tételnek a számítógépes „bizonyítása" elfogadható-e.

A továbbiakban síkbeli gráfokról szeretnénk megmutatni, hogy öt színnel mindig kiszínezhetők. Azt belátni, hogy legalább 4 szín szükséges, könnyű. Például a tetraéder síkba rajzolt gráfja csak 4 színnel színezhető ki.

14.15. tétel (ötszín-tétel). Bármely síkba rajzolható gráf kiszínezhető öt színnel.



14.4. ábra. *Síkba rajzolt tetraéder*

A tétel belátása: Indukciót alkalmazunk a gráf pontszáma szerint. Kétpontú gráfra nyilvánvalóan igaz a tétel állítása. Ezután feltesszük, hogy minden n pontú G_n síkgráf kiszínezhető legfennebb 5 színnel, és bebizonyítjuk, hogy ebből következik ugyanez minden $(n+1)$ pontú G_{n+1} síkgráfra. A 13.6. tételben kijelentettük (és bebizonyítottuk), hogy minden síkgráfnak van legfennebb 5 fokszámú pontja. Legyen v egy ilyen pontja G_{n+1}-nek. Ha v fokszáma legfennebb négy, akkor színezzük ki a $G_{n+1} - \{v\}$ n pontú gráfot legfennebb öt színnel, majd adjunk v-nek egy olyan színt, amilyet nem használtunk a négy szomszédja esetében.

Tekintsük most azt az esetet, amikor v fokszáma pontosan 5. Ha v minden szomszédja össze van kötve, akkor G_{n+1}-ben szerepel egy K_6, ami viszont ellentmond a síkbarajzolhatóságnak. Legyen u_1 és u_2 két olyan szomszédja v-nek, amelyek nincsenek összekötve. Vonjuk össze a v, u_1 és u_2 pontokat egy ponttá, majd az így kapott $(n-1)$ pontú gráfot színezzük ki legfennebb 5 színnel. Ez a színezés persze nem jó G_{n+1}-ben (miután az összevont pontokat szétválasztjuk), hiszen a v, u_1 és u_2 pontok azonos színt kaptak, holott u_1 és u_2 szomszédai v-nek. Változtassunk G_{n+1} színezésén a következőképpen: mivel a v pont többi három szomszédja legfentebb 3 színt foglal le, fessük át az u_1 és u_2 pontokat a negyedik színre (lehetséges, mert nem szomszédok), és legyen az 5-ödik szín a v ponté. Ezzel találtunk egy legfennebb ötszínű helyes színezést a G_{n+1} gráfra.

14.16. tétel (négyszín-tétel). Bármely síkba rajzolható gráf kiszínezhető négy színnel.

Megjegyzések:

- A bizonyítás több száz oldalas.
- Egy backtracking algoritmussal generálható az összes helyes négy színnel való színezés (lásd a B. függeléket).

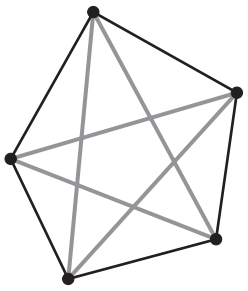
15. FEJEZET

NÉHÁNY RÉSZGRÁFOKKAL KAPCSOLATOS TÉTEL

Néhány részgráfokkal kaocsolatos eredmény Ramsey nevéhez fűződik. Ramsey, alig 17 évnyi életideje alatt, jelentőset alkotott mind a matematika, mind a filozófia és a gazdaságtan területén. Matematikai tételeire gyakran úgy utalnak, mint a Dirichlet-féle skatulyaelv általánosítására. Másfelől úgy is jellemzik kutatási eredményeit, mint annak bizonyítékát, hogy a világ nem lehet teljesen véletlen, valamilyen szabályosságoknak törvényszerűen létezniük kell.

Közismert feladat, hogy hat ember között mindig találunk vagy 3 olyat, akik ismerik egymást, vagy 3-at, akik nem ismerik egymást. A gráfelmélet nyelvén megfogalmazva az alábbi tételhez jutunk.

15.1. tétel. Bármely 6 pontú gráfban vagy van 3 olyan pont úgy, hogy bármely kettő között van él, vagy van 3 olyan pont úgy, hogy bármely kettő között nincs él.



15.1. ábra. *A K_5 gráf olymódon kiszínezve, hogy nincs benne egyszínű háromszög. (A fehér szín helyett szürkét használtunk.)*

Egy n pontú gráfot felfoghatunk úgy is, hogy az n pontú teljes gráf azon éleit, amelyek szerepelnek a gráfunkban, feketére festjük ki, a többit pedig fehérre. Ebből a szempontból az előbbi tétel azt jelenti, hogy a K_6 gráf éleit bárhogy festenénk ki két színnel (fekete/fehér), biztos lesz benne vagy egy fekete, vagy egy fehér háromszög. A tétel éles abban az

értelemben, hogy az öt csúcspontú teljes gráf éleit lehet úgy feketére és fehérre színezni, hogy nem lesz benne sem fekete, sem fehér háromszög (lásd a 15.1. ábrát). Ramsey tétele a 15.1. tétel általánosításának tekinthető.

15.2. tétel (Ramsey). Adott p és q pozitívak mellett létezik egy olyan legkisebb $r(p,q)$ szám, hogy e számnál nagyobb vagy egyenlő pontszámú ($n \geqslant r(p,q)$) teljes gráfok (K_n) esetén, ezek két színnel (fekete/fehér) való kiszínezése után van a gráfban vagy fekete K_p, vagy fehér K_q. (Bizonyítás végett lásd a [6], 87. oldalt.)

A *Ramsey-féle számok* értelmezése szemléletesebben: ha az $r(p,q)$ pontú teljes gráf éleit tetszés szerint kiszínezzük fehérre vagy feketére, akkor $K_{r(p,q)}$-nak vagy lesz egy K_p-vel izomorf fekete részgráfja, vagy egy olyan, amelyik K_q-val izomorf és összes éle fehér. Továbbá az $r(p,q)-1$ pontú teljes gráf éleinek létezik olyan fekete/fehér színezése, hogy nincs benne se K_p-vel izomorf fekete részgráf, se K_q-val izomorf és fehér részgráf.

15.3. tétel (Erdős–Szekeres). $r(p,q) \leqslant r(p-1,q)+r(p,q-1)$. (Bizonyítás végett lásd a [6], 87. oldalt.)

A gráfelmélet történelme során felmerült a kérdés, hogy legfennebb hány éle lehet egy n pontú gráfnak anélkül, hogy tartalmazna háromszöget. Sejthető volt, hogy a legtöbb éle annak a páros gráfnak van, amelyben a két osztály pontszáma legfennebb 1-gyel tér el egymástól. Turán Pál e részgráfok témakörébe tartozó tételt általánosabban is bebizonyította.

15.4. definíció. Értelmezzük a $T_{n,m}$ ($n \geqslant m$) gráfot a következőképpen. Legyen $n = qm + r$ (maradékos osztás). A gráf pontjait osszuk m osztályba, r darab osztályban legyen egyenként $(q+1)$ pont, a többiekben pedig egyenként q pont. A gráfban két pont akkor és csakis akkor van összekötve, ha különböző osztályokhoz tartoznak. Továbbá *m-osztályú gráf*nak nevezzük azt a gráfot, amelynek pontjai m osztályba oszthatók úgy, hogy az egy osztályba eső pontok között nincs él. A *$T_{n,m}$ gráfot m-osztályú teljes gráf*nak nevezzük.

15.5. tétel (Turán). Ha egy n pontú G gráf nem tartalmaz $(m+1)$ pontú teljes részgráfot, akkor éleinek száma kisebb vagy egyenlő a gráf m-osztályú teljes gráf élszámánál ($T_{n,m}$). Továbbá, ha G éleinek száma pontosan egyenlő $T_{n,m}$ élszámával, akkor G izomorf $T_{n,m}$-mel. (Bizonyítás végett lásd a [6], 89. oldalt.)

FÜGGELÉKEK

A. FÜGGELÉK

NP-BELI PROBLÉMÁK[1]

Sok problémával kapcsolatban, amelyet e könyvben tárgyaltunk, felmerül a kérdés, hogyan tudnánk számítógéppel eldönteni, hogy például rendelkezik-e egy adott tulajdonsággal egy bizonyos gráf. Eldöntési problémának nevezzük az olyan problémákat, amikor az input egy olyan kérdés (például: „G összefüggő-e?"), amire az output „igen" vagy „nem". Jelöljük **P**-vel az eldöntési problémáknak azon osztályát, amelyek az input méretének polinomiális függvényével felülről becsülhető időben megoldhatók. Például egy gráf összefüggőségének eldöntése **P**-ben van.

„Sajnos" vannak olyan problémák, amelyekre eddig senki sem tudott polinomiális algoritmust adni. Ilyen például a Hamilton-kör probléma:

Input: G gráf. *Kérdés:* Van-e a G-ben Hamilton-kör?

Ha most egy varázsló megmondja nekünk, hogy G-ben van Hamilton-kör, és meg is mutatja, melyik az a kör, akkor bárki be tudja bizonyítani, hogy ebben a gráfban van Hamilton-kör. Ha azonban nincs a gráfban Hamilton-kör, akkor a varázsló sem tud jobbat, mint hogy minden egyes körről megmutatja nekem, hogy az nem Hamilton-kör. Ez viszont egy nagyobb gráf esetén rengeteg időbe telhet. Az olyan eldöntési problémák osztályát, amelyeknél az igenlő válasz esetén a varázsló biztosan tud olyan segítséget adni, amellyel polinom időben be tudjuk bizonyítani, hogy a válasz igen, **NP**-nek nevezzük. Pontosabban, egy eldöntési probléma akkor **NP**-beli, ha minden olyan I inputhoz, melyre a válasz igenlő, létezik egy olyan, I-től függő T „tanú", hogy egyrészt T hossza felülről becsülhető I hosszának egy polinomjával, másrészt I és T ismeretében polinom időben ellenőrizhető, hogy a válasz igenlő.

Természetesen vannak olyan problémák is, amelyeknél a nemleges választ tudjuk polinom időben bebizonyítani, ha segít a varázsló. Az ilyen problémák osztálya a **co-NP** osztály. Egyből adódik, hogy ilyen feladat például a következő.

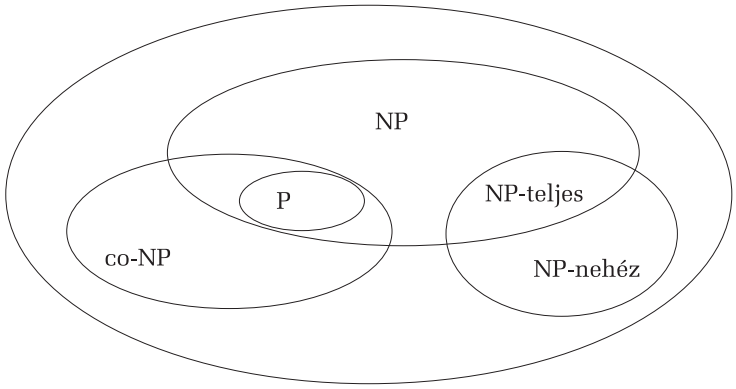
Input: G gráf. *Kérdés:* Igaz-e, hogy a G-ben nincs Hamilton-kör?

1 [6], 96–103. oldalak

Az eddigiekből nyilvánvaló, hogy **P** $\subseteq$ **NP**, hiszen **P**-beli problémák esetén polinom időben meg tudjuk határozni a választ, és ezáltal, a varázsló segítsége nélkül, be is tudjuk bizonyítani, hogy a válasz igenlő. A bonyolultságelmélet talán legérdekesebb megoldatlan kérdése, hogy itt valódi tartalmazás vagy egyenlőség áll fenn.

A következő ábrán, mely e függelékben szereplő problémaosztályok legvalószínűbb (de nem biztos) viszonyát mutatja, látható, hogy nyilván vannak problémák, amelyek **NP** $\cap$ **co-NP**-ben vannak. Ilyen például a következő:

Input: G gráf. Kérdés: Igaz-e, hogy a G gráf síkba rajzolható?



A.1. ábra. *A problémaosztályok legvalószínűbb viszonya*

Ez a probléma nyilván benne van **NP**-ben, hiszen ha a varázsló megmutatja a gráf egy síkba rajzolását, akkor könnyen ellenőrizhető. Másfelől, ha a varázsló mutat a gráfban valamelyik Kuratowski-gráffal topologikusan izomorf részgráfot, akkor a nemleges válasz is polinom időben eldönthető.

A síkbarajzolhatósági probléma (és sok más ehhez hasonló) nemcsak **NP** $\cap$ **co-NP**-ben van benne, hanem **P**-ben is. Vannak, akik azt sejtik, hogy **P** = **NP** $\cap$ **co-NP**, hiszen a *legtöbb* **NP** $\cap$ **co-NP**-beli problémáról már kiderült, hogy **P**-beli, de *egyről sem* tudta még senki belátni, hogy *bizonyítottan* ne lenne benne **P**-ben.

Tegyük most fel, hogy van egy olyan számítógépünk, amellyel egységnyi idő alatt meg tudjuk oldani a P_2 problémát. Ha ezzel a számítógéppel polinom idő alatt megoldható a P_1 probléma, akkor azt mondjuk, hogy P_1 polinomiálisan visszavezethető P_2-re. Ha $P_2 \in$ **P**, akkor természetesen $P_1 \in$ **P**.

Egy problémát **NP-nehéz**nek nevezünk, ha minden **NP**-beli probléma polinomiálisan visszavezethető rá. Ha ez a probléma maga is eleme **NP**-nek, akkor **NP-teljes**nek nevezzük. Ha egyetlen **NP-teljes** problémát megoldanánk polinom időben, akkor minden **NP**-beli probléma megoldható lenne polinom időben. Cook és Levin bebizonyították, hogy létezik **NP-teljes** probléma. Később sikerült belátni, hogy a Hamilton-kör probléma **NP-teljes**.

Az előbbi fejtegetéssel összhangban egy problémáról úgy is bebizonyítható, hogy **NP-teljes**, hogy visszavezetjük egy „elismerten" **NP-teljes** problémára. Például viszonylag könnyen belátható, hogy a „van-e Hamilton-út egy gráfban?" probléma is **NP-teljes**. Hogyan? Egyfelől triviális, hogy a probléma eleme **NP**-nek. Másfelől nem nehéz kimutatni, hogy a Hamilton-körre vonatkozó eldöntési probléma visszavezethető a Hamilton-út problémára.

Miért lehet hasznos mindez? Ha van egy problémánk, amire nem találunk polinom idejű algoritmust, viszont nyilván **NP**-ben van, akkor megpróbálhatjuk visszavezetni egy **NP-teljes** problémára. Ha ez sikerül, akkor nem leszünk felette szomorúak előbbi sikertelenségünk miatt, hiszen amennyiben megoldottuk volna az eredeti problémánkat, „automatikusan" több száz olyan problémát is megoldottunk volna, amelyeken

A.1. táblázat. *Közismert polinomrendű, illetve NP-teljes problémák*

P-beli problémák	**NP-teljes** problémák
Van-e G-ben legalább k darab független él?	Van-e G-ben legalább k darab független pont?
Lefogható-e G minden pontja legfeljebb k éllel?	Lefogható-e G minden éle legfeljebb k éllel?
Van-e G-ben legfeljebb k hosszúságú út?	Van-e G-ben legalább k hosszúságú út?
Van-e G-ben legfeljebb k hosszúságú kör?	Van-e G-ben legalább k hosszúságú kör?
Kiszínezhetőek-e G pontjai legfeljebb 2 színnel?	Kiszínezhetőek-e G pontjai legfeljebb k ($k \geqslant 3$) színnel?
Van-e egy hálózatban legfeljebb k értékű vágás?	Van-e egy hálózatban legalább k értékű vágás?

számos világhírű matematikus és informatikus évtizedek óta rágódik „eredménytelenül".

Általában igaz, hogy ha egy probléma **NP-teljes**, akkor egy általánosabb **NP**-beli problémának is **NP-teljes**nek kell lennie. Forditva viszont nem igaz. Például **NP-teljes** probléma annak eldöntése, hogy egy általános gráf kiszínezhető-e 5 színnel. Ezzel szemben a síkba rajzolható (speciális eset) gráfokról tudjuk, hogy mindig kiszínezhetők 5 színnel. (Lásd a 14.3. alfejezetet.)

B. FÜGGELÉK

ALGORITMUSTERVEZÉSI STRATÉGIÁK[1]

Számos gráfalgoritmus valamelyik nevezetes programozási technikát (algoritmustervezési stratégiát) alkalmazza. Egyes esetekben, könyvünk előző fejezeteiben, nem adtunk részletes leírást a szóban forgó algoritmusokról, csak felvázoltuk az irányelveket és megjelöltük az alkalmazandó programozási technikát. E függelék keretein belül röviden áttekintjük a négy legismertebb algoritmustervezési stratégiát (visszalépéses keresés, oszd meg és uralkodj, mohó, valamint dinamikus programozás). A függelék célja segíteni az olvasónak jobban megérteni az illető technikákra épülő algoritmusokat, illetve megírni azokat, amelyeket csak felvázoltunk, vagy amelyekre csak utaltunk. Számos gráfelméleti feladat számítógépes megoldásában is jó hasznát veheti az olvasó e függelék anyagának.

A visszalépéses keresés módszere (backtracking)

Néhány gráffeladat, amely ezzel a módszerrel oldható meg:
- Az összes zárt Euler-vonal megkeresése egy gráfban.
- Az összes Hamilton-kör megkeresése egy gráfban.
- Az utazó ügynök feladat.
- Egy gráf kiszínezése négy színnel az összes lehetséges módon.

A backtracking módszer alapvetően a következő feladatot oldja meg:

Egy ismert halmazsorozatból $(A_1, A_2, \ldots, A_n)$ az összes lehetséges módon összeválogat elemeket, halmazonként egy-egy elemet, szem előtt tartva, hogy az összeválogatott sorozatok (vektorok) elemei között teljesülniük kell bizonyos feltételeknek.

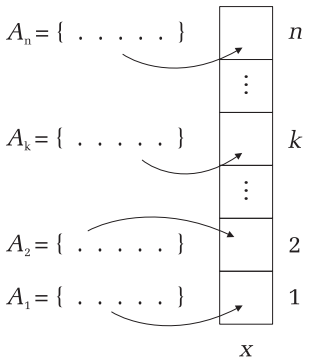
Ezek az összeválogatott elemekből álló sorozatok lesznek a feladat megoldásai. Gyakran egyszerűen arról van szó, hogy meg kell keresnünk az $A_1 \times A_2 \times \ldots \times A_n$ Descartes-szorzat azon elemeit, amelyeket bizonyos *belső tulajdonságok* jellemeznek, és így megoldásnak számítanak.

1 Részletek a szerző „Algoritmusok felülnézetből" című könyvéből

Jegyezzük meg (mint fontos következtetést ennél a pontnál), hogy *egy „backtracking-feladat" megoldásai vektor alakúak*.

Egy másik fontos szempont az, hogy az $A_1, A_2, \ldots, A_n$ halmazokat általában nem kell tárolnunk a számítógép memóriájában, a backtracking algoritmus generálja ezeket. Erre a célra a módszer felhasznál egy egydimenziós tömböt, amelyet x-szel fogunk jelölni. Az A_i halmaz elemei rendre bekerülnek az $x[i]$ tömbelembe.

Mindezt szemléletesen mutatja be az alábbi ábra, amelyet egy backtracking feladat alapábrájának is nevezhetnénk:



B.1. ábra.

Hogyan helyezi el a backtracking az $x[i]$ tömbelembe rendre az A_i halmaz egy-egy elemét? Egymás után állítja elő őket, egyikből a másikat. Ez azt jelenti, hogy az $A_1, A_2, \ldots, A_n$ halmazok nem lehetnek akármilyenek, elemeik valamilyen szabályosság szerint kell hogy kövessék egymást.

Az eddig kifejtettek alapján elmondhatjuk, hogy backtracking feladatként felfogni egy feladatot azt jelenti, hogy

1. a feladat megoldásait egy vektorban kódoljuk,
2. azonosítjuk az $A_1, A_2, \ldots, A_n$ halmazokat, ahonnan a megoldásvektorok elemei származnak.

Mikor oldható meg egy feladat a backtracking módszer segítségével?

1. A feladat megoldásai vektor formájában kódolhatók.
2. Azonosítani tudjuk az $A_1, A_2, \ldots, A_n$ halmazokat, ahonnan a megoldásvektorok elemei származnak, és ezek elemei valamilyen szabályosság szerint követik egymást.

Ugyanakkor figyelembe kell vennünk, hogy a backtracking algoritmusok általában exponenciális bonyolultságúak. Például a backtracking

legprimitívebb változata előállítja az $A_1 \times A_2 \times \ldots \times A_n$ Descartes-szorzat összes $n_1 \times n_2 \times \ldots \times n_n$ elemét (az n királynő feladata esetén ez n_n elemet jelent), és kiválasztja közülük a megoldásokat. Bár a módszer filozófiája az, hogy megpróbálja csökkenteni a generálandó megoldások számát, gyakran az optimalizálások után is megmarad az exponenciális bonyolultság. Ezért a backtrackinget általában csak akkor alkalmazzuk, ha nincs más választásunk.

A backtracking módszer stratégiája jobban nyomon követhető, ha a feladatot fa formájában ábrázoljuk.

A gyökértől a levelekhez vezető utak az $A_1 \times A_2 \times \ldots \times A_n$ Descartes-szorzat elemeit ábrázolják. Az első szintre az $x[1]$ tömbelembe n_1-féleképpen választhatunk elemet az A_1 halmazból, tehát a fa gyökeréből n_1 első szinti fiúcsomópont ágazik le, amelyekhez az A_1 halmaz elemeit rendeljük. Minden első szinti választáshoz a második szintre, az $x[2]$-be n_2-féleképpen választhatunk elemet az A_2 halmazból. A fán ez abban tükröződik, hogy minden első szinti csomópontnak n_2 fia lesz a második szinten, amelyekhez az A_2 halmaz elemeit rendeljük. Ez összesen $n_1 \times n_2$ második szinti csomópontot eredményez. Végül az n-edik szinten a fának $n_1 \times n_2 \times \ldots \times n_n$ csomópontja (levele) lesz.

Általános szabályként megjegyezhető, hogy ha a csomópont a fa k-adik szintjén található, és az apacsomópontjának ez az i-edik fia, akkor az A_k halmaz i-edik elemét rendeljük hozzá. Mivel a fa bármely csomópontjához a gyökérből pontosan egy út vezet, ezért elmondható, hogy minden csomópont képviseli ezt az utat. A levelek mindegyike a Descartes szorzat valamelyik elemét képviseli – azt, amelyiket a gyökérből az illető levélhez vezető út ábrázol.

Hogyan jelennek meg a fában a feladat megoldásvektorai? Ha a megoldások a Descartes-szorzat adott tulajdonságú elemei, akkor ezeket azok az utak ábrázolják, amelyek a gyökértől egy-egy levélhez vezetnek. Nevezzük ezeket *megoldásutak*nak vagy *megoldásá*gaknak, a hozzájuk tartozó leveleket pedig *megoldáslevelek*nek. Egyes feladatokban a megoldásokat nem feltétlenül gyökér–levél utak ábrázolják, hanem a gyökértől adott tulajdonságú csomópontokhoz vezető utak. Tehát, általános esetben, *megoldáscsomópontok*ról beszélünk. Ezek után nem nehéz átlátni, miért nevezzük a feladathoz rendelt fát a *megoldások teré*nek.

Mindezeket figyelembe véve egy backtracking feladat az alábbi módon is megfogalmazható: keressük meg a feladathoz rendelhető fa megoldáscsomópontjait, illetve építsük fel azokat a megoldásutakat, amelyek a gyökérből ezekhez vezetnek, és amelyekhez, természetesen, éppen a

megoldásvektorok vannak rendelve. Mindezt az x tömbben fogjuk megvalósítani, amelyet – amint látni fogjuk – úgy használunk, mint egy vermet[2].

Indulunk a gyökérből. Sorra megvizsgáljuk a gyökér utáni első szint csomópontjait, a gyökér fiait, és fellépünk ahhoz, amelyről elsőként találjuk azt, hogy potenciálisan megoldáscsomóponthoz vezet. Itt hasonlóképpen járunk el: megvizsgáljuk a fiait (a belőle ágazó második szint csomópontjait), és ahogy olyat találunk, amely ígéretes irányba vezet, oda lépünk. Így járunk el egészen addig, míg olyan csomóponthoz nem jutunk, amelynek nyilvánvalóan egyetlen fia sem vezet megoldáshoz. Ilyenkor visszalépünk az előző szinti apacsomóponthoz, ahol folytatjuk a keresést, további ígéretes fiak után kutatva. Ha találunk egy újabb csomópontot, amely potenciálisan megoldáscsomóponthoz vezet, akkor újból fellépünk a következő szintre. Ha egy csomópontból már minden irányt ellenőriztünk – az ígéreteseket be is járva –, akkor innen is visszalépünk. Az algoritmus akkor ér véget, amikor a gyökérből ágazó összes első szinti csomóponttal foglalkoztunk már. Valahányszor megoldáscsomópontot találunk, a gyökérből hozzá vezető út egy megoldásvektornak felel meg.

A fa ily módon történő bejárását mélységi bejárásnak nevezzük (lásd a bevezető fejezetet). Vegyük észre, hogy nem jártuk be a teljes fát, csak azt a részfát, amely potenciálisan tartalmazza a megoldásokat. Ezt nevezhetnénk a fa *élő részé*nek, a többi *száraz ág* a feladatra nézve. Nyilván, minél jobban sikerül leszűkíteni a bejárt részfát, annál hatékonyabb az algoritmusunk. Ez attól függ, hogy mennyire hatékonyan tudjuk meghatározni, hogy egy bizonyos irány ígéretes-e.

De mit is jelent az, hogy egy irány ígéretes vagy hogy az illető fiúcsomópont potenciálisan megoldáscsomóponthoz vezet? Először is kövessük nyomon, miként alakul át a keresés alatt az aktuális út (a gyökérből az aktuális csomóponthoz vezető út) megoldásutakká. Amikor felfele lépünk a fában, új csomópont kerül az aktuális út végére. Visszalépéskor egy bizonyos csomópont eltűnik a végéről. Nos, egy csomópont akkor vezet potenciálisan megoldáscsomópont felé, ha ígéretesen bővíti a gyökérből az aktuális csomóponthoz vezető utat. Hogyan értendő ez? Emlékezzünk, hogy egy út attól megoldásút, hogy a csomópontjaihoz

2 A verem egy lineáris adatszerkezet, amelynek ugyanazon végén (a „tetején") történik mind a betevés, mind a kivevés. Ebből kifolyólag az utoljára betett elem lesz az elsőnek kivett. (**L**ast **I**n **F**irst **O**ut)

rendelt elemek megoldásvektort alkotnak. Más szóval, az illető úthoz rendelt elemek rendelkeznek a megoldásvektorokat azonosító belső tulajdonsággal. A backtracking módszer kulcsötlete az, hogy amennyiben a szóban forgó fiúcsomóponthoz rendelt elem máris nem fér össze – a megoldásvektorokat azonosító belső tulajdonság szempontjából – a már az aktuális úton lévő csomópontokhoz rendelt elemekkel, akkor értelmetlen az illető csomópontra építve keresni az újabb megoldásokat. Tehát, egy megvizsgált fiúcsomópont akkor ígéretes, ha – az imént bemutatott értelemben – nem kerül „konfliktusba" az apacsomópontjához vezető aktuális út már ígéretesnek talált csomópontjaival.

Az a feltétel, amelynek alapján eldönthető, hogy az illető csomóponttal ígéretesen bővíthető-e az aktuális út, a feladat megoldásait azonosító belső tulajdonságok alapján határozható meg. Amint megfigyelhettük, ennek a feltételnek kulcsszerepe van a backtrackingben, hiszen alapvetően ez határozza meg, mekkora lesz a bejárt részfa.

És most lássuk, miként valósítható meg a fán bemutatott algoritmus az x tömbben. Nem fogjuk felépíteni a fát a számítógép memóriájában, csak szimuláljuk a bejárását az x tömb segítségével.

Az algoritmus tanulmányozása a következő fontos észrevételekhez vezet:

1. Az x tömböt úgy kell használnunk, mint egy vermet. Figyeljük meg, hogy a fa bejárásának egy adott pillanatában a tömb az aktuális úton található csomópontok képviselte elemeket tartalmazza. Amikor felfele lépünk a fában, a verem tetejére új elem kerül. Visszalépéskor eltűnik a verem tetején lévő elem.
2. A fa bejárásakor minden érintett csomópontban alapvetően ugyanúgy kellett eljárnunk: sorra megvizsgáltuk a fiait, és valahányszor ígéretest találtunk, felléptünk hozzá, ahol hasonlóképpen jártunk el. Természetesen minden csomópontban ellenőriznünk kell, hogy nem képvisel-e éppen megoldást.
3. A fának a fenti algoritmus szerinti bejárását úgy is felfoghatjuk, hogy egy csomóponthoz tartozó részfa bejárását visszavezetjük ígéretes fiú-részfáinak bejárására (egy részfa akkor ígéretes, ha potenciálisan tartalmaz megoldáscsomópontot).

Mindhárom észrevétel azt sugallja, hogy az algoritmust rekurzívan valósítsuk meg. Ez egy olyan eljárás megírását feltételezi, amely mindig az aktuális csomóponton dolgozik: sorra ellenőrzi a fiait, és valahányszor ígéretest talál, meghívja magát az illető fiúcsomópontra. A visszalépés

automatikusan történik a rekurzió mechanizmusa által, amikor az aktuális csomópont összes fiának a vizsgálata befejeződött. Ha az a részfa, amely potenciálisan tartalmazza a megoldásokat, véges, akkor nem kell tartanunk a végtelen rekurzív hívásoktól: mivel a leveleknek nincsenek ígéretes fiaik, ezekből az algoritmus nem hívja meg önmagát.

A kérdés már csak az, hogy miként tudjuk szimulálni mindezt az x tömbben.

Mivel az azonos szintű csomópontok fiaihoz ugyanazok az elemek vannak rendelve, a rekurzív eljárásnak mint paraméterre az x tömbön kívül (amit *cím szerint* kap meg) alapvetően csak *az aktuális szint értékére* van még szüksége. Tegyük fel, hogy az aktuális csomópont a k-adik szinten található. Ebben az esetben a veremben, azaz az x tömb első k elemében, az aktuális úton lévő csomópontokhoz rendelt értékek találhatók. Mivel minden k-adik szinti csomópont fiaihoz az A_{k+1} halmaz elemei vannak rendelve, ezért a következőképpen járhatunk el: az x tömb $(k+1)$-edik szintjére sorra előállítjuk az A_{k+1} halmaz elemeit, és valahányszor ígéretest találunk, meghívjuk az eljárást rekurzívan a $(k+1)$-edik szintre. Természetesen emellett azt is ellenőriznünk kell, hogy nem tartunk-e megoldáscsomópontnál, azaz az x tömbben nem épült-e fel egy megoldásvektor, amelyet ki kell íratnunk. Hosszas vajúdás után, íme a visszalépéses keresést implementáló rekurzív eljárás:

```
eljárás BACKTRACKING(x[],k)
   ha megoldás(x,k) akkor
      kiír(x,k)
   vége ha
   minden i ← 1,n_{k+1} végezd
      * előállítjuk x[k+1]-ben az A_{k+1} halmaz i-edik elemét
      ha ígéretes(x,k+1) akkor
         BACKTRACKING(x,k+1)
      vége ha
   vége minden
vége BACKTRACKING
```

A legtöbb feladat esetében (ide tartozik az n királynő feladata is), ha $(x_1, x_2, \ldots, x_k)$ megoldásvektor, akkor nem bővíthető ígéretesen újabb megoldásvektorrá. Más szóval kizárt, hogy egyik megoldásvektor része legyen egy másiknak. Ilyenkor a **minden** ciklust tehetjük a **ha** utasítás **különben** ágára. Ha nem így járnánk el, és a fa valamely levele megoldáscsomópont lenne, akkor a fenti eljárás még generálná e levél

(képzeletbeli) fiait is, amelyek között persze egy ígéretest sem találna, és így ezt követően visszalépne.

```
eljárás BACKTRACKING(x[],k)
   ha megoldás(x,k) akkor
      kiír(x, k)
   különben
      minden i ← 1,n_{k+1} végezd
         * előállítjuk x[k+1]-ben az A_{k+1} halmaz
         * i-edik elemét
         ha ígéretes(x,k+1) akkor
            BACKTRACKING}(x,k+1)
         vége ha
      vége minden
   vége ha
vége BACKTRACKING
```

A backtracking algoritmus kulcsfüggvénye, az `ígéretes(x,k+1)` függvényhívás alkalmával ellenőrzi, hogy az `x[k+1]` rekeszbe elhelyezett elem ígéretesen összefér-e, a megoldásvektorokat azonosító belső tulajdonság szempontjából, a már ígéretesnek elfogadott és az `x[1]`, `x[2]`, ..., `x[k]` rekeszekben található elemekkel.

A `megoldás(x,k)` függvényhívás ellenőrzi, hogy az `x` tömb első `k` elemében megoldásvektor épült-e ki. Ez a függvény akkor lesz eredményes, ha figyelembe veszi, hogy amikor hozzá kerül az ellenőrzésre váró `x[1..k]` tömbszakasz, már átment az ígéretes függvény kezén. A `kiír(x,k)` eljáráshívás kiírja az `x` tömb első `k` helyén felépült megoldásvektort.

„Oszd meg és uralkodj" módszer (divide et impera)

Milyen feladatok oldhatók meg a *divide et impera* („oszd meg és uralkodj") módszer segítségével? Azok a feladatok, amelyek visszavezethetők, más szóval lebonthatók, két vagy több hasonló, de egyszerűbb (kisebb méretű) részfeladatra. Ezen részfeladatok hasonlóak lévén az eredeti feladathoz, maguk is visszavezethetők további hasonló, de még egyszerűbb részfeladatokra. Addig járunk így el, míg banálisan egyszerű (triviális) részfeladatokhoz jutunk, amelyek tovább nem bonthatók. Megint csak egy fát látunk magunk előtt. A gyökérben található az eredeti feladat. A fa első szintjén helyezkednek el azok a részfeladatok,

amelyekre első lépésből bontható le a feladat. Mely részfeladatok kerülnek a második szintre? Nyilván azok, amelyek közvetlenül adódnak az első szinti részfeladatok lebontásából. Végül a fa leveleibe a lebontásból adódó triviális részfeladatok kerülnek.

Mivel az imént bemutatott fa minden csomópontjában hasonló feladatok találhatók, a részfeladatok általános alakjáról beszélhetünk. Az eredeti feladat úgy tekinthető, mint az általános feladat határesete, abban az értelemben, hogy méretben a legnagyobb. A triviális részfeladatok a másik határesetként foghatók fel, hiszen méretben a lehető legkisebbek (tovább nem darabolhatók).

Egy divide et impera algoritmus rekurzívan közelíti meg a feladatot. Ebből adódik eleganciája. Az algoritmust megvalósító rekurzív eljárást (vagy függvényt) nyilván az általános feladatra kell megírnunk és az eredeti feladatra meghívnunk. Az eljárásnak (függvénynek) különbséget kell tennie triviális és nem triviális részfeladat között. Más szóval két forgatókönyvet tartalmaz:

- Triviális esetben fel kell vállalnia az illető részfeladat teljes megoldását. Ez fog a rekurzió megállási feltételeként szolgálni.
- Ha nem triviális a feladat, megoldását, rekurzív hívások által, vissza kell vezetnie a részfeladatai megoldására. Ez három lépésben valósítható meg:
  1. Meghatározzuk a részfeladatokat, amelyekre az általános feladat lebontható („Oszd meg . . . ").
  2. Rekurzív hívások által megoldjuk az így nyert részfeladatokat („. . . és uralkodj").
  3. A részfeladatok megoldásaiból felépítjük az általános feladat megoldását.

Íme egy divide et impera algoritmus váza:

```
eljárás DIVIDE_et_IMPERA(az_általános_feladat_paraméterei)
   ha triviális akkor
      * oldd meg
   különben
      * határozd meg a részfeladatait
      * rekurzív hívások által oldd meg ezeket
      * építsd fel a megoldást
   vége ha
vége DIVIDE_et_IMPERA
```

Hogyan kerül bejárásra egy divide et impera algoritmusban a feladat lebontásából adódó fastruktúra? Nyilván mélységében, hiszen mindenik

csomópont esetén először sorra megoldjuk a fiú-részfák által (azon részfák, amelyeknek gyökerei a csomópont fiai) képviselt részfeladatokat, majd ezt követően az illető csomóponthoz kapcsolódó feladatot.

Mi történik, ha a feladat lebontásakor a fa különböző ágain azonos részfeladatok jelennek meg? Példaként tegyük fel, hogy n elem k-adrendű kombinációinak számát szeretnénk kiszámítani divide et impera algoritmussal a $C(n,k) = C(n-1,k)+C(n-1,k-1)$ képlet alapján. Első lépésben a $C(n,k)$ kiszámítása a $C(n-1,k)$ és a $C(n-1,k-1)$ kiszámítására vezetődik vissza. A második lépében megjelenő részfeladatok pedig a következők lesznek: $C(n-2,k)$ és $C(n-2,k-1)$, valamint $C(n-2,k-1)$ és $C(n-2,k-2)$. Íme, máris megjelent két azonos részfeladat. Mivel a divide et impera a részfeladatokat, amelyekre egy feladat visszavezetődik, egymástól függetlenül oldja meg, az azonos részfeladatok újra és újra megoldásra kerülnek, ahányszor csak találkozunk velük a fa bejárása közben. Ilyen feladatok megoldására előnytelen divide et impera algoritmust írni. Például megtörténhet, hogy a fa összes csomópontjainak száma exponenciálisan függ a bemenet méretétől, bár a különböző részfeladatokat képviselőinek száma csak polinomiális érték.

Hogyan közelítsünk meg egy divide et impera feladatot?

Ha tisztázzuk az alábbi kérdéseket, akkor ezekből az algoritmus természetszerűen fog adódni:

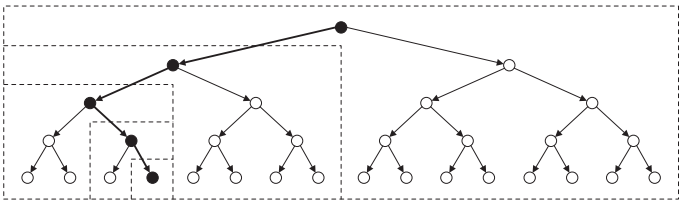
- Hogyan vezethető vissza a feladat hasonló, de egyszerűbb részfeladatokra?
- Mi az általános feladat alakja? Mik a paraméterei? Ezek lesznek az eljárás formális paraméterei!
- Milyen paraméterértékekre kapjuk az eredeti feladatot? Ezekre hívjuk meg kezdetben az eljárást!
- Mi a trivialitás feltétele? Hogyan oldhatók meg a triviális részfeladatok? Nem triviális esetben mik az általános feladat részfeladatainak a paraméterei? Ezek lesznek a rekurzív hívások aktuális paraméterei!
- Hogyan építhető fel a részfeladatok megoldásaiból az általános feladat megoldása?

Mohó módszer (greedy)

Néhány gráffeladat, amely mohó módszerrel oldható meg:
- A minimális súlyú feszítőfa meghatározása (Kruskal- és Prim-algoritmusok).
- Találni egy jó színezést egy gráfnak.
- Felső becslést találni az utazó ügynök feladatára.
- Egy pontból az összes többihez vezető legrövidebb utak megkeresése egy gráfban (Dijkstra algoritmusa).

A greedy módszert optimalizálási feladatok megoldására használjuk. A greedy feladatokban általában arról van szó, hogy egy döntéssorozattal kell meghatározni az optimális megoldást. Egyes esetekben egyszerűen egy optimális sorrendet kell megtalálni, máskor egy halmaznak valamilyen szempontból vett optimális részhalmazát stb. Fontos megjegyezni azonban, hogy a második esetben is általában az vezet el az optimális részhalmazhoz, hogy optimális sorrendben vizsgáljuk meg a halmaz elemeit.

Újra egy fa elevenedik meg előttünk, éspedig egy döntési fa, amely a feladathoz rendelhető.



B.2. ábra.

Minden döntéssel a feladat kisebb és kisebb méretű hasonló feladatokká redukálódik, amelyek az eredeti feladat egymásba ágyazott részfeladatainak tekinthetők (ezt szemléltettük a szaggatott vonalú keretekkel). Az eredeti fa az első döntéssel leszűkül valamelyik fiú-részfájára (azzal, hogy választunk, mintha lemetszenénk a fáról a többi fiú-részfát), majd a következő döntéssel ennek valamelyik fiú-részfájára, és így tovább, míg már csak egy levél marad belőle.

Az optimális döntéssorozatot, amely elvezet az optimális megoldáshoz, a fa valamelyik gyökér–levél útja képviseli. Nevezzük el ezt az utat *optimális* útnak, az illető levelet pedig optimális levélnek. Ezek után úgy is megfogalmazható egy greedy feladat, hogy keressük meg a feladathoz rendelhető döntési fa optimális gyökér–levél útját.

A greedy módszer stratégiája

A greedy algoritmusok filozófiája nagyon egyszerű, és azokra az emberekre emlékeztet, akik a mának élnek. Neve jelentésével összhangban (mohó), mindig az adott lépésben optimálisnak látszó (legígéretesebb) döntést hozza, nem számolva az esetleges hosszú távú következményekkel. Úgy gondolkodik, hogy a lokális optimum majd globális optimumhoz vezet. Amennyiben ez nem igaz az illető feladatra, vagy nem talál megoldást, vagy nem találja meg az optimálisat (legfennebb véletlenül bizonyos sajátos esetekben).

Megjegyzések: Mivel a greedy módszer alapgondolata, miszerint a lokális optimum globális optimumhoz vezet, általánosságban nem igaz, ezért egy teljes megoldáshoz az is hozzátartozik, hogy bizonyítjuk, hogy az illető feladatra viszont igaz. Ennek ellenére – ha beérjük kevesebbel is, mint az optimális megoldás – célszerű lehet minden olyan esetben alkalmazni (még ha nem is bizonyítható a nyert algoritmus helyessége), amikor minden más megközelítés túl bonyolult vagy túl nagy időigényű algoritmust eredményezne.

Számos greedy feladat esetében az alábbi helyzet ismerhető fel: Létezik egy úgynevezett „kandidátusok halmaza" (az elemei arra kandidálnak, hogy az optimális megoldás részét képezzék) és egy bizonyos „célfüggvény", amely a kandidátushalmaz részhalmazainak halmazán van értelmezve. A feladat abból áll, hogy a kandidátushalmaz adott tulajdonságú részhalmazai közül meg kell határozni azt a részhalmazt, amelyik optimalizálja a célfüggvényt. Íme a greedy algoritmus erre a helyzetre:

```
eljárás GREEDY(K)
   S ← Φ
   amíg (nem megoldás(S) ÉS K ≠ Φ) végezd
      x ← legígéretesebb(K)                          (1)
      K ← K \ {x}                                    (2)
      ha bővíthető(S,x) akkor
         S ← S ∪ {x}                                 (3)
      vége ha
   vége amíg
   ha megoldás(S) akkor
      kiír(S)
   különben
      kiír: "Nincs megoldás"
   vége ha
vége GREEDY
```

Ahol
- `K` – a kandidátusok halmaza
- `S` – ebben a halmazban építjük fel a megoldást
- `legígéretesebb(K)` – a mohó döntés alapján kiválasztja a kandidátushalmaznak az adott pillanatban legígéretesebbnek tűnő elemét. Mivel ennek a függvénynek a szerepe a célfüggvény optimalizálása, ezért ez utóbbiból következtethető ki. A `legígéretesebb` függvény a lokális optimumot biztosítja, a célfüggvény pedig a globálist.
- `bővíthető(S,x)` – Ellenőrzi, hogy az `x` elemmel bővíthető-e az `S` halmaz, azzal a kilátással, hogy végül a kívánt tulajdonságú legyen. Tehát onnan vezethető le, hogy a kandidátushalmaz mely tulajdonságú részhalmazai között keressük az optimálisat. Ha a feladat azt kéri, hogy az eredeti halmazra határozzuk meg a célfüggvény optimumát (lásd az 1. példafeladatot), akkor a bővíthető függvény elveszíti szerepét.
- `megoldás(S)` – ellenőrzi, hogy felépült-e a megoldás
- `kiír(S)` – kiírja a megoldást

A greedy stratégia kulcsgondolata az eljárás (1), (2) és (3) soraiban testesül meg:

1. mindig az adott pillanatban legígéretesebbnek látszó kandidátust választjuk (a döntési fa aktuális csomópontjának legígéretesebb fiát);
2. a kiválasztott kandidátust „örökre" eltávolítjuk a kandidátushalmazból;
3. ha a megoldás halmaz bővíthető az illető elemmel, akkor végérvényesen beletesszük, ha nem, akkor végképp lemondunk róla.

Tehát egy greedy algoritmusban nincs visszalépés (*backtrack*). Amint mondani szokás, mindent feltesz egy lapra.

Hogyan állapítható meg, hogy egy feladat megoldható-e a greedy módszerrel? Nincs általános módszer, azonban van két alapelv, amelyek ha érvényesek az illető feladatra, akkor bizonyíthatóan alkalmazható rá a greedy stratégia. Szerencsére az esetek nagy többségében ez járható út.

A mohó-választás alapelve

Emlékezzünk, hogy a greedy stratégia szerint mindig az adott lépésben legjobbnak tűnő választást hajtjuk végre, bármelyik legyen is az,

majd ezt *követően* megoldjuk azokat a részfeladatokat, amelyekre választásunk nyomán a feladat leszűkült. Ezzel összhangban az aktuális választás függhet (figyelembe tudja venni) az előző döntésektől, de nem függhet a későbbiektől, hiszen mindez még a jövő titka. Úgy is fogalmaztunk, hogy a greedy algoritmusok fentről lefele haladva, egymást követő mohó döntések által a feladatot mind kisebb és kisebb méretű feladattá redukálják. Ez azt jelenti, hogy a döntési fának egyetlen gyökér–levél útját generálják, bízva abban, hogy pontosan ez lesz az optimális.

Ezek után a mohó-választás alapelve a következőképpen jelenthető ki:

1. A feladat *optimális* megoldása mohó-választással *kezdődik* (vagy módosítható úgy, hogy mohó-választással kezdődjön),
2. és ezen választás nyomán a feladat *hasonló* feladattá redukálódik.

Természetesen, ha a mohó-választás nyomán nyert feladat hasonló, akkor ennek az *optimális* megoldása is mohó-választással fog kezdődni (vagy módosítható, hogy azzal kezdődjön), és így tovább... A feladat optimális megoldásának ez a tulajdonsága szükséges feltétel annak bizonyításához, hogy a globális optimum felépíthető lokális optimumok (mohó döntések) sorozataként. Mi hiányzik még a teljes bizonyításhoz?

Legyen $D_1, D_2, \ldots, D_n$ a feladat optimális megoldása. Pontosabban, az az *optimális* döntéssorozat, amely a globális optimumot biztosítja. Azt szeretnénk bebizonyítani, hogy ezen döntések mindenike egyenként mohó-választás, azaz lokális optimum. Mit biztosít ebből a mohó-választás alapelve? Azt, hogy D_1 biztosan mohó-választás. Mire van még szükség ahhoz, hogy a mohó-választás alapelvéből következzen a D_2 döntés mohó volta is? Ehhez a $D_2, D_3, \ldots, D_n$ döntéssorozatnak is *optimálisnak* kell lennie, annak a részfeladatnak az optimális megoldásának, amellyé a feladat az első mohó döntés nyomán redukálódott. Általánosan: a D_i $(i = 2, n)$ döntések mohó voltának bizonyításához további szükséges feltétel, hogy a $D_i, D_{i+1}, \ldots, D_n$ alakú rész-döntéssorozatok ugyancsak optimálisak legyenek. Pontosan ezt mondja ki a következő alapelv.

Az optimalitás alapelve

A feladat optimális megoldása a részfeladatok (amelyekre a feladat a mohó döntések nyomán redukálódik) optimális megoldásaiból épül

fel. Ez azt jelenti, hogy ha $D_1, D_2, \ldots, D_n$ optimális döntéssorozat, akkor ebből következik, hogy a $D_i, D_{i+1}, \ldots, D_n$ alakú rész-döntéssorozatok ugyancsak optimálisak az illető részfeladatokra nézve.

Végkövetkeztetésként leszögezhetjük, hogy ha egy feladat optimális megoldására érvényes a mohó-választás, valamint az optimalitás alapelve, akkor megoldható greedy módszerrel.

Hogyan közelítsünk meg egy greedy feladatot?

Sajnos (vagy éppen ez a szép benne) nincs recept ezt illetően. Talán ez a módszer az, amelyik, bár a legegyszerűbb, mégis a legtöbb leleményességet követeli. Különösen ahhoz kell leleményesség, hogy helyesen azonosítsuk a mohó-választást, amely a feladatot hasonló feladattá redukálja. Mindenképpen ez az első lépés. Az vezethet nyomra ebben, hogy mit is kell optimalizálni. Ezért mondtuk, hogy a legígéretesebb függvény a célfüggvényből következtethető ki. Ha az előbbiekben bemutatott algoritmus-váz alkalmazható a feladatra, akkor második lépésben a bővíthető függvényt kellene megírni. A többi függvény általában magától értetődő. A fenti algoritmus-vázzal azonban nem az volt a célunk, hogy sablont adjunk az olvasónak, inkább azt ajánljuk, hogy sajátítsa el a greedy gondolkodás szellemét, és sajátosan alkalmazza az egyes feladatokra.

Dinamikus programozás

Különösen az optimális út problémák témakörében jelenik meg a dinamikus programozás (lásd a 6. fejezetet):

- Topologikus sorrenden alapuló legrövidebb út algoritmusa.
- Kritikus út módszere tevékenységgráfban.
- Legrövidebb utak keresése egy gráf összes pontpárja között.
- Dinamikus programozásos elemek lelhetők fel a Dijkstra- és Bellman–Ford-algoritmusokban.

A bemutatott technikák között a dinamikus programozás nevet viselő programozási technika rendelkezik a legvonzóbb tulajdonságokkal. Számos olyan feladat van, amelyet bár megold „valahogy" a divide et impera is, a dinamikus programozás hatékonyan teszi ezt meg. Nem kevés azon optimalizálási feladatok száma sem, amelyeknek esetében a

greedy megközelítés nem kielégítő, míg a dinamikus programozás sikeresen alkalmazható.

A dinamikus programozást gyakran optimalizálási feladatok megoldására használjuk. Rendszerint arról van szó, hogy létezik egy célfüggvény, amelyet egy (optimális) döntéssorozat által optimalizálni kell. Minden optimalizálási feladathoz rendelhető egy gyökeres fa (fastruktúra), amelyet döntési fának fogunk nevezni. A gyökér a feladat kezdeti állapotát jelképezi, az első szinti csomópontok azokat az állapotokat, amelyekbe a feladat az első döntés nyomán kerülhet, a második szinten lévők azokat, amelyek a második döntésből adódhatnak stb. Egy csomópontnak annyi fia lesz, ahány lehetőség közül történik a választás az illető döntés alkalmával.

Két esetet különítünk el:

- *I. típusú döntési fa:* Minden döntéssel a feladat egy kisebb méretű hasonló feladattá redukálódik, amelyet az aktuális csomópont valamelyik részfája ábrázol.
- *II. típusú döntési fa:* Minden döntéssel a feladat két vagy több kisebb méretű hasonló feladatra esik szét, amelyeket az aktuális csomópont megfelelő részfái ábrázolnak.

Dönteni nem jelent mást, mint választani. Természetesen attól függően, hogy miként választunk az egyes döntések alkalmával, más-más részfeladatokhoz jutunk. Kijelenthetjük hát, hogy egy döntési fa részfái azokat a részfeladatokat képviselik, amelyekké a feladat – az egyes döntéssorozatok által – redukálódhat (1. eset), illetve széteshet (2. eset).

Tekintettel arra, hogy a részfeladatok hasonlóak, beszélhetünk ezek általános alakjáról. Általános alakon mindig egy paraméteres alakot értünk. Átlátni egy feladat szerkezetét, többek között, az alábbiak tisztázását foglalja magába:

- Mi a részfeladatok általános alakja?
- Milyen paraméterek írják ezt le?
- Mely paraméterértékekre kapjuk az általános feladat határeseteiként az eredeti feladatot, illetve a triviális részfeladatokat?

Az összevont döntési fa

Bár a döntési fa csomópontjainak száma exponenciálisan függ az optimális döntéssorozat döntéseinek számától, gyakran megtörténik, hogy számos identikus csomópontot tartalmaz, amelyek nyilván azonos

állapotokat ábrázolnak, és amelyeket természetesen ugyanazok a paraméterértékek jellemeznek. Ez a helyzet akkor alakul ki, ha különböző rész-döntéssorozatok révén a feladat ugyanazon részfeladattá redukálódik, vagy lebontásakor identikus részfeladatok jelennek meg. Csúsztassuk egymásra a fa identikus állapotait képviselő csomópontokat. Az így kapott adatszerkezetet *összevont döntési fá*nak fogjuk nevezni. Az összevont döntési fa már nem fastruktúra, hanem egy irányított gráf. Az összevont fának pontosan annyi csomópontja lesz, ahány különböző állapotba kerülhet a feladat (ez a szám rendszerint csak polinom függvénye az optimális megoldáshoz vezető döntések számának).

Az optimalitás alapelve

A dinamikus programozás az optimalitás alapelvére épül. Úgy is fogalmazhatnánk, hogy ennek az alapelvnek az implementálása. Az optimalitás alapelve a következőképpen fogalmazható meg: *az optimális megoldás optimális részmegoldásokból épül fel.* Más szóval a feladat optimális megoldása felépíthető a részfeladatok optimális megoldásaiból. Ezt a stratégiát követi a dinamikus programozás: kiindul a triviális részfeladatok optimális megoldásaiból, és felépíti az egyre bonyolultabb részfeladatok optimális megoldásait, végül az eredeti feladatét. Ezért is szokás azt mondani, hogy az egyszerűtől halad a bonyolult fele, vagy hogy lentről felfele oldja meg a feladatot.

A dinamikus programozás egy másik jellemvonása, hogy nyilvántartja a már megoldott részfeladatok *optimális* megoldásait képviselő optimumértékeket (az optimalizálandó célfüggvény optimumértékeit az illető részfeladatokra vonatkozóan). Erre a célra rendszerint tömböt használunk, amelyet c-vel fogunk jelölni. Ez a tömb attól függően lesz egy-, két- vagy többdimenziós, hogy a részfeladatok általános alakját hány paraméter írja le. A tömb felhasznált elemeinek a száma természetesen azonos az összevont döntési fa csomópontjainak a számával, azaz az egymástól különböző részfeladatok számával.

Az optimalitás alapelve konkrétan egy rekurzív képletben testesül meg, amely leírja matematikailag, hogy miként épülnek fel az egyszerűbb részfeladatok optimális megoldásából az egyre bonyolultabbak optimális megoldásai. Nyilván egy olyan képletről van szó, amelybe bele lett építve az optimális döntéshozás módja. A rekurzív képlet alapvetően a c tömb elemeire van megfogalmazva, tehát a részfeladatok optimumértékeivel dolgozik.

Hogyan tükröződik az optimalitás alapelvének implementálása a döntési fán, az összevont döntési fán, illetve a részfeladatok optimumértékeit tároló tömbben?

1. Ha a növekvő döntési fa koronáján identikus állapotokat képviselő csomópontok jelennek meg, a fa csak azon ág irányába fog tovább növekedni, amelyik az illető részfeladat optimális megoldását képviseli.
2. A rekurzív képlet diktálta sorrendben oly módon messük meg az összevont döntési fa csomópontjait, hogy mindenikhez egyetlen út vezessen, mégpedig az, amelyik az optimális megoldást ábrázolja.
3. Az algoritmus magva alapvetően a c tömb megfelelő elemeinek a rekurzív képletből adódó stratégia szerinti feltöltését jelenti. Bár a c tömb egy az egyben csak a részfeladatok optimumértékeit tárolja, elegendő információt tartalmaz az optimális döntéssorozat rekonstruálásához. Gyakran célszerű, hogy a c tömb feltöltésekor magukat az optimális választásokat is eltároljuk valamilyen módon. Ez megkönnyítheti, sőt felgyorsíthatja az optimális döntéssorozat rekonstruálását.

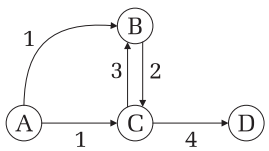
Ha egy feladatra érvényes az optimalitás alapelve, ez jelentősen lecsökkentheti az optimális megoldás megépítésének idejét, hiszen az építkezésben támaszkodhatunk kizárólag a részfeladatok *optimális* megoldásaira.

Megjegyzések:

– Figyeljünk fel arra, hogy az optimalitás alapelve nem azt mondja ki, hogy a feladat megoldásának a részfeladatok optimális megoldásaiból való *bármely* felépítése optimális lesz. Tekintsük példaként azt a feladatot, amikor egy adott összeget minimális számú ismert értékű pénzérmékkel kell kifizetni (létezik 1 értékű pénzérme, és mindenik fajta pénzérméből bármennyi van). A feladat e változatára igaz az optimalitás alapelve, de nem érvényes a mohó választás alapelve, így a leghatékonyabb megoldást a dinamikus programozás nyújtja. Szolgáljon példaként az az eset, amikor 13 eurót kell kifizetni és 1, 5, valamint 10 eurósok állnak rendelkezésünkre. A 6 euró optimális kifizetése 5 + 1, a 7 euróé pedig 5 + 1 + 1. Ennek ellenére a 13 euró optimális kifizetése nem 5 + 1 + 5 + 1 + 1, hanem a 10 + 1 + 1 + 1, amely viszont felfogható a 10 és 3, vagy a 11 és 2, valamint a 12 euró és az 1 euró optimális kifizetéseinek az összegeként.

– Azt mondtuk, hogy a dinamikus programozást elsősorban optimalizálási feladatok megoldásához használjuk. Célszerű azonban alkalmazni minden olyan esetben, amikor sok az egymásra tevődő részfeladat. Példaként tekintsük az n-edik Fibonacci-szám meghatározásának feladatát. A klasszikus algoritmus erre az, hogy az első két Fibonacci-szám összegeként meghatározzuk a harmadikat, majd a második és harmadik összegeként a negyediket, és így tovább, míg végül az $(n-2)$-edik és $(n-1)$-edik összegeként ki nem számítjuk az n-ediket. Mindez feltételezi, hogy minden lépésben eltároljuk legalább az utolsó két kiszámított értéket, hiszen ezekből épül majd fel a következő. Bár nem optimalizálási feladatról van szó, a bemutatott elemzés felszínre hozta a dinamikus programozás néhány alapvonását: a részfeladatokat az egyszerűtől a bonyolult fele haladva oldottuk meg, és mindig eltároltuk azoknak a részfeladatoknak a megoldásait, amelyekre később szükségünk volt az építkezésben. Ez a megoldás összehasonlíthatatlanul hatékonyabb, mint a divide et impera megközelítés (az n-edik Fibonacci-szám kiszámítását rekurzívan visszavezetjük az $(n-2)$-edik és az $(n-1)$-edik elem egymástól független meghatározására), ami azonos részfeladatok többszöri megoldásához vezetne.
– Amikor a feladat minden döntéssel egyetlen hasonló egyszerűbb részfeladattá redukálódott, az optimalitás alapelvének érvényessége nyilvánvaló. Nem ez a helyzet a 2. típusú döntési feladatok esetében. Megtörténhet ugyanis, hogy a részfeladatoknak – amelyekre a feladatot felbontjuk – az optimalizálása konfliktusba kerül. Legyen egy példa erre a következő (*leghosszabb út*): Adott egy város utcahálózata a terek közti közvetlen *egyirányú* utcák hossza által. Határozzuk meg két tér között azt a *leghosszabb* utat, amely *nem érinti kétszer ugyanazt a teret*.

Példa:



B.3. ábra.

Az A és D terek között az ABCD a leghosszabb út (1 + 2 + 4 = 7), de az A és B terek között van hosszabb út is, mint az optimális megoldásbeli AB közvetlen utca (1), nevezetesen az ACB (1 + 3 = 4) út.

Milyen esetben folyamodjunk a dinamikus programozáshoz?

Nincs egyértelmű válasz erre a kérdésre, mégis van néhány nyomra vezető jel.

1. A feladatra érvényes kell legyen az optimalitás alapelve, miszerint: az optimális megoldás optimális részmegoldásokból épül fel. Ez a kritérium önmagában még nem utal egyértelműen dinamikus programozásra. Megtörténhet például, hogy elegendő a greedy megközelítés, ami sokkal egyszerűbb és hatékonyabb algoritmust eredményez.
2. Ne legyen érvényes a mohó kiválasztás alapelve. Ez azt jelenti, hogy a globálisan optimális döntéssorozatban nem föltétlenül lokális optimum minden egyes döntés. Ez a feltétel kizárja a greedy megoldást, de még mindig versenyben marad a divide et impera stratégia, amely ugyancsak egyszerűbb, mint a dinamikus programozás.
3. A részfeladatok között, amelyekre a feladat lebomlik, sok az azonos. Ilyen esetben a divide et impera algoritmus nem hatékony. Mivel fentről lefele haladva egymástól függetlenül oldaná meg a részfeladatokat, rengeteget dolgozna fölöslegesen.

Ha a fenti három feltétel egyidőben teljesül egy adott feladatra, akkor ez arra mutat, hogy a feladat nagy valószínűséggel dinamikus programozással oldható meg hatékonyan.

Hogyan közelítsünk meg egy dinamikus programozás feladatot?

Az alábbi lépéssorozatot ajánljuk:

1. Meghatározzuk a részfeladatok általános alakját. Hogyan bontható le a feladat kisebb méretű hasonló részfeladatokra oly módon, hogy érvényes legyen rá az optimalitás alapelve? Az optimalitás alapelvének ellenőrzése azért is szükséges, mert ezáltal alapozzuk meg matematikailag, hogy a dinamikus programozás tényleg az optimális megoldást nyújtja. Ennél a lépésnél megállapítjuk az általános részfeladat paramétereit, és azt, hogy mely paraméterértékekre kapjuk az eredeti feladatot, illetve a triviális részfeladatokat. Ugyancsak itt tisztázzuk, hogy a részfeladatok növekedése a paraméterek növekedésével vagy csökkenésével jár kéz a kézben.

2. Eldöntjük, hol fogjuk eltárolni az egyes részfeladatok optimális megoldásait jellemző optimumértékeket. Általában annyi dimenziós tömböt használunk, ahány független paramétere van az általános részfeladatnak. Mely tömbrekeszekben kerül eltárolásra a fő feladat optimumértéke, illetve a triviális részfeladatok optimumértékei? Érdekes megfigyelni, hogy a felhasznált tömbrekeszek száma azonos a különböző részfeladatok számával. Ha a feladat nem kéri magát az optimális megoldást is, csak az ehhez tartozó optimumértéket, akkor szükségtelen lehet a teljes tömb eltárolása.
3. Megkeressük azt a rekurzív képletet, amely matematikailag leírja, miként épül fel az általános részfeladat optimális megoldását jellemző optimumérték a részfeladatok optimumértékéből. Segíthet ennél a lépésnél, ha behatároltuk, hogy melyik formájában igaz a feladatra az optimalitás alapelve.
4. A rekurzív képlet alapján – az egyszerűtől haladva a bonyolult fele – feltöltjük a tömböt a részfeladatok optimumértékeivel. Tehát a képletet nem rekurzívan alkalmazzuk, hanem lentről felfele építkezünk a segítségével. A tömböt oly módon kell bejárnunk, hogy egy adott részfeladat optimumértékének kiszámításakor rendelkezésre álljanak már azon optimumértékek, amelyekből – a képlet alapján – ez meghatározható.
5. A 4. lépésben csak a megoldás optimumértéke kerül kiszámításra. Ezek után az optimális döntéssorozat meghatározása megvalósítható lineáris időben, és aszerint történik, hogy melyik formájában volt érvényes az optimalitás alapelve a feladatra.

Ahhoz, hogy ezen optimálismegoldás-meghatározás tényleg lineáris időben történjen, egyes esetekben szükség lehet arra, hogy az optimális részmegoldások optimumértékeinek építésekor (a 4. lépésben) eltároljuk az optimális választásokat.

C. FÜGGELÉK

A MAGYAR GRÁFELMÉLETI ISKOLA KIMAGASLÓ SZEMÉLYISÉGEI[1]

A gráfelmélet a matematika, ezen belül a kombinatorika egyik fontos ága. Kialakításához jelentős mértékben hozzájárultak a magyar kombinatorikai iskola tagjai: Kőnig Dénes, Erdős Pál, Gallai Tibor, Rényi Alfréd, Lovász László, Pósa Lajos.

Kőnig Dénes (1884. szeptember 21., Budapest – 1944. október 19., Budapest) magyar matematikus, az első gráfelméleti tankönyv szerzője.

Matematikai tanulmányait 1902-től a budapesti és a göttingeni egyetemen végezte; Göttingenben nagy hatással volt rá Minkowski előadása a négyszínsejtésről. 1907-ben doktorált a Budapesti Műszaki Egyetemen, témavezetője Kürschák József volt. A professzori címet 1935-ben kapta meg. Témavezetője volt Gallai Tibornak. A gráfelmélet mellett főleg topológiával foglalkozott. A második világháború alatt segített az üldözött matematikusoknak; a nyilasok hatalomátvétele után öngyilkos lett.

Erdős Pál (1913. március 26., Budapest – 1996. szeptember 20., Varsó), a XX. század egyik legkiemelkedőbb matematikusa, az MTA tagja.

Elsősorban számelmélettel (ezen belül főleg elemi számelmélettel) és kombinatorikával, halmazelmélettel, analízissel és valószínűségszámítással foglalkozott, de a matematika szinte minden ágában alkotott. Számelméleti, illetve kombinatorikai kutatásaival ún. *magyar iskolá*t teremtett. Életében ő volt a kombinatorika kutatásának és alkalmazásának talán legnagyobb egyénisége. Meghonosította a *Ramsey-típusú jelenségek* vizsgálatát és nagy úttörője volt a véletlen módszerek alkalmazásának. Zsenialitása nemcsak bizonyításaiban mutatkozott meg, hanem nagy problémafelvető is volt: művészi szintre fejlesztette a fontos problémák meglátásának képességét.

Élete utolsó évtizedeiben valamelyest híresség vált, nemcsak Magyarországon, de az egész világon is. Ebben nemcsak hatalmas életműve

1 Wikipédia – szabad lexikon

játszott szerepet, de sajátos, örökké utazó életformája is, valamint olyan, az újságírók számára hálás téma is, mint sajátos beszédmódja *(„Erdős-nyelv"* vagy *„Erdős-szótár"):* úr (nő), rab (férfi), epszilon (gyerek), a Jordan-tételt tanulmányozza (börtönben van), meghalt (abbahagyta a matematikai kutatást), méreg (alkohol).

Tagja volt a magyar (1956), az amerikai (1979), az indiai (1988), az angol (1989) és más tudományos akadémiáknak; munkásságáért több külföldi tudományos akadémia választotta tiszteletbeli tagjává. 1500 cikke jelent meg, több mint 500 társszerzővel dolgozott, 15 egyetemnek volt a díszdoktora. 1983-ban megkapta a legmagasabb nemzetközi elismerést, a Nobel-díjjal egyenértékű Wolf-díjat. Magyarországon Kossuth-díjjal és Állami Díjjal tüntették ki.

A matematikusok máig számon tartják (félig-meddig viccesen, félig-meddig Erdősnek emléket állítandó) az ún. Erdős-számokat, ami azt fejezi ki, hogy a publikálásban mennyire kerültek közel hozzá.

Élete évszámokban

- 1913. március 26., Budapest: szülei matematikatanárok.
- 1930: Budapesten egyetemi tanulmányok kezdete; a Pázmány Péter Tudományegyetem és a Műszaki Egyetem között ingázva folytatta tanulmányait, mindkét egyetem professzorainak (Fejér Lipót, Kürschák József, Kőnig Dénes) előadásait hallgatva.
- 1934: doktorátus – Manchesterbe megy, ott 4 évet tölt.
- 1938–39: Princeton, Institute for Advanced Study.
- 1943: Purdue University.
- 1948: rövid látogatásra Magyarországra utazik.
- 1949: Atle Selberg és Erdős elemi bizonyítást adtak a prímszámtételre.
- 1951: Cole Prize (American Mathematical Society), számelméleti cikkei, de főleg a prímszámtétel elemi bizonyítása miatt.
- 1952: University of Notre Dame.
- 1954: a McCarthy-féle antikommunista kampány miatt kitiltják az USA-ból – 10 évet Európában, Izraelben és számos más helyen tölt.
- 1963. november: megkapja a vízumot az USA-ba.
- 1964. november: ettől kezdve édesanyja elkíséri utazásain.
- 1971: Szele Tibor Emlékérem (Bolyai János Matematikai Társulat).

– 1971: *Anyuka* meghal egy kanadai úton, Calgaryban. Ezt Erdős soha nem heveri ki.
– 1973: a Londoni Matematikai Társaság tiszteletbeli tagjává választotta.
– 1975: vendégprofesszor a cambridge-i Trinity College-ban.
– 1983: Wolf-díj.
– 1991: Akadémiai Aranyérem (Magyar Tudományos Akadémia).
– 1996. szeptember 20., Varsó: szívroham.

Gallai Tibor (Grünwald Tibor, 1912. július 15., Budapest – 1992. január 2., Budapest) magyar matematikus, az MTA levelező tagja volt.

Doktori fokozatát a Budapesti Műszaki Egyetemen szerezte. Témavezetője Kőnig Dénes volt. Középiskolai tanára volt Rényi Katónak és T. Sós Verának. Tanítványa volt Lovász László és Pósa Lajos is.

Kombinatorikával, gráfelmélettel foglalkozott. A gráfok faktoraira vonatkozó struktúratételt igazolt. Dilworth-tól függetlenül, sőt előbb, bebizonyította a Dilworth-tételt. Bebizonyította, hogy ha egy véges irányított gráfban minden független halmaznak legfeljebb k eleme van, akkor a gráf lefedhető k irányított úttal. Igazolta a van der Waerden-tétel többdimenziós általánosítását.

Erdős Pállal való barátsága az Anonymus-csoportban kezdődött és életük végéig tartott. Nem meglepő tehát, hogy Gallai Erdős-száma 1.

Rényi Alfréd (1921. március 20., Budapest – 1970. február 1., Budapest) magyar matematikus, akadémikus.

A budapesti egyetemen Fejér Lipót tanítványa volt, majd a II. világháború után a szegedi egyetemen Riesz Frigyesnél doktorált. Kandidátusiját 1947-ben Moszkvában szerezte. 1949-től a debreceni tudományegyetem professzora lett. 1950-ben ő hozta létre a Magyar Tudományos Akadémia gyorsan világhírűvé vált Alkalmazott Matematikai Intézetét, amelynek haláláig igazgatója volt. 1952-től az ELTE valószínűségszámítási tanszékét is vezette. 1949-től az Akadémiának levelező, 1956-tól rendes tagja lett. Emlékére az Akadémia Matematikai Kutató Intézete 1972-ben Rényi Alfréd-díjat alapított.

A matematikában a kombinatorika, a gráfelmélet és főleg a valószínűségszámítás területén ért el eredményeket. 1947-ben továbbfejlesztette a Linnyiktől származó nagy szitát, és ennek felhasználásával világraszóló eredményt ért el a Goldbach-sejtéssel kapcsolatban: igazolta, hogy minden elég nagy páros szám egy prímszám és egy olyan szám

összege, amelynek legfeljebb K prímosztója van, ahol K egy konkrét, megadható szám. Mintegy 350 cikkéből 32 Erdős Pállal közös. Jelentős volt a tudománynépszerűsítő tevékenysége is.

„A matematika lételeme volt; séta közben, síelés vagy autóvezetés közben is mindig hajlandó volt matematikai diszkusszióba belemenni. Ez a szüntelen lobogás, ez a cigarettával és feketekávéval állandóan élesztett izzás égette el fiatalon, munkaereje és kedve teljében, pillanatra sem érezve a hanyatlás fájdalmát." (Turán Pál)

Lovász László (1948. március 9., Budapest) magyar matematikus, az MTA tagja.

Elsősorban kombinatorikával és számítógép-tudománnyal foglalkozik.

Számos eredménye közül kiemelkedik a gyenge perfekt gráf sejtés igazolása, a Kneser-gráfokra vonatkozó sejtés bizonyítása, a Shannon-féle ötszögprobléma megoldása. Nevéhez fűződik a Lovász-féle lokális lemma és a Lovász-féle bázisredukciós algoritmus: a Lenstra–Lenstra–Lovász (LLL) algoritmus.

- 1962–1966: a budapesti Fazekas Mihály Fővárosi Gyakorló Gimnázium speciális matematika tagozata (osztályfőnök: Komlós Gyula, matematikatanár: Rábai Imre).
- 1966–1971: ELTE matematikus szak.
- 1969: Grünwald Géza-díj.
- 1970: kandidátus.
- 1971: matematikus diploma (ELTE).
- 1971–1975: tudományos főmunkatárs az ELTE Geometria tanszékén.
- 1975–1982: a JATE Geometria tanszékét vezeti, docensként, majd egyetemi tanárként.
- 1979: a Magyar Tudományos Akadémia levelező tagja.
- 1979: Pólya-díj (SIAM).
- 1981: Erdős Pállal és Babai Lászlóval létrehozza a Combinatorica című folyóiratot.
- 1981: Best Information Theory Paper Award (IEEE).
- 1982: az ELTE professzora, a Számítógéptudományi Tanszék vezetője.
- 1982: Fulkerson-díj.
- 1985: az MTA rendes tagja.
- 1985: Állami Díj.

- 1985: Babai Lászlóval létrehozza a Budapest Semesters in Mathematicsot.
- 1985: John von Neumann Professor, Universität Bonn.
- 1987–1993: a Princeton Egyetem professzora.
- 1987–1994: a Nemzetközi Matematikai Unió (IMU) Végrehajtó Bizottságának választott tagja.
- 1991: Szele Tibor-Emlékérem.
- 1991: Academia Europaea.
- 1992: Brouwer Medal (Holland Akadémia).
- 1993–1999: a Yale Egyetem professzora.
- 1998: a Magyar Köztársaság Érdemrend középkeresztje.
- 1999: Wolf-díj.
- 1999: Knuth-díj.
- 1999–2006: a Microsoft tudományos kutatója.
- 2001: Corvin-lánc.
- 2001: Gödel-díj.
- 2004–2006: az Abel-díj öttagú bírálóbizottságának tagja.
- 2006: a kanadai Calgary Egyetem díszdoktora.
- 2006: John von Neumann Theory Prize: Neumann János Elméleti díj.
- 2006–2008: az ELTE TTK Matematikai Intézetének igazgatója.
- 2007–2010: a Nemzetközi Matematikai Unió (IMU) Végrehajtó Bizottságának elnöke.
- 2007: a Svéd Királyi Akadémia külső tagja.
- 2007: Bolyai-díj.

Pósa Lajos (1947. december 9., Budapest)

1966 és 1971 között az ELTE matematikus szakára járt, és ott szerzett diplomát, de matematikusi munkássága sokkal korábban kezdődött. Még általános iskolás korában anyja barátja, Péter Rózsa bemutatta Erdős Pálnak, aki meghívta ebédre, és a vendéglői ebéd alatt matematikai kérdésekkel bombázta. Pósa gyorsabban végzett a problémákkal, mint a levessel, és ez mély benyomást tett Erdősre, aki maga is csodagyerek volt és sok szeretettel és hozzáértéssel támogatta a fiatalabb tehetségeket. Így született Pósa első cikke 13 éves korában, Erdős Pállal közösen (ebből következően Erdős-száma 1). Később jó néhány jelentős cikket publikált a gráfelmélet témakörében. (A Dirac-tételben szereplőnél gyengébb feltételek mellett igazolta Hamilton-kör létezését gráfban. Bebizonyította Erdős és Rényi sejtését, ami szerint n szögponton $cn\log n$ éllel rendelkező

véletlen gráf majdnem biztosan tartalmaz Hamilton-kört.) Felfedezettjei közül Erdős őrá volt legbüszkébb, mindig emlegette, szomorúan, hogy Pósa abbahagyta a kutatást, jellegzetesen erdősi terminológiával: Pósa meghalt.

Pósa 1971 és 1982 között az ELTE Analízis Tanszékén tanított, és 1983-ban szerzett egyetemi doktori címet véletlen gráfok Hamilton köreiről írt disszertációjával. 1984-től 2002-ig az ELTE Számítógép-tudományi Tanszékén dolgozott. 2002-től az MTA Rényi Alfréd Matematikai Kutatóintézet kutatója.

A '70-es évek elején került kapcsolatba a komplex matematikatanítási kísérlettel. Tagja lett a Varga Tamás körül kialakult csoportnak, mely az iskolai matematikatanítás megreformálásán munkálkodott. Pósa a gimnáziumi matematika megújításán dolgozott.

D. FÜGGELÉK

DINAMIKUS PROGRAMOZÁS ÉS „d_gráfok"[1]

A dinamikus programozás elméletét mint módszert, amellyel optimalizálási feladatokat lehet megoldani, *Richard Bellman* dolgozta ki. Első könyve, amely a dinamikus programozást tárgyalta, 1957-ben jelent meg. Haláláig (1982) számos könyvet és cikket írt erről a témáról. 1962-ben Bellman és Dreyfus kiadták az „Applied Dynamic Programming" című könyvet, amelyben felhívták a figyelmet arra, hogy a dinamikus programozási feladatok megfogalmazhatók gráfkeresési feladatokként is. Ezt az állítást később számos cikk tárgyalta. Például Georgescu és Ionescu bevezették a PD gráf fogalmát.

Ebben a cikkben be fogunk mutatni egy speciális gráfot, melyet *d_gráf*nak nevezünk (*division graph*). Ennek segítségével speciális eszközt biztosítunk olyan optimalizálási feladatok elemzésére, melyek két vagy több részfeladatra bomlanak minden döntés során.

Számos programozási feladat hatékony megoldása feltételezi a feladat optimális módon való részfeladataira bontását. A jelen dolgozatban olyan optimalizálási feladatokkal foglalkozunk, amelyekre igazak az alábbi feltételek:

- Létezik egy célfüggvény, amelyet optimalizálni kell.
- A célfüggvény optimalizálása a feladatnak részfeladataira való lebontását feltételezi.
- Ez döntéssorozatot foglal magába.
- Ami a részfeladatokra bontást illeti, minden döntéssel (vágással) a feladat (I. típusú optimalizálási feladatok) egy hasonló, de kisebb méretű részfeladattá redukálódik, vagy két vagy több hasonló, de kisebb méretű részfeladatra esik szét (II. típusú optimalizálási feladatok).
- A célfüggvényt a feladat részfeladatainak halmazán definiáljuk.
- A feladatra érvényes az optimalitás alapelve, miszerint a feladat optimális megoldása felépíthető a részfeladatainak optimális

1 Kátai Zoltán, Studia-Informatica, 2006/2, Cluj-Napoca

megoldásaiból (a célfüggvénynek a feladatra vonatkozó optimumértéke meghatározható a részfeladatokra vonatkozó optimumértékeiből).
- A feladat különböző lebontási módjai közül azt (vagy azt a döntéssorozatot) tekintjük optimálisnak, amelyik – az optimalitás alapelvével összhangban – a feladat optimális megoldásának a felépítését vonja maga után.
- Egy részfeladatot akkor nevezünk triviálisnak, ha a célfüggvény rávonatkozó értéke a feladat input adataiból triviálisan adódik.

Egy ilyen optimalizálási feladatot a dinamikus programozásnak nevezett programozási technika old meg hatékonyan.

Példa: Számítsuk ki az $A_1 \times A_2 \times \ldots \times A_n$ mátrixszorzatot (a mátrixok méretei: $d_0 \times d_1, d_1 \times d_2, \ldots, d_{n-1} \times d_n$). A mátrixszorzás asszociativitásából adódóan ezt sokféleképpen megtehetjük. Határozzuk meg a szorzat egy olyan zárójelezését (részfeladatokra bontását), amelynek megfelelően a mátrixok szorzásának sorrendje minimális számú elemi szorzást (a célfüggvényt) feltételez.

Például, ha

$$n = 4, \quad A_1(1 \times 10),\ A_2(10 \times 1),\ A_3(1 \times 10),\ A_4(10 \times 1).$$

Az optimális részfeladatokra bontás: $(A_1 \times A_2) \times (A_3 \times A_4)$, amely 21 elemi szorzást feltételez. Egy legrosszabb megoldás 210 szorzást feltételezne: $((A_1) \times (A_2 \times A_3)) \times (A_4)$.

Egy optimalizálási feladat szerkezete jól leírható egy *d_gráf* (division graph) segítségével, amelyet az alábbiakban definiálunk.

d_gráfok

D.1. definíció. A $G_d(V_p, V_d, E_p, E_d, C)$ összefüggő súlyozott irányított gráfot *d_gráf*nak nevezzük, ha teljesülnek az alábbi feltételek:

1. $V = V_p \cup V_d$ és $E = E_p \cup E_d$.
2. V_p – a gráf p típusú csomópontjainak halmaza (p_csomópontok). $V_p = \{p_1, p_2, \ldots, p_{nr\_p}\}$, ahol nr_p – p_ a csomópontok száma.
3. A V_p halmaznak pontosan egy eleme forrás (f).
4. Jelöljük $NY(G_d)$-vel a G_d gráf p típusú nyelő-csomópontjainak halmazát (nr_ny jelöli a nyelők számát).
5. V_d – a gráf d típusú csomópontjainak halmaza (d_csomópontok); ahol nr_d – d típusú csomópontok száma.

6. A d_csomópontok összes szomszédja p típusú, és fordítva, minden p_csomópont szomszédjai d típusúak. Minden d_csomópontnak pontosan egy p típusú be-szomszédja van, amit a *p_apjá*nak nevezünk. A p_csomópontok ki-szomszédjait pedig nevezzük *d_fiak*nak.
   Minden d_csomópontnak van legalább egy p típusú ki-szomszédja, és ezekre a d_fiak megnevezéssel fogunk utalni.
7. A d_csomópontokat két indexszel azonosítjuk:
   például a d_{ik} jelölés a p_i p_csomópont d_fiai közül a k-val azonosítottra utal.
8. E_p – a gráf p típusú irányított éleinek halmaza (p_élek).
   $E_p = \{(p_i, d_{ik}) / p_i \in V_p, d_{ik} \in V_d\}$.
9. E_d – a gráf d típusú irányított éleinek halmaza (d_élek).
   $E_d = \{(d_{ik}, p_j) / d_{ik} \in V_d, p_j \in V_p, i < j\}$. Figyeljük meg, hogy bármely p_csomópont p típusú utódai nála nagyobb indexűek. Tehát minden d_gráf esetében a forrás az 1-es csomópont.
10. A $C : E_p \to \mathbb{R}$ függvény minden p_élhez hozzárendel egy költséget. A d_éleket nulla költségűeknek tekintjük.

D.2. tétel. Minden d_gráf körmentes.

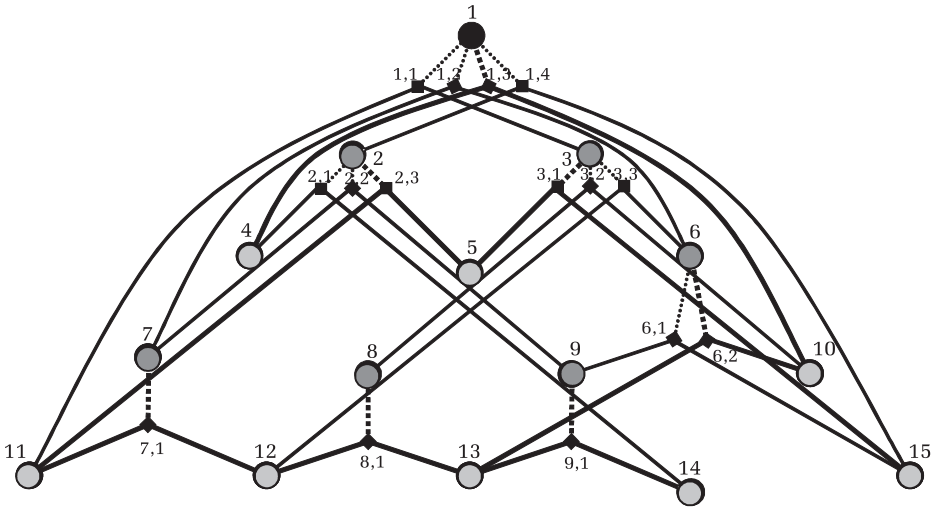
Bizonyítás: Tegyük fel, hogy létezne egy irányított kör valamely d_gráfban. A definíció hatodik pontjával összhangban a p és d típusú csomópontok egymást váltogatják a körön. Ha a kör egy p_csomópontból és egy d_csomópontból áll, akkor a p_csomópont a d_csomópontnak p_apja is és p_fia is. Ez viszont ellentmond a definíció kilencedik pontjának, miszerint a d_csomópontok p_fiai mindig nagyobb indexűek a p_apjánál. Ha a körön van legalább két csomópont mindkét típusúból, akkor legyen p_i és p_j a kör két egymás utáni p_csomópontja. Mivel p_i őse is és utódja is p_j-nek, ezért – ugyancsak a definíció kilencedik pontja alapján – i kisebb is és nagyobb is kellene legyen j-nél, ami nyilván lehetetlen. Tehát minden d_gráf körmentes.

Következmény: Minden d_gráf p_csomópontjai topologikus sorrendbe állíthatók.

Az alábbi ábra egy olyan d_gráfot mutat be, amelyben minden d_csomópontnak pontosan két p_fia van.

D.3. definíció. A $g_d(v, e, c)$ d_gráfot a $G_d(V, E, C)$ d_gráf *d_részgráfjának* nevezzük, ha

– $v_p \subseteq V_p$, $v_d \subseteq V_d$, $e_p \subseteq E_p$, $e_d \subseteq E_d$ és $NY(g_d) \subseteq NY(G_d)$;



– p_csomópontok (a fekete forrás, a világosszürkék nyelők)
– d_csomópontok
– irányított p_élek
– irányított d_élek
(Az ábra túlzsúfolásának elkerüléséért nem rajzoltuk be expliciten az élek irányításait. Mindig fentről lefelé mutatnak.)

D.1. ábra.

- c: $e_p \to \mathbb{R}$ és $c(x) = C(x)$ bármely $x \in e_p$;
- g_d bármely p, illetve d típusú csomópontja d, illetve p típusú fiainak halmaza megegyezik a g_d és G_d d_gráfokban.

A fenti definícióból következik, hogy egy d_gráf minden p_csomópontja egyértelműen azonosítja azt a d_részgráfot, amelyiknek az illető csomópont a forrása.

D.4. definíció. *D_fának* nevezünk egy olyan d_gráfot, amelyben minden p_csomópontnak (kivéve a nyelőket) pontosan egy fia van. Egy d_fa forrását *d_gyökér*nek, nyelőit pedig *d_levelek*nek nevezzük. A T_d d_fa leveleinek halmazát jelöljük $L(T_d)$-vel.

D.5. definíció. A $t_d(vt, et, c)$ d_fát a $T_d(Vt, Et, C)$ d_fa *d_részfájának* nevezzük, ha

- $vt_p \subseteq Vt_p$, $vt_d \subseteq Vt_d$, $et_p \subseteq Et_p$, $et_d \subseteq Et_d$ és $L(t_d) \subseteq L(T_d)$,
- $c : et_p \to \mathbb{R}$ és $c(x) = C(x)$ bármely $x \in et_p$,
- t_d bármely d_csomópontja p_fiainak halmaza megegyezik a t_d és T_d d_fákban.

D.6. definíció. Egy $T_d(Vt, Et, C)$ d_fát a $G_d(V, E, C)$ d_gráf *d_részfájának* nevezzük, ha

- $Vt_p \subseteq V_p$, $Vt_d \subseteq V_d$, $Et_p \subseteq E_p$, $Et_d \subseteq E_d$, $L(T_d) \subseteq NY(G_d)$,
- $c : Et_p \to \mathbb{R}$ és $c(x) = C(x)$ bármely $x \in Et_p$,
- T_d bármely d_csomópontja p_fiainak halmaza megegyezik a T_d d_fában és a G_d d_gráfban.

Ha T_d gyökere megegyezik G_d forrásával, akkor *feszítő d_részfá*ról beszélünk.

D.7. definíció. Egy *d_fa költsége* alatt a p_élei összköltségét értjük.

D.8. definíció. Egy d_gráf legkisebb költségű feszítő d_részfáját *minimális költségű feszítő d_részfá*nak nevezzük.

D.9. definíció (optimalitás alapelve). Egy d_gráfot *optimális szerkezetű*nek nevezünk, ha az optimális (minimális költségű) feszítő d_részfájának minden d_részfája maga is optimális feszítő d_részfája a gyökere meghatározta d_részgráfnak.

Optimális szerkezetű d_gráfok

Legyen $G_d(V, E, C)$ egy d_gráf. Az alábbiakban egy olyan C p_élköltségfüggvényt definiálunk, amely mellett minden d_gráf optimális szerkezetű lesz. Ezt megelőzően definiáljuk a w_p és w_d csomópontsúlyozó függvényeket. A p_i p_csomópont d_fiainak halmazát d_fiak(p_i)-vel jelöltük, a d_{ik} d_csomópont p_fiaiét pedig p_fiak(d_{ik})-vel.

A w_p súlyfüggvény:

$w_p : V_p \to \mathbb{R}$ minden p_i p_csomópontnak megfelelteti a következő értékeket:

$$w_p(p_i) = \text{optimum}\{w_d(d_{ik})\}, \text{ ha } p_i \notin NY(G_d), d_{ik} \in \text{d\_fiak}(p_i),$$
$$w_p(p_i) = h_r, \text{ ha } p_i \text{ a d\_gráf } r\text{-edik nyelője,}$$

ahol $\{h_1, h_2, \ldots, h_{nr\_ny}\} \subset \mathbb{R}$ egy bemeneti halmaz, amely a G_d d_gráfot jellemzi.

Minden p_csomópont w_p súlya (kivéve a nyelőket) egyenlő az „optimális d_fia" w_d súlyával.

A w_d súlyfüggvény:

$w_d : V_d \to \mathbb{R}$ minden d_{ik} d_csomópontnak megfelelteti a következő értékeket:

$$w_d(d_{ik}) = \phi(\{w_p(p_j)/p_j \in \text{p\_fiak}(d_{ik})\}).$$

A ϕ függvény leírja matematikailag, miként *számítható ki* egy d_csomópont w_d súlya a p_fiai w_p súlyaiból. A ϕ függvény ugyanakkor jellemzi a G_d d_gráfot is.

A fenti súlyfüggvények bevezetése után definiáljuk a C^* költségfüggvényt az alábbi módon:

$$C^* : E_p \to \mathbb{R}, C^*((p_i, d_{ik})) = |w_p(p_i) - w_d(d_{ik})|.$$

D.10. tétel. Minden $G_d(V, E, C^*)$ d_gráf optimális szerkezetű.

Bizonyítás: Mivel p_csomópontok súlyának az optimális d_fiúk súlyát választottuk, ezért minden p_csomópontra illeszkedik legalább egy nulla költségű p_él. Ebből adódóan a minimális költségű feszítő d_részfa, és ennek nyilván minden d_részfája, nulla költségű lesz. Lévén, hogy C^*, definíciójából adódóan, pozitív költségeket rendel a p_élekhez, magától értetődően a minimális költségű feszítő d_részfa minden d_részfája minimális költségű feszítő d_részfája lesz a gyökerével megegyező forrású d_részgráfnak.

Az optimális feszítő d_részfa meghatározása az optimalitás alapelvének implementálásával

Legyen $G_d(V, E, C^*)$ optimális szerkezetű d_gráf. Az optimalitás alapelve értelmében G_d bármely g_d d_részgráfjának az optimális feszítő d_részfája meghatározható g_d fiú d_részgráfjainak optimális feszítő d_részfáiból. Következésképpen fordított topologikus sorrendben fogjuk meghatározni a $p_i \in V_p$ $(i = 1, 2, \ldots, nr\_p)$ csomópontokhoz tartozó optimális feszítő d_részfákat. Ez a sorrend úgy biztosítható, ha mélységi bejárásakor az egyes csomópontokkal az elhagyásuk pillanatában foglalkozunk.

Használjuk a `WP[1..nr_p]` és `WD[1..nr_d]` tömböket a G_d d_gráf p, illetve d típusú csomópontjai súlyainak tárolására. Kezdetben a `WP` tömb nyelőknek megfelelő elemeit ezeknek h_i (`i=1..nr_ny`) súlyával, a többi elemét pedig a `NIL` értékkel töltjük fel. Az optimális feszítő d_részfa tárolására legyen az `ODS[1..nr_p]` tömb, amely a p_csomópontok optimális d_fiat fogja tárolni. Ezt a tömböt `NIL` értékekkel inicializáljuk. Az `inicializál` eljárás, attól függően, hogy milyen természetű optimumot kell számítani, egy megfelelő kezdőértéket ad a paraméterként kapott $WP[p_i]$ tömbelemnek. A `jobb_e` függvény azt vizsgálja, hogy az optimum természetének megfelelően az első paraméter jobbnak számít-e a másodiknál.

```
eljárás optimális_lebontás(p_i)
   inicializál(WP[p_i])
   minden d_ik ∈d_fiak(p_i) végezd
      minden p_j ∈p_fiak(d_ik) végezd
         ha WP[p_j]=NIL akkor
            optimális_lebontás(p_j)
         vége ha
      vége minden
      WD[d_ik] ← φ({WP[p_j]/p_j ∈p_fiak(d_ik)})
      ha jobb_e(WD[d_ik],WP[p_i]) akkor
         WP[p_i] ← WD[d_ik]
         ODS[p_i] ← d_ik
      vége ha
   vége minden
vége optimális_lebontás
```

Az `optimális_lebontás` eljárást természetesen a forrás csomópontra hívjuk meg, feltéve, ha ez nem nyelő is egymagában. A nyelők `ODS` értékei

NIL értékűek maradnak. Az alábbi rekurzív eljárás az ODS tömb alapján kiírja az optimális feszítő d_részfa p_éleit preorder sorrendben.

```
eljárás optimális_fa(p_i)
   kiír: (p_i,ODS[p_i])
   minden p_j ∈p_fiak(ODS[p_i]) végezd
      ha ODS[p_j]≠NIL akkor
         optimális_fa(p_j)
      vége ha
   vége minden
vége optimális_fa
```

Az optimalizálási feladatok és a d_gráfok

Minden, a bevezetőben leírt típusú optimalizálási feladathoz rendelhető egy d_gráf.

- A p_csomópontok képviselik a feladat lebontásából adódó különböző részfeladatokat. A forrás az eredeti feladatot, a nyelők a triviálisakat ábrázolják.
- A p_csomópontok számozása és az ebből adódó körmentesség kéz a kézben jár azzal, hogy a lebontás során a feladatot mind egyszerűbb és egyszerűbb részfeladatokra vezetjük vissza.
- Egy p_csomópontnak annyi d_fia lesz, ahányféleképpen részfeladataira bontható – az illető döntés alkalmával – az általa képviselt részfeladat. Ezen választási lehetőségeket ábrázolják a p_élek.
- A d_csomópontok azt ábrázolják, hogy az egyes döntésekből adódó különböző választások alkalmával az illető részfeladat hogyan esik szét részfeladataira.
- Egy d_csomópontnak annyi p_fia lesz, ahány részfeladatra esik szét az általa képviselt döntés (választás) alkalmával a p_apja ábrázolta részfeladat. Ezen részfeladatokra bontást ábrázolják a d_élek.
- Ha valamely feladat lebontásakor különböző részdöntéssorozatok ugyanahhoz a részfeladathoz vezetnek, akkor az illető p_csomópontnak különböző leszármazási ágakon azonos p_utódai lesznek.
- Egy d_gráf d_részgráfjai azt ábrázolják, hogy ezeknek forrásai által képviselt részfeladatai milyen különböző módokon bonthatók további még kisebb részfeladataira.

– Egy d_gráf valamely részfája a gyökere által képviselt részfeladat egyik részfeladataira bontását ábrázolja. Egy d_gráf feszítő d_részfái azt ábrázolják, hogy az eredeti feladat milyen különböző módokon bontható le részfeladataira.
– A d_gráfok optimális szerkezete azt tükrözi, hogy a feladat optimális megoldása a részfeladatok optimális megoldásaiból épül fel. Más szóval, az optimális döntéssorozat megfelelő részdöntéssorozatai ugyancsak optimálisak.
– Az optimális feszítő d_részfa a feladat optimális részfeladatokra bontását ábrázolja (minden p_éle az optimális döntéssorozat egyik döntését képviseli).
– A w_p függvény nem más, mint a feladat optimalizálandó célfüggvényének áthangolása a G_d d_gráfra.
– A h_1, h_2, ..., h_{nr_ny} valós értékek a célfüggvénynek a nyelők képviselte triviális részfeladatokra vonatkozó optimumértékei.
– Az optimum függvény természete direkt módon adódik a feladat célfüggvényéből, és gyakran a minimum vagy maximum függvények egyike.
– A ϕ függvényt a feladat szerkezete határozza meg, vagyis az, hogy milyen általános szabály szerint épül fel valamely részfeladat megoldása a fiúrészfeladatainak megoldásaiból.

Ezek után egy optimalizálási feladat úgy is felfogható, mint a megfelelő d_gráf forrása súlyának (a célfüggvény optimumértéke az eredeti feladatra vonatkozóan), valamint az optimális feszítő d_részfájának (optimális döntéssorozat, illetve optimális részfeladatokra bontás) a meghatározása.

Azt a stratégiát, mely szerint az `optimal_division` eljárás az optimalitás alapelvét implementálja, dinamikus programozásnak nevezzük.

Egy példaértékű feladatmegoldás

Tömörítés: *Adott egy n elemű bitsorozat. Továbbá rendelkezünk m bitszekvenciával*[2]*, amelyek közül nem hiányoznak az egymagában 0-s bitet, illetve egymagában 1-es bitet tartalmazó szekvenciák. Helyettesítsük a bitsorozatot minimális számú szekvenciával.*

Példa: Legyen a bitsorozat 01011.

2 A szekvenciák ugyancsak bitsorozatok. Az n elemű bitsorozattól való világos megkülönböztetésük végett használtuk a szekvencia kifejezést.

Továbbá a szekvenciák legyenek: `1:0`, `2:1`, `3:11`, `4:010`, `5:101`.

Látható, hogy a bitsorozat többféleképpen is felbontható a megadott szekvenciákra:

```
(0)(1)(0)(1)(1), (0)(1)(0)(11), (010)(11),
        (0)(101)(1), (010)(1)(1)
```

Az optimális megoldást nyilván a harmadik változat jelenti, melynek tömörített kódja 43.

Hogyan bontható a feladat részfeladataira? Amennyiben a bitsorozat nem egyike a megadott szekvenciáknak, akkor vágjuk kettőbe ezt a bitsorozatot, visszavezetve ezáltal optimális tömörítését a vágás jobb és bal felére eső bitszakaszok optimális tömörítésére. Ezt addig folytatjuk, míg olyan bitszakaszokhoz nem jutunk, amelyek szerepelnek a megadott szekvenciák között (egyetlen szekvenciával helyettesíthető triviális részfeladatok).

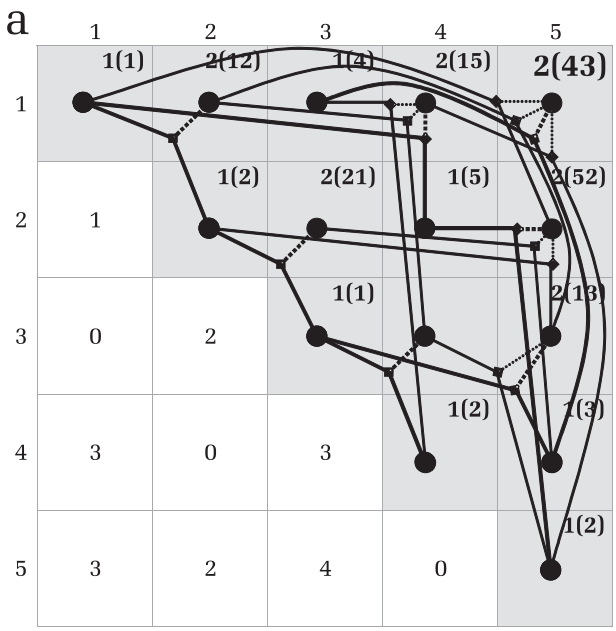
Az általános részfeladatot a bitsorozat `i..j` szakaszának optimális tömörítése jelenti. Ezek az indexek éppen azonosítják a feladathoz rendelhető d_gráf p_csomópontjait. A példafeladat d_gráfjának a forrása tehát az 15-ös (olvasd egyöt-ös) p_csomópont lesz. A `WP` tömb szerepét egy `a[1..n,1..n]` kétdimenziós tömb főátló és főátló feletti része fogja betölteni. E tömbterület felfogható a feladat d_gráfja implicit ábrázolása gyanánt. A p_csomópontokat a megfelelő tömbelemek képviselik, és w_p súlyoknak az optimális tömörítés kódjának hosszúságait tekinthetjük. Az `ij` indexű p_csomópontnak – amennyiben nem nyelő (az `i..j` bitszakasz nem része a megadott szekvenciáknak) – `(j-i+1)` darab d_fia lesz, amelyek p_fiúpárjai az `(ik, (k+1)j)` (`k=i..j-1`) p_csomópontpárok lesznek. Tehát a d_csomópontok, illetve p és d típusú élek csak implicit vannak jelen a d_gráf ezen ábrázolásában. Természetesen ez azt is maga után vonja, hogy nem használunk `WD` tömböt. Erre nincs is szükség, ahogy a d_gráfok d_csomópontjainak súlyozása is kikerülhető az (1) és (2) képletek összevonásával (bármely nem nyelő p_csomópont súlya meghatározható a közvetlen p_utódai súlyából):

$$w_p(p_i) = \text{optimum}\{\phi(\{w_p(p_k)/p_k \in \text{p\_fiak}(d_j)\})\}, \quad \text{ha } p_i \notin NY(G_d),\ dj \in \text{d\_fiak}(p_i)$$

Az `ODS` tömb szerepében jól felhasználható a kétdimenziós tömb főátló fölötti része. Az `a[j,i]` tömbelem (`i<j`) az `i..j` bitszakasz optimális

vágásához tartozó optimális `k` érték tárolásával implicit képviseli az `ij` indexű p_csomópont optimális d_fiát. Ha az `ij` indexű p_csomópont (`i<j`) nyelő, akkor az `a[j,i]` elem nulla értéket fog kapni. Az `ii` indexű (`i=1..n`) p_csomópontok nyilván mind nyelők.

Az alábbi ábra megeleveníti a példafeladat d_gráfját, ahogy az a részfeladatok optimumértékeit tároló a tömbben el van rejtve. Kiemelten rajzoltuk minden `ij` forrású d_részgráf optimális feszítő d_részfáját.



D.2. ábra.

A d_gráf ily módon történő ábrázolása mellett a nem nyelő p_csomópontok fordított topologikus sorrendben való bejárása megvalósítható az a tömb főátló feletti elemeinek egyszerű bejárásával (például soronként lentről felfele, balról jobbra). Mivel bármely bitszakasz optimális kódja az optimális vágásából adódó részszakaszok optimális kódjainak az egymás után fűzése, a ϕ függvény egy egyszerű összeadó függvény lesz. Lévén, hogy a legrövidebb kódú tömörítést keressük, az optimum függvény minimumot fog számolni. Az input adatokat az `n`, `m`, `b[1..n]` és `sequences[1..m]` változók tárolják. A `szekvencia(i,j)` függvény ellenőrzi, hogy a `b[i..j]` bitszakasz szerepel-e a megadott szekvenciák

között. Ha igen, akkor visszatéríti a kódját (a sequences tömbbeli sorszámát), ha nem, nullát térít vissza. Az a tömb feltöltésével párhuzamosan egy KOD[1..n,1..n] tömbben eltároljuk magukat az optimális kódokat is (a fenti ábrán zárójelben tüntettük fel ezeket). A ragaszt(cod1,cod2) függvény egymás után fűzi a paraméterként kapott kódokat.

```
minden i ← 1,n,1 végezd
   a[i,i] ← 1
   KOD[i,i] ← szekvencia(i,i)
vége minden
minden i ← n-1,1,-1 végezd
   minden j ← i+1,n,1 végezd
      cod ← szekvencia(i,j)
      ha cod>0 akkor
         a[i,j] ← 1
         KOD[i,j] ← cod
         a[j,i] ← 0
      különben
         a[i,j] ← 0
         minden k ← i,j-1,1 végezd
            ha a[i,k]+a[k+1,j]>a[i,j] akkor
               a[i,j] ← a[i,k]+a[k+1,j]
               KOD[i,j] ← ragaszt(KOD[i,k],KOD[k+1,j])
               a[j,i] ← k
            vége ha
         vége minden
      vége ha
   vége minden
vége minden
```

Az eredeti bitsorozat optimális kódja a KOD[1,n] tömbelembe kerül. Ha szeretnénk a bitsorozat optimális zárójelezését is, akkor ez a d_gráf optimális feszítő d_részfájának mélységi bejárásával állítható elő az a tömb alapján.

```
eljárás DFS(i,j)
   kiír: '('
   ha i=j VAGY a[j,i]=0 akkor
      minden k ← i,j,1 végezd
         kiír: b[k]
      vége minden
   különben
      DFS(i,a[j,i])
      DFS(a[j,i]+1,j)
```

```
    vége ha
    kiír ')'
vége DFS
```

Következtetések

Elsősorban fontos észrevenni azt, hogy az I. típusú optimalizálási feladatokhoz rendelt d_gráfokat „normál_gráfokra" lehet redukálni (minden d_csomópontnak egyetlen p_fia van, ezért ezeket ki lehet hagyni a gráfból úgy, hogy a d_csomópont p_apját az egyetlen p_fiával párosítjuk). Ebben a speciális esetben az optimális megoldást a gráf gyökér–levél útja fogja képviselni.

A d_gráfok bevezetésével lehetőség nyílik számos optimalizálási feladat egységes tárgyalásához, és az ezekhez kapcsolódó dinamikus programozásos stratégiák elméleti megalapozásához. Hasonló kapcsolatról van szó, mint a mohó algoritmusok és a matroidok elmélete között.

GRÁFELMÉLETI FOGALMAK SZÓTÁRA

Magyar	**Română**	**English**
be-fok	grad interior	in-degree
csúcs, szögpont	nod, vârf	node, vertex
egyszerű gráf	graf simplu	graph
él	muchie	edge
fa	arbore	tree
faváz, feszítőfa	arbore de acoperire, arbore parțial	spanning tree
fok	grad	degree
folyam	flux	flow
forrás	sursă	source
gráf nagysága	dimensiunea grafului	size of graph
gráf rendje	ordinul grafului	order of graph
hurok	buclă	loop
incidencia mátrix, illeszkedési mátrix	matrice de incidență	incidence matrix
irányított él	arc	arc
irányított gráf	graf orientat	directed graph, digraph
irányított kör	circuit elementar	directed cycle, cycle in digraph
irányított séta	drum	walk in digraph

Magyar	**Română**	**English**
irányított út	drum elementar	path in digraph, directed path
irányított vonal	drum simplu	trail in digraph
irányított zárt séta	circuit	closed walk in digraph
irányított zárt vonal	circuit simplu	directed circuit, circuit in digraph
ki-fok	grad exterior	out-degree
komplementer gráf, kiegészítő gráf	graf complementar	complement of a graph
komponens	componentă	component
kör	ciclu elementar	cycle
közlekedési hálózat	rețea de transport	network
kritikus út	drum critic	critical path
kromatikus szám	număr cromatic	chromatic number
liget	pădure	forest
mélységi keresés	căutare în adâncime	depth-first search
mohó algoritmus	algoritm greedy	greedy algorithm
multigráf	multigraf	multigraph
nyelő	puț	sink
összefüggő gráf	graf conex	connected graph
páros gráf	graf bipartit	bipartite graph
párosítás	cuplaj	matching
reguláris gráf	graf regular	regular graph
részgráf	subgraf, graf parțial	subgraph
séta	lanț	walk
síkgráf	graf planar	planar graph

Magyar	**Română**	**English**
szélességi keresés	căutare în lățime	breadth-first search
szögpont, csúcs	vârf, nod	vertex, node
szomszédossági mátrix	matrice de adiacență	adjacency matrix
teljes gráf	graf complet	complete graph
út	lanț elementar	path
vonal	lanț simplu	trail
zárt séta	ciclu	closed walk
zárt vonal	ciclu simplu	circuit

SZAKIRODALOM

[1] ANDRÁSFAI B.
1987 *Introductory graph theory.* North Holland, Akadémiai Kiadó
[2] CORMEN, T. H.–LEISERSON, C. E.–RIVEST, R. R.
1998 *Algoritmusok.* Budapest, Műszaki Kiadó
[3] CSEKE V.
1972 *A gráfelmélet és alkalmazásai.* Bucureşti, Editura Ştiinţifică
[4] CRISTEA, V.–ATHANASIU, I.–KALISZ, E.–IORGA, V.
1993 *Tehnici de programare.* Bucureşti, Editura Teora
[5] IONESCU, C.–BĂLAN, A.
2004 *Informatică pentru grupele de performanţă* (clasa a XI-a). Cluj-Napoca, Editura Dacia
[6] KATONA Gy.–RECSKI A.–SZABÓ Cs.
2002 *A számítástudomány alapjai.* Budapest, Typotex Kiadó
[7] KÁTAI Z.
2007 *Algoritmusok felülnézetből.* Kolozsvár, Scientia Kiadó
[8] LOVÁSZ L.
1997 *Kombinatorikai problémák és gyakorlatok.* Budapest, Typotex Kiadó
[9] ODĂGESCU, I.–FURTUNA, F.
1998 *Metode şi tehnici de programare.* Editura Libris
[10] TUDOR, S.
1996 *Tehnici de programare.* Bucureşti, Editura Teora
[11] `http://hu.wikipedia.org`
[12] `www.math.klte.hu/~turjanyi`
[13] `http://www.cs.ubbcluj.ro/~kasa`

ABSTRACT

The Graph Algorithms book is an introduction in graph theory. The author emphasizes the algorithmic aspects of the graph theory. The first chapters present the basic graph algorithms like breadth-first and depth-first search and their applications, minimum spanning tree and shortest path algorithms, etc. The second part of the book treats subjects with stronger mathematical flavour. All subjects are presented through efficient didactical methods.

REZUMAT

Cartea „Algoritmica grafurilor" pune accent pe latura algoritmică a teoriei grafurilor. Autorul prezintă algoritmii de bază a teoriei grafurilor cu foarte mută exigență. Primii capitoli tratează probleme de teoria grafurilor care au aplicare directă în domeniul programării calculatoareleor (parcurgerea grafurilor, arbori minime, drumuri optime, rețele de fluxuri, etc). În partea a doua a cărții autorul tratează teme mai apropiate de matematică (grafuri planare, problemele de colorare a grafurilor, etc). Cartea este scrisă cu o exigență didactică deosebită pentru a fi accesibilă atât liceenilor, cât şi studenților.

A SZERZŐRŐL

Kátai Zoltán 1968. március 13-án született Nagyváradon. Középiskolai tanulmányait a marosvásárhelyi Bolyai Farkas Elméleti Líceumban végezte 1982–1986 között, egyetemi tanulmányait pedig a Kolozsvári Műszaki Egyetem Automatizálás és Számítógépek Karán 1987–1992 között.

1992–2005 között informatika tanár a marosvásárhelyi Bolyai Farkas Elméleti Líceumban, 1996–2000 között kvalifikált oktató a Gábor Dénes Főiskola marosvásárhelyi karán, 1995–1997 között pedig óraadó tanár a marosvásárhelyi Petru Maior Egyetemen. 2002-től a Sapientia Erdélyi Magyar Tudományegyetem Műszaki és Humántudományok Kara Matematika–Informatika Tanszékének adjunktusa.

Fő kutatási területe a programozási technikák, dinamikus programozás, valamint az érzékszervek párhuzamos bevonása a tanítás-tanulás folyamatába.

JEGYZETEK

A SAPIENTIA –
ERDÉLYI MAGYAR TUDOMÁNYEGYETEM JEGYZETEI

Bege Antal
: Számelméleti feladatgyűjtemény. Marosvásárhely, Műszaki és Humán Tudományok Kar, Matematika–Informatika Tanszék. 2002.

Bege Antal
: Számelmélet. Bevezetés a számelméletbe. Marosvásárhely, Műszaki és Humán Tudományok Kar, Matematika–Informatika Tanszék. 2002.

Vofkori László
: Gazdasági földrajz. Csíkszereda, Csíkszeredai Kar, Gazdaságtan Tanszék. 2002.

Tőkés Béla – Dónáth-Nagy Gabriella
: Kémiai előadások és laboratóriumi gyakorlatok. Marosvásárhely, Műszaki és Humán Tudományok Kar, Gépészmérnöki Tanszék. 2002.

Irimiaş, George
: Noţiuni de fonetică şi fonologie. Csíkszereda, Csíkszeredai Kar, Humán Tudományok Tanszék. 2002.

Szilágyi József
: Mezőgazdasági termékek áruismerete. Csíkszereda, Csíkszeredai Kar, Gazdaságtan Tanszék. 2002.

Nagy Imola Katalin
: A Practical Course in English. Marosvásárhely, Műszaki és Humán Tudományok Kar, Humán Tudományok Tanszék. 2002.

Balázs Lajos
: Folclor. Noţiuni generale de folclor şi poetică populară. Csíkszereda, Csíkszeredai Kar, Humán Tudományok Tanszék. 2003.

Popa-Müller Izolda
Műszaki rajz. Marosvásárhely, Műszaki és Humán Tudományok Kar, Gépészmérnöki Tanszék. 2004.

Fodorpataki László – Szigyártó Lídia – Bartha Csaba
Növénytani ismeretek. Kolozsvár, Természettudományi és Művészeti Kar, Környezettudományi Tanszék. 2004.

Marcuş, Andrei – Szántó Csaba – Tóth László
Logika és halmazelmélet. Marosvásárhely, Műszaki és Humán Tudományok Kar, Matematika–Informatika Tanszék. 2004.

Kakucs András
Műszaki hőtan. Marosvásárhely, Műszaki és Humán Tudományok Kar, Gépészmérnöki Tanszék. 2004.

Biró Béla
Drámaelmélet. Csíkszereda, Gazdasági és Humántudományi Kar, Humántudományi Tanszék. 2004.

Biró Béla
Narratológia. Csíkszereda, Gazdasági és Humántudományi Kar, Humántudományi Tanszék. 2004.

Márkos Zoltán
Anyagtechnológia. Marosvásárhely. Műszaki és Humán Tudományok Kar, Gépészmérnöki Tanszék. 2004.

Grecu, Victor
Istoria limbii române. Csíkszereda, Gazdasági és Humántudományi Kar, Humántudományi Tanszék. 2004.

Varga Ibolya
Adatbázis-kezelő rendszerek elméleti alapjai. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2004.

Csapó János
Biokémia. Csíkszereda, Műszaki és Társadalomtudományi Kar, Műszaki és Természettudományi Tanszék. 2004.

Csapó János – Csapóné Kiss Zsuzsanna
Élelmiszer-kémia. Csíkszereda, Műszaki és Társadalomtudományi Kar, Műszaki és Természettudományi Tanszék. 2004.

Kátai Zoltán
Programozás C nyelven. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2004.

Weszely Tibor
Analitikus geometria és differenciálgeometria. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2005.

Györfi Jenő
A matematikai analízis elemei. Csíkszereda, Gazdaság- és Humántudományok Kar, Matematika–Informatika Tanszék. 2005.

Finta Béla – Kiss Elemér – Bartha Zsolt
Algebrai struktúrák – feladatgyűjtemény. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2006.

Antal Margit
Fejlett programozási technikák. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2006.

Csapó János – Salamon Rozália
Tejipari technológia és minőségellenőrzés. Csíkszereda, Műszaki és Társadalomtudományok Kar, Élelmiszertudományi Tanszék. 2006.

Oláh-Gál Róbert
Az informatika alapjai közgazdász- és mérnökhallgatóknak. Csíkszereda, Gazdaság- és Humántudományok Kar, Matematika–Informatika Tanszék. 2006.

Józon Mónika
Általános jogelméleti és polgári jogi ismeretek. Csíkszereda, Gazdaság- és Humántudományok Kar, Üzleti Tudományok Tanszék. 2007.

Kátai Zoltán
Algoritmusok felülnézetből. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2007.

Csapó János – Csapóné Kiss Zsuzsanna – Albert Csilla
Élelmiszer-fehérjék minősítése. Csíkszereda, Műszaki és Társadalomtudományi Kar, Élelmiszertudományi Tanszék. 2007.

Ágoston Katalin – Domokos József – Márton Lőrinc
Érzékelők és jelátalakítók. Laboratóriumi útmutató. Marosvásárhely, Műszaki és Humántudományok Kar, Villamosmérnöki Tanszék. 2007.

Szász Róbert
Komplex függvénytan. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2007.

Kakucs András
A végeselem-módszer alapjai. Marosvásárhely, Műszaki és Humántudományok Kar, Gépészmérnöki Tanszék. 2007.

Antal Margit
Objektumorientált programozás. Marosvásárhely, Műszaki és Humántudományok Kar, Matematika–Informatika Tanszék. 2007.

Majdik Kornélia – Tonk Szende-Ágnes
Biokémiai alkalmazások. Kémiai laboratóriumi jegyzet. Kolozsvár, Természettudományi és Művészeti Kar, Környezettudományi Tanszék. 2007.

A PARTIUMI KERESZTÉNY EGYETEM JEGYZETEI

Kovács Adalbert
: Alkalmazott matematika a közgazdaságtanban. Lineáris algebra. Nagyvárad, Alkalmazott Tudományok Kar, Közgazdaságtan Tanszék. 2002.

Horváth Gizella
: A vitatechnika alapjai. Nagyvárad, Bölcsészettudományi Kar, Filozófia Tanszék. 2002.

Angi István
: Zeneesztétikai előadások. I. Nagyvárad, Alkalmazott Tudományok Kar, Zenepedagógiai Tanszék. 2003.

Péter György – Kinter Tünde – Pajzos Csaba
: Makroökonómia. Feladatok. Nagyvárad, Alkalmazott Tudományok és Művészetek Kar, Közgazdaságtan Tanszék. 2003.

Angi István
: Zeneesztétikai előadások. II. Nagyvárad, Alkalmazott Tudományok Kar, Zenepedagógiai Tanszék. 2005.

Tonk Márton
: Bevezetés a középkori filozófia történetébe. Nagyvárad, Bölcsészettudományi Kar, Filozófiai Tanszék. 2005.

Scientia Kiadó
400112 Kolozsvár (Cluj-Napoca)
Mátyás király (Matei Corvin) u. 4. sz.
Tel./fax: +40-264-593694
E-mail: kpi@kpi.sapientia.ro

Korrektúra:
Jancsik Pál

Műszaki szerkesztés:
Lineart Kft.

Tipográfia:
Könczey Elemér

Készült a kolozsvári Gloria nyomdában
150 példányban, 16 nyomdai ív terjedelemben
Igazgató: Nagy Péter