Dinamikus programozás

(az egyszerűtől a bonyolult fele)

https://codeforces.com/blog/entry/43256

A DP stratégia (optimalizálási) kulcspontjai

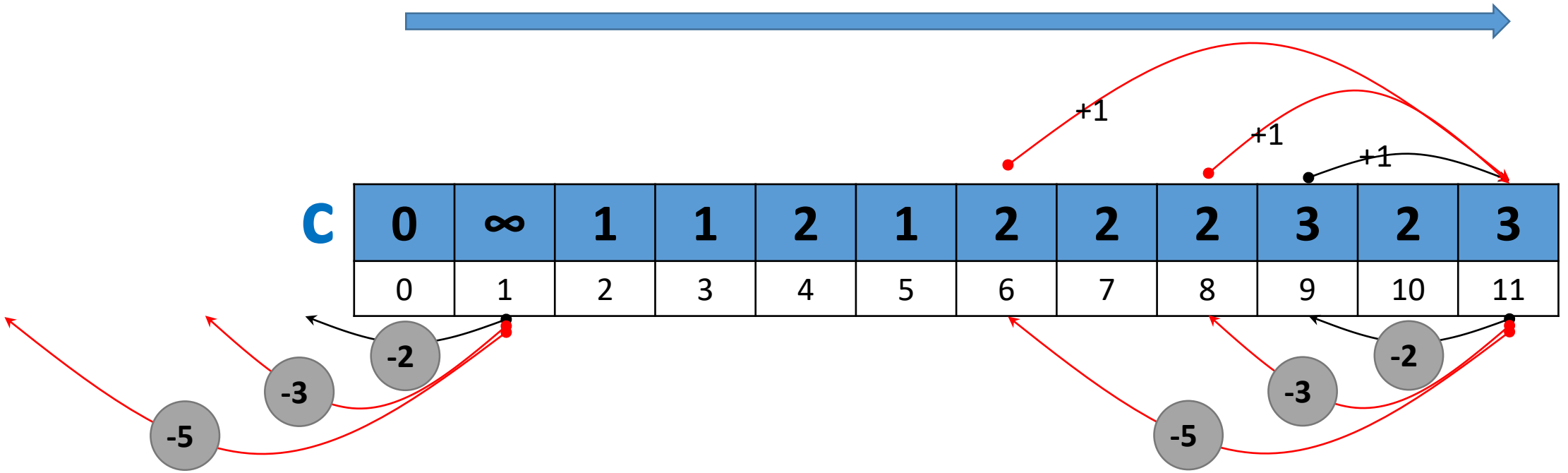
- **Átlátjuk**, hogy a feladat miként épül fel hasonló részfeladatokból
  - Hogyan bomlik, mely részfeladatira, lépésről lépésre (apa-fiú kapcsolat)
  - Meghatározzuk a részfeladatok általános (paraméteres) alakját
- **Egyszerűtől bonyolult f**ele haladva oldjuk meg a részfeladatokat
- Részfeladatonként egy értéket (az optimumot) tároljuk el (**tömbben**)
  - A cella-indexek és a részfeladatok paraméterei kéz-a-kézben járnak
  - optimalitás alapelve: **„a feladat optimuma felépíthető a részfeladatok optimumaiból”**
- Meghatározunk egy **rekurzív képletet**, amely leírja, hogy a kurrens részfeladat:
  - optimuma **miként építhető fel** a közvetlen fiúrészfeladatok optimumaiból (optimális építkezés: *optimumokból optimálisan*)

Pénzérmék (értékről – értékre)

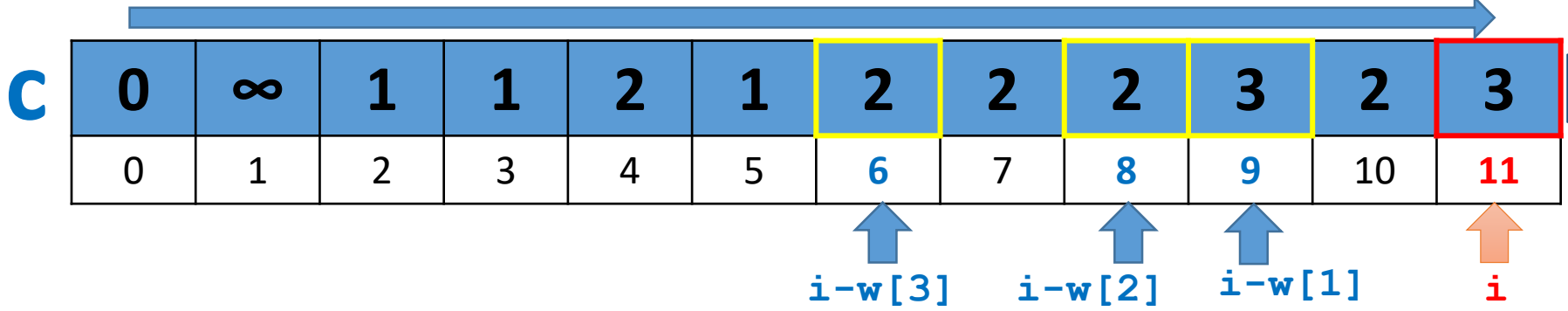
- Adott n féle pénzérme, melyek értékeit a w[1…n] tömb tárolja, valamint egy S összeg. Fizessük ki az S összeget minimális pénzérme felhasználásával (bármelyikből bármennyi felhasználható).
  - INPUT:
    - S=11
    - n=3, w[1..3] = {2,3,5}
  - OUPTUP:
    - 11 = 3+3+5

Összeg	Min-érme-szám	Felhasznált érmék
0	**0**	**-**
1	**∞**	**-**
2	**1**	**2**
3	**1**	**3**
4	**2**	**2, 2**
5	**1**	**5**
6	**2**	**3, 3**
7	**2**	**2, 5**
8	**2**	**3, 5**
9	**3**	**2, 2, 5**
10	**2**	**5, 5**
11	**3**	**3, 3, 5**

S=11, n=3, w[1..3] = {2,3,5}



- A megoldást **felépítjük**, lépésről-lépésre:
  - Megoldjuk a feladatot rendre i = 0, 1, …, S összegekre
    - Adott i-re a minimális pénzérme-számot **tárolja** c[i]
    - Kurrens i-re a c[i]-t, a már rendelkezésre álló c[j]-kből (j<i) számítjuk ki
      - Kell egy **rekurzív képlet**, amely ezt az építkezést matematikailag leírja
  - i=0 esetre a minimális pénzérme-szám 0 (c[0] = 0)
  - Az i összeg, mely j összegekből (j<i) építhető fel egy pénzérme hozzáadásával?
    - c[i] = **min**{c[i-w[k]] + 1; minden k=1..n-re, ahol w[k] <= i}

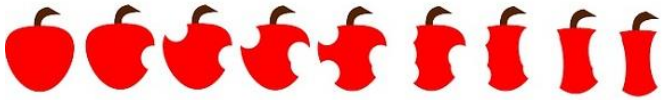


c[11] = 1 + **min** { c[9], c[8], c[6] }

```
c[0] = 0;
for ( i = 1 ; i <= S ; ++i ){
   c[i] = VEGTELEN;
   for ( k = 1 ; k <= n ; ++k ){
      if ( i-w[k] >= 0 && c[i-w[k]] < c[i] ){
         c[i] = c[i-w[k]];
      }
   }
   ++c[i];
}
cout << c[S];
```

- c[S] csak a minimális pénzérme-számot adja meg.
- Hogyan határoznád meg, hogy mely pénzérmék biztosítják ezt az optimális értéket?
  Például: 3 + 3 + 5 = 11.
- És ha az is érdekel, hogy hányféleképpen lehet minimális számú pénzérmével kifizetni S-t? És melyek ezek?

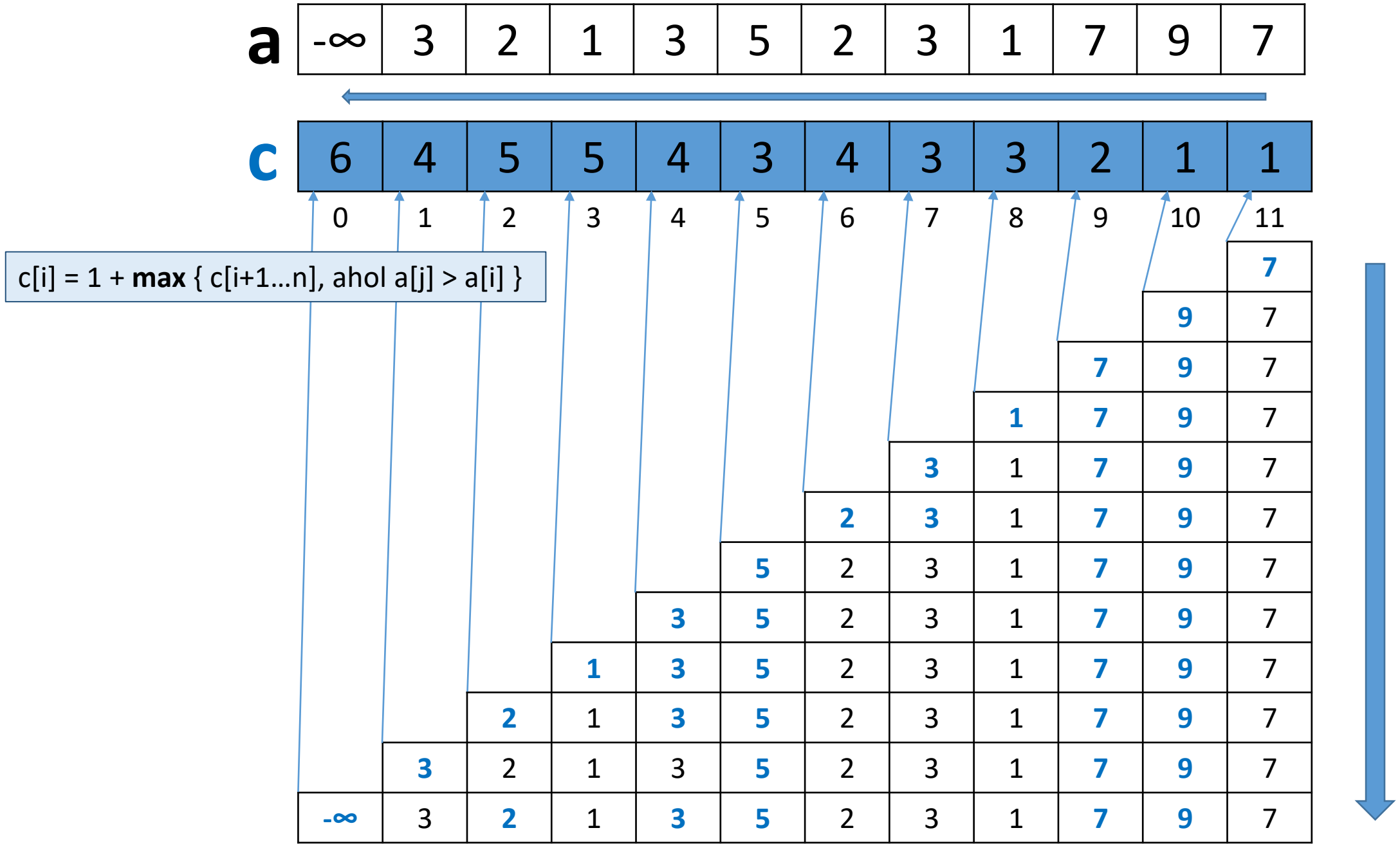
Leghosszabb növekvő részsorozat (szuffixről – szuffixre)

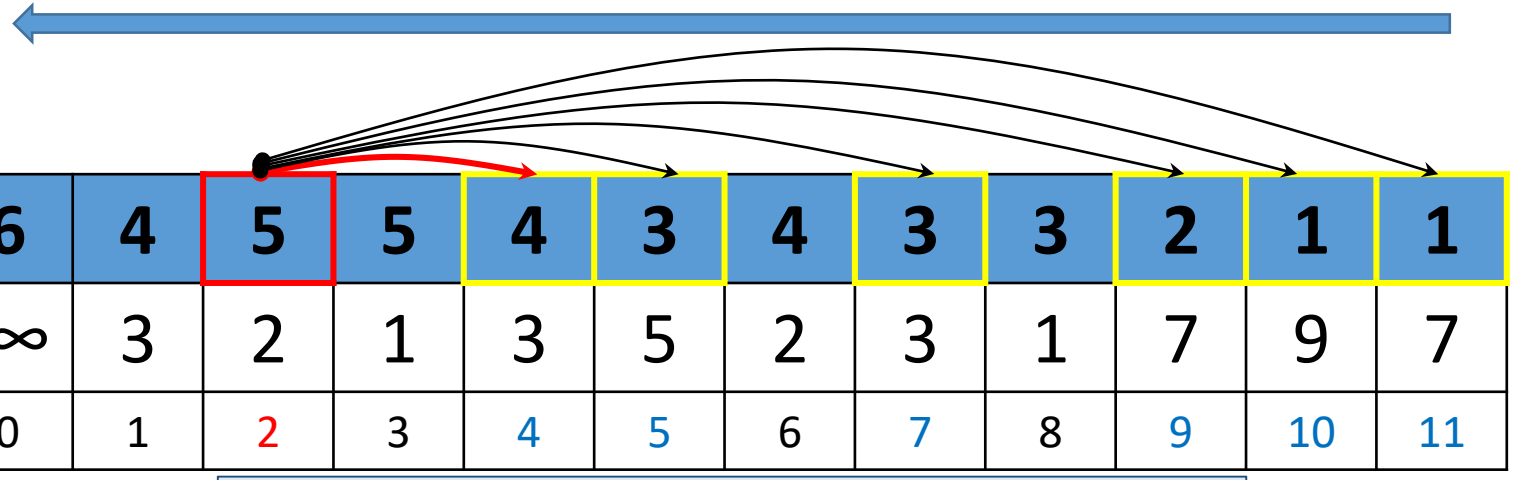


- Adott az a[1..n] számsorozat. Határozzuk meg a leghosszabb **szigorúan** növekvő részsorozatát.

a	-∞	3	2	1	3	5	2	3	1	7	9	7

- Sorra meghatározzuk az a[n], a[n-1], …, a[1], a[0] elemekkel kezdődő leghosszabb növekvő részsorozatokat.
  - Tárolja c[i] az a[i] elemmel kezdődő leghosszabb részsorozat hosszát.
- c[n] = 1
- Minden i-re (n-1)-től 0-ig:
  - c[i] = 1 + **max** { c[j], ahol j=i+1..n és a[j] > a[i] }


a
-∞ 3 2 1 3 5 2 3 1 7 9 7
c
6 4 5 5 4 3 4 3 3 2 1 1
0 1 2 3 4 5 6 7 8 9 10 11
c[i] = 1 + max { c[i+1…n], ahol a[j] > a[i] }



c	**6**	**4**	**5**	**5**	**4**	**3**	**4**	**3**	**3**	**2**	**1**	**1**
a	-∞	3	2	1	3	5	2	3	1	7	9	7
	0	1	2	3	4	5	6	7	8	9	10	11

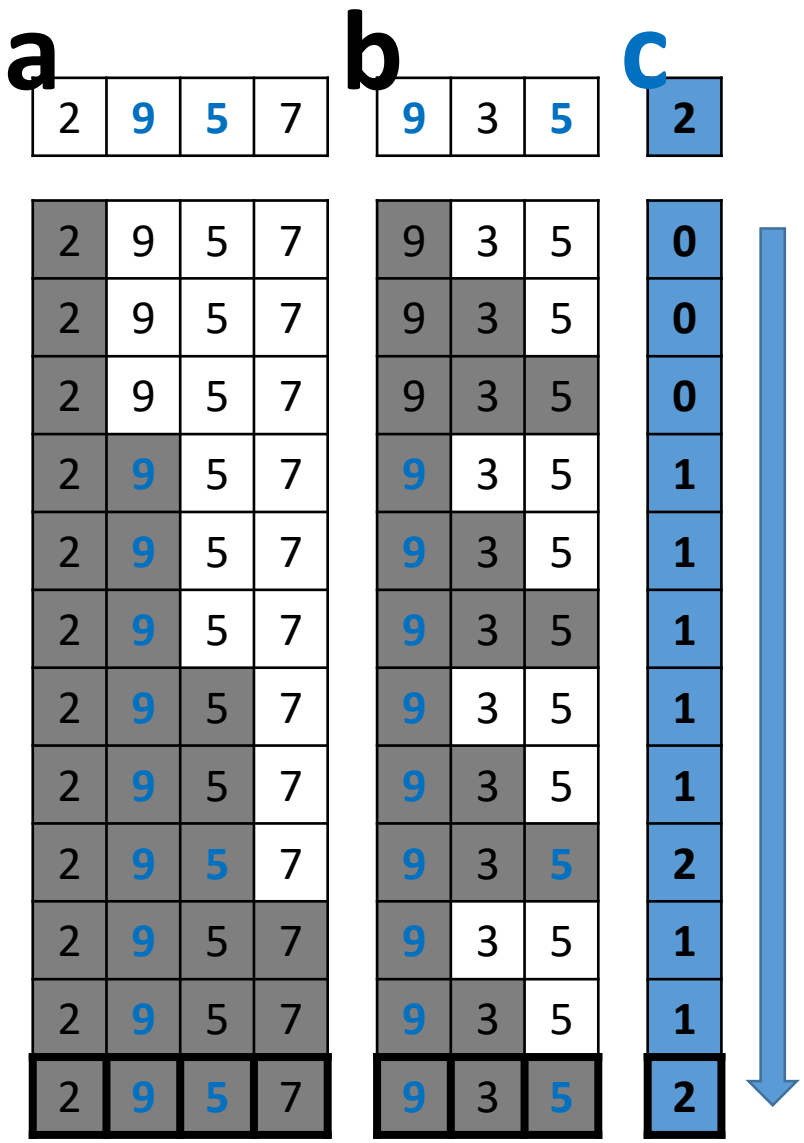
c[2] = 1 + **max** { c[4], c[5], c[7], c[9], c[10], c[11] }

```
c[n] = 1;
for ( i = n-1 ; i >= 0 ; --i ){
   c[i] = 1;
   for ( j = i + 1 ; i <= n ; ++i ){
      if ( a[i] < a[j] && c[j]+1 > c[i] ){
         c[i] = c[j]+1;
      }
   }
}
cout << c[0] - 1;
```

- És ha érdekel a leghosszabb részsorozat is? Például: 2, 3, 5, 7, 9.
- És ha az is, hogy hány ilyen van, és melyek ezek?

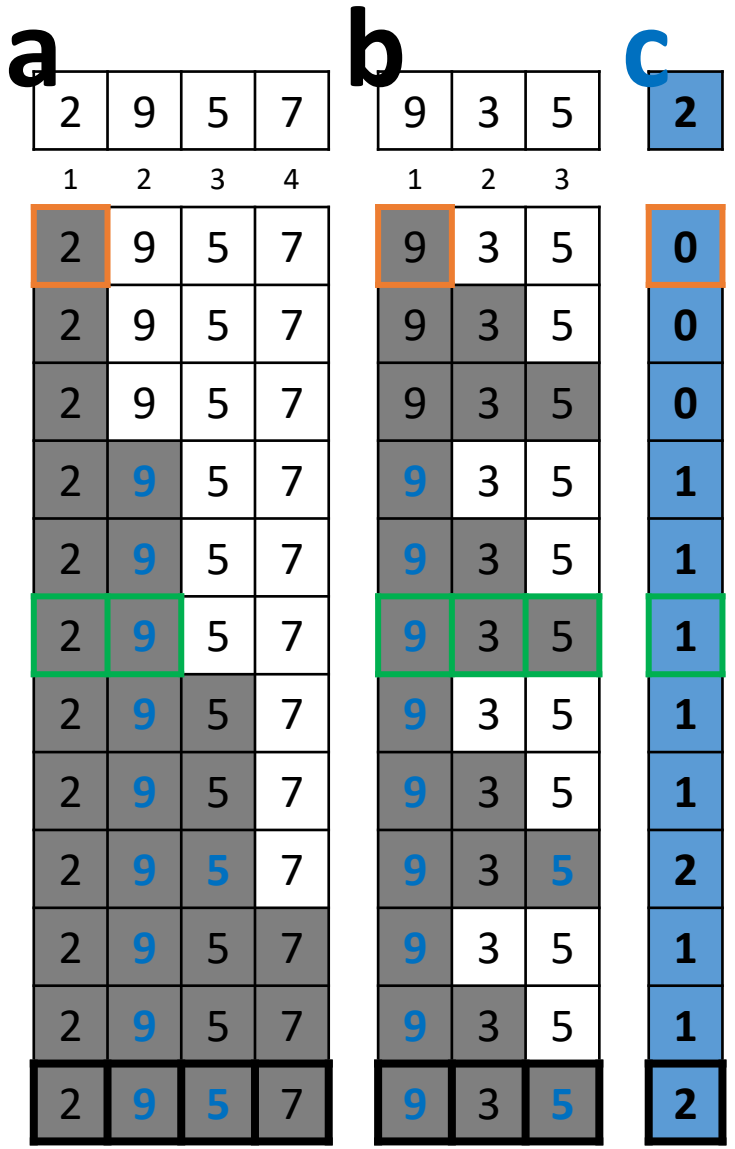
Leghosszabb közös részsorozat (prefix-párról – prefix-párra)

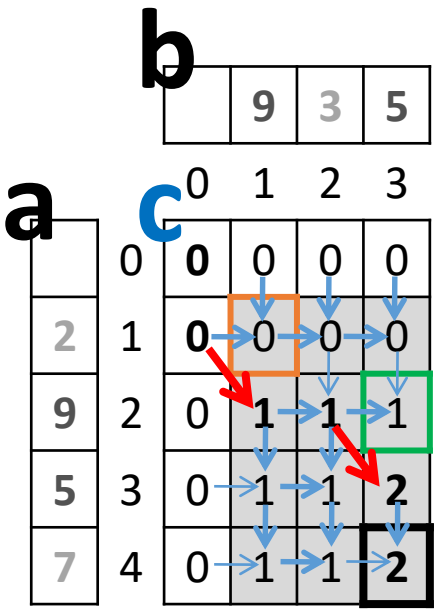
- Adottak az a[1..n] és b[1..m] számsorozatok. Határozzuk meg a leghosszabb **közös** részsorozatot.



- Sorra meghatározzuk minden a[1..i], b[1..j] részsorozat-párra (i=0..n, j=0..m) a leghosszabb közös részsorozatot.
  - Tárolja c[i][j] az a[1..i], b[1..j] részsorozat-pár leghosszabb közös részsorozatának hosszát.
- c[0][j] = c[i][0] = 0
- Minden (i,j)-re (1,1)-től (n,m)-ig:
  - ha a[i] == b[j] (bekerülnek)
    - c[i][j] = 1 +c[i-1][j-1]
  - ha a[i] != b[j] (valamelyik kerül**het** be)
    - c[i][j] = **max** { c[i-1][j], c[i][j-1] }

- És ha érdekel a leghosszabb közös részsorozat is? Például: 9, 5.
- És ha az is, hogy hány ilyen van, és melyek ezek?





Leghosszabb palindrom részsorozat (szakaszról – szakaszra)

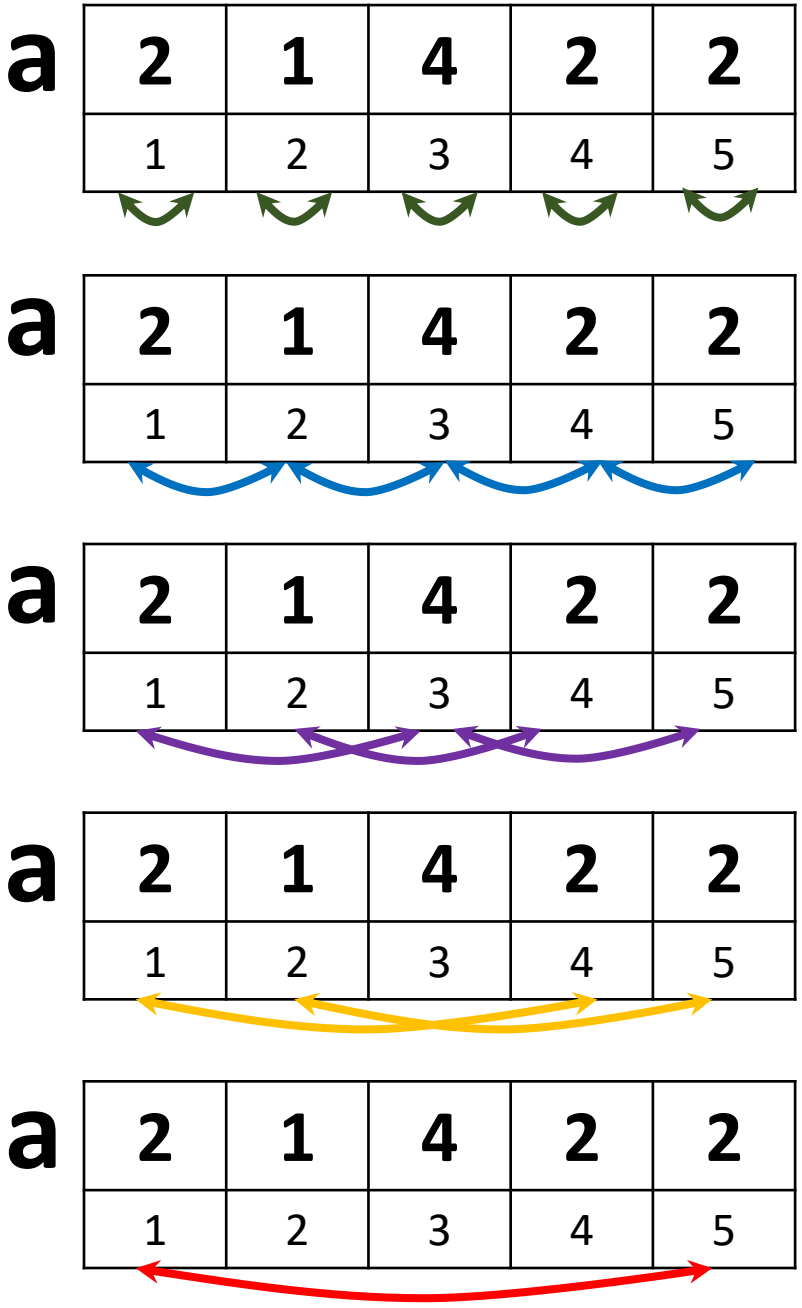
- Határozzuk meg az a[1..n] számsorozat/karakterlánc leghosszabb palindrom részsorozatának **hosszát** (esetleg ezek számát is).
  - INPUT: a[1..5] = {2, 1, 4, 2, 2}
  - OUTPUT: **3**

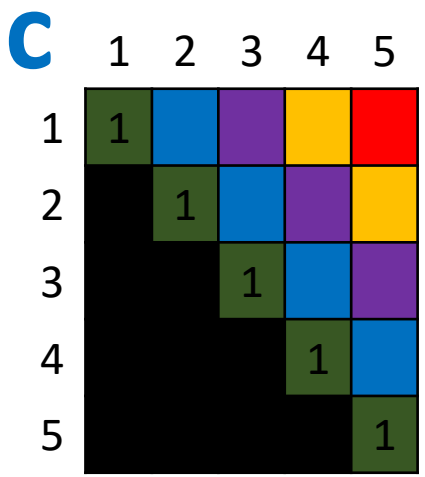
    (2,1,2); (2,1,2); (2,4,2); (2,4,2); (2,2,2)

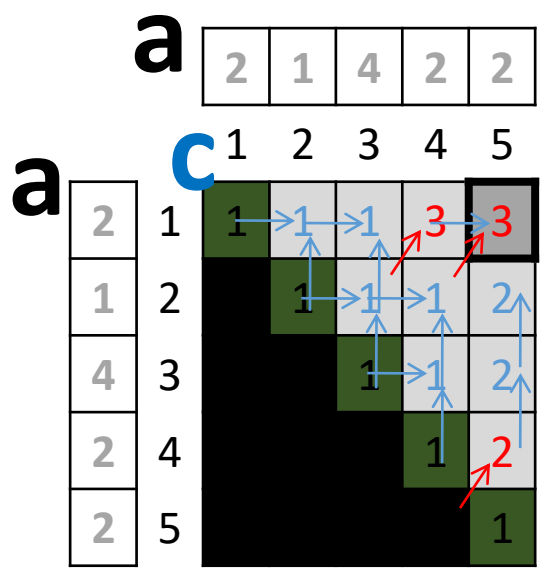
2	1	4	2	2	**1**
2	**1**	4	2	2	**1**
2	1	**4**	2	2	**1**
2	1	4	**2**	2	**1**
2	1	4	2	**2**	**1**
2	**1**	4	2	2	**1**
2	**1**	**4**	2	2	**1**
2	1	**4**	**2**	2	**1**
2	1	4	**2**	**2**	**2**
2	**1**	4	2	2	**1**
2	**1**	4	**2**	2	**1**
2	1	**4**	**2**	**2**	**2**
2	**1**	4	2	2	**3**
2	**1**	4	2	**2**	**2**
2	1	4	2	2	**3**

- Sorra meghatározzuk az 1, 2, …, n hosszú szakaszok tartalmazta leghosszabb palindrom hosszát.
  - n darab 1-hosszú, (n-1) darab 2-hosszú, … 1 darab n-hosszú szakasz van.
  - Tárolja c[i][j] az a[i..j] szakasz leghosszabb palindromjának hosszát.
- c[i][i] = 1, minden i=1..n
- Minden j > i esetén
  - ha a[i] == a[j] (bekerülnek)
    - c[i][j] = 2 + c[i+1][j-1],
  - ha a[i] != a[j] (valamelyik bekerülhet)
    - c[i][j] = max{ c[i][j-1], c[i+1][j] },

- **(2,** 1, 4, 2, **2)**
  - 2, **1, 4, 2,** 2
- **(1,** 4, 2, **2)**
  - 1, 4, 2, 2
  - 1, 4, 2, 2





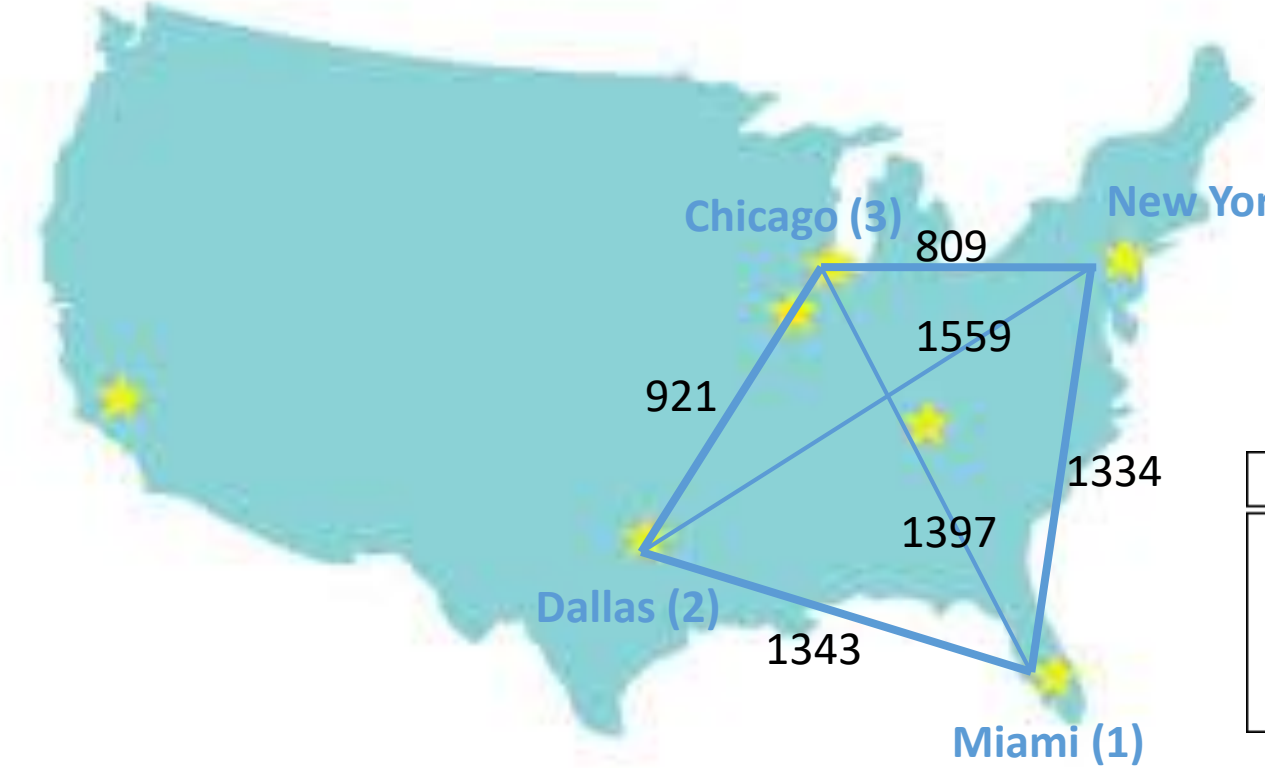



```
for ( h = 1 ; h <= n ; ++h ){
   for ( i = 1 ; i <= n - h + 1 ; ++i ){
      j = i + h - 1;
      //Foglalkozz az i..j szakasszal
      //amelyre vonatkozó értéket c[i][j] tárolja
   }
}
```

- És ha érdekel valamelyik leghosszabb parindrom is?
  Például: 2, 4, 2.
- És, ha az összes leghosszabb érdekel?

Utazókereskedő probléma (részhalmazról – részhalmazra)

- Legyen n város (0,1,…,n-1). Az a[0..n-1][0..n-1] mátrix valamely a[i][j] eleme azt tárolja, hogy mekkora az i és j városok között a közvetlen távolság. Határozzuk meg a legrövidebb Hamilton kör hosszát!



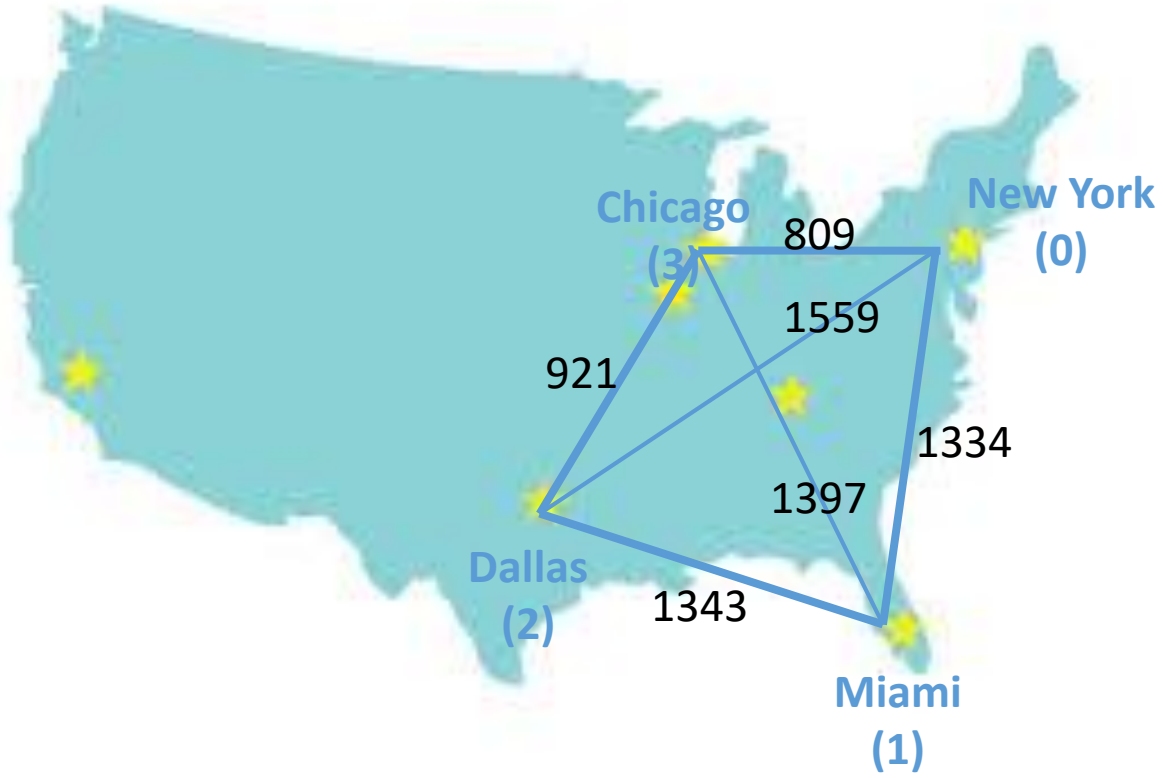
a

	New York	Miami	Dallas	Chicago
New York	—	1334	1559	809
Miami	1334	—	1343	1397
Dallas	1559	1343	—	921
Chicago	809	1397	921	—

Minden i = 1..n-1

- h[i]: a min-út hossza 0-ból i-be,
  a teljes város-halmaz érintésével

Optimum = min { h[i] + a[i][0], ahol i = 1..n-1 }



0	1	2	3	0
0	**1**	2	3	1334
0	1	**2**	3	1559
0	1	2	**3**	809
0	**1**	2	3	1559+1343
0	1	**2**	3	1334+1343
0	**1**	2	3	809+1397
0	1	2	**3**	1334+1397
0	1	**2**	3	809+921
0	1	2	**3**	1559+921
0	**1**	2	3	809+921+1343
0	1	**2**	3	809+1397+1343
0	1	2	**3**	1334+1343+921

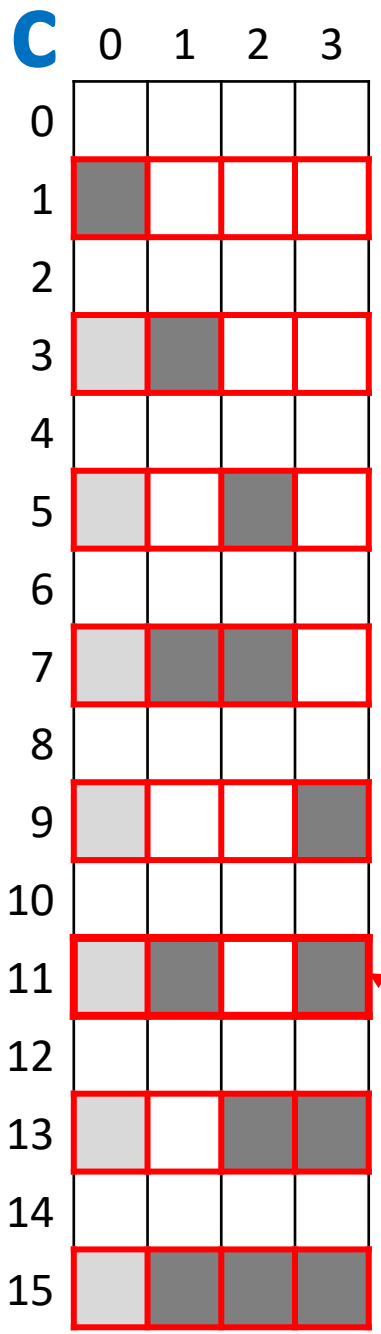
+921 +1397 +1343 +1343 +921 +1392

Meghatározzuk 0-t tartalmazó **összes** részhalmazra

- Mennyi a min-út a részhalmaz egyes elemeihez, a részhalmaz összes elemének érintésével

- El kell tároljunk tömbben, adott részhalmazra vonatkozó értéket…
- Hogyan lehet egy tömb indexe halmaz?
- Egy tömb adott halamzadik eleme???

	3	2	1	0	
{}	0	0	0	0	**0**
{0}	0	0	0	1	**1**
{1}	0	0	1	0	**2**
{0,1}	0	0	1	1	**3**
{2}	0	1	0	0	**4**
{0,2}	0	1	0	1	**5**
{1,2}	0	1	1	0	**6**
{0,1,2}	0	1	1	1	**7**
{3}	1	0	0	0	**8**
{0,3}	1	0	0	1	**9**
{1,3}	1	0	1	0	**10**
{0,1,3}	1	0	1	1	**11**
{2,3}	1	1	0	0	**12**
{0,2,3}	1	1	0	1	**13**
{1,2,3}	1	1	1	0	**14**
{0,1,2,3}	1	1	1	1	**15**



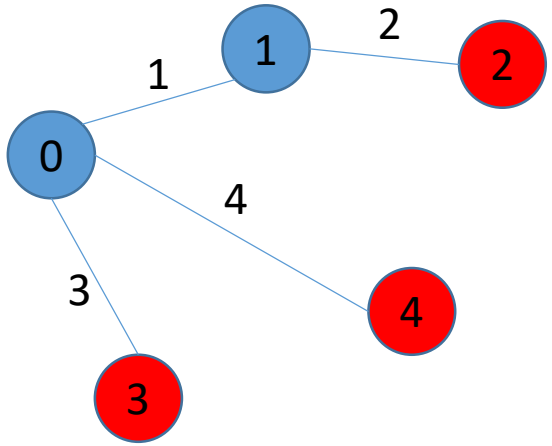
Ha i-ben generálom a $0..2^n-1$ számsort, akkor i bit-mintáiban megjelennek, lexikográfikus sorrendben, a részhalmazok kódjai.

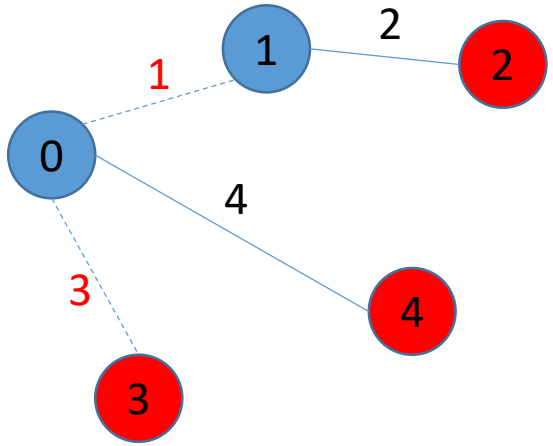
```
for (int s = 0; s < (1<<n); s++){
    if ( !(s & 1) ) { continue; }
    for (int t = 0; t < n ; t++){
        ...
    }
}
```

A **c** tömb {0,1,3}-adik sora

Izoláld a gócpontokat (részfáról –részfára)

- Legyen egy n (1<n<51) pontú fa (körmentes, összefüggő gráf), amelynek csomópontjai vagy kékek vagy pirosak. Ismerve minden kapcsolat felszámolásának költségét, izoláld egymás között a gócpontokat (pirosak), minimális költséggel.





Jó étvágyat!