# DINAMIKUS PROGRAMOZÁS

monadikus(monadic) / poliadikus(polyadic)

soros(serial) / nem-soros(non-serial)

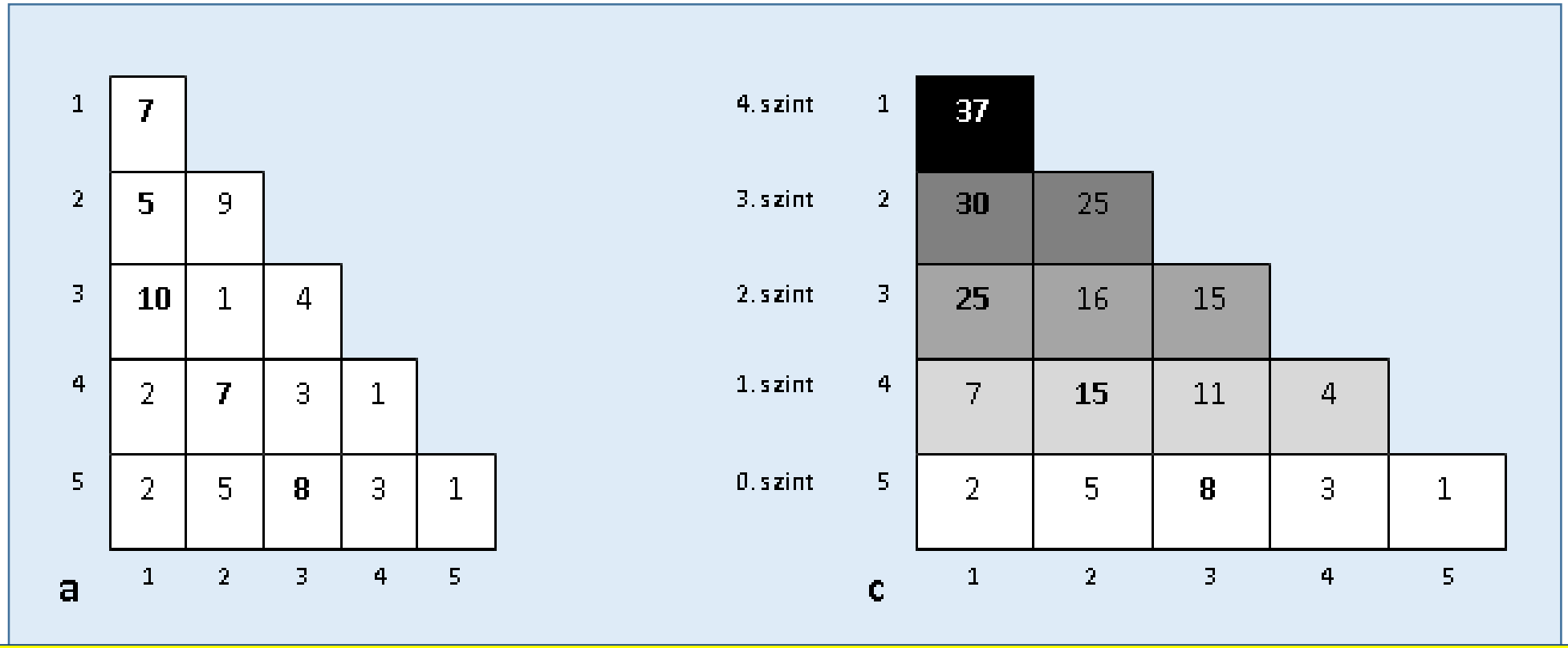
# Szintről szintre, lentről felfele

- A 0. szinten találhatók a triviális részfeladatok, amelyek optimális megoldásai implicite adódnak az input adatokból;
- Az 1. szinten azok a részfeladatok oldhatók meg, amelyek optimális megoldásai közvetlenül adódnak a „triviális optimumokból”;
- Általánosan: A k. szinti részfeladatok optimális megoldásai kizárólag 0..(k-1) szinti optimumoktól függnek;
- A legfelső szinten található, nyilván, az eredeti feladat.

# **monadikus**(monadic) / **poliadikus**(polyadic) **soros**(serial) / **nem-soros**(non-serial)

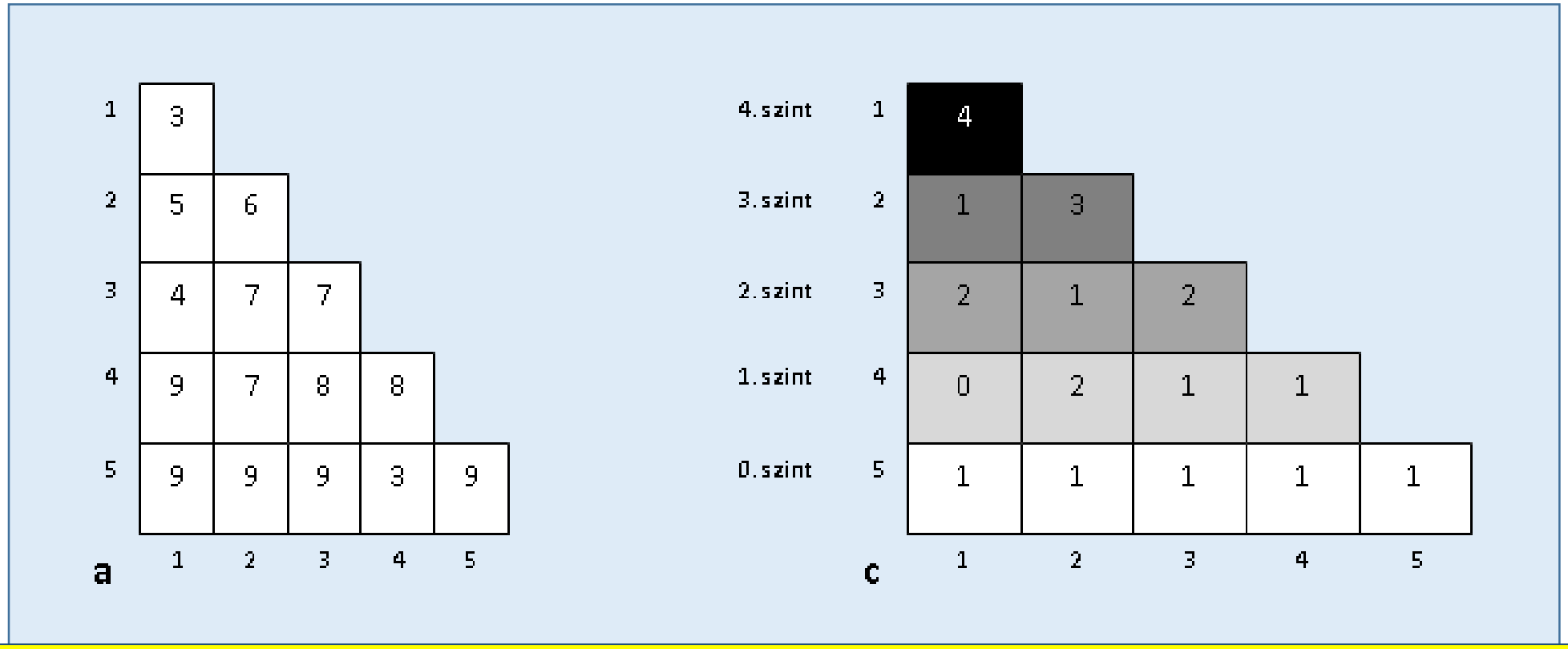
- Ha a k. szinti feladatok megoldásai *kizárólag (k-1). szintiek* megoldásaitól függnek (vagy függhetnek), akkor a feladatot sorosnak nevezzük,
  - különben nem-sorosnak;
- Ha bármely részfeladat optimális megoldásába *egyetlen* alsóbb szinti részfeladat optimuma épül be, akkor monadikus feladatról beszélünk,
  - különben poliadikusról.

# Monadikus–soros
(HÁROMSZÖG-optimalizálás)




```
c[n][j] = a[n][j] , 1≤j≤n;
c[i][j] = max {a[i][j] + c[i+1][j], a[i][j] + c[i+1][j+1]}, 1≤i<n, 1≤j≤i.
```

# HÁROMSZÖG-számlálás

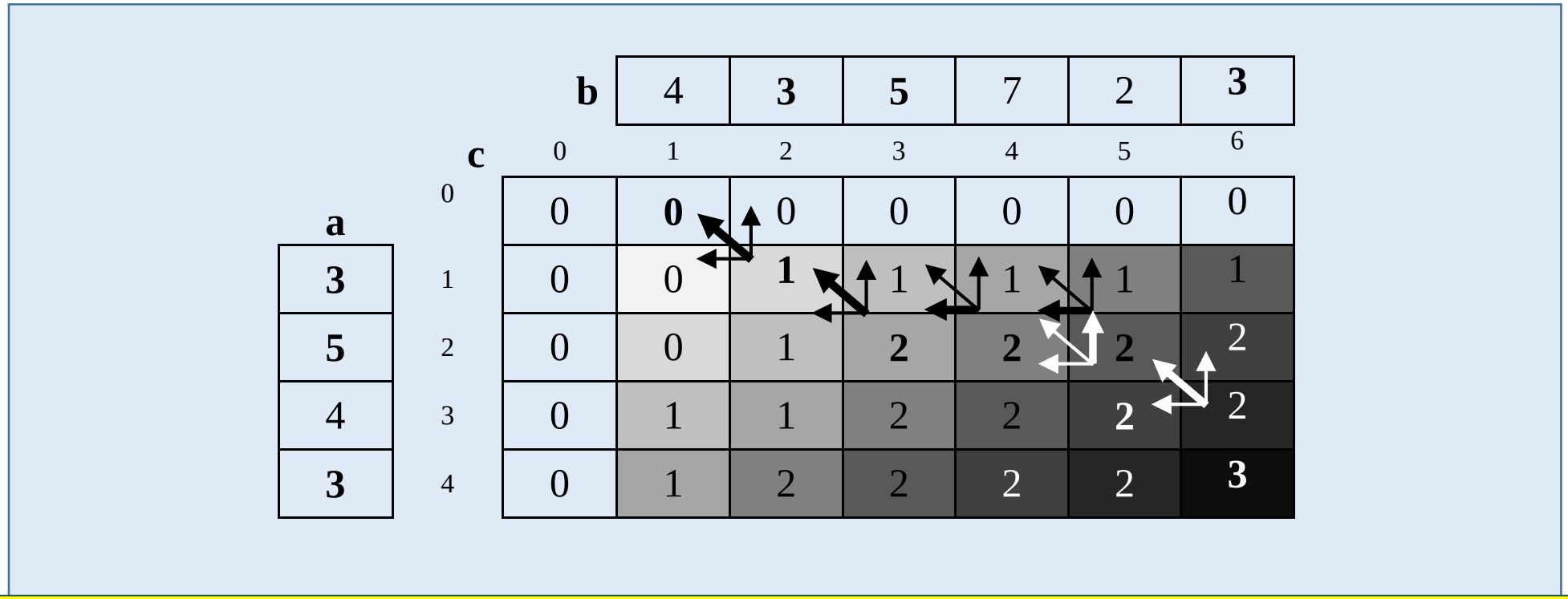



`c[n][j]` = **???**, $1 \le j \le n$;

**`c[i][j]`** = **???**, $1 \le i < n$, $1 \le j \le i$.

# Monadikus – nem-soros

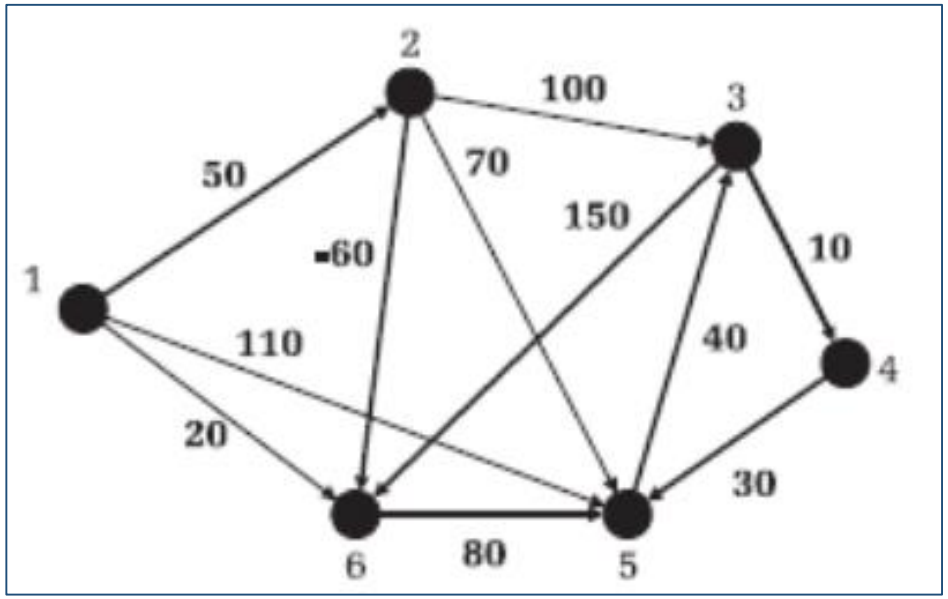
(Határozzuk meg az a[1..n] és  b[1..m] tömbök leghosszabb közös részsorozatát)




```
c[i][0] = 0; c[0][j] = 0, ahol 0≤i≤n, 0≤j≤m.
c[i][j] = c[i-1][j-1] + 1, ha 1≤i≤n, 1≤j≤m, a[i]=b[j].
c[i][j] = max {c[i][j-1], c[i-1][j]}, ha 1≤i≤n, 1≤j≤m, a[i]≠b[j].
```

# Poliadikus – soros (FLOYD algoritmusa)

Legyen n város. Az a[1..n][1..n] tömb a[i][j] cellája azt tárolja, hogy mekkora az i és j városok közti direkt útvonal költsége (nem létező útvonalon a költség ∞; a negatív költségértékek nyereségként tekintendők). Határozzuk meg minden várospár közt a legelőnyösebb utak árát.



# Kövessük a receptet (1)

- *Általános alak:* Az (i,j) várospár között azon legelőnyösebb út meghatározása, amely legfennebb az 1..k köztes állomásokon halad át (k=0,1,..,n).
  - *Triviális részfeladatok*: k=0 (direkt utak; nincsenek köztes állomások).
  - *Eredeti feladat*: k=n (bármely város lehet köztes állomás).
- *Optimum-értékek tömbje:* c[1..n][1..n]
  - a kurrens k-ra tárolja az optimumokat;
  - a k. szinti optimumok felülírhatók a (k+1). szintiekkel

# Kövessük a receptet (2)

- *Rekurzív képlet*: az (i,j) várospár közötti, legfennebb az 1..k köztes állomásokon áthaladó, legelőnyösebb út ($c_k$[i][j] apa-optimum) tartalmazhatja (a $c_{k-1}$[i][k] és $c_{k-1}$[k][j] fiú-optimumokra épül), vagy nem (a $c_{k-1}$[i][j] fiú-optimumra épül), a k. állomást.
  - $c_0$[i][j] = a[i][j], ahol $1 \le i \le n$, $1 \le j \le n$.
  - **$c_k$[i][j]** = min {$c_{k-1}$[i][j], **$c_{k-1}$[i][k] + $c_{k-1}$[k][j]**}, ahol $1 \le i \le n$, $1 \le j \le n$.

(Azért soros, mert a k. szinti optimumok csak (k-1). szintiektől függnek. Azért poliadikus, mert megtörténhet, hogy valamely optimum-érték *két* másik összegéből adódik)

$$c_k[i][j] = \min\{c_{k-1}[i][j],\ c_{k-1}[i][k] + c_{k-1}[k][j]\}$$

$c_0$

| | 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 |

→

$c_1$

| | 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 |

$c_1$

| | 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 |

→

$c_2$

| | 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 |

$c_2$

| | 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 |

→

$c_3$

| | 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 |

$c_3$

| | 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 |

→

$c_4$

| | 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 |

$c_4$

| | 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 |

→

$c_5$

| | 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 |

$c_5$

| | 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 |

→

$c_6$

| | 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 |

# Poliadikus – nem-soros (TÜKÖRSZÓ)

Az s[1..n] karakterláncot bontsunk minimális számú tükörszóra.

- *Általános alak:* Egy s[i..j] karakterlánc-szakasz minimális tükörszóra bontása.
  - *Méretében triviális részfeladatok*: i=j (egy elemű szakaszok).
  - *Minőségükben triviális részfeladatok*: az s[i..j] szakasz tükörszó.
  - *Eredeti feladat*: i=1, j=n.
- *Optimum-értékek tömbje:* c[1..n][1..n] (főátló és főátló feletti háromszög).

- *Rekurzív képlet*: az s[i..j] nem szimmetrikus szakasz, egy vágással, j-i féleképpen bontható s[i..k], s[k+1..j] alakú párosra, ahol k=i,j–1.
  - c[i][i] = 1, 1≤i≤n;
  - c[i][j] = 1, ha s[i..j] tükörszó;
  - **c[i][j]** = min { **c[i][k] + c[k+1][j]**, minden **k=i,j–1**}, ha s[i..j] nem tükörszó.

| | | b | b | a | b | a |
|---|---|---|---|---|---|---|
| | **c** | 1 | 2 | 3 | 4 | 5 |
| b | 1 | 1 (b) | 1 (bb) | 2 (bba) | 2 (bbab) | 2 (bbaba) |
| b | 2 | 0 | 1 (b) | 2 (ba) | 1 (bab) | 2 (baba) |
| a | 3 | 2 | 2 | 1 (a) | 2 (ab) | 1 (aba) |
| b | 4 | 1 | 0 | 0 | 1 (b) | 2 (ba) |
| a | 5 | 2 | 2 | 0 | 0 | 1 (a) |

# Iteratívan

```
minden i = 1,n végezd // méretükben is triviális optimumok
  c[i][i] = i
vége minden
minden i = n-1,1 végezd // lentről felfele
  minden j = i+1,n végezd // balról jobbra
    ha tükörszó(s,i,j) akkor // minőségükben triviális optimumok
      c[i][j] = 1
      c[j][i] = 0 // nem igényel vágást
    különben // nem triviális optimumok
      c[i][j] = n // „nagy” kezdőérték a minimumszámoláshoz
      minden k = i,j-1 végezd
        ha c[i][k] + c[k+1][j] < c[i][j] akkor
          c[i][j] = c[i][k] + c[k+1][j]
          c[j][i] = k // eltárolom a vágás helyét
        vége ha
      vége minden
    vége ha
  vége minden
vége minden
ki: c[n][n]
```

# Rekurzívan

```
Optimális_megoldás(s[],c[][],i,j)
  ki: '('
  ha i == j vagy c[j][i] == 0 akkor // s[i..j] tükörszó
    ki: s[i..j]
  különben
    Optimális_megoldás(s[],c[][],i,c[j][i])
    Optimális_megoldás(s[],c[][],c[j][i]+1,j)
  vége ha
  ki: ')'
vége Optimális_megoldás
```