Dinamikus programozás

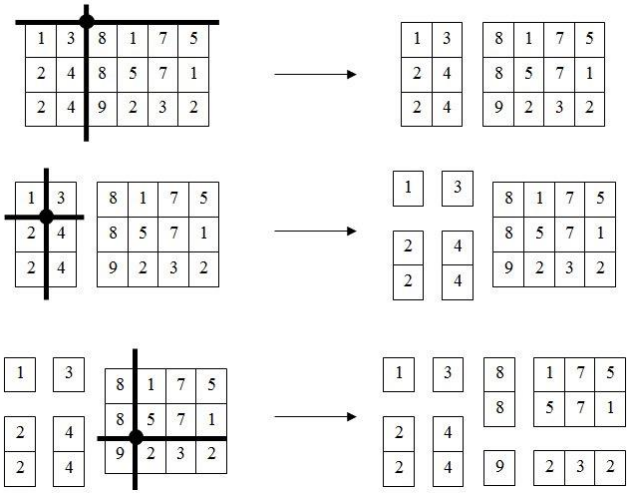
II. rész

Cikksorozatunk előző részében egy 5 lépéses módszert ajánlottunk dinamikus programozásos feladatok megoldásához:

1. Meghatározzuk a részfeladatok általános, paraméteres alakját. (Lévén szó „különböző méretű" hasonló részfeladatok egy családjáról, nyilván beszélhetünk ezek általános alakjáról)
2. Eldöntjük, hogy hol tároljuk a részfeladatok optimális megoldásait képviselő optimum értékeket. (Az optimalizálandó célfüggvény optimum-értékeit a részfeladatokra vonatkoztatva)
3. Meghatározunk egy rekurzív képletet, amely matematikailag leírja az optimális építkezés módját. (Mi a módja, hogy a már rendelkezésre álló „fiú-optimumokból" „apa-optimumokat" építsünk)
4. Implementáljuk az iteratív algoritmust, amely a rekurzív képlet alapján („lentről-felfelé irányba") feltölti az optimum-értékek tömbjét. (A „triviális szegélyen" lévő celláktól „optimum-hidat" építünk az „ellenkező oldalon" található célcellához)
5. Az optimum-tömbből kiolvassuk („fentről-lefelé irányba") az optimális döntéssorozatot (amely az optimális megoldást eredményezi). (Meghatározzuk az „optimum-híd" célcellába vezető „optimális útját")

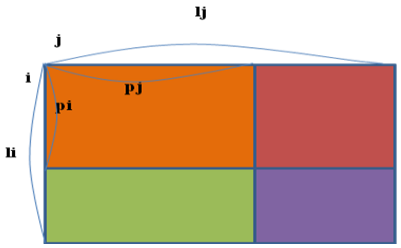
Emlékeztetőül, lássuk egy újabb feladatra alkalmazva a fenti módszert (ebben a cikkben C nyelv közeli jelöléseket használunk a tömbökre vonatkozóan). *Legyen egy nxm méretű téglalap alakú mozaikkép, amelyet egy nxm méretű mátrix kódol ($1 \le n,m \le 32$). Az (i,j) pozíciójú tömbelem az(i,j) pozíciójú képelem színkódját jelenti (0..255). A mozaikképet egy golyósorozat éri, amely téglalap alakú darabokra töri (lásd az 1. ábrát). Minimum hány, középpontjaikra nézve szimmetrikus színösszetételű téglalapra (középpontra szimmetrikus pozíciójú képelem-párok színe azonos) darabolható a kép? (Sapientia-ECN programozás verseny, F feladat, 2011; mitis.ro/ecn)*

1. *Általános alak*:(i,j,li,lj)-feladat, azaz az (i,j) sarkú lixlj oldalhosszú téglalap optimális feldarabolása.
   - *Triviális részfeladatok*: szimmetrikus színösszetételű résztéglalapok (az 1x1-es méretűek értelemszerűen azok).
   - *Eredeti feladat*: (1,1) sarkú nxm oldalhosszú téglalap.
2. *Optimum-értékek tömbje*:c[1..32][1..32][1..32][1..32].
3. *Rekurzív képlet* (lásd a 2. ábrát):
   c[i][j][li][lj] = min {c[i][j][pi][pj]+ c[i+pi][j][li-pi][pj]+ c[i][j+pj][pi][lj-pj]+ c[i+pi][j+pj][li-pi][lj-pj]}
   pi=0,li-1
   pj=0,lj-1



1. ábra

Egy példa mátrix minimális számú szimmetrikus elemre való szétdarabolására 3 lépésben



2. ábra

Az (i,jli,lj) téglalap egy lehetséges felbontása

4. *Iteratív algoritmus* (C nyelven) a c és b tömbök feltöltésére (a b tömb az optimális döntéseket tárolja):

```
for(li=1;li<=n;li++) //minden rész-téglalapra méretük szerint növekvő sorrendben
 for(lj=1;lj<=m;lj++)
   for(i=0;i<=n-li;i++)
     for(j=0;j<=m-lj;j++)
       if (szimetrikus(i,j,li,lj))
         {c[i][j][li][lj]=0;b[i][j][li][lj].pi=-1; b[i][j][li][lj].pj=-1;}
       else
       {
        min=n*m;
        for(pi=0;pi<li;pi++) //a kurrens téglalap lehetséges feldarabolásai egy lövéssel
          for(pj=0;pj<lj;pj++)
          {
```

```
            if(!(pi==0 && pj==0))
             {
              x=c[i][j][pi][pj]+c[i+pi][j][li-pi][pj]+c[i][j+pj][pi][lj-
pj]+c[i+pi][j+pj][li-pi][lj-pj];
            if (x<min) {min=x; mpi=pi;mpj=pj;}
             }
            }
         a[i][j][li][lj]=min+1; b[i][j][li][lj].pi=mpi; b[i][j][li][lj].pj=mpj;
        }
```

5. Az optimális felbontás kiíratása maradjon az olvasóra.

Dinamikus programozásos feladatok osztályozása

Ahogy az eddigi két példa is érzékeltette, a dinamikus programozásos feladatok igen sokfélék lehetnek. Hogyan lehetne mégis, egy viszonylag átfogó képet kapni róluk? Célszerű lehet követni az alábbi osztályozást: (1) monadikus(monadic)/poliadikus(polyadic); (2) soros(serial)/nem-soros(non-serial).

Valószínű, hogy az eddigiek alapján máris érzékelte a kedves olvasó, hogy adinamikus programozásos építkezés szintről szintre halad, lentről felfelé. A 0. szinten találhatók a triviális részfeladatok, amelyek optimális megoldásai implicite adódnak az input adatokból. Első 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.

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. Az alábbiakban felsorolunk mindenik kategóriából egy-egy példát, megtesszük a módszerünk szerinti 1,2,3 lépéseket, és arra buzdítjuk az olvasót, próbálja meg maga implementálni a 4,5 lépéseket. Mivel klasszikus feladatokról van szó, ezért google-barát is sokat segíthet, az esetleges zsákutcákból kikerülni.

Monadikus–soros (Nemzetközi Informatika Olimpiász, Svédország, 1994): Határozzuk meg a csúcsból alapra vezető legjobb utat (amely mentén a legnagyobb az összeg), ha a megengedett irányok le vagy átlósan-jobbra (lásd a 3. ábrát).

1. *Általános alak:* (i,j) pozícióból induló alapra vezető legjobb út meghatározása.
   - *Triviális részfeladatok*: (n,j) pozíciókból induló legjobb utak meghatározása.
   - *Eredeti feladat*: (1,1) induló legjobb út meghatározása.
2. *Optimum-értékek tömbje:* c[1..n][1..n].
3. *Rekurzív képlet*: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.

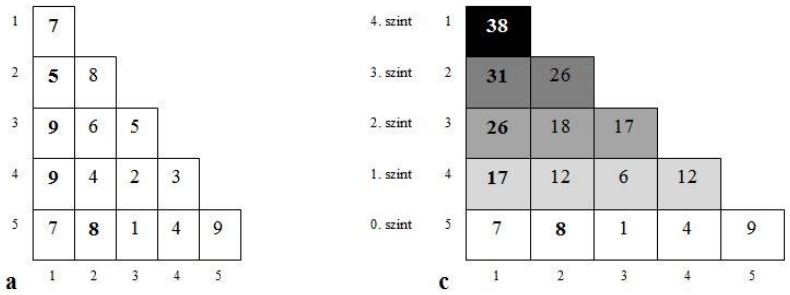
Monadikus – nem-soros: Határozzuk meg az a[1..n] és b[1..m] tömbök leghosszabb közös részsorozatát.

1. *Általános alak*:aza[1..i] és b[1..j] tömbszakaszok leghosszabb közös részsorozatának meghatározása.
   - *Triviális részfeladatok*: i=0 vagy j=0 (valamelyik részsorozat üres).
   - *Eredeti feladat*: i=n, j=m.
2. *Optimum-értékek tömbje:* c[0..n][0..m].
3. *Rekurzív képlet*:

   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]}, 1≤i≤n, 1≤j≤m, a[i]≠b[j].



3. ábra

Példatömbök a háromszög feladathoz (a[1..5][1..5], c[1..5][1..5]).
A feladat, azért soros, mert a k. szinti optimumok csak (k-1). szintiektől függnek.

		b	4	3	5	7	2	3
	c	0	1	2	3	4	5	6
a	0	0	**0**	0	0	0	0	0
3	1	0	0	**1**	1	1	1	1
5	2	0	0	1	**2**	**2**	**2**	2
4	3	0	1	1	2	2	**2**	2
3	4	0	1	2	2	2	2	**3**

4. ábra

Leghosszabb közös részsorozat: 3,5,3. A 0. szint háttere fehér.
Az „egyszínű" átlókon található cellák ugyanazon szinten levő részfeladatokat képviselnek.
A feladat azért „nem-soros", mert a k. szinti optimális megoldások függhetnek, mind (k-1). mind (k-2). szinti optimumoktól
(lásd a példaként berajzolt nyilakat).

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 út hossza (nem létező út hossza ∞). Határozzuk meg minden várospár közt a legrövidebb utakat.

1. *Általános alak:* Az (i,j) várospár között azon legrövidebb út meghatározása, amely legfennebb az 1..k köztes állomásokon halad át. A szintről-szintre való egyszerűtől bonyolult felé haladás a k növekedésével (k=0,1,..,n) jár kéz a kézben.
   - *Triviális részfeladatok*: k=0 (direkt élek; nincsenek köztes állomások).
   - *Eredeti feladat*: k=n (bármely város lehet köztes állomás).
2. *Optimum-értékek tömbje:* c[1..n][1..n] (a kurrens k-ra tárolja az optimumokat; a k. szinti optimumok felülírhatók a (k+1). szintiekkel).
3. *Rekurzív képlet*:

   $c_0[i][j] = a[i][j]$, 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)

Poliadikus – nem-soros: Mátrixsorozat optimális összeszorzása.

Ez egy klasszikus feladat, amely dinamikus programozásos megoldásának felkutatását az olvasóra bízzuk. Miért tekinthető poliadikusnak, illetve nem-sorosnak ez a példa?

Reméljük, hogy e kis algoritmika kúra hozzájárult ahhoz, hogy a kedves olvasó tisztábban lássa a programozás világának e „dinamikus területét". Ha kedvet kapott a „mélyüléshez", akkor miért ne nézne máris utána az első rész bevezetőjében említett valós feladatok dinamikus programozásos vetületeinek.

Kátai Zoltán,
Sapientia-EMTE, Matematika-informatika Tanszék

Kémiatörténeti évfordulók

III. rész

290 éve született

Bayen, Pierre: 1725. február 7-én Chalons-sur-Marne-on (Franciaország). Gyógyszerészetet és kémiát tanult. Vizsgálta a franciaországi ásványvizeket. Felfedezte a higany-fulminátot (1774). A vörös higany-oxid hevítésével oxigént nyert, de azt nem tekintette kémiai elemnek. Kevéssel Lavoisier előtt már ellenezte a flogiszton-elméletet. 1798. febr. 19-én halt meg.