Algoritmusok gyakorlat, 2020

4. hét, szeptember 28.

4. gyakorlat

A rekurzív algoritmusok sokszor azért lassúak, mert bizonyos részproblémákat többször is kiszámolnak. Ezt tudjuk elkerülni azzal, ha az egyszer már kiszámolt részmegoldásokat eltároljuk es újra felhasználjuk.
A több időt több tárra ,,cseréljük": a rekurzióval egyébkent exponenciális idő alatt megoldható problémákat polinomidőben megoldhatjuk. Ehhez viszont el kell tároljuk a részproblémák megoldását.

4. gyakorlat

Részfeladatokra való bontással oldjuk meg a problémát.

DP (dinamikus programozás)– és nem oszd-meg-és-uralkodj – ha a részproblémák nem függetlenek, azaz *közös részproblémáik* vannak (optimalizálási feladatoknál tipikus!).

Alapgondolat: a már megoldott részproblémák optimális megoldásának értékét *memorizáljuk* és ha még egyszer fel kell használni felidézzük

4. gyakorlat

Jelölje P(n) azt a számot, ahányféleképpen mehetünk fel egy n lépcsőfokból álló lépcsőn, ha egyszerre csak 1 vagy 2 lépcsőfokot léphetünk. Adjunk egy dinamikus programozási eljárást a P(n) érték kiszámítására!

P(8) = ?

4. gyakorlat

P(1) = 1 (ha egy lépcső van, egyféleképpen mehetünk)
P(2) = 2 (ha két lépcső van, vagy kétszer egyet lépünk, vagy egyszer kettőt)
P(n) = P(n-1) + P(n-2), ha n >=3 (utolsó lépésként egyet vagy kettőt léphetünk)

i	1	2	3	4	5	6	7	8
P(i)	1	2	3	5	8	13	21	34

https://visualgo.net/en/recursion

4. gyakorlat

```
static int findStep(int n)
{
    int f[] = new int[n];
    f[0] = 1;
    f[1] = 2;
    for (int i = 2; i < n; i++)
       f[i] = f[i-1] + f[i-2];
      return f[n];
 }
```

4. gyakorlat

Jelölje R(k; n) azt a számot, ahányféleképpen eljuthatunk egy k x n méretű sakktábla bal alsó sarkából a jobb felső sarkába, ha csak a jobbra vagy a felfelé szomszédos mezőre léphetünk. Adjunk meg két dinamikus programozási eljárást (egyet négyzetes, egyet lineáris táblázatkitöltéssel) az R(k; n) érték kiszámítására!

R(5;4) = ?

4. gyakorlat

R(k; 1) = 1 (csak felfele mehetünk)
R(1; n) = 1 (csak jobbra mehetünk)
R(k; n) = R(k; n-1)+R(k-1; n), ha n, k >= 2

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

4. gyakorlat

5	1	5	15	35
4	1	4	10	20
3	1	3	6	10
2	1	2	3	4
1	1	1	1	1
	1	**2**	**3**	**4**

4. gyakorlat

```
static int numberOfPaths(int k, int n)
 {
    int count[][] = new int[k][n];
    for (int i = 0; i < k; i++)
       count[i][0] = 1;
    for (int j = 0; j < n; j++)
       count[0][j] = 1;
     for (int i = 1; i < k; i++) {
       for (int j = 1; j < n; j++)
            count[i][j] = count[i - 1][j] + count[i][j - 1];
    }
    return count[k - 1][n - 1];
 }
```

4. gyakorlat

```
static int numberOfPaths(int k, int n)
 {
    int count[] = new int[n];
    for (int j = 0; j < n; j++)
       count[j] = 1;
     for (int i = 1; i < k; i++) {
       for (int j = 1; j < n; j++)
            count[j] = count[j] + count[j - 1];
    }
    return count[n - 1];
 }
```

4. gyakorlat

Oldjuk meg a pénzváltási feladat alábbi változatát a következő értékekre:

$P_1=1$, $P_2=5$, $P_3=7$, $F=11$

Adottak különböző pénzérmék korlátlan mennyiségben és egy összeg. Adjuk meg hogy összesen hány **különböző** módon lehet felváltani az összeget a megadott pénzérmékkel!

4. gyakorlat

1. eset: P_n-t bevesszük és az F-P_n összes felváltási lehetőségét tekintjük.
2. P_n-t nem vesszük bele és F összes felváltási lehetőségét tekintjük a többi érmével.

$$C(P,n,F)=C(P,n,F-P_n)+C(P,n-1,F)$$

4. gyakorlat

$$C(P,n,F)=C(P,n,F-P_n)+C(P,n-1,F)$$

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

4. gyakorlat

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

4. gyakorlat

```
public static int count( int P[], int n, int F )
{
    int table[]=new int[F+1];
    table[0] = 1;
    for(int i=0; i<n; i++)
        for(int j=P[i]; j<=F; j++)
            table[j] += table[j-P[i]];
     return table[n];
}
```

4. gyakorlat

Adott egy k x n-es tábla. Minden mezőre meg van adva egy c_{ij} pozitív szám, ami a mezőről begyűjthető érték. Egy játékos a bal alsó sarokból szeretne eljutni a jobb felső sarokba úgy, hogy csak jobbra és felfelé léphet a szomszédos mezőre. Az útja során összegyűjtheti a mezőkről az értékeket. Mennyi értéket tudunk összeszedni maximálisan? Hogyan határoznánk meg ezt az utat, amin a maximális értéket gyűjthetjük?

1 2 4 2
2 5 1 3
1 4 4 2

4. gyakorlat

```
private static int minCost(int cost[][], int k, int n)
{
      int tc[][]=new int[k+1][n+1];
       tc[0][0] = cost[0][0];
       for (int i = 1; i <= k; i++)
          tc[i][0] = tc[i-1][0] + cost[i][0];
       for (int j = 1; j <= n; j++)
          tc[0][j] = tc[0][j-1] + cost[0][j];
       for (int i = 1; i <= k; i++)
          for (int j = 1; j <= n; j++)
              tc[i][j] = min( tc[i-1][j],tc[i][j-1]) + cost[i][j];

      return tc[k][n];
}
```

4. gyakorlat

Keressük meg azon mátrixok összeszorzásának optimális zárójelezését, ahol a méretek sorozata

(5, 10, 3, 12, 5, 50, 6)!

A_1:5x10 $A_1(A_2A_3)$=5x10x12+10x3x12 = 960

A_2:10x3 $(A_1A_2)A_3$=5x10x3+5x3x12 = 330

A_3:3x12 Lehetséges zárójelezések száma:

A_4:12x5

A_5:5x50

A_6:50x6

$$P(n) = \begin{cases} 1, & ha \;\; n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k), & ha \;\; n \geq 2 \end{cases}$$

4. gyakorlat

Részproblémákra bontás:

$A_i \ldots A_j \longrightarrow A_i \ldots A_k \times A_{k+1} \ldots A_j$

$m_{ij} = m_{ik} + m_{k+1j} + p_{i-1} p_k p_j$

$$m_{ij} = \begin{cases} 0, & \text{ha } i = j \\ \min\limits_{i \le k < j} \{ m_{ik} + m_{k+1,j} + p_{i-1} p_k p_j \} & \text{ha } i < j \end{cases}$$

4. gyakorlat

(5, 10, 3, 12, 5, 50, 6)

$$m_{ij} = \begin{cases} 0, & \text{ha } i = j \\ \min\limits_{i \le k < j} \{m_{ik} + m_{k+1,j} + p_{i-1} p_k p_j\} & \text{ha } i < j \end{cases}$$

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

4. gyakorlat

(5, 10, 3, 12, 5, 50, 6)

$$m_{ij} = \begin{cases} 0, & \text{ha } i = j \\ \min\limits_{i \le k < j} \{m_{ik} + m_{k+1,j} + p_{i-1} p_k p_j\} & \text{ha } i < j \end{cases}$$

6	0					
5	1500	0				
4	1860	3000	0			
3	1770	930	180	0		
2	1950	2430	330	360	0	
1	2010	1655	405	330	150	0
	6	**5**	**4**	**3**	**2**	**1**

4. gyakorlat

```
static int MatrixChainMultiplication(int p[], int n)
{
      int m[][] = new int[n][n];
      for (int i = 1; i < n; i++)
         m[i][i] = 0;
       for (int l = 2; l < n; l++) {
         for (int i = 1; i < n - l + 1; i++) {
            j = i + l - 1;
           m[i][j] = Integer.MAX_VALUE;
            for (int k = i; k < j ; k++) {
               int q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
               if (q < m[i][j])
                  m[i][j] = q;
            }
         }
      }
      return m[1][n - 1];
}
```

4. gyakorlat

Határozzuk meg az

(a, b, b, a, b, a, b, a)

és a

(b, a, b, a, a, b, a, a, b)

sorozatok leghosszabb közös részsorozatát!

Házi feladat

4. gyakorlat

Magyarázatok, megoldások:

Gelle Kitti: Algoritmusok és adatszerkezetek I. gyakorlati jegyzet.

http://www.inf.u-szeged.hu/~kgelle/sites/default/files/upload/alga-gyak-04.pdf