Fibonacci számok

Fibonacci 1202-ben vetette fel a kérdést: hány nyúlpár születik n év múlva, ha feltételezzük, hogy

- az első hónapban csak egyetlen újszülött nyúl-pár van;
- minden nyúlpár, amikor szaporodik egy újabb nyúlpárt hoz létre;
- a nyulak a születésük után a következő két évben egyszer-egyszer szaporodnak

```
Fibonacci(n)
  If n=1 or n=2 return 1
  Else return Fibonacci(n-1)+Fibonacci(n-2)
```

Az algoritmus az n és n-1 argumentumra 1-szer, az n-2-re 2-szer (n és n-1), az n-3-ra 3-szor (n-1 és 2-szer n-2), az n-4-re 5-ször (2-szer n-2 és 3-szor n-3) lesz végrehajtva.

Rekurzíómemórizálás

```
FibonacciM(n)
  If T[n]> 0 then return T[n]
  Else If n=1 or n=2 then T[n]:=1
                          return T[n]
  Else T[n]:=FibonacciM(n-1)+FibonacciM(n-2)
       return T[n]
```

Dinamikus programozással

```
FibonacciDP(n)
    T[1]:=1
    T[2]:=2
    for i=3 to n
       T[i]:=T[i-1]+T[i-2]
    return T[n]
```

Partíció probléma

Az n természetes szám egy partíciója olyan $\pi = (a_1, \ldots, a_k)$ sorozat, amelyre teljesül:
- $a_1 \geq a_2 \geq \cdots \geq a_k > 0$
- $\sum_{i=1}^{k} a_i = n$

Jelölje $P(n)$ az n szám összes partíciójának számát.

Partíciószám számító algoritmus

Jelölje $P2(n,k)$ n azon partícióinak számát, amelyben minden elem legfeljebb k.
Ekkor a következő összefüggések teljesülnek:

- $P2(1,k) = 1, P2(n,1) = 1$
- $P2(n,n) = 1 + P2(n,n-1)$
- $P2(n,k) = P2(n,n)$ ha $n < k$
- $P2(n,k) = P2(n,k-1) + P2(n-k,k)$ ha $k < n$

- A megoldás: $P(n) = P2(n,n)$

Partíció négyzetes táblázatkitöltéssel

A rekurziót teljesen kiküszöbölhetjük táblázat-kitöltéssel. A T2[n,k] táblázatelem tartalmazza a P2(n,k) részprobléma megoldását.

A táblázat első sora azonnal kitölthető, mert $P2(n,1) = 1$. Olyan kitöltési sorrendet keresünk, hogy minden (n,k), $k > 1$ részprobléma kiszámítása esetén azok a részproblémák, amelyek szükségesek $P2(n,k)$ kiszámításához, már korábban kiszámítottak legyenek.

Általánosan, rekurzív összefüggéssel definiált problémamegoldás esetén egy *r* (rész)probléma összetevői azok a részproblémák, amelyek megoldásától *r* megoldása függ. Tehát a táblázatkitöltés alkalmazásához meg kell állapítani a részproblémáknak egy olyan sorrendjét, hogy minden r részprobléma minden összetevője előrébb álljon a sorrendben, mint *r*.

A rekurzív összefüggések megadják az összetevőket:

- P2(1,k) = 1, P2(n,1) = 1,
- P2(n,n) = 1+P2(n,n-1),
- P2(n,k) = P2(n,n) ha n < k,
- P2(n,k) = P2(n,k-1)+P2(n-k,k) ha k < n.

Összetevők:

- P2(1,k)-nak és P2(n,1)-nek nincs összetevője,
- P2(n,n) összetevője P2(n,n-1),
- P2(n,k) összetevője P2(n,n), ha (n < k),
- P2(n,k) összetevői: P2(n,k-1) és P2(n-k,k), ha (k < n).

Tehát a táblázat kitöltése (k-szerint) soronként balról jobbra haladó lehet. Az algoritmus futási ideje és tárigénye is $O(n^2)$.

```
ParticioDin
   for i:=1 to n  T[i,1]:=1
   for j:=2 to n
           {T[j,j]=T[j,j-1]+1
            for r:=j+1 to n
               {p:=min(r-j,j)
                T[r,j]:=T[r,j-1]+T[r-j,p]}}
   return T[n,n]
```

Partíció lineáris táblázatkitöltéssel

Látható, hogy elegendő lenne a táblázatnak csak két sorát tárolni, mert minden (n,k) részprobléma összetevői vagy a k-adik, vagy a k-1-edik sorban vannak. Sőt, elég egy sort tárolni balról jobbra (növekvő n-szerint) haladó kitöltésnél, mert amelyik részproblémát felülírjuk (n-k,k), annak később éppen az új értéke kell összetevőként.

1. táblázat. A partíció algoritmus táblázata

-	-	-	-	7
-	-	-	5	6
-	-	3	4	5
-	2	2	3	3
1	1	1	1	1

2. táblázat. A partíció algoritmus teljes táblázata

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

```
ParticioDin2
   for i:=1 to n do T[i]:=1
   for j:=2 to n
           {T[j]=T[j]+1
           for r:=j+1 to n T[r]:=T[r]+T[r-j]}
   return T[n]
```

A dinamikus programozás stratégiája.

A dinamikus programozás, mint probléma-megoldási stratégia az alábbi öt lépés végrehajtását jelenti.

1. Az [optimális] megoldás szerkezetének elemzése.

2. Részproblémákra és összetevőkre bontás úgy, hogy az összetevőktől való függés körmentes legyen. Minden részprobléma [optimális] megoldása kifejezhető legyen (rekurzívan) az összetevők [optimális] megoldásaival.

3. Részproblémák [optimális] megoldásának kifejezése (rekurzívan) az összetevők [optimális] megoldásaiból.

Az 1-3 pontok lényegében egy rekurzív algoritmus megtervezését jelentik.

4. Részproblémák [optimális] megoldásának kiszámítása alulról-felfelé haladva. (A részproblémák kiszámítási sorrendjének meghatározása. Olyan sorba kell rakni a részproblémákat, hogy minden p részprobléma minden összetevője előrébb szerepeljen a felsorolásban, mint p. A részproblémák kiszámítása a sorrendnek megfelelően haladva, azaz táblázat-kitöltéssel.

5. Egy [optimális] megoldás előállítása a 4. lépésben kiszámított (és tárolt) információkból. Visszafejtéses vagy feljegyzéses módszer.

Mikor érdemes dinamikus programozást használni?

Optimális résztruktúrájú feladat: a probléma egy részfeladatának optimális megoldása önmagán belül a további részfeladatok optimális megoldásait is tartalmazza.

Átfedő részfeladatok: egy rekurzív algoritmus, ismételten visszatér ugyanazokra a részfeladatokra. (Oszd meg és uralkodj típusú rekurzív algoritmusoknál általában nincs ilyen probléma.)

Leghosszabb közös részsorozat

Egy sorozat, akkor részsorozata egy másiknak, ha abból elemeinek elhagyásával megkapható. A feladat két sorozat $X = (x_1, \ldots, x_m)$ és $Y = (y_1, \ldots, y_n)$ leghosszabb közös részsorozatának meghatározása.

A továbbiakban X_i az X sorozat i hosszú prefixét jelöli $X_i = (x_1, \ldots, x_i)$ és hasonlóan jelöljük a prefixeket az Y és Z sorozatokra is.

Lemma: Legyen $X = (x_1, \ldots, x_m)$ és $Y = (y_1, \ldots, y_n)$ két sorozat és $Z = (z_1, \ldots, z_k)$ ezek LKR-je. Ekkor:
- Ha $x_m = y_n$, akkor $z_k = x_m = y_n$ és Z_{k-1} az X_{m-1} és Y_{n-1} sorozatok egy LKR-je.
- Ha $x_m \neq y_n$, akkor Z az X_{m-1} és Y vagy az X és Y_{n-1} sorozatok egy LKR-je.

Megoldás dinamikus programozással:
Részprobléma: X_i és Y_j LKR-je. Az LKR hossza legyen c[i,j]. Nyilvánvalóan c[0,j]=c[i, 0]= 0.
Rekurzív összefüggés: A lemma alapján

$$c[i,j] = \begin{cases} 0, & \text{ha } i = 0 \text{ vagy } j = 0, \\ c[i-1, j-1] + 1, & \text{ha } x_i = y_j, \\ \max\{c[i-1, j], c[i, j-1]\} & \text{egyébként,} \end{cases}$$

Táblázatkitöltés: c[i,j]-hez használjuk az c[i,j-1] és c[i-1,j] értékeket, ezeknek kell meglenni a c[i,j] érték számításánál. Így a helyes kitöltési sorrend soronként minden sorban a nagyobb j érték felé.

A megoldás meghatározását feljegyzéses módszerrel oldjuk meg, S[i,j]-ben feljegyezzük, hogy mi volt az optimális döntés c[i,j] számításakor.

```
LKR
for i:=0 to m c[i, 0]:=0
for j:=1 to n  c[0,j]:= 0
for i=:1 to m
    {for j:=1 to n
        {if x[i]=y[j]
           then {c[i,j]:=c[i-1,j-1]+1
                 S[i,j]:=2}
        else if c[i-1,j]>= c[i,j-1]
            then {c[i,j]:=c[i-1,j]
                  S[i,j]:=1}
        else   {c[i,j]:=c[i,j-1]
               S[i,j]:= 0}}}
```

Megoldás meghatározása

Ez a szakasz kitölti a c és S táblázatokat, a kiíratás S alapján egy rekurzív algoritmussal megtehető.

```
KIIR(i,j)
if i=0 or j=0 then return
if S[i,j]=2
    then {KIIR(i-1,j-1)
          Print "x[i]"}
else if  S[i,j]=1 then KIIR(i-1,j)
else KIIR(i,j-1)
```

Példa Határozzuk meg az (a,b,b,a,b,a,b,a) és (b,a,b,a,a,b,a,a,b) sorozatok leghosszabb közös részsorozatát!

Tehát az LKR hossza 6. Az LKR-t megkapjuk, ha felírjuk az S táblázatot, vagy visszafejtéssel, ahol az átlós érték növekszik, ott van közös betű. Az i-edik sor j-edik elemének, az X i-edik és az Y j-edik betűje felel meg. Következésképp egy LKR (b,b,a,a,b,a).

Mátrixszorzás probléma

3. táblázat. Az c[i,j] értékek táblázata

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

Ha egy $i \times j$ méretű mátrixot és egy $j \times k$ méretű mátrixot szorzunk össze a skalár műveletek száma ijk.

A mátrixok szorzása asszociatív, az elvégzendő skalár műveletek száma függ a zárójelezéstől.

Példa:

Legyenek A_1, A_2, A_3 méretei 2×3, 3×4, 4×5. Ekkor:

- $(A_1A_2)A_3$ 24+40=64 műveletet hajt végre
- $A_1(A_2A_3)$ pedig 60+30=90 műveletet.

A mátrixszorzás probléma feladata egy adott szorzás optimális zárójelezésének megtalálása. Az input: $A_1, A_2, \ldots A_n$, ahol A_i mérete $p_{i-1} \times p_i$.

Részprobléma: $A_i \ldots A_j$ optimális zárójelezése minden i, j párra, a megoldás értéke legyen $m[i, j]$. Nyilvánvaló, hogy $m[i, i] = 0$.

Rekurzív összefüggés: Ha a szorzásnál az első zárójelpár hátsó zárójele az A_k után kerül, akkor a költség: $m[i,k] + m[k+1, j] + p_{i-1}p_kp_j$. Ezen lehetőségek közül választjuk a legjobbat, így ha $i < j$, akkor

$m[i, j] = \min_{i \le k < j}\{m[i,k] + m[k+1, j] + p_{i-1}p_kp_j\}$.

Táblázatkitöltés: m[i,j]-hez használjuk az m[i,k] és m[k+1,j] értékeket, ezeknek kell meglenni az m[i,j] érték számításánál. Így a helyes kitöltési sorrend átlónként megy (elsőként a j=i, értékek, aztán j=i+1, majd j=i+2 és így tovább).

A megoldás meghatározását feljegyzéses módszerrel oldjuk meg, S[i,j]-ben feljegyezzük, hogy mi volt az optimális döntés m[i,j] számításakor.

```
MATRIXSZORZAS
  for i:=1 to n m[i,i]=0
  for  z:=2 to n
      {for i:=1 to n-z+1
          {j:=i+z-1
           m[i,j]:=INF
           for k:=i to j-1
               {q:= m[i,k]+m[k+1,j]+p(i-1)p(k)p(j)
                If q<m[i,j]
                   then {m[i,j]:=q
                         S[i,j]:=k}}}}
```

A fenti a szakasz kitölti az m és S táblázatokat, a kiíratás S alapján egy rekurzív algoritmussal megtehető.

```
KIIR(i,j)
  If j>i
  Then Print "("
```

```
    KIIR(i,S[i,j])
    KIIR(S[i,j]+1,j)
    Print ")"
Else Print "A(i)"
```

Példa:
$A_1(6\times 7), A_2(7\times 3), A_3(3\times 1), A_4(1\times 2), A_5(2\times 4)$

4. táblázat. Az m[i,j] értékek táblázata

0	0	0	0	0
0	0	0	0	8
0	0	0	6	20
0	0	21	35	57
0	126	63	75	95

$m[1,3] = \min\{[m[1,1]+m[2,3]+6\cdot 7\cdot 1, m[1,2]+m[3,3]+6\cdot 3\cdot 1\} = 63$
$m[2,4] = \min\{[m[2,2]+m[3,4]+7\cdot 3\cdot 2, m[2,3]+m[4,4]+7\cdot 1\cdot 2\} = 35$
$m[3,5] = \min\{[m[3,3]+m[4,5]+3\cdot 1\cdot 4, m[3,4]+m[5,5]+3\cdot 2\cdot 4\} = 20$
$m[1,4] = \min\{[m[1,1]+m[2,4]+6\cdot 7\cdot 2, m[1,2]+m[3,4]+6\cdot 3\cdot 2, m[1,3]+m[4,4]+6\cdot 1\cdot 2\} = 75$

5. táblázat. Az S[i,j] értékek táblázata

0	0	0	0	0
0	0	0	0	4
0	0	0	3	3
0	0	2	3	3
0	1	1	3	3

Megoldás: $((A_1)(A_2A_3))(A_4A_5)$

Kérdések

- Partíció négyzetes táblázatkitöltéssel
- Mátrixszorzás (KIIR is)
- Leghosszabb Közös Részsorozat (KIIR is)
- Két adott string esetén azok leghosszabb közös részsorozatának meghatározása

Szorgalmi feladat

Adott egy pozitív egész K és egy számjegyekből álló n hosszú S sztring. A sztring felbontható $S(0), S(1), S(2), \ldots, S(M)$ részekre, amelyekt egymás után írva visszakapjuk S-t. Minden ilyen felbontáshoz hozzárendelhető egy polinom, $P(x) = a(0) + a(1)x + a(2)x^2 + \cdots + a(M)x^M$, ahol $a(i)$ az a szám, amit az $S(i)$ sztring megad. Például az $S = 1204$ sztring felbontható $S = 1 \cup 204$ alakba (ami a $P(x) = 1 + 204x$ polinom), de $S = 1 \cup 2 \cup 04$ alakba is, ami a $P(x) = 1 + 2x + 4x^2$ polinom) és még egyéb felbontások is vannak. A feladat egy olyan legfeljebb $O(n^2)$ futási idejű dinamikus programozási algoritmus megadása, amely kiszámolja melyik S-ből előállítható polinomra a legkisebb a $P(K)$ érték.

Beküldés: cimreh@inf.u-szeged.hu,
Pszeudókod vagy forrás + magyarázat + futási idő

- első négy megoldó 8 - 8 pont
- második négy megoldó 5-5 pont

A szerzett plusszpontok a vizsga minimumkövetelményébe nem számítanak bele.