



Algoritmuselmélet 4. témakör

Pusztai Pál
pusztai@sze.hu

Tartalom

- Dinamikus programozás
 - Véges sok mátrix összeszorzásának problémája
 - A leghosszabb közös részsorozat probléma
- Mohó algoritmusok
 - Az esemény-kiválasztási probléma
 - Bináris karakterkód tervezése
 - A Huffman-kód
- Közelítő algoritmusok
 - A hibakorlát és a hibakorlát-függvény
 - A halmazlefogási probléma
 - Egy mohó megoldó algoritmus és hibakorlát függvénye

Dinamikus programozás

- Az **oszd meg és uralkodj** módszer a megoldandó feladatot **független** részfeladatokra osztja, amelyeket megold és a részfeladatok megoldásait az eredeti feladat megoldása céljából egyesíti.
- A **dinamikus programozás**
 - A feladatot szintén részfeladatokra való osztással oldja meg. Akkor alkalmazható, ha a részfeladatok **nem függetlenek**, azaz közös részproblémák vannak.
 - Minden egyes részfeladatot pontosan egyszer old meg, az eredményt egy táblázatban tárolja, így elkerülhető egy részfeladat ismételt kiszámítása.
 - **Optimalizálási feladatok** megoldására használjuk. Általában sok megengedett megoldás létezik amelyek mindegyikének van értéke.
 - Cél az optimális (minimális vagy maximális) értékű megoldás megtalálása, amelyet **egy optimális megoldásnak** nevezünk.

Egy dinamikus programozási algoritmus kifejlesztése az alábbi lépésekre bontható:

1. Jellemezzük az optimális megoldás szerkezetét.
2. Rekurzív módon definiáljuk az optimális megoldás értékét.
3. Kiszámítjuk az optimális megoldás értékét alulról felfelé történő módon.
4. A kiszámított információk alapján megszerkesztünk egy optimális megoldást.

Mátrixok véges sorozatainak szorzása

MÁTRIXSZORZÁS(A, B)

```

1  if oszlop[ $A$ ]  $\neq$  sor[ $B$ ]
2      error „nem összeillő dimenzió”
3  else
4      for  $i \leftarrow 1, \textit{sor}[A]$ 
5          for  $j \leftarrow 1, \textit{oszlop}[B]$ 
6               $C[i, j] \leftarrow 0$ 
7              for  $k \leftarrow 1, \textit{oszlop}[A]$ 
8                   $C[i, j] \leftarrow C[i, j] + A[i, k] B[k, j]$ 
9      return  $C$ 
    
```

Megjegyzés: Az A és B mátrixot csak akkor szorozhatjuk össze, ha A oszlopainak száma egyenlő B sorainak számával. Ha A egy $p \times q$ méretű mátrix, B pedig $q \times r$ méretű, akkor eredményül kapott C mátrix mérete $p \times r$.

Műveletigény: A C kiszámításához szükséges idő döntő része a 8. sorban található skalár szorzások mennyisége, ami pqr . A következőkben a számítási időt a skalár szorzások számával fejezzük ki.



Mátrixok véges sorozatainak szorzása

Tegyük fel, hogy adott n mátrix A_1, A_2, \dots, A_n és ezek $A_1 A_2 \dots A_n$ szorzatát akarjuk kiszámítani.

A mátrixok szorzása **asszociatív**, így bármilyen zárójelezés ugyanazt az eredményt adja.

Mátrixok egy szorzata **teljesen zárójelezett**, ha vagy egyetlen mátrix, vagy két zárójelbe tett teljesen zárójelezett mátrixszorzat szorzata.

Példa: ha a sorozat A_1, A_2, A_3, A_4 , akkor az $A_1 A_2 A_3 A_4$ szorzat teljes zárójelezései:

$$(A_1 (A_2 (A_3 A_4))),$$

$$(A_1 ((A_2 A_3) A_4)),$$

$$((A_1 (A_2 A_3)) A_4),$$

$$((A_1 A_2)(A_3 A_4)),$$

$$(((A_1 A_2) A_3) A_4).$$

Egy szorzat zárójelezése alapvetően befolyásolja a kifejezés kiértékelésének költségét.

Példa: Legyen $n=3$ és az A_1, A_2, A_3 mátrixok mérete rendre 10×100 , 100×5 és 5×50 .

Az $((A_1 A_2) A_3)$ zárójelezés szerint: $10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 5000 + 2500 = 7500$ szorzás.

Az $(A_1 (A_2 A_3))$ zárójelezés esetén $100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50 = 25000 + 50000 = 75000$ szorzás.

Mátrixok véges sorozatainak szorzása

A véges sok mátrix összeszorzásának problémája:

- Adott mátrixoknak egy A_1, A_2, \dots, A_n véges sorozata, ahol az A_i mátrix mérete $p_{i-1} \times p_i$ ($i=1, 2, \dots, n$).
- Keresendő az $A_1 A_2 \dots A_n$ szorzat azon teljes zárójelezése, amely minimalizálja a szorzat kiszámításához szükséges skalár szorzások számát.

Megjegyzés: A teljes zárójelezések száma exponenciális n -ben.

A dinamikus programozási megoldás lépései:

1. Az optimális megoldás jellemzése:

Jelölje $A_{i..j}$ az $A_i A_{i+1} \dots A_j$ szorzat eredményét. Az optimális zárójelezés kettévágja az $A_i A_{i+1} \dots A_j$ szorzatot valamely A_k és A_{k+1} mátrix között, ahol $i \leq k < j$.

Ekkor az optimális zárójelezés költsége az $A_{i..k}$ és $A_{k+1..j}$ mátrixok kiszámításának és ezek összeszorzásának együttes költsége.

Vegyük észre, hogy az $A_{i..k}$ és $A_{k+1..j}$ részsorozatok zárójelezésének is optimálisnak kell lennie!

A véges sok mátrix összeszorzása problémájának optimális megoldása minden esetben tartalmazza a részfeladatok optimális megoldását.

Ez a tulajdonság, az optimális megoldáson belüli optimális részszerkezet, általánosan jellemzi a dinamikus programozás alkalmazhatóságát.



Mátrixok véges sorozatainak szorzása

2. Az optimális érték kifejezése a részproblémák optimális értékeinek segítségével:

A részprobléma az $A_i A_{i+1} \dots A_j$ szorzat optimális zárójelezésének meghatározása, ahol $1 \leq i \leq j \leq n$.

Legyen $m[i, j]$ az $A_{i..j}$ mátrix kiszámításához minimálisan szükséges skalár szorzások száma, így az $A_{1..n}$ kiszámításának lehetséges legkisebb költsége $m[1, n]$.

Ha $i=j$, akkor $A_{i..i}=A_i$, azaz nincs szükség szorzásra a szorzat meghatározásához.

Ha $i < j$, akkor tegyük fel, hogy az optimális zárójelezés az A_k és A_{k+1} mátrixok között vágja szét az $A_i A_{i+1} \dots A_j$ szorzatot, ahol $i \leq k < j$.

Mivel az A_i mátrixok mérete $p_{i-1} \times p_i$, az $A_{i..k}$ és $A_{k+1..j}$ szorzat kiszámítása $p_{i-1} p_k p_j$ szorzást igényel, ezért azt kapjuk, hogy:

$$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$$

Az $i \leq k < j$ miatt csak $j-i$ különböző értéke lehet k -nak ($k=i, i+1, i+2, \dots, j-1$), ezért:

$$m[i, j] = 0, \quad \text{ha } i=j,$$

$$m[i, j] = \min_{i \leq k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\}, \quad \text{ha } i < j.$$

Megjegyzés: Az optimális megoldás előállítása érdekében definiáljuk az $s[i, j]$ mennyiségeket, amelyek azt a k indexet adják, ahol az optimális zárójelezés kettévágja az $A_i A_{i+1} \dots A_j$ szorzatot, azaz amelyre $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$ teljesül.

Mátrixok véges sorozatainak szorzása

3. Az optimális megoldás értékének meghatározása alulról felfelé történő megközelítéssel:

MÁTRIX-SZORZÁS-SORREND(p)

```

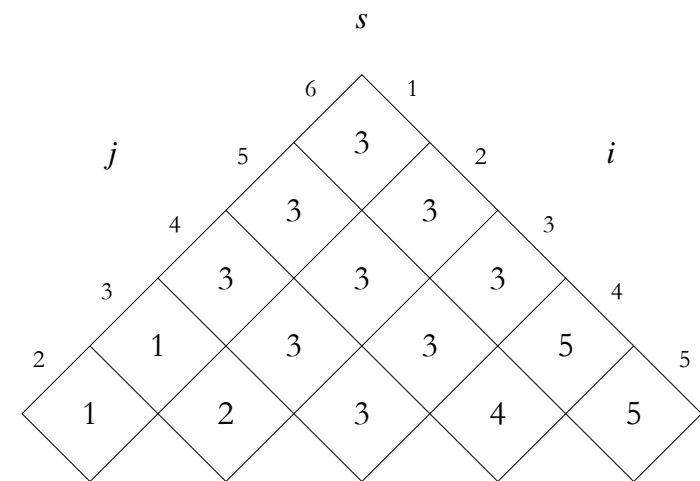
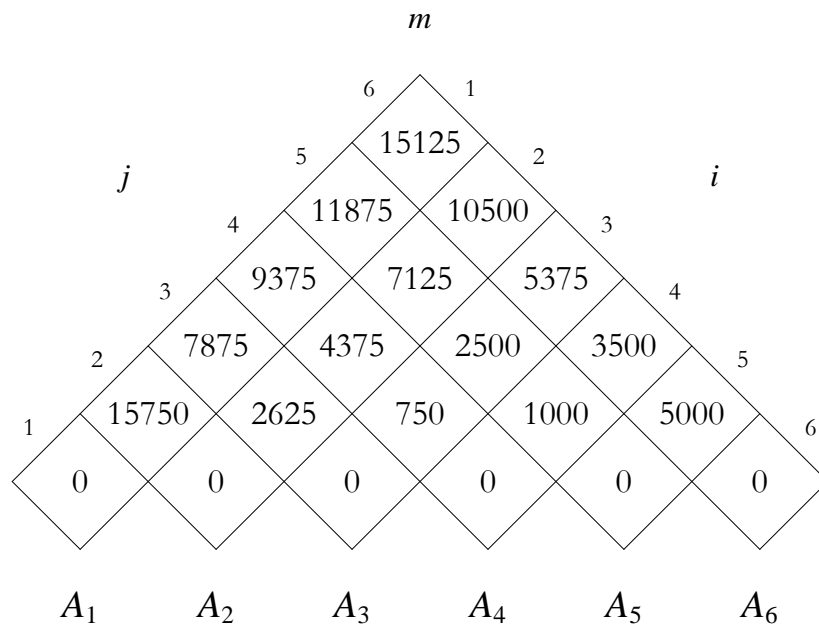
1   $n \leftarrow \text{hossz}[p]-1$ 
2  for  $i \leftarrow 1, n$ 
3       $m[i, i] \leftarrow 0$ 
4  for  $l \leftarrow 2, n$ 
5      for  $i \leftarrow 1, n-l+1$ 
6           $j \leftarrow i+l-1$ 
7           $m[i, j] \leftarrow \infty$ 
8          for  $k \leftarrow i, j-1$ 
9               $q \leftarrow m[i, k]+m[k+1, j]+ p_{i-1} p_k p_j$ 
10             if  $q < m[i, j]$ 
11                  $m[i, j] \leftarrow q$ 
12                  $s[i, j] \leftarrow k$ 
13 return  $m, s$ 
```

Megjegyzés: l azt mutatja, hogy milyen hosszú szorzatok minimális költségeit számoljuk éppen.

Hatékonyság: Az algoritmus $O(n^3)$ futási időt és $\Theta(n^2)$ memóriát igényel.



Mátrixok véges sorozatainak szorzása



$A_1 : 30 \times 35$

$A_2 : 35 \times 15$

$A_3 : 15 \times 5$

$A_4 : 5 \times 10$

$A_5 : 10 \times 20$

$A_6 : 20 \times 25$

$$m[2, 2] + m[3, 5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000,$$

$$m[2, 5] = \min \{ m[2, 3] + m[4, 5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \dots \}$$

$$m[2, 4] + m[5, 5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375$$

A MÁTRIX-SZORZÁS-SORREND eljárás



Mátrixok véges sorozatainak szorzása

4. Az optimális megoldás előállítása a számított információk alapján:

MÁTRIX-LÁNC-SZORZÁS(A, s, i, j)

```

1  if  $j > i$ 
2       $X \leftarrow$  MÁTRIX-LÁNC-SZORZÁS( $A, s, i, s[i, j]$ )
3       $Y \leftarrow$  MÁTRIX-LÁNC-SZORZÁS( $A, s, s[i, j] + 1, j$ )
4      return MÁTRIXSZORZÁS( $X, Y$ )
5  else
6      return  $A_i$ 
```

Megjegyzés: Az előző mátrixokkal a MÁTRIX-LÁNC-SZORZÁS($A, s, 1, 6$) hívás a mátrixok szorzatát a $((A_1(A_2 A_3))((A_4 A_5) A_6))$ zárójelezés szerint számítja.

OPTIMÁLIS-ZÁRÓJELEZÉS-NYOMTATÁS(s, i, j)

```

1  if  $j = i$ 
2      Ki "A" $i$ 
3  else
4      Ki "("
5      OPTIMÁLIS-ZÁRÓJELEZÉS-NYOMTATÁS ( $s, i, s[i, j]$ )
6      OPTIMÁLIS-ZÁRÓJELEZÉS-NYOMTATÁS ( $s, s[i, j] + 1, j$ )
7      Ki ")"
```



Feladatok

- Hány skalár szorzást végzünk legjobb és hányat legrosszabb esetben az alábbi méretekkel megadott mátrix sorozat szorzatmátrixának kiszámításakor?
 - $(5, 2, 3, 4)$
- Keressük meg azon mátrixok összeszorzásának optimális zárójelezését, ahol a méretek sorozata az alábbi!
 - $(5, 2, 3, 4, 2)$
- Adjuk meg a MÁTRIX-SZORZÁS-SORREND(p) eljárás m és s eredménytömbjeit az alábbi p bemenő tömb esetén!
 - $p=(5, 2, 3, 5, 4, 2)$



Mátrixok véges sorozatainak szorzása

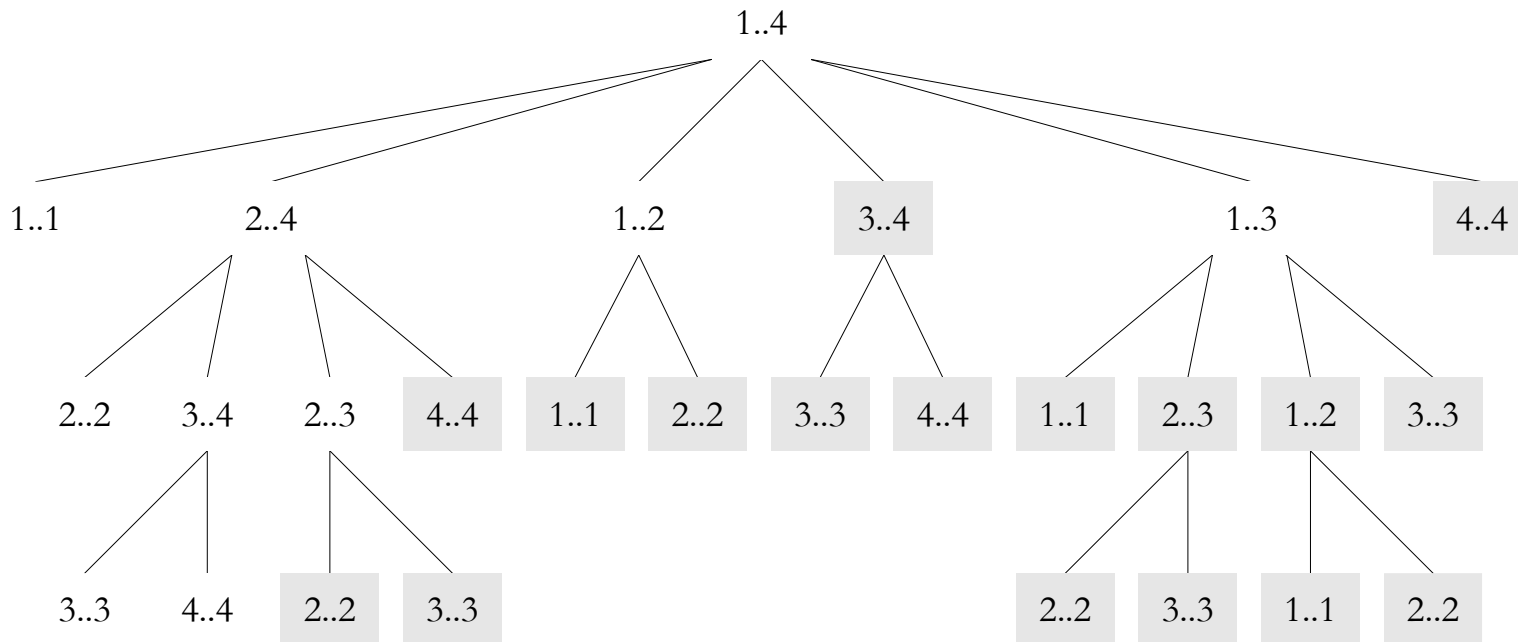
REKURZÍV-MÁTRIX-LÁNC(p, i, j)

```

1  if  $i = j$ 
2      return 0
3   $m[i, j] \leftarrow \infty$ 
4  for  $k \leftarrow i, j-1$ 
5       $q \leftarrow \text{REKURZÍV-MÁTRIX-LÁNC}(p, i, k) +$ 
         $\text{REKURZÍV-MÁTRIX-LÁNC}(p, k+1, j) + p_{i-1} p_k p_j$ 
6      if  $q < m[i, j]$ 
7           $m[i, j] \leftarrow q$ 
8  return  $m[i, j]$ 
    
```

Hatékonyság: Az algoritmus számításigénye n -ben legalább exponenciális.

Mátrixok véges sorozatainak szorzása



A REKURZÍV-MÁTRIX-LÁNC(p , 1, 4) rekurziós fája

Mátrixok véges sorozatainak szorzása

FELJEGYZÉSES-MÁTRIX-LÁNC(p)

```

1   $n \leftarrow \text{hossz}[p]-1$ 
2  for  $i \leftarrow 1, n$ 
3      for  $j \leftarrow i, n$ 
4           $m[i, j] \leftarrow \infty$ 
5  return LÁNCOT-KERES( $p, 1, n$ )
    
```

LÁNCOT-KERES(p, i, j)

```

1  if  $m[i, j] < \infty$ 
2      return  $m[i, j]$ 
3  if  $i = j$ 
4       $m[i, j] \leftarrow 0$ 
5  else
6      for  $k \leftarrow i, j-1$ 
7           $q \leftarrow \text{LÁNCOT-KERES}(p, i, k) + \text{LÁNCOT-KERES}(p, k+1, j) + p_{i-1} p_k p_j$ 
8          if  $q < m[i, j]$ 
9               $m[i, j] \leftarrow q$ 
10 return  $m[i, j]$ 
    
```

Hatékonyság: Az algoritmus $O(n^3)$ futási időt és $\Theta(n^2)$ memóriát igényel.



A leghosszabb közös részsorozat

Egy sorozat részsorozata egy olyan sorozat, amit az adott sorozatból néhány (esetleg nulla) elem elhagyásával nyerünk.

Formálisan, ha az adott sorozat $X=(x_1, x_2, \dots, x_m)$, akkor egy másik $Z=(z_1, z_2, \dots, z_k)$ sorozat akkor **részsorozata** X -nek, ha létezik X indexeinek egy szigorúan növő (i_1, i_2, \dots, i_k) sorozata, hogy minden $j=1, 2, \dots, k$ esetén $x_{i_j} = z_j$.

Példa: $Z=(B, C, D, B)$ részsorozata az $X=(A, B, C, B, D, A, B)$ -nek, és ekkor az indexek megfelelő sorozata $(2, 3, 5, 7)$.

Adott két sorozat X és Y . Azt mondjuk, hogy egy Z sorozat **közös részsorozatuk**, ha Z részsorozata X -nek is és Y -nak is.

Példa: Ha $X=(A, B, C, B, D, A, B)$ és $Y=(B, D, C, A, B, A)$, akkor a (B, C, A) közös részsorozata X -nek és Y -nak. Azonban (B, C, A) nem a **leghosszabb** közös részsorozat, mert (B, C, B, A) ill. (B, D, A, B) is közös részsorozat, amelyek hossza 4.

A leghosszabb közös részsorozat probléma:

- Adott két sorozat $X=(x_1, x_2, \dots, x_m)$, és $Y=(y_1, y_2, \dots, y_n)$.
- Feladat: megtalálni a leghosszabb közös részsorozatukat.

Megjegyzés: A leghosszabb közös részsorozat kifejezést LKR-ként rövidítjük.



A leghosszabb közös részsorozat

Egy $X=(x_1, x_2, \dots, x_m)$ sorozat i -edik **prefixe** az $X_i=(x_1, x_2, \dots, x_i)$ sorozat, ahol $i=0, 1, \dots, m$.

Példa: ha $X=(A, B, C, B, D, A, B)$, akkor $X_4=(A, B, C, B)$, és X_0 az üres sorozat.

Tétel: Legyen $X=(x_1, x_2, \dots, x_m)$, és $Y=(y_1, y_2, \dots, y_n)$ két sorozat és $Z=(z_1, z_2, \dots, z_k)$ ezek egy LKR-je. Ekkor igazak a következő állítások:

1. Ha $x_m = y_n$, akkor $z_k = x_m = y_n$, és Z_{k-1} az X_{m-1} és Y_{n-1} egy LKR-je.
2. Ha $x_m \neq y_n$, akkor $z_k \neq x_m$ esetén Z az X_{m-1} és Y egy LKR-je.
3. Ha $x_m \neq y_n$, akkor $z_k \neq y_n$ esetén Z az X és Y_{n-1} egy LKR-je.

Következmény: Mivel az LKR rendelkezik az optimális részstruktúra tulajdonsággal, így hatékonyan megoldható a dinamikus programozás módszerével.

Legyen $c[i, j]$ az X_i és Y_j LKR-jének hossza. Ekkor (az előző tétel szerint):

$$\begin{aligned} c[i, j] &= 0, & \text{ha } i=0 \text{ vagy } j=0, \\ c[i, j] &= c[i-1, j-1] + 1, & \text{ha } i, j > 0 \text{ és } x_i = y_j, \\ c[i, j] &= \max\{c[i, j-1], c[i-1, j]\}, & \text{ha } i, j > 0 \text{ és } x_i \neq y_j. \end{aligned}$$

A leghosszabb közös részsorozat

LKR-HOSSZ(X, Y)

```

1   $m \leftarrow \text{hossz}[X]$ 
2   $n \leftarrow \text{hossz}[Y]$ 
3  for  $i \leftarrow 1, m$ 
4       $c[i, 0] \leftarrow 0$ 
5  for  $j \leftarrow 0, n$ 
6       $c[0, j] \leftarrow 0$ 
7  for  $i \leftarrow 1, m$ 
8      for  $j \leftarrow 1, n$ 
9          if  $x_i = y_j$ 
10              $c[i, j] \leftarrow c[i-1, j-1] + 1$ 
11              $b[i, j] \leftarrow \text{„}\nwarrow\text{”}$ 
12         else
13             if  $c[i-1, j] \geq c[i, j-1]$ 
14                  $c[i, j] \leftarrow c[i-1, j]$ 
15                  $b[i, j] \leftarrow \text{„}\uparrow\text{”}$ 
16             else
17                  $c[i, j] \leftarrow c[i, j-1]$ 
18                  $b[i, j] \leftarrow \text{„}\leftarrow\text{”}$ 
19 return  $c, b$ 
    
```

Hatékonyság: Az algoritmus $O(mn)$ futási időt és $\Theta(mn)$ memóriát igényel.



A leghosszabb közös részsorozat

		j	0	1	2	3	4	5	6
		y_j		B	<i>D</i>	C	<i>A</i>	B	A
i	x_i								
0	x_i		0	0	0	0	0	0	0
1	<i>A</i>		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	<i>D</i>		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	<i>B</i>		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Az LKR-HOSSZ működése

A leghosszabb közös részsorozat

LKR-NYOMTAT(b, X, i, j)

```

1  if  $i = 0$  vagy  $j = 0$ 
2      return
3  if  $b[i, j] = „\nwedge”$ 
4      LKR-NYOMTAT( $b, X, i-1, j-1$ )
5      Ki:  $x_i$ 
6  else
7      if  $b[i, j] = „\uparrow”$ 
8          LKR-NYOMTAT( $b, X, i-1, j$ )
9      else
10         LKR-NYOMTAT( $b, X, i, j-1$ )
    
```

Megjegyzés: A kezdeti hívás: LKR-NYOMTAT($b, X, \text{hossz}[X], \text{hossz}[Y]$).

Hatékonyság: Az algoritmus futási ideje $O(m+n)$.



Feladatok

- Határozzuk meg az alábbi sorozatok egy LKR-jét!
 - $(1, 0, 0, 1, 0, 1, 0, 1), (0, 1, 0, 1, 1, 0, 1, 1, 0)$
- Adjuk meg az LKR-HOSSZ(X, Y) eljárás c és b eredmény tömbjeit és LKR-jét az alábbi bemenő sorozatok esetén!
 - $X=(1, 0, 0, 1, 0)$
 - $Y=(0, 1, 0, 1)$
- Megírható-e az LKR-HOSSZ algoritmus a b tömb nélkül? Mi lesz ekkor az LKR-NYOMTAT eljárással?
- Csökkenthető-e a $\Theta(mn)$ memóriaigény, ha csak az LKR hosszára vagyunk kíváncsiak (és magára az LKR-re nem)?

Mohó algoritmusok

Sok optimalizálási probléma esetén a **dinamikus programozási** megoldás túl sok esetet vizsgál meg (alulról-felfelé haladva) annak érdekében, hogy az optimális választást meghatározza.

A **mohó algoritmusok** mindig az adott lépésben optimálisnak látszó döntést hozzák abban a reményben, hogy a lokális optimumok majd a globális optimumhoz vezetnek.

A mohó stratégia általában felülről-lefelé halad, az egyes választások függhetnek az addig elvégzett választásoktól, de nem függhetnek a későbbi választásoktól, a részproblémák megoldásától.

A **mohó-választási tulajdonság**: globális optimális megoldás elérhető lokális optimum (mohó) választásával.

Az **optimális részproblémák tulajdonság**: az optimális megoldás felépíthető a részproblémák optimális megoldásából.

A mohó algoritmusok nem mindig adnak optimális megoldást, sok esetben csak közelítik azt.

Mohó algoritmusok

■ Mohó stratégia, vagy dinamikus programozás?

A 0-1 hátizsák feladat

Adott n darab tárgy, az i -edik tárgy használati értéke v_i , a súlya pedig w_i , ahol v_i és w_i egész számok.

Feladat: Kiválasztandó a tárgyaknak olyan részhalmaza, amelyek használati értékének összege a lehető legnagyobb, de a súlyuk nem nagyobb, mint a hátizsák W kapacitása, amely egész szám.

A **töredékes hátizsák feladat** csak abban különbözik az előzőtől, hogy a tárgyak töredéke is választható.

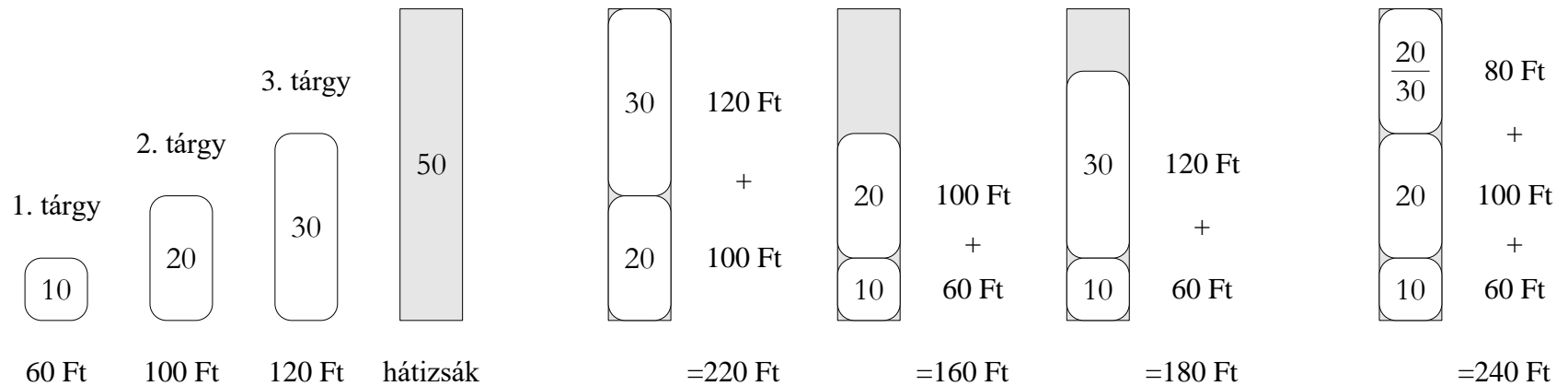
Tulajdonságok:

Mindkét probléma teljesíti az optimális részproblémák tulajdonságát.

A 0-1 feladat nem teljesíti a mohó választási tulajdonságot, így nem oldható meg mohó stratégiával, viszont dinamikus programozással igen.

A töredékes feladat teljesíti a mohó választási tulajdonságot, így megoldható a mohó stratégiával (a használati érték per súly hányados (v_i / w_i) szerinti mohó választással).

Mohó algoritmusok



A 0-1 és a töredékes hátizsák feladat

Mohó algoritmusok

■ Az esemény-kiválasztási probléma

Adott események egy $S = \{a_1, a_2, \dots, a_n\}$ n elemű halmaza, amelyek egy közös erőforrást (pl. egy előadótermet) kívánnak használni, amit egy időben csak az egyik használhat.

Minden a_i eseményhez adott az s_i **kezdő időpont** és az f_i **befejező időpont**, ahol $0 \leq s_i < f_i < \infty$.

Ha az a_i eseményt kiválasztjuk, akkor ez az $[s_i, f_i)$ félig nyitott időintervallumot foglalja le.

Az a_i és a_j események **kompatibilisek**, ha az $[s_i, f_i)$ és $[s_j, f_j)$ intervallumok nem fedik egymást (azaz ha $s_i \geq f_j$ vagy $s_j \geq f_i$).

Feladat: Kiválasztandó kölcsönösen kompatibilis események egy legnagyobb elemszámú halmaza.

Mohó algoritmusok

MOHÓ-ESEMÉNYKIVÁLASZTÓ(s, f)

```

1   $n \leftarrow \text{hossz}[s]$ 
2   $A \leftarrow \{a_1\}$ 
3   $i \leftarrow 1$ 
4  for  $m \leftarrow 2, n$ 
5      if  $s_m \geq f_i$ 
6           $A \leftarrow A \cup \{a_m\}$ 
7           $i \leftarrow m$ 
8  return  $A$ 
    
```

Feltétel: Az s és f adatokat tömb ábrázolja és a bemeneti események a befejező időpont szerint rendezettek, azaz $f_1 \leq f_2 \leq \dots \leq f_n$.

Megjegyzés: Mivel az eseményeket a befejező időpontok szerint nemcsökkenő sorrendben vizsgáljuk, így f_i mindig az A -beli események befejező időpontjainak maximuma, vagyis $f_i = \max\{f_k : a_k \in A\}$.

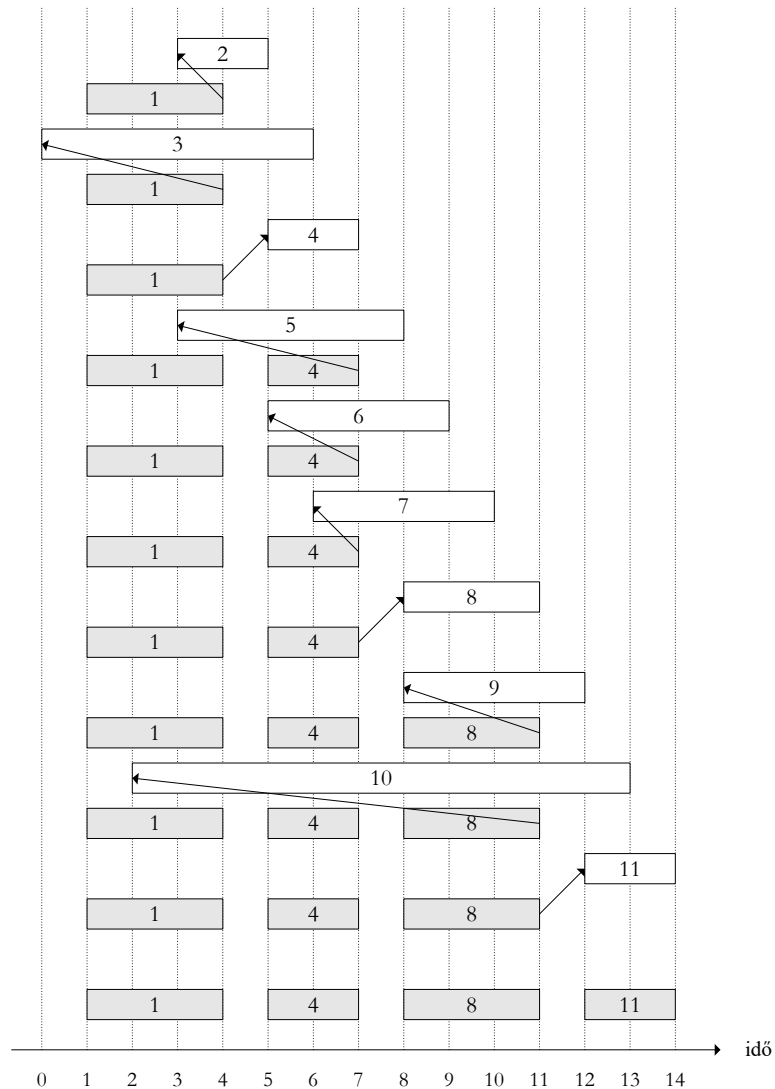
Hatékonyság: Az események n elemű S halmazát az algoritmus $\Theta(n)$ idő alatt ütemezi.

Tétel: Tekintsünk egy S_i nem üres részproblémát és legyen a_m a legkisebb befejezési idejű esemény S_i -ben, azaz $f_m = \min\{f_k : a_k \in S_i\}$. Ekkor a_m eleme S_i valamely maximális elemszámú, kölcsönösen kompatibilis eseményekből álló részalmazának.

Következmény: A MOHÓ-ESEMÉNYKIVÁLASZTÓ algoritmus maximális elemszámú megoldását adja az esemény-kiválasztási problémának.

Mohó algoritmusok

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



A MOHÓ-ESEMÉNYKIVÁLASZTÓ működése

Feladatok

- Melyik eseményeket választja ki a MOHÓ-ESEMÉNYKIVÁLASZTÓ algoritmus az alábbi s és f tömbök esetén?
 - $s = \langle 8, 3, 5, 12, 3, 6, 10, 6, 17 \rangle$
 - $f = \langle 12, 6, 8, 14, 7, 9, 15, 7, 20 \rangle$



Mohó algoritmusok

■ Bináris karakterkód tervezése

Hogyan tároljunk egy karakterekből álló adatállományt tömörítetten úgy, hogy minden karaktert egy bináris jelsorozattal ábrázolunk?

Prefix-kódnak nevezzük az olyan kódolást, amelyben egyik kódszó sem kezdőszelete egy másiknak.

Állítás: A karakterkóddal elérhető optimális adattömörítés mindig megadható prefix-kóddal is.

A prefix-kódok előnyösek, mert egyszerűsítik a kódolást (tömörítés) és a dekódolást.

A dekódoláshoz szükség van a prefix-kód olyan alkalmas ábrázolására, amely lehetővé teszi, hogy a kódszót könnyen azonosítani tudjuk. Az olyan bináris fa, amelynek levelei a kódolandó karakterek, egy ilyen ábrázolás. Az egyes karakterek kódját az adott karakterig vezető út adja.

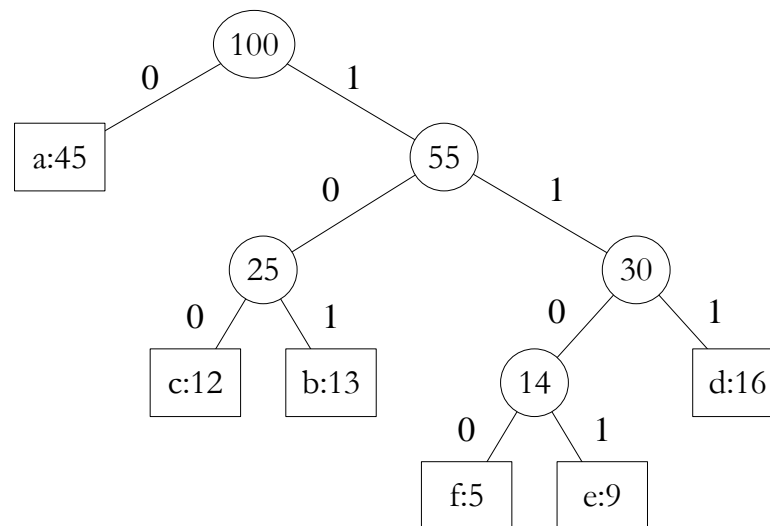
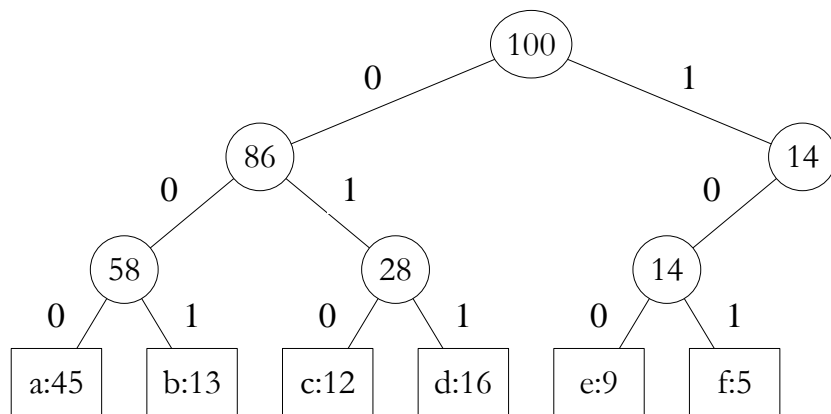
Állítás: Egy adatállomány optimális kódját mindig teljes bináris fa adja, azaz olyan fa, amelyben minden nem levél pontnak két gyereke van.

Következmény: Ha C az az ábécé, amelynek elemei a kódolandó karakterek, akkor az optimális prefix-kód fájának $|C|$ levele és pontosan $|C|-1$ belső pontja van.

Feladat: Meghatározandó egy adott C ábécéhez (az egyes karakterek gyakoriságának ismeretében) az optimális prefix-kód.

Mohó algoritmusok

	a	b	c	d	e	f	
Gyakoriság (ezrekben)	45	13	12	16	9	5	
Fix hosszú kódszó	000	001	010	011	100	101	(300 000 bit)
Változó hosszú kódszó	0	101	100	111	1101	1100	(224 000 bit)



A karakter kódolási probléma

Mohó algoritmusok

HUFFMAN(C)

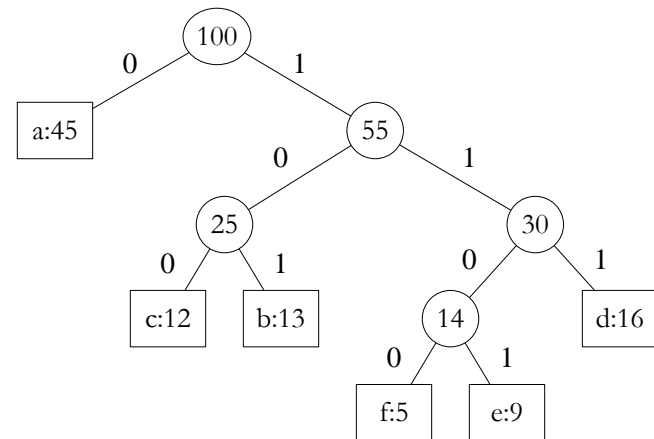
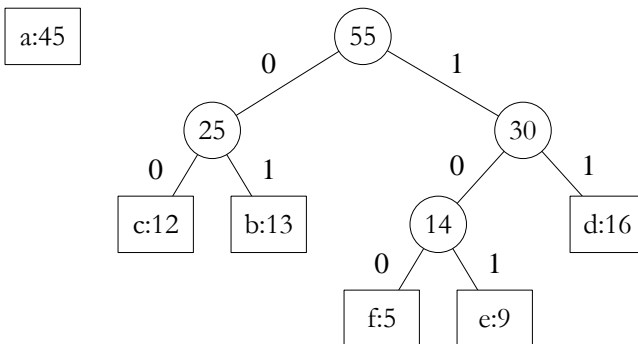
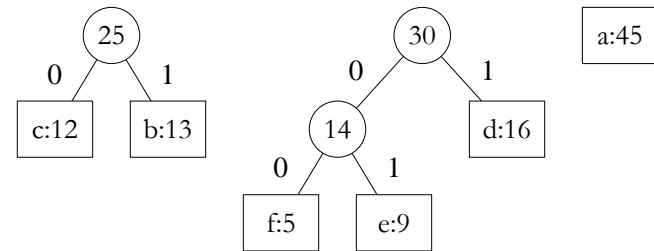
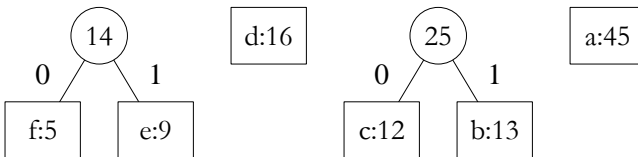
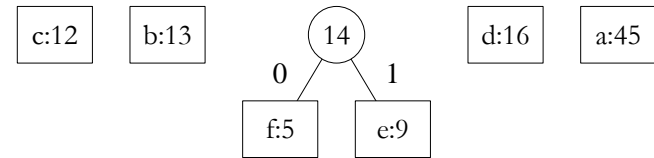
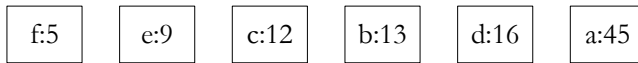
```

1   $n \leftarrow |C|$ 
2   $Q \leftarrow C$ 
3  for  $i \leftarrow 1, n-1$ 
4       $z \leftarrow$  egy újonnan létrehozott csúcs
5       $x \leftarrow bal[z] \leftarrow$  KIVESZ-MIN( $Q$ )
6       $y \leftarrow jobb[z] \leftarrow$  KIVESZ-MIN( $Q$ )
7       $f[z] \leftarrow f[x] + f[y]$ 
8      BESZÚR( $Q, z$ )
9  return KIVESZ-MIN( $Q$ )
    
```

Megjegyzés: Minden $c \in C$ karakter gyakorisága az $f[c]$ érték.

Hatékonyság: Ha a Q elsőbbségi sort bináris kupaccal valósítjuk meg, akkor a 2. sor $O(n)$ idejű (lásd KUPACOT-ÉPÍT), az elsőbbségi sorok műveletei (KIVESZ-MIN, BESZÚR) $O(\lg n)$ idejűek, így a Huffman algoritmus $O(n \lg n)$ idejű, minden n karaktert tartalmazó C halmazra.

Mohó algoritmusok



Feladatok

- Milyen fát állít elő a HUFFMAN algoritmus az alábbi karakterek és előfordulási darabszámok esetén?
 - a: 23, b: 15, c: 12, d: 13, e: 6, f: 10, g: 4, h: 17
- A fix hosszú kódoláshoz képest hány százalékos lesz a megtakarítás a teljes szöveg kódolásakor?
- Mi az optimális Huffman-kódja annak az ábécének, ahol a karakterek gyakoriságát a Fibonacci számok adják, azaz:
 - a: 1, b: 1, c: 2, d: 3, e: 5, f: 8, g: 13, h: 21, ...

Mohó algoritmusok

■ Közelítő algoritmusok

Az optimális közeli megoldást adó algoritmust **közelítő algoritmusnak** nevezzük.

Tegyük fel, hogy egy optimalizálási probléma minden lehetséges megoldásának pozitív költsége van.

Optimális megoldás: **minimális** vagy **maximális** költséggel rendelkező megoldás.

Egy közelítő algoritmus **hibakorlát-függvénye** $\rho(n)$, ha az általa adott megoldás C költsége – minden n méretű bemenetre – az optimális megoldás C^* költségének legfeljebb $\rho(n)$ -szerese, ill. legalább $\rho(n)$ -ed része, azaz:

$$\max(C/C^*, C^*/C) \leq \rho(n).$$

Ha egy algoritmus biztosítja a $\rho(n)$ hibakorlát-függvény betartását, akkor **$\rho(n)$ -közelítő algoritmusnak** nevezzük.

Megjegyzés: Ha a hibakorlát-függvény független n -től, akkor ρ hibakorlátról és ρ -közelítő algoritmusról beszélünk (pl. egy 1-közelítő algoritmus optimális megoldást ad).

Mohó algoritmusok

■ A halmazlefedési probléma

Adott egy X véges, nem üres halmaz, és X részhalmazainak egy olyan \mathcal{F} összessége, hogy X minden eleme legalább egy \mathcal{F} -beli részhalmazhoz tartozik, azaz:

$$X = \bigcup_{S \in \mathcal{F}} S.$$

Azt mondjuk, hogy egy $S \in \mathcal{F}$ **lefogja/lefed**i az elemeit.

Feladat: Meghatározandó egy olyan minimális méretű $\mathcal{C} \subseteq \mathcal{F}$ részhalmaz, amelynek tagjai lefedik X -et, azaz $X = \bigcup_{S \in \mathcal{C}} S$.

Példa: Tegyük fel, hogy X olyan képességek halmazát jelöli, amelyek szükségesek egy probléma megoldásához, és adott azon emberek halmaza, akik a problémán dolgozhatnak. Kiválasztandó egy minimális létszámú bizottság úgy, hogy X minden képességéhez legyen (legalább) egy ember a bizottságban, aki rendelkezik azzal a képességgel.

Mohó algoritmusok

MOHÓN-HALMAZT-LEFOG(X, \mathcal{F})

```

1   $U \leftarrow X$ 
2   $\mathcal{C} \leftarrow \emptyset$ 
3  while  $U \neq \emptyset$ 
4      válasszunk ki egy olyan  $(S \in \mathcal{F})$ -et, amelyre  $|S \cap U|$  maximális
5       $U \leftarrow U - S$ 
6       $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$ 
7  return  $\mathcal{C}$ 
    
```

Hatékonyság: A ciklus legfeljebb $\min(|X|, |\mathcal{F}|)$ -szer fut le, és a ciklusmag megvalósítható $O(|X||\mathcal{F}|)$ idő alatt, azaz az algoritmus polinomiális futási idejű.

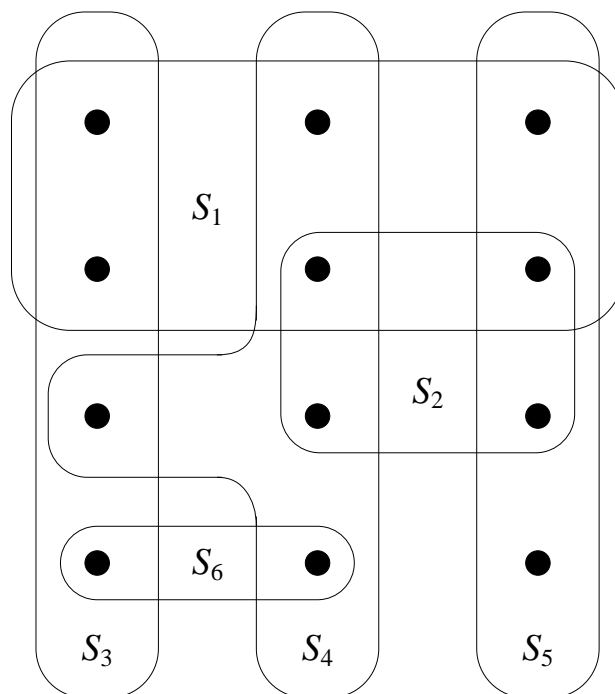
Jelölés: jelölje $H(d)$ a d -edik harmonikus számot, azaz $H(d) = \sum_{i=1, \dots, d} 1/i$.

Tétel: A MOHÓN-HALMAZT-LEFOG polinomiális $\rho(n)$ -közelítő algoritmus, ahol

$$\rho(n) = H(\max\{|S| : S \in \mathcal{F}\}).$$

Következmény: Mivel $\sum_{i=1, \dots, d} 1/i \leq \ln d + 1$, ezért a MOHÓN-HALMAZT-LEFOG algoritmus polinomiális $(\ln |X| + 1)$ -közelítő algoritmus.

Mohó algoritmusok



A lefogási probléma egy esete

Feladatok

- Mi a minimális méretű halmaz lefedés az előző dián szereplő adatok esetén és milyen megoldást ad (mely halmazokat választja ki és milyen sorrendben) a MOHÓN-HALMAZT-LEFOG algoritmus?
- Az alábbi szavakat betűk halmazának véve milyen megoldást ad a MOHÓN-HALMAZT-LEFOG algoritmus, ha a holtversenyt annak a szónak a javára döntjük el, amelyik előbb jelenik meg a szótárban.
 - {arid, dash, drain, heard, lost, nose, shun, slate, snare, thread}