Kombinatorikus optimalizálás 4. hét

Pusztai Pál
pusztai@sze.hu

Tartalom

- Dinamikus programozás
  - A leghosszabb közös részsorozat probléma
- Mohó algoritmusok
  - Az esemény-kiválasztási probléma
- Hátizsák feladatok

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 lehetséges 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.

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.

- Mohó stratégia, vagy dinamikus programozás?

Ha egy feladat teljesíti a mohó választási tulajdonságot, akkor mohó algoritmussal megkapható az optimális megoldás, egyébként a mohó algoritmus csak közelítő megoldást garantál.

Ha egy feladat teljesíti az optimális részproblémák tulajdonságot, akkor megoldható dinamikus programozással.

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 $X=(x_1, x_2, \ldots, x_m)$, akkor egy másik $Z=(z_1, z_2, \ldots, z_k)$ sorozat akkor **részsorozata** X-nek, ha létezik X indexeinek egy szigorúan növő $(i_1, i_2, \ldots, i_k)$ sorozata, hogy minden $j=1, 2, \ldots, k$ 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 $X=(x_1, x_2, \ldots, x_m)$, és $Y=(y_1, y_2, \ldots, y_n)$.
- Feladat: megtalálni a leghosszabb közös részsorozatukat.

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

A leghosszabb közös részsorozat

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

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

1. Jellemezzük az optimális megoldás szerkezetét.

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

- 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.
- Ha $x_m \neq y_n$, akkor $z_k \neq x_m$ esetén Z az X_{m-1} és Y egy LKR-je.
- 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.

2. Rekurzív módon definiáljuk az optimális megoldás értékét.

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

$$c[i, j] = 0, \qquad \text{ha } i=0 \text{ vagy } j=0,$$

$$c[i, j] = c[i-1, j-1]+1, \qquad \text{ha } i, j >0 \text{ és } x_i = y_j,$$

$$c[i, j] = \max\{c[i, j-1], c[i-1, j]\}, \qquad \text{ha } i, j >0 \text{ és } x_i \neq y_j.$$

A leghosszabb közös részsorozat

3. Kiszámítjuk az optimális megoldás értékét alulról felfelé történő módon.

```
LKR-HOSSZ(X, Y)
1   m ← hossz[X]
2   n ← hossz[Y]
3   for i ← 1, m
4           c[i, 0] ← 0
5   for j ← 0, n
6           c[0, j] ← 0
7   for i ← 1, m
8           for j ← 1, n
9                   if x_i = y_j
10                          c[i, j] ← c[i−1, j−1]+1
11                          b[i, j] ← „↖”
12                  else
13                          if c[i−1, j] ≥ c[i, j−1]
14                                  c[i, j] ← c[i−1, j]
15                                  b[i, j] ← „↑”
16                          else
17                                  c[i, j] ← c[i, j−1]
18                                  b[i, j] ← „←”
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
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	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	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Az LKR-HOSSZ működése

A leghosszabb közös részsorozat

4. A kiszámított információk alapján megszerkesztünk egy optimális megoldást.

```
LKR-NYOMTAT(b, X, i, j)
1   if i = 0 vagy j = 0
2         return
3   if b[i, j] = „↖”
4         LKR-NYOMTAT(b, X, i−1, j−1)
5         Ki: x_i
6   else
7         if b[i, j] = „↑”
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*, *hossz*[*X*], *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)
- 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)?

Feladatok

- 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)

Mohó algoritmusok

- Az esemény-kiválasztási probléma

Adott események egy $S=\{a_1, a_2, \ldots, a_n\}$ 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 $s_i \leq f_i$.

Ha az 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 ← hossz[s]
2   A ← {a_1}
3   i ← 1
4   for m ← 2, n
5         if s_m ≥ f_i
6                   A ← A ∪ {a_m}
7                   i ← m
8   return A
```

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

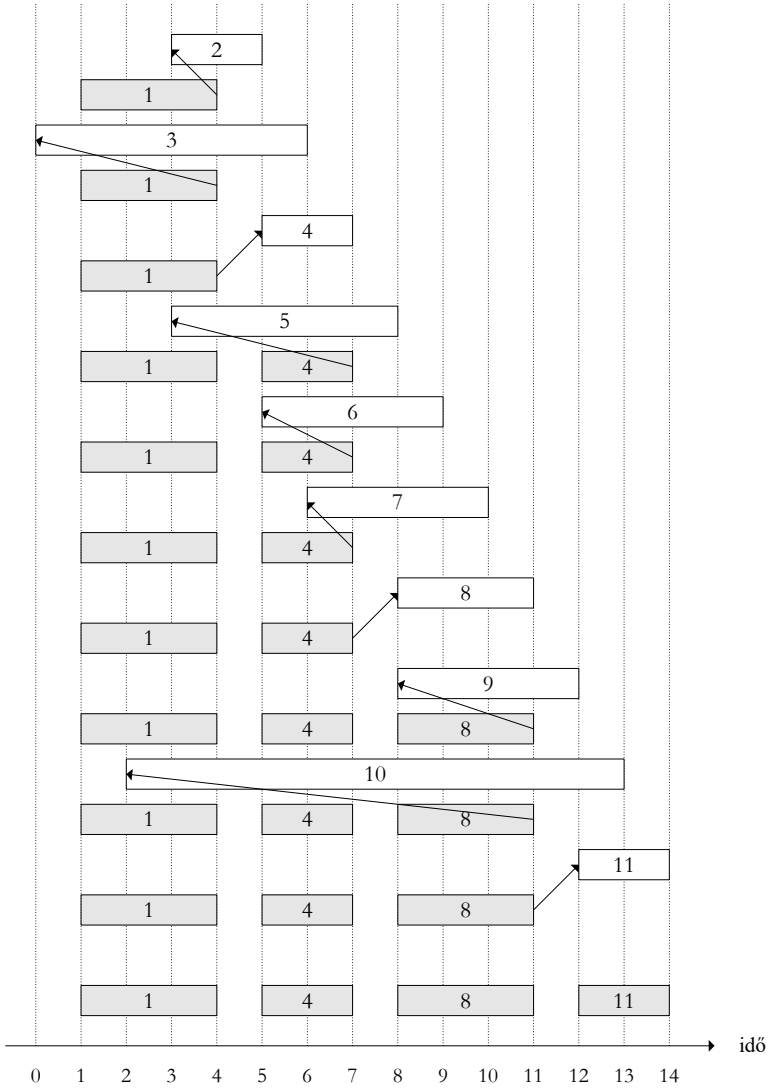
Megjegyzés: Mivel az eseményeket a befejező időpontok szerint nemcsökkenő sorrendben vizsgáljuk, így f_i mindig az A-beli események befejező időpontjainak maximuma, vagyis $f_i = \max\{f_k: k \in A\}$.

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

Tétel: 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*=< 8, 3, 5, 12, 3, 6, 10, 6, 17 >
  - *f*=< 12, 6, 8, 14, 7, 9, 15, 7, 20 >

Hátizsák feladat

- A bináris (vagy 0−1) hátizsák feladat

Adott egy hátizsák és különböző használati tárgyak egy halmaza. Minden egyes tárgynak adott a súlya és az értéke, valamint a hátizsákban elszállítható rakomány maximális súlya.

Feladat: A rendelkezésre álló tárgyakból egy olyan rakomány összeállítása, amely szállítható (a súlya nem haladja meg a rakomány súlykorlátját) és emellett maximális értékű.

- Jelölések
  - m : a használati tárgyak száma,
  - a_j : a j-edik tárgy súlya, (j=1, …, m),
  - c_j : a j-edik tárgy értéke, (j=1, …, m),
  - b : a hátizsák rakományának súlykorlátja,
  - $x_j = \begin{cases} 1, & \text{ha a } j\text{−edik tárgy bekerül a rakományba,} \\ 0 & \text{különben.} \end{cases}$

- Optimumszámítási modell

$$\sum_{j=1}^{m} a_j x_j \leq b$$
$$x_j \in \{0,1\}, (j = 1, \ldots, m)$$
$$\sum_{j=1}^{m} c_j x_j \rightarrow \max$$

Hátizsák feladat

- Töredékes hátizsák feladat

A tárgyak töredéke is választható.

- Egészértékű hátizsák feladat

A bináris hátizsák feladat egyik **általánosítása** az, amikor a rendelkezésre álló tárgyak több példányban, esetleg korlátlan mennyiségű példányban használhatók fel. Ilyenkor az x_j változó azt adja meg, hogy a j-edik tárgyból hány példány kerül a rakományba.

- Optimumszámítási modell

$$\frac{\begin{gathered}\sum_{j=1}^{m} a_j x_j \leq b \\ x_j \geq 0 \text{ és } x_j \text{ egész } (j = 1, \ldots, m)\end{gathered}}{\sum_{j=1}^{m} c_j x_j \rightarrow \max}$$

- Az egészértékű hátizsák feladatok fajtái
  - **Korlátos**: Minden x_j változóra adott egy felső korlát.
  - **Nemkorlátos**: Nincs minden x_j változóra felső korlát.
  - **Általános hátizsák feladat**: Egyetlen x_j változóra sincs felső korlát.

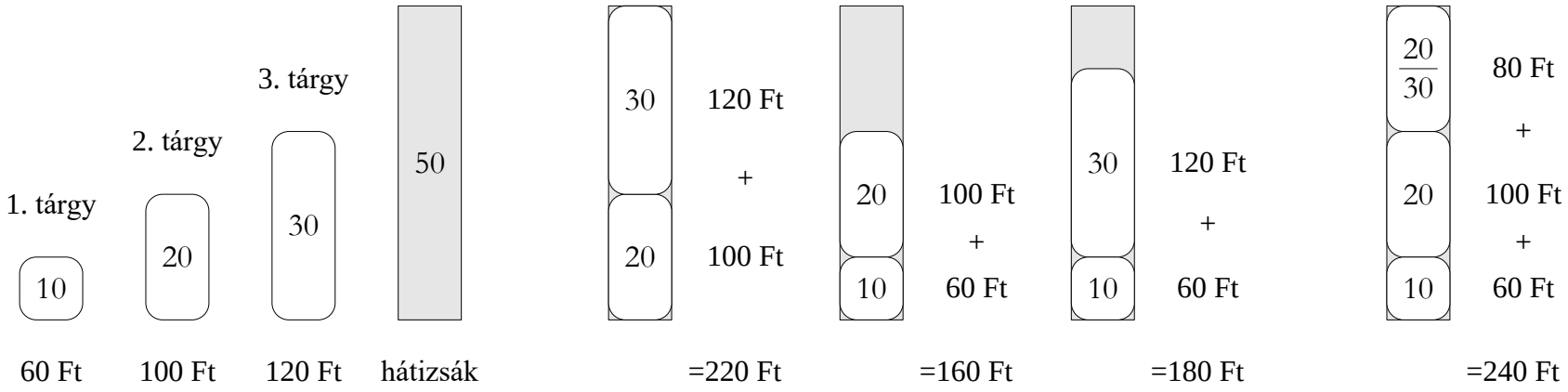
Megjegyzés: Csak olyan feladatokat fogunk vizsgálni, ahol a_j, c_j, (j=1,…, m) és b egész.

Hátizsák feladat

A 0-1, és a töredékes hátizsák feladat is teljesíti az optimális részproblémák tulajdonságot.

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 (c_j/a_j) szerinti mohó választással).



A 0-1 és a töredékes hátizsák feladat

Hátizsák feladat

Tétel: A 0-1 hátizsák feladat optimális megoldás létezését és meghatározását illetően elegendő olyan típusú feladatok vizsgálatára szorítkozni, amelyekben minden a_j, c_j együttható pozitív egész.

Az alábbi eljárás tetszőleges 0-1 hátizsák feladathoz (ahol a_j, c_j egész együtthatók ≤ 0 értékűek is lehetnek) konstruál egy olyan új hátizsák feladatot (ahol a_j, c_j együtthatók pozitív egészek), hogy a két feladatnak egyidejűleg létezik optimális megoldása, és az új feladat optimális megoldásából közvetlenül származtatható a kiindulási feladat optimális megoldása.

1. Minden olyan $j \in \{1, \ldots, m\}$ indexre, amelyre $c_j \leq 0$ és $a_j \leq 0$, helyettesítsük x_j -t $(1 - x'_j)$ -vel.
2. Minden olyan $j \in \{1, \ldots, m\}$ indexre, ha $c_j > 0$ és $a_j \leq 0$, adjuk az x_j változónak az 1 értéket és rendezzük át a feladatot.
3. Minden olyan $j \in \{1, \ldots, m\}$ indexre, ha $c_j \leq 0$ és $a_j > 0$, adjuk az x_j változónak a 0 értéket.

Feladatok

- Milyen új hátizsák feladatot konstruál az előző eljárás az alábbi feladathoz?

$$\frac{x_2 + 2x_3 + 3x_4 - x_5 - 2x_6 - 2x_7 \leq 4}{x_1 + 3x_3 - x_4 - 2x_5 + x_6 - x_7 = z \rightarrow \max}$$