Haladó informatikai algoritmusok

Mohó algoritmusok – „Élj a mának!”

(Dinamikus programozás – „Az intelligens ember hosszútávon gondolkodik.”)

Bevezetés

- A dinamikus programozás bizonyos feladatok esetében túl sok esetet vizsgál.
- Mohó algoritmus: az adott lépésben optimálisnak látszó választást teszi → lokális optimumot választ abban a reményben, hogy ez globális optimumhoz fog vezetni

A mohó stratégia elemei

1. A probléma optimális szerkezetének meghatározása.
2. Rekurzív megoldás kifejlesztése.
3. Annak bizonyítása, hogy minden rekurzív lépésben az egyik optimális választás a mohó választás. Tehát mindig biztonságos a mohó választás.
4. Annak igazolása, hogy a mohó választás olyan részproblémákat eredményez, amelyek közül legfeljebb az egyik nem üres.
5. A mohó stratégiát megvalósító rekurzív algoritmus kifejlesztése.
6. A rekurzív algoritmus átalakítása iteratív algoritmussá.

A mohó stratégia jellemzői

- A mohó algoritmus mindig az adott lépésben optimálisnak látszó választást teszi.
- Csak egy részproblémát kell vizsgálni az optimális megoldáshoz.
- Minden részproblémát felülről-lefelé haladó módon meg tudunk oldani.

Esemény-kiválasztási probléma

- Tegyük fel, hogy adott események egy $S = \{a_1, a_2, \ldots, a_n\}$ n elemű halmaza, amelyek egy közös erőforrást használnak, például egy előadótermet vagy egy nyomtatót, amit egy időben csak egyik használhat.
- Minden a_i eseményhez adott az s_i kezdő időpont és az f_i befejező időpont, ahol $s_i < f_i$.
- Ha az a_i eseményt kiválasztjuk, akkor ez az esemény az $[s_i, f_i)$ félig nyitott idő intervallumot 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 a_i és a_j kompatibilisek, ha $s_i \geq f_j$ vagy $s_j \geq f_i$).
- Az esemény-kiválasztási probléma azt jelenti, hogy kiválasztandó kölcsönösen kompatibilis eseményeknek egy legnagyobb elemszámú halmaza.

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

Optimális részproblémák szerkezete 1

- **1. feladat**: megtaláljuk az optimális szerkezetet, és felépítjük a feladat optimális megoldását a részproblémák optimális megoldásaiból.
- Definiáljuk a következő halmazokat: $S_{i,j} = \{a_k \in S | f_i \leq s_k < f_k \leq s_j\}$ ($S_{i,j}$ azokat az S-beli eseményeket tartalmazza, amelyek a_i befejeződése után kezdődhetnek, és befejeződnek az a_j kezdete előtt)
- A teljes probléma kezeléséhez egészítsük ki az eseményhalmazt az a_0 és a_{n+1} eseményekkel, ahol $f_0 = 0$, $s_{n+1} = \infty$
  Ekkor $S = S_{0,n+1}$, és a részproblémák indexeinek tartománya:
  $0 \leq i, j \leq n + 1$.

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

$$S_{1,11} = \{a_k \in S | f_1 \leq s_k < f_k \leq s_{11}\} = \{a_4, a_6, a_7, a_8, a_9\}$$

Optimális részproblémák szerkezete 2

- Tegyük fel, hogy az események a befejezésük szerint monoton nem-csökkenő sorrendbe rendezettek.

  $f_0 \leq f_1 \leq f_2 \leq \ldots \leq f_n < f_{n+1}$

- A részprobléma szerkezetének meghatározásához tekintsünk egy nem üres $S_{i,j}$ részproblémát, és tegyük fel, hogy valamely a_k eleme a megoldásnak, azaz

  $f_i \leq s_k < f_k \leq s_j$.

- Az a_k eseményt használva két részproblémát kaphatunk:

  $S_{i,k}$, $S_{k,j}$

- $S_{i,j}$ megoldását megkapjuk, ha az $S_{i,k}$ és $S_{k,j}$ megoldásának egyesítéséhez hozzávesszük az a_k eseményt.

Optimális részproblémák szerkezete 3

Optimális részproblémák szerkezete:

- Tegyük fel, hogy $A_{i,j}$ egy optimális megoldása az $S_{i,j}$ részproblémának és $a_k \in A_{i,j}$.
- Ekkor az $A_{i,k}$ megoldás optimális megoldása kell legyen az $S_{i,k}$ részproblémának, és az $A_{k,j}$ megoldás optimális megoldása kell legyen az $S_{k,j}$ részproblémának.
- Ezután megmutatjuk, hogy az eredeti probléma optimális megoldása felépíthető a részproblémák optimális megoldásiból.

Rekurzív megoldás

- Legyen ***c[i, j]*** az $S_{i,j}$ részprobléma maximális elemszámú, kölcsönösen kompatibilis eseményeket tartalmazó részhalmaz elemszáma.
- ***c[i, j] = 0***, ha $S_{i,j} = \emptyset$ és ***c[i, j] = 0***, ha ***i > j***
- Kapjuk a következő rekurzív összefüggést:

  c[i, j] = c[i, k] + c[k, j] + 1

- ***k*** értékét nem ismerjük, összesen ***j-i-1*** lehetséges értéket vehet fel (*k=i+1, … , j-1*)
- A teljes rekurzív alak:

$$c[i,j] = \begin{cases} 0, \text{ ha } S_{i,j} = \emptyset \\ max_{i<k<j;\ a_k \in S_{i,j}} \{c[i,k] + c[k,j] + 1\}, \text{ ha } S_{i,j} \neq \emptyset \end{cases}$$

$$c[i,j] = \begin{cases} 0, \text{ ha } S_{i,j} = \emptyset \\ max_{i<k<j;\ a_k \in S_{i,j}} \{c[i,k] + c[k,j] + 1\}, \text{ ha } S_{i,j} \neq \emptyset \end{cases}$$

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

$S_{1,11} = \{a_k \in S | f_1 \leq s_k < f_k \leq s_{11}\} = \{a_4, a_6, a_7, a_8, a_9\}$

$k = 4, 6, 7, 8, 9$

Megoldás lehet:

1. $\begin{pmatrix} 5 & 8 \\ 7 & 11 \end{pmatrix} = (a_4, a_8)$

2. $\begin{pmatrix} 5 & 8 \\ 7 & 12 \end{pmatrix} = (a_4, a_9)$

Átalakítás

A dinamikus programozási megoldás átalakítása mohó megoldássá:

Tétel: Tekintsünk egy $S_{i,j}$ nem üres részproblémát, és legyen

a_m a legkisebb befejezési idejű esemény $S_{i,j}$ -ben.

$$f_m = min\{f_k | a_k \in S_{i,j}\}$$

5	5	6	8	8
7	9	10	11	12

Pl. $S_{1,11} = \{a_4, a_6, a_7, a_8, a_9\}$

a_m

Ekkor

1. a_m eleme $S_{i,j}$ valamely maximális elemszámú, kölcsönösen kompatibilis eseményekből álló részhalmazának.
2. Az $S_{i,m}$ részprobléma üres, tehát a_m választásával legfeljebb az $S_{m,j}$ nem üres.

(Dinamikus programozás → 2 részprobléma vizsgálata)

Tétel segítségével→ csak 1 **részprobléma kell az optimális megoldáshoz**

Minden részproblémát **felülről-lefelé haladó** módon oldunk meg.

Tétel alapján:

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

$S_{0,n+1}$ -ből indulunk: $S_{0,n+1} = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}\}$

$a_{m1} = a_1$ m1=1

$S_{1,n+1} = \{a_k \in S \mid 4 \leq s_k < f_k \leq \infty\} = \{a_4, a_6, a_7, a_8, a_9, a_{11}\}$

$a_{m2} = a_4$ m2=4

$S_{4,n+1} = \{a_k \in S \mid 7 \leq s_k < f_k \leq \infty\} = \{a_8, a_9, a_{11}\}$

$a_{m3} = a_8$ m3=8

$S_{8,n+1} = \{a_k \in S \mid 11 \leq s_k < f_k \leq \infty\} = \{a_{11}\}$

$a_{m4} = a_{11}$ m4=11

$S_{11,n+1} = \{a_k \in S \mid 14 \leq s_k < f_k \leq \infty\} = \emptyset$

Megoldás:

a_1	a_4	a_8	a_{11}
1	5	8	12
4	7	11	14

Rekurzív mohó algoritmus 1

Bemenő paraméterei:

- az események kezdő és befejező időpontjait tartalmazó ***s*** és ***f*** tömb
- a megoldandó $S_{i,n+1}$ részproblémát meghatározó ***i*** és ***n*** sorszám

Az eljárás $S_{i,n+1}$ **egy maximális elemszámú, kölcsönösen kompatibilis eseményeket tartalmazó részhalmaz**át adja eredményül.

Feltételezzük, hogy az ***n*** bemeneti esemény befejezési idő szerint monoton nem-csökkenő sorrendbe rendezett.

A kiindulási probléma megoldását a

REKURZIV – ESEMENY – KIVALASZTO(s, f, 0, n)

eljáráshívás adja.

Rekurzív mohó algoritmus 2

$REKURZIV - ESEMENY - KIVALASZTO(s, f, i, n)$

1. $m \leftarrow i + 1$
2. **while** $m \leq n$ és $s_m < f_i$
3. **do** $m \leftarrow m + 1$
4. **if** $m \leq n$
5. **then** return $\{a_m\} \bigcup REKURZIV - ESEMENY - KIVALASZTO(s, f, m, n)$
6. **else** return $\emptyset$

Rekurzív mohó algoritmus alapján:

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

1. REK(s,f,0,11) m:=1 1≤11? és 1<0?; 1≤11? **a_1**
2. REK(s,f,1,11) m:=2 2≤11? és 3<4? m:=2+1=3; 3≤11? és 0<4? m:=3+1=4; 4≤11? és 5<4?; 4≤11? **a_4**
3. REK(s,f,4,11) m:=5 5≤11? és 3<7? m:=5+1=6; 6≤11? és 5<7? m:=6+1=7; 7≤11? és 6<7?; m:=7+1=8; 8≤11? és 8<7?; 8≤11? **a_8**
4. REK(s,f,8,11) m:=9 9≤11? és 8<11? m:=9+1=10; 10≤11? és 2<11? m:=10+1=11; 11≤11? és 12<12?; 11≤11? **a_{11}**
5. REK(s,f,11,11) m:=12 12≤11? Nincs további

Megoldás: **a_1, a_4, a_8, a_{11}**

$REKURZIV-ESEMENY-KIVALASZTO(s,f,i,n)$

1. $m \leftarrow i+1$
2. **while** $m \leq n$ és $s_m < f_i$
3. **do** $m \leftarrow m+1$
4. **if** $m \leq n$
5. **then return** $\{a_m\} \bigcup REKURZIV-ESEMENY-KIVALASZTO(s,f,m,n)$
6. **else return** $\emptyset$

Iteratív mohó algoritmus

A MOHÓ-ESEMÉNY-KIVÁLASZTÓ eljárás egy **iteratív változata** a REKURZÍV-ESEMÉNY-KIVÁLASZTÓ eljárásnak.

Ez ***feltételezi***, hogy a bemeneti események befejezési idejük szerint monoton nem-csökkenő sorrendbe rendezettek.

```
1. n ← hossz[s]
2. A ← {a_1}
3. i ← 1
4. for m ← 2 to n
5.     do if s_m ≥ f_i
6.         then A ← A ⋃ {a_m}
7.              i ← m
8. return A
```

f_i mindig a legnagyobb befejezési idejű esemény az A halmazban, tehát $f_i = \max\{f_k \mid a_k \in A\}$.

Futási idő: Θ(n)

Iteratív mohó algoritmus alapján:

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

n:=11; A:={a_1};

i:=1 m=2 3≥4?; m=3 0≥4?; m=4 5≥4?
A:=AU{a_4};

i:=4 m=5 3≥7?; m=6 5≥7?; m=7 6≥7?; m=8 8≥7? A:=AU{a_8};

i:=8 m=9 8≥11?; m=10 2≥11?; m=11 12≥11?
A:=AU{a_{11}};

i:=11

```
1. n ← hossz[s]
2. A ← {a_1}
3. i ← 1
4. for m ← 2 to n
5.     do if s_m ≥ f_i
6.         then A ← A ⋃ {a_m}
7.              i ← m
8. return A
```

Megoldás: a_1, a_4, a_8, a_{11} maximális elemszámú kölcsönösen kompatibilis eseményekből álló halmaz

Mohó stratégia vagy dinamikus programozás

- az optimális részproblémák tulajdonságot kihasználjuk mind a mohó, mind a dinamikus programozási stratégiáknál,
- előfordulhat, hogy dinamikus programozási megoldást próbálunk adni akkor, amikor mohó megoldás is célravezető lenne, és fordítva

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.

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

Mely tárgyakat rakjuk a hátizsákba?

A töredékes hátizsák feladat: a tárgyak töredéke is választható, nem kell 0 – 1 bináris választást tenni.

Feladatok elemzése 1

Mindkét hátizsák feladat teljesíti az **optimális részproblémák tulajdonság**ot.

- A 0 – 1 **feladat esetén** tekintsünk egy olyan választást, amely a legnagyobb használati értéket adja, de a tárgyak összsúlya nem haladja meg a ***W*** értéket. Ha kivesszük a ***j***-edik tárgyat a hátizsákból, akkor a bennmaradt tárgyak használati értéke a legnagyobb lesz azon feltétel mellett, hogy az összsúly nem nagyobb, mint $\boldsymbol{W - w_j}$, és ***n – 1*** tárgyból választhatunk, kizárva az eredeti tárgyak közül a ***j***-ediket.
- A **töredékes hátizsák feladatnál** ha egy optimális választásból kiveszünk a ***j*** tárgyból ***w*** mennyiséget, akkor a megmaradt választás optimális lesz arra az esetre, amikor legfeljebb ***W –*** w összsúlyt érhetünk el és a ***j***-edik tárgyból legfeljebb $\boldsymbol{w_j - w}$ mennyiséget választhatunk.

Feladatok elemzése 2

- **a** 0 – 1 **hátizsák feladat megoldható dinamikus programozással**: ha egy tárgy beválasztásáról döntünk, akkor előbb össze kell hasonlítani annak a két részproblémának a megoldását, amely a tárgy beválasztásával, illetve kihagyásával adódik. Az így megfogalmazott probléma sok, egymást átfedő részproblémát eredményez
- **a töredékes hátizsák feladat megoldható mohó stratégiával**: megoldásához előbb számítsuk ki minden tárgyra a v_i/w_i használati érték per súly hányadost. A mohó stratégiát követve először a legnagyobb hányadosú tárgyból választunk amennyit csak lehet. Ha elfogyott, de még nem telt meg a hátizsák, akkor a következő legnagyobb hányadosú tárgyból választunk amennyit csak lehet, és így tovább, amíg a hátizsák meg nem telik.

Annak bemutatására, hogy a mohó stratégia nem működik a 0-1 hátizsák feladatra, tekintsük a következő feladatot:

i	1	2	3	4	5
v_i	40	55	140	180	120
w_i	20	50	35	60	20

W=100

i	1	2	3	4	5
v_i	120	140	180	40	55
w_i	20	35	60	20	50

Mohó választás: 20+35=55

Dinamikus választás: 35+60=95

Egységnyi végrehajtású munkák ütemezése

Egy vállalkozó minden munkája egy napig tart és egyszerre csak egy munkán tud dolgozni. Minden megrendelés határidős, így amit elvállal, azt határidőre el kell végeznie. Minden elvégzett munka után meghatározott haszon jár.

A vállalkozó a következő időszakra beérkezett megrendelések közül ki akar választani egy olyan részhalmazt, amely a lehető legtöbb hasznot eredményezi.

Adjunk meg egy megoldást a következő időszaki megrendelések egy lehető legnagyobb elemszámú részhalmazának a kiválasztására és ütemezésére annak érdekében, hogy a kiválasztott munkákat határidőre el tudja végezni, és az összes haszon a lehető legnagyobb legyen.

Egységnyi végrehajtású munkák ütemezése

Bemenet: Az *utemez_be* állomány első sora a megrendelések ***n*** számát (1 ≤ n ≤ 1000) tartalmazza. A következő ***n*** sor mindegyikében két pozitív egész szám van egy-egy szóközzel elválasztva, az adott megrendelés ***h*** határideje (1 ≤ n ≤ 365), és a munka után járó ***p*** haszon.

Kimenet: Az *utemez_ki* állomány első sorában a kiválasztott munkák ***k*** száma legyen. A következő ***k*** sor mindegyikébe két számot kell írni egy-egy szóközzel elválasztva. Az első szám a kiválasztott munka száma legyen, a másik annak a napnak a sorszáma, amelyiken az adott munkát el kell végezni.

utemez_be:
n=6

	h	p
1.	3	7
2.	2	4
3.	7	2
4.	4	6
5.	2	4
6.	1	3

A határidők szerint a munkákat sorba rendezzük:

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

Legyen ***M*** a még beosztásra váró munkák halmaza, ***S*** a még szabad napok halmaza: M={6,2,5,1,4,3}; S={1,2,3,4,5,6,7}

1. A legnagyobb hasznú munkát választjuk és ha van olyan szabad nap, amely nem nagyobb, mint a munka határideje, akkor ütemezzük be a munkát a legnagyobb ilyen szabad napra.
2. A munkát töröljük a beosztandó munkák halmazából.

1. M={6,2,5,1,4,3}; S={1,2,3,4,5,6,7}; 1. (3 7) ➔ M={6,2,5,4,3}; S={1,2,4,5,6,7}; → **(1 3)**
2. M={6,2,5,4,3}; S={1,2,4,5,6,7}; 4. (4 6) ➔ M={6,2,5,3}; S={1,2,5,6,7}; → **(4 4)**
3. M={6,2,5,3}; S={1,2,5,6,7}; 2. (2 4) ➔ M={6,5,3}; S={1,5,6,7}; → **(2 2)**
4. M={6,5,3}; S={1,5,6,7}; 5. (2 4) ➔ M={6,3}; S={5,6,7}; → **(5 1)**
5. M={6,3}; S={5,6,7}; 6. (1 3) ➔ M={6,3}; S={5,6,7}; → Nem tudom választani, mert az 1. napra nem tudom betenni
6. M={6,3}; S={5,6,7}; 3. (7 2) ➔ M={6}; S={5,6}; → **(3 7)**

utemez_ki:
k=5

1.	5	1
2.	1	3
3.	2	2
4.	4	4
5.	3	7