Algoritmuselmélet

Dinamikus programozás

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Maximális hosszú növekvő intervallum

Input: $A[1:n]$ különböző egész számokból álló tömb

Output: Egy maximális méretű $A[i,j]$ résztömb hossza, aminek az elemei növekszenek

Példa: A $7, 5, \underbrace{2, 8, 9}, 3, 1, 6, 4$ tömbben egy maximális hosszú növekvő intervallum a $2, 8, 9$.
Négy hosszú növekvő intervallum nincs.

```
1:  procedure MNI(A[1 : n])
2:      hossz := 1                    ▷ az utolsó növekvő intervallum hossza
3:      max := 1                      ▷ az eddigi leghosszabb
4:      for i = 2 to n do
5:          if A[i] > A[i − 1] then
6:              hossz := hossz + 1    ▷ az aktuális intervallum nőtt
7:          else                      ▷ az aktuális intervallum vége
8:              if max < hossz then
9:                  max := hossz
10:             end if
11:             hossz := 1            ▷ új intervallum kezdődik
12:         end if
13:     end for
14:     if max < hossz then
15:         max := hossz
16:     end if                        ▷ ha az utolsó intervallum a leghosszabb
17: end procedure
```

Lépésszám: $O(n)$

Maximális hosszú növekvő részsorozat

Input: $A[1:n]$ különböző egész számokból álló tömb

Output: Egy maximális méretű növekvő részsorozat $A[i_1] < A[i_2] < \cdots < A[i_k]$ $(i_1 < i_2 < \ldots < i_k)$ hossza.

Példa: A $9, 5, \underline{2}, 8, 7, \underline{3}, 1, 6, \underline{4}$ tömbben egy maximális hosszú növekvő részsorozat a $2, 3, 6$ és a $2, 3, 4$ is.
Négy hosszú növekvő részsorozat nincs.

Ötlet: A leghosszabb $A[j]$-re végződő részsorozat vagy csak $A[j]$-ből áll, ha nincsen ennél kisebb korábbi elem vagy egy korábbi i-nél végződő monoton növő folytatása, amennyiben $A[i] < A[j]$.

Definíció

$L[i] :=$ *a leghosszabb* $A[i]$*-re végződő részsorozat hossza.*

Maximális hosszú növekvő részsorozat

Lemma

$$L[j] := 1 + \max\{L[i] \mid i < j \text{ és } A[i] < A[j]\}$$

Bizonyítás.

Ha lenne ennél hosszabb $A[j]$-re végződő részsorozat, akkor ennek utolsó elemét elhagyva egy olyan részsorozatot kapunk, ami valamely $A[i] < A[j]$-re végződik ($i < j$), de hosszabb, mint $L[i]$. Ez ellenmond annak, hogy az ilyenek maximumát számoltuk ki. □

Lemma

A leghosszabb növekvő részsorozat hossza $\max\{L[i] \mid 1 \leq i \leq n.\}$

Bizonyítás.

A leghosszabb növekvő részsorozat valamelyik $A[i]$-re végződik. Ezt biztos tekintetbe vesszük a maximumnál. □

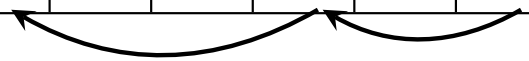
Maximális hosszú növekvő részsorozat

Hogyan találhatjuk meg egy maximális hosszú növekvő részsorozat elemeit?

- $L[j]$ kiszámolásakor jegyezzük fel, hogy melyik $L[i]$-nél vette fel a maximumot.
- Legyen ez az $i = \text{előző}[j]$, ez egy leghosszabb $A[j]$-re végződő részsorozat utolsó előtti eleme.
- Amikor a végén az összes $L[i]$ közül kiválasztjuk a maximumot, ott is jegyezzük meg, hogy melyik volt a maximális.
- Innen visszafelé lépkedve végignézhetjük a maximális részsorozat elemeit.

Maximális hosszú növekvő részsorozat

$i =$	1	2	3	4	5	6	7	8	9
$A[i] =$	9	5	2	8	7	3	1	6	4
$L[i] =$	1	1	1	2	2	2	1	3	3
előző$[j] =$	-	-	-	2	2	3	-	6	6



Például:

$L[8] = 1 + \max\{L[2], L[3], L[6], L[7]\} = 1 + \max\{1, 1, 2, 1\} = 1 + 2 = 3$

Leghosszabb növekvő részsorozat hossza:

$$\max\{1, 1, 1, 2, 2, 2, 1, 3, 3\} = 3$$

A maximum például $i = 8$-nál van.
előző$[8] = 6$, előző$[6] = 3 \implies$ egy maximális növekvő részsorozat $A[3], A[6], A[8]$, azaz $2, 3, 6$.

Maximális hosszú növekvő részsorozat

```
1:  procedure MNR(A[1 : n])
2:      L[1] := 1
3:      for i = 2 to n do
4:          L[i] := 0
5:          for j = 1 to i − 1 do
6:              if A[j] < A[i] és L[j] > L[i] then
7:                  L[i] := L[j]
8:              end if
9:          end for
10:         L[i] := L[i] + 1
11:     end for
12:     return max{L[i] | 1 ≤ L[i] ≤ n}
13: end procedure
```

Lépésszám: $O(\sum_{i=1}^{n-1} i) + O(n) = O(n^2)$

Maximális összegű intervallum

Input: $A[1:n]$ különböző egész számokból álló tömb

Output: Egy maximális összegű $A[i,j]$ résztömb összege

Ha minden $A[i] \geq 0$, akkor könnyű a feladat: $A[1:n]$ lesz a maximális.

Példa: $-2, 1, -3, 4, -1, 2, 1, -5, 4$ esetén 6 a maximális összeg.

Ötlet: Mikor érdemes $A[i]$-t hozzávenni az előző intervallumhoz és mikor nem?

Definíció

$L[i] :=$ *a legnagyobb összegű* $A[i]$*-re végződő intervallum összege.*

Maximális összegű intervallum

Lemma

$$L[i] := \max\{A[i]; L[i-1] + A[i]\}$$

Bizonyítás.

Ha az $A[i]$-re végződő maximális összegű intervallum 1 hosszú, akkor az összeg $A[i]$. Különben az $A[i-1]$-re végződő maximális összegű intervallumhoz vesszük hozzá $A[i]$-t. □

Lemma

A maximális összegű intervallum összege $\max\{L[i] \mid 1 \leq i \leq n.\}$

Bizonyítás.

A leghosszabb növekvő részsorozat valamelyik $A[i]$-re végződik. Ezt biztos tekintetbe vesszük a maximumnál. □

Maximális összegű intervallum

$i =$	1	2	3	4	5	6	7	8	9
$A[i] =$	-2	1	-3	4	-1	2	1	-5	4
$L[i] =$	-2	1	-2	4	3	5	6	1	5

Például: $L[4] = \max\{A[4], L[3] + A[4]\} = \max\{4, -2\} = 4$
$L[7] = \max\{A[7], L[6] + A[7]\} = \max\{1, 6\} = 6$

Maximális összegű intervallum összege:

$$\max\{-2, 1, -2, 4, 3, 5, 6, 1, 5\} = 6$$

Lépésszám: $O(n) + O(n) = O(n)$

Dinamikus programozás

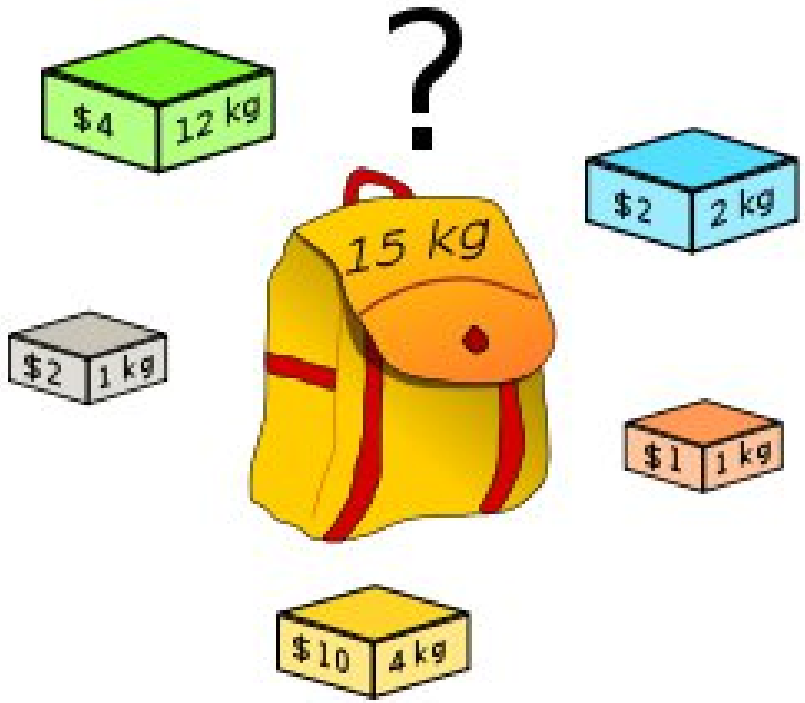
Optimalizálási feladatok megoldására használható módszer. Richard Bellman fejlesztette ki 1950 környékén részben a hadsereg megrendelésére. Az elnevezést a jobb eladhatóság miatt adta neki.

- Az eredetihez hasonló részfeladatokat tűzünk ki, ami lehet akár általánosabb is az eredeti feladatnál.
- A részfeladatokat általában kisebb inputra oldjuk először meg.
- A részfeladatok eredményét eltároljuk.
- Sorba rendezem a feladatokat úgy, hogy a későbbi feladatok megoldásánál fel tudjam használni a korábbiak eredményét.
- Az első néhány részfeladat legyen könnyen megoldható önmagában is.
- A későbbi feladatok eredményét a korábbiak eredményét felhasználva kapjuk. A részfeladatokat úgy határozzuk meg, hogy ez könnyen menjen.
- Az összes részfeladat megoldásából már könnyen megkapható az eredeti feladat megoldása.

Hátizsák probléma

Probléma

Adottak néhány tárgy, $s_1, \ldots, s_m$ a tárgyak súlyai, $v_1, \ldots, v_m$ a tárgyak értékei és C egy súlykorlát. (Minden érték pozitív egész.) Ki akarunk választani a tárgyaknak egy $I \subseteq \{1, \ldots, m\}$ részhalmazát, melyre teljesül, hogy $\sum_{i \in I} s_i \leq C$ és $\sum_{i \in I} v_i$ maximális. Mekkora lehet ez a maximum?



Hátizsák probléma, példa

	1	2	3	4
s=	4	7	5	2
v=	6	11	7	1
	↑		↑	

$C = 10$

$I = \{1, 3\}$ $\quad \sum_{i \in I} s_i = 9 \leq C$ $\quad \sum_{i \in I} v_i = 13$

Brute-force: $\Omega(2^m)$ lépés, ennyi részhalmaz van.

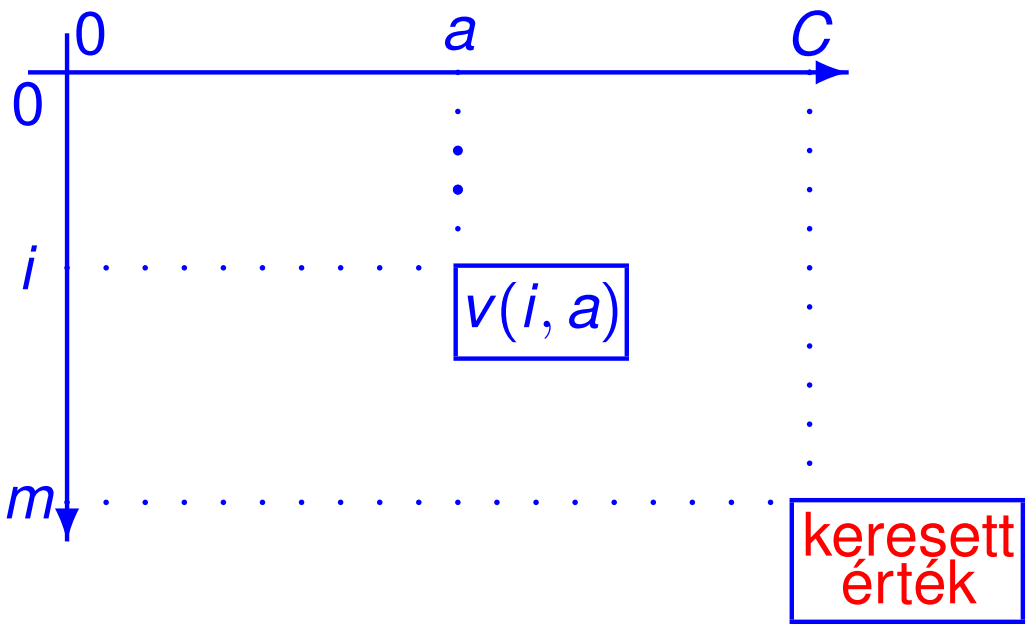
Hátizsák probléma megoldása

Először kisebb problémára oldjuk meg: $v(i, a)$ a maximális elérhető érték az $s_1, \ldots, s_i$ súlyokkal, $v_1, \ldots, v_i$ értékekkel és a súlykorláttal megadott feladatra.

$v(0, a) = 0 \ \forall a$-re $\Longleftarrow$ Ha üres a hátizsák, nulla az összérték.

$v(i, 0) = 0 \ \forall i$-re $\Longleftarrow$ Ha nulla méretű a hátizsák, nulla az összérték.

cél $\rightarrow v(m, C)$



Hátizsák probléma megoldása

Az optimális pakolásnál vagy betesszük az i-edik tárgyat a hátizsákba, vagy nem.

- Ha nem tesszük be: A maradék a helyre pakoljunk minél nagyobb értéket az első $i-1$ tárgyból: $v(i-1,a)$.
- Ha betesszük: A tárgy értéke v_i, súlya s_i. A maradék $a-s_i$ helyre pakoljunk minél nagyobb értéket az első $i-1$ tárgyból: $v(i-1,a-s_i)$.
  Ha $a-s_i<0$, akkor ez nem megy, ilyenkor 0 az összérték.

$$v(i,a)=\max\{v(i-1,a);v_i+v(i-1,a-s_i)\}$$

$\Longrightarrow$ A táblázat soronként kitölthető $\Longleftarrow$ minden érték két felette levő sorban található értékből számolható.

Összköltség: $O(Cm)$

C-től függ, nem $\log C$-től, ami az input mérete, így ez exponenciális idejű algoritmus. Ha C sokjegyű szám, ez sok idő.

Hátizsák probléma, példa

	1	2	3	4
s=	4	7	5	2
v=	6	11	7	1

$C = 10$

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	6	6	6	6	6	6	6
2	0	0	0	0	6	6	6	11	11	11	11
3	0	0	0	0	6	7	7	11	11	13	13
4	0	0	1	1	6	7	7	11	11	13	13

Animáció

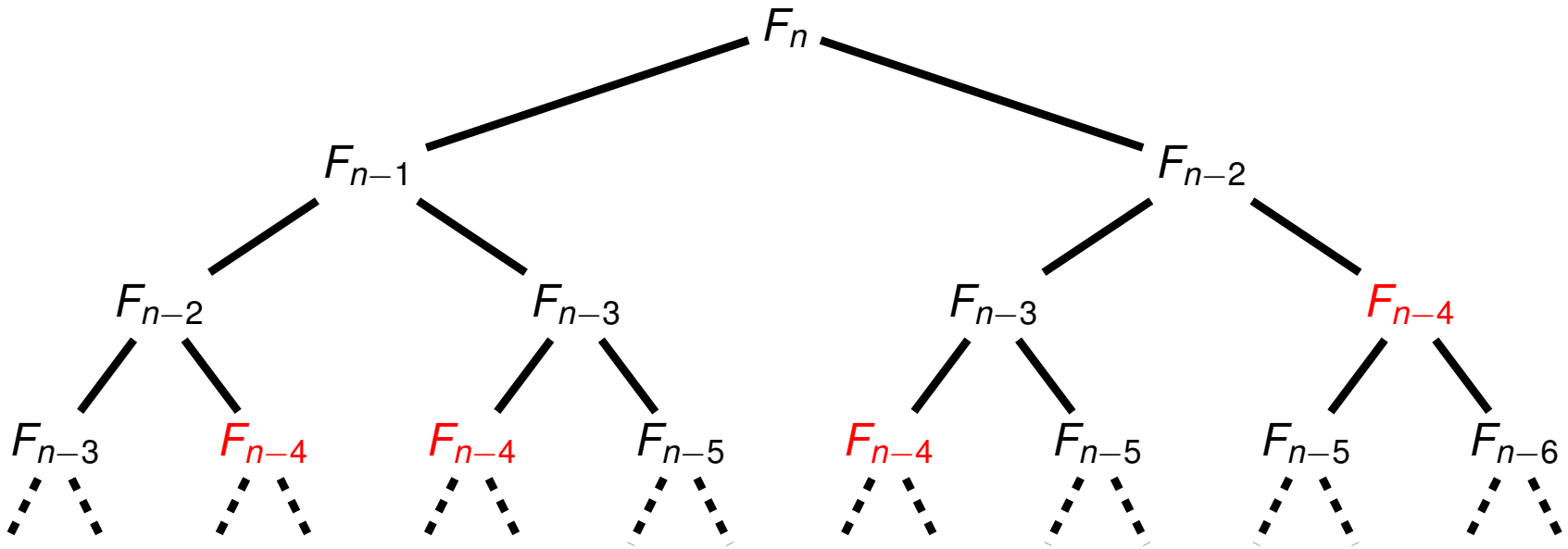
DP összehasonlítása a rekurzióval

A Fibonacci számok kiszámolása:

$$F_1 = F_2 = 0 \quad F_n = F_{n-1} + F_{n-2}$$

F_n dinamikus programozással (ami itt elég egyszerű) kiszámolható n összeadással.

DP összehasonlítása a rekurzióval



Fontos különbségek:

- A rekurzió egy-egy részfeladatot esetleg sokszor kiszámol, a dinamikus programozásnál mindent csak egyszer és az eredményt eltárolja.
- A rekurzió csak azokat számolja ki, ami feltétlen szükséges a végeredményhez. A dinamikus programozás minden részfeladatot kiszámol.