

Dinamikus programozás

- A dinamikus programozás (DP) bevezetése
- Példa: 0/1 hátizsák probléma
- Dinamikus programozás jellemzői
- Optimális részstruktúra ÉS átfedő részproblémák
- Példa: Leghosszabb közös részsorozat (LCS)

Hallgatói tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük *szükséges, de nem elégséges* feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, az előadásokon és gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint az előadásokon, és gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

A dinamikus programozás áttekintése

- **A *dinamikus programozás* (DP) széles körben használt optimalizációs problémák megoldására:**
 - ütemezési feladatok
 - legrövidebb út keresése
 - utazó ügynök
 - hasonló fehérjesorozatok, vagy aminosavak keresése
 - string-feladatok
 - elválasztási problémák
 - dinamikus idő-vetemítés
 - sztereó képfeldolgozás
 - csomagolási probléma, stb.
- **A problémákat részproblémákra bontjuk és ezek megoldásait kombináljuk, hogy a nagyobb probléma megoldását megtaláljuk.**
- **Az „oszd meg és uralkodj” típusú megoldási módszerrel szemben, itt a részproblémák között kapcsolatok, átfedések állhatnak fent (*átfedő részproblémák*) ÉS optimális részstruktúrákból építkezünk.**
- **Az elnevezés matematikai optimalizálásra utal.**

0/1 HÁTI ZSÁK PROBLÉMA

0/1 hátizsák probléma

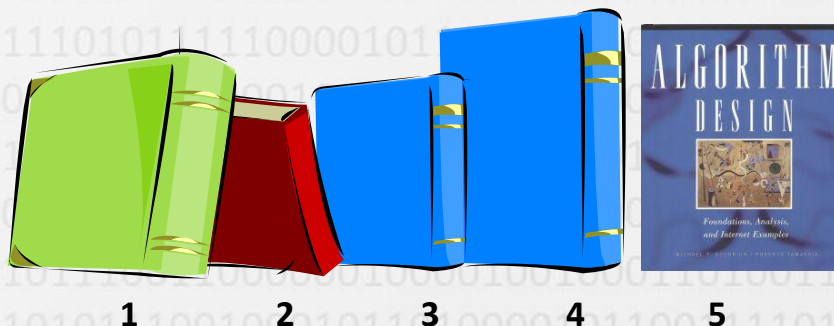
0/1 knapsack problem

- „A thief is robbing the King’s treasury, but he can only carry a load weighing at most W ...”
- Adott: S halmaz n elemmel és egy súlyhatár c , minden i elemnek
 - p_i – pozitív egész értéke van (*profit*)
 - w_i – és pozitív egész súlya
- Cél: úgy válasszuk ki az elemeket, hogy az összérték maximális legyen, de az összsúly kisebb legyen, mint c .
 - T jelölje azokat az elemeket, amit kiválasztunk, $T \subseteq S$
 - Cél: összérték maximalizálása $\sum_{i \in T} p_i$
 - Feltétel: a súlyhatáron belül kell maradni $\sum_{i \in T} w_i \leq c$

0/1 hátizsák probléma: példa

- Adott: n elemű S halmaz, minden elem
 - p_i – pozitív értékű és w_i – pozitív súlyú
- Cél: válasszunk ki elemeket, hogy az összérték maximális legyen, de az összsúly ne haladja meg c -t.
- Összesen 2^n eset lehet! Minden eset kipróbálása: „brute force”

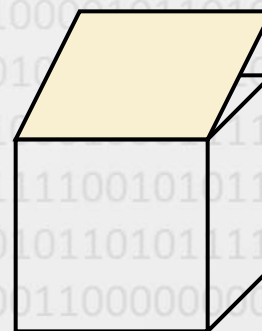
Elemek:



Súly (w_i): 4 N 2 N 2 N 6 N 2 N

Érték (p_i): 20 3 6 25 80

“zsák”



$c = 9$ N

Megoldás:

- 5 (2 N)
- 3 (2 N)
- 1 (4 N)

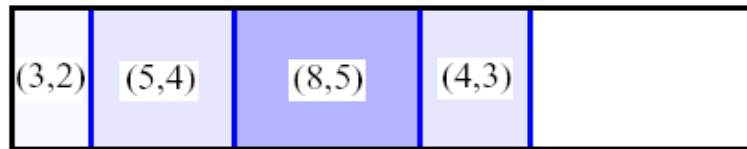
Összérték:

- $80 + 6 + 20$
 $= 106$

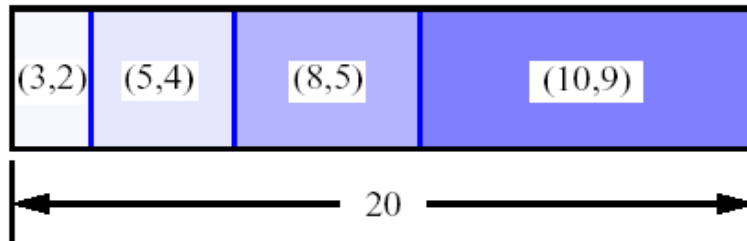
0/1 hátizsák probléma, első próbálkozás

- S_i : A halmaz elemeinek azonosítója 1-től i -ig ($i \leq n$).
- Definiálja $F[i]$ = a legjobb kiválasztást S_i -ből, azaz az elemek 1-től i -ig.
- Legyen pl. $S = \{(3,2), (5,4), (8,5), (4,3), (10,9)\}$ érték-súly párok, és $c = 20$ kapacitás

A legjobb S_4 ,
ha csak négyet veszünk ki:
Összsúly: 14, érték: 20



A legjobb S_5 :
az 5. benne van, de a 4. nem



- Rossz hír: S_4 nem része az S_5 optimális megoldásnak

0/1 hátizsák probléma, más megközelítés

- S_i : Az elemek halmaza 1-től i -ig, $i \leq n$.
- Definiálja $F[i, x]$ = az S_i halmazból a legjobb kiválasztást, ahol a súly legfeljebb x – további paraméter, és a 0 lehetséges súlyt is figyelembe vesszük: $x = 0, 1, 2, \dots, c$
- Ez optimális részprobléma megoldáshoz vezet.
- Az S_i legjobb kiválasztás legfeljebb x súllyal, két esetet jelenthet:
 - Ha $w_i > x$, akkor az i . elemet nem lehet hozzávenni, mert súlya nagyobb, mint az aktuális határ
 - Egyébként: ha a legjobb S_{i-1} részhalmaz súlya $x - w_i$ és ehhez vagy jön i , vagy nem eredményez nagyobb értéket

$$F[i, x] = \begin{cases} F[i-1, x] & \text{ha } w_i > x \\ \max\{F[i-1, x], F[i-1, x - w_i] + p_i\} & \text{egyébként} \end{cases}$$

0/1 hátizsák algoritmus

- $F[i, x]$ rekurzív formula:

$$F[i, x] = \begin{cases} F[i-1, x] & \text{ha } w_i > x \\ \max\{F[i-1, x], F[i-1, x - w_i] + p_i\} & \text{egyébként} \end{cases}$$

- $F[i, x]$ = az 1-től i -ig tartó elemekből a legjobb kiválasztás, ahol az összsúly legfeljebb x
- Alapeset: $i = 0$, nem került elem kiválasztásra, azaz ha nem veszünk figyelembe elemet a halmazból, akkor az **összérték 0**.
- A feladat megoldása: a legnagyobb érték az utolsó sorban (n), utolsó oszlopban (c)
- Futási idő: $O(n \times c)$.
Nem 2^n ideig tart.
- *Kivételesen a tömb indexeit 0-tól indítjuk, hogy az előzőkkel összhangban legyünk.*

0/1 hátizsák algoritmus

Függvény *0-1Knapsack*(S, n, c)

// **Input:** S halmaz elemei p_i értékkel, w_i súllyal $i = 1 \dots n$; valamint a max. súly c

// **Output:** a legjobb részhalmaz értéke ($F[n, c]$), hogy teljesül: összsúly $\leq c$

// Mivel 0 elem, vagy 0 súly esetén az összérték 0, ezért kivételesen 0-tól indexelünk

Ciklus $x \leftarrow 0$ -tól c -ig

$F[0, x] \leftarrow 0$

Ciklus vége

Ciklus $i \leftarrow 1$ -től n -ig

$F[i, 0] \leftarrow 0$

Ciklus vége

Ciklus $i \leftarrow 1$ -től n -ig

Ciklus $x \leftarrow 1$ -től c -ig

Ha $w_i \leq x$, **akkor**

$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$

Különben

$F[i, x] \leftarrow F[i - 1, x]$

Elágazás vége

Ciklus vége

Ciklus vége

return $F[n, c]$

Függvény vége

0/1 hátizsák példa

- $n = 4$ az adatok száma
- $c = 5$ kapacitás (maximális összsúly)
- Elemek:

i	súly (w_i),	profit (p_i)
1	2	3
2	3	4
3	4	5
4	5	6

Példa

$i \backslash x$	0	1	2	3	4	$5 = c$
0	0	0	0	0	0	0
1						
2						
3						
$4 = n$						

Ciklus $x \leftarrow 0$ -tól c -ig

$F[0, x] \leftarrow 0$

Ciklus vége

Példa

$i \backslash x$	0	1	2	3	4	$5 = c$
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
$4 = n$	0					

Ciklus $i \leftarrow 1$ -től n -ig

$F[i, 0] \leftarrow 0$

Ciklus vége

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 1$
 $p_i = 3$
 $w_i = 2$
 $x = 1$
 $x - w_i = -1$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3			
2	0					
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 1$
 $p_i = 3$
 $w_i = 2$
 $x = 2$
 $x - w_i = 0$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3		
2	0					
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 1$
 $p_i = 3$
 $w_i = 2$
 $x = 3$
 $x - w_i = 1$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	
2	0					
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 1$
 $p_i = 3$
 $w_i = 2$
 $x = 4$
 $x - w_i = 2$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0					
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 1$
 $p_i = 3$
 $w_i = 2$
 $x = 5$
 $x - w_i = 3$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0				
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 2$
$p_i = 4$
$w_i = 3$
$x = 1$
$x - w_i = -2$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3			
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 2$
$p_i = 4$
$w_i = 3$
$x = 2$
$x - w_i = -1$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4		
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 2$
$p_i = 4$
$w_i = 3$
$x = 3$
$x - w_i = 0$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 2$
$p_i = 4$
$w_i = 3$
$x = 4$
$x - w_i = 1$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 2$
$p_i = 4$
$w_i = 3$
$x = 5$
$x - w_i = 2$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0 ↓	3 ↓	4 ↓	4	7
3	0	0 ↓	3 ↓	4 ↓		
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 3$
$p_i = 5$
$w_i = 4$
$x = 1..3$
$x - w_i = -3..-1$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 3$
$p_i = 5$
$w_i = 4$
$x = 4$
$x - w_i = 0$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	5 = c
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4 = n	0					

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 3$
$p_i = 5$
$w_i = 4$
$x = 5$
$x - w_i = 1$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	$5 = c$
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
$4 = n$	0	0 ↓	3 ↓	4	5 ↓	

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben


$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 4$
$p_i = 6$
$w_i = 5$
$x = 1..4$
$x - w_i = -4..-1$

Példa

$w_i; p_i$
1: (2; 3)
2: (3; 4)
3: (4; 5)
4: (5; 6)

$i \setminus x$	0	1	2	3	4	$5 = c$
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
$4 = n$	0	0	3	4	5	7 

Ha $w_i \leq x$, akkor

$$F[i, x] \leftarrow \max(F[i - 1, x], F[i - 1, x - w_i] + p_i)$$

Különben

$$F[i, x] \leftarrow F[i - 1, x]$$

Elágazás vége

$i = 4$
$p_i = 6$
$w_i = 5$
$x = 5$
$x - w_i = 0$

Példa

- Az algoritmus a maximális összűlyt vette figyelembe úgy, hogy a zsákba tehető $F[n, c]$ érték a lehető legnagyobb legyen
- Az elemek kiolvasásához egy visszafele haladó algoritmus szükséges, amely a táblázatot használja
- S kimenet a kiválasztott elemek indexeit tartalmazza a kiolvasás végén:

Eljárás Kiolvas ($F[n, c], S$)

$S \leftarrow \emptyset$

$i \leftarrow n, x \leftarrow c$

Ciklus amíg ($i > 0$) és ($x > 0$)

Ha $F[i, x] \neq F[i - 1, x]$

// Jelöljük meg az i . elemet, hogy a zsákban van

$S \leftarrow S \cup \{i\}$

$x \leftarrow x - w_i$

Elágazás vége

$i \leftarrow i - 1$

Ciklus vége

0/1 hátizsák probléma

F tábla

n								
i					$F[i, j]$			
2								
1								
Súlyok	0	$j - w_i$		j		$c - 1$	c	

**A 0/1 hátizsák probléma F táblájának számítása. Az $F[i, j]$ meghatározásához szükséges kommunikáció az $i - 1$ sorban található elemekkel, amelyek tartalmazzák $F[i - 1, j]$ és $F[i - 1, j - w_i]$ elemeket.
Az i értéke ebben a táblázatban lentől felfelé nő!**

DINAMIKUS PROGRAMOZÁS

ELVE

A dinamikus programozás áttekintése

- **A problémamegoldás menete:**
 - Az optimális megoldás jellemzése.
 - Részproblémákra osztás úgy, hogy az összetevőktől való függés „körmentes” legyen
 - A részproblémák optimális megoldása rekurzívan.
 - Az optimális megoldások felhasználása az eredeti feladat optimális megoldásának megtalálásához: egy részprobléma megoldását gyakran egy, vagy több az előző szintű részproblémák függvényeként adják meg.
- **A rekurzív DP kifejezést *funkcionális egyenletnek*, vagy *optimalizációs egyenletnek* is nevezzük.**
- **A dinamikus programozás (DP) során az (átfedő) részmegoldások eredményeit gyakran táblázatokban tároljuk, hogy azokat – szemben a rekurzív megközelítésekkel – ne kelljen újra és újra kiszámolni (*memoization*, *NEM memorization*).**

Dinamikus programozás

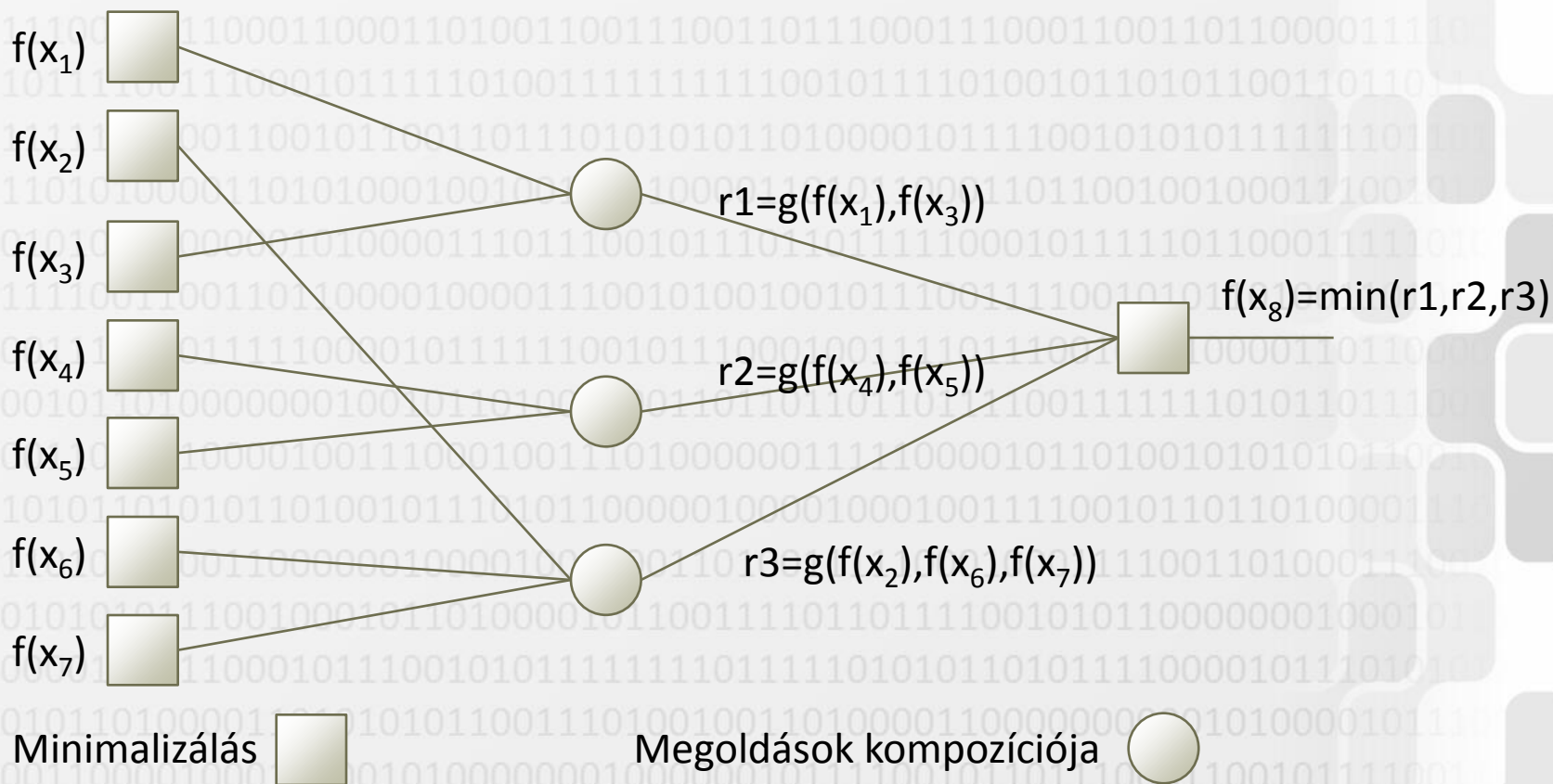
- Eleget kell tennünk bizonyos feltételeknek
- A DP formátumú probléma megoldását tipikusan a lehetséges megoldások minimumaként vagy maximumaként fejezzük ki.
 - Ha r az x_1, x_2, \dots, x_l részproblémák kompozíciójából meghatározott megoldás költségét jelenti, akkor r a következő alakban írható fel

$$r = g(f(x_1), f(x_2), \dots, f(x_l)).$$

Itt g az un. kompozíciós függvény.

Ha minden probléma optimális megoldása az átfedő részfeladatok optimális megoldásának optimális kompozíciójaként kerül meghatározásra (azaz optimális részstruktúra) és a minimum (vagy maximum) érték kiválasztásra kerül, akkor DP formátumú megoldásról beszélünk. (Metamódszer, nem konkrét algoritmus.)

Dinamikus programozás: példa



Az $f(x_8)$ megoldás meghatározásának kompozíciója és számítása részfeladatok megoldásából.

LEGHOSSZABB KÖZÖS RÉZSOROZAT (LCS)

Leghosszabb közös részsorozat (LCS)

- Adott egy $X = \langle x_1, x_2, \dots, x_n \rangle$ sorozat; az X részsorozatát kapjuk, ha valahány elemet törölünk belőle.
- Cél: Adott két sorozat: $X = \langle x_1, x_2, \dots, x_n \rangle$ és $Y = \langle y_1, y_2, \dots, y_m \rangle$, keressük meg a leghosszabb sorozatot, ami részsorozata egyaránt X -nek és Y -nak.
- Példa: ha $X = \langle c, a, d, b, r, z \rangle$ és $Y = \langle a, s, b, z \rangle$, a leghosszabb közös részsorozata (*longest common subsequence*) X -nek és Y -nak $\langle a, b, z \rangle$.
- Feladat: ha $X = \langle A, B, C, B, D, A, B \rangle$ és $Y = \langle B, D, C, A, B, A \rangle$, mi X -nek és Y -nak a leghosszabb közös részsorozata?

Leghosszabb közös részsorozat (Longest-Common-Subsequence: LCS)

Stratégia:

1. Az LCS hosszát vizsgáljuk

2. Utána magát az LCS-t határozzuk meg

- Nem a közös részsorozatokat, hanem az X és Y „elejét” tekintjük
- Jelölje $F[i, j]$ az X első i elemének és Y első j elemének leghosszabb közös részsorozat hosszát. Az LCS célja, hogy megtaláljuk $F[n, m]$ értékét (és a sorozat elemeit).
- Ekkor igaz, hogy:

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

LCS rekurzív algoritmus váz

Függvény $LCS(X, Y, n, m)$

// **Input:** X és Y sorozat n , illetve m elemmel, valamint $i \leq n$ és $j \leq m$

// **Output:** $F[i, j]$ az X és Y i és j hosszú elejének LCS hossza

// Mivel az alapesetek 0 hosszú részsorozatok, itt is 0-tól indexeljük az $n + 1 \times m + 1$ tömböt

Ha $(i = 0)$ vagy $(j = 0)$ akkor

$F[i, j] \leftarrow 0$

Különben

Ha $x_i = y_j$ akkor

$F[i, j] \leftarrow LCS(X, Y, i - 1, j - 1) + 1$

Különben

$F[i, j] \leftarrow \max(LCS(X, Y, i - 1, j), LCS(X, Y, i, j - 1))$

Elágazás vége

Elágazás vége

return $F[i, j]$

Függvény vége

Futási idő: exponenciális

LCS dinamikus programozással

Eljárás $LCS(X, Y, n, m, F)$

// **Input:** X és Y sorozat n , illetve m elemmel, valamint $i \leq n$ és $j \leq m$

// **Output:** $F[n+1, m+1]$ az X és Y közös részsorozatainak hosszait tartalmazó tömb

// Mivel az alapesetek 0 hosszú részsorozatok, itt is 0-tól indexeljük az $n + 1 \times m + 1$ tömböt

Ciklus $j \leftarrow 0$ -tól m -ig

$F[0, j] \leftarrow 0$

Ciklus vége

Ciklus $i \leftarrow 1$ -től n -ig

$F[i, 0] \leftarrow 0$

Ciklus vége

Ciklus $i \leftarrow 1$ -től n -ig

Ciklus $j \leftarrow 1$ -től m -ig

Ha $x_i = y_j$ **akkor**

$F[i, j] \leftarrow F[i - 1, j - 1] + 1$

Különben

$F[i, j] \leftarrow \max(F[i - 1, j], F[i, j - 1])$

Elágazás vége

Ciklus vége

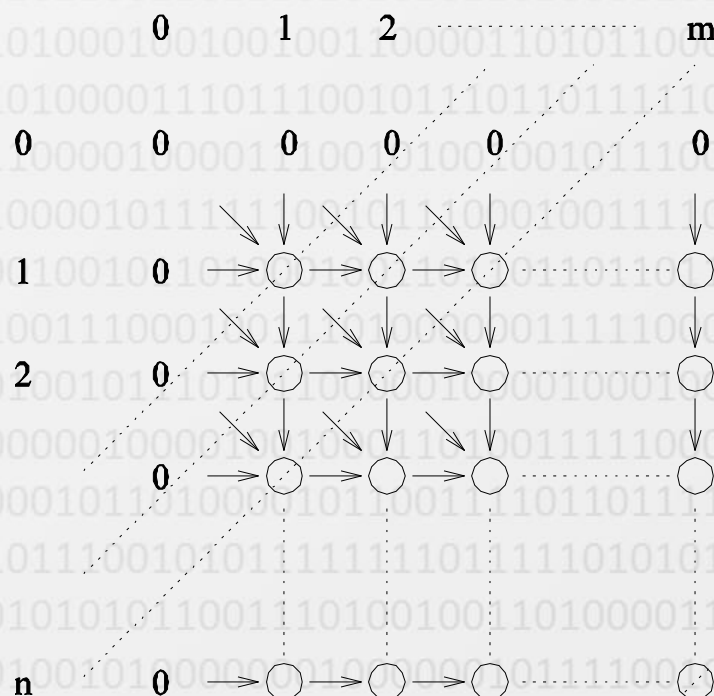
Ciklus vége

Eljárás vége

Futási idő: $O(n \times m)$

Leghosszabb közös részsorozat

- Az algoritmus kiszámolja a két-dimenziós F táblát (*memoization*) sor-oszlop sorrendben $\Rightarrow O(n*m)$, konstans idő.
- Az átlós csomópontok mindegyike két részproblémához kapcsolódik, az előző szinthez és az azt megelőző szinthez.



Az LCS táblázat számítási elemei. A számítás a jelzett átlós irányban halad.

Leghosszabb közös részsorozat: példa

- Legyen két aminosavnak a szekvenciája H E A G A W G H E E és P A W H E A E, ahol A: Alanine, E: Glutamic acid, G: Glycine, H: Histidine, P: Proline és W: Tryptophan.

		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	1	1	1	1	1	1	1	1
W	0	0	0	1	1	1	2	2	2	2	2
H	0	1	1	1	1	1	2	2	3	3	3
E	0	1	2	2	2	2	2	2	3	4	4
A	0	1	2	3	3	3	3	3	3	4	4
E	0	1	2	3	3	3	3	3	3	4	5

- Az LCS: A W H E E.
- Feladat: írjon algoritmust az LCS kiolvasására

Az általános dinamikus programozási technika

- Általában olyan feladatoknál alkalmazzuk, amelyek első ránézésre rengeteg időt vesznek igénybe.

Részei:

- **Egyszerű részproblémák:** a részproblémákat néhány változó függvényeként kell definiálni.
- **Részprobléma optimalitás:** a globális optimum a részproblémák optimumaként definiálható.
- **Átfedő, kapcsolódó részproblémák:** a részfeladatok nem függetlenek, hanem átfedőek (ezért bottom-up konstrukcióban kell feldolgozni).

Megoldás során:

- Rekurziós összefüggés felírása
- Az összefüggés elemzése. Tömbre átírás (a változók diszkrét, korlátozottak)
- A tömbben a kitöltési irány meghatározása (minden elem esetén definiált legyen az összes olyan elem értéke, amit használ)
- Végző megoldás helyének megadása
- Ha kell, optimális megoldás visszakeresése
- Ellenkező esetben memóriaigény csökkentése (ha lehet)

Felhasznált és javasolt irodalom

- [1] A. Grama, A. Gupta, G. Karypis, V. Kumar

Introduction to Parallel Computing

Addison-Wesley, ISBN 0-201-64865-2, 2003, 2nd ed., angol, 636 o.