

1. előadás (2025. szeptember 9.)

1.1. Stabil párosítás

Könyv: ../konyv.pdf#page=105

- Van n fiú (f_1, f_2, \dots, f_n)
- és n lány (l_1, l_2, \dots, l_n)
- Minden fiúnak van egy preferencialistája a lányokról: $f_i \rightarrow (l_{i_1}, l_{i_2}, \dots, l_{i_n})$
- és minden lánynak van egy preferencialistája a fiúkról: $l_j \rightarrow (f_{j_1}, f_{j_2}, \dots, f_{j_n})$.
- A preferencialisták "teljesek" és "holtversenymentesek".

Pl.

András (C, D) — Cili (B, A)

Béla (C, D) — Dalma (A, B)

Erre a párosításra Béla és Cili instabilitást jelent: jobban tetszenek egymásnak, mint amennyire a párosításbeli párjuk tetszik nekik.

Kérdés: van-e olyan (teljes) párosítás, amelyekre nézve nincs instabilitási tényező?

A példában igen:

András — Cili
Béla — Dalma

Algoritmus tervezése

Teljes párosítás, instabilitás nélkül \rightarrow stabil házasság.

Kérdés (újra): tetszőleges preferencialistához van-e stabil házasság? Ha van, akkor hogyan található ilyen (gyorsan)?

Naiv algoritmus

- Kiindulunk egy tetszőleges M_0 teljes párosításból.
- Ha található instabilitást jelentő f fiú és l lány erre a párosításra, akkor cseréljük a "négyesben" a párokat $\rightarrow M_1$ párosítás.
- Ezt addig ismételjük, amíg található instabilitás.

Pozitívum: megállás után stabil házasságunk van.

Negatívum: nem feltétlenül terminál az algoritmus.

Közgazdasági Nobel-díjas algoritmus

Gale-Shapley algoritmus (1962). Ez egy házassági rituálénak is felfogható, ami több napig tart.

- Minden reggel a lányok kiállnak az erkélyükre és várják hogy a fiúk jöjjenek szerenádolni.
- Minden fiú ahhoz a lánynak megy szerenádolni, aki a legjobban tetszik neki azok közül, akiktől még nem kapott kosarat.
- Ha valamelyik fiú már minden lánytól kosarat kapott, másnap otthon marad.
- Ha egy lány erkélye alatt legalább egy fiú szerenádol, akkor a lány a szerenádózói közül a neki legjobban tetszőnek azt mondja, hogy várja másnap is, a többinek meg azt, hogy ne jöjjenek többet mert semmiképp nem lesz a feleségük (kikosarazás).
- Este azok a fiúk, akiket kikosaraztak, kihúzzák a kikosarazó lányt a preferencialistájukról.

Megállási feltétel: minden lány erkélye alatt LEGFELJEBB egy fiú szerenádol.

Párosítás a terminálás után: lány \leftrightarrow ablaka alatt szerenádoló fiú.

Algoritmus elemzése

Hatékonyság

Legrosszabb esetben hány napig tart (egyáltalán befejeződik-e véges sok lépésben)?

Állítás: az algoritmus legfeljebb az $(n^2 + 1)$. napon befejeződik.

Bizonyítás: Egy olyan nap, amikor az algoritmus nem fejeződik be, van olyan lány, akinek az erkélye alatt legalább két fiú szerenádozik, következésképpen van olyan fiú, aki kosarat kap és kihúzza egy lányt a preferencialistájáról. Kezdetben a fiúk preferencialistájának összhossza n^2 . Ha a megállás előtt ezekről minden nap legalább egy lány kihúzásra kerül, akkor az $(n^2 + 1)$. napon már nincs kit kihúzni. Következésképpen ekkorra az algoritmus biztosan befejeződik.

Helyesség

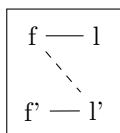
- teljes párosítás
- stabilitás

Állítás: az algoritmus teljes párosítást ad.

Bizonyítás: Indirekt tegyük fel, hogy a párosítás nem teljes. Ekkor van olyan fiú, akinek nincs párja. Ez úgy lehetséges, hogy mindenkinél szerenádozott és mindenkitől kosarat kapott. Ezen kívül kell lenni olyan lánynak akinek nincs párja. Ez úgy lehetséges, hogy nála soha senki nem szerenádozott → **ellentmondás**.

Állítás: az algoritmus stabil párosítást ad.

Bizonyítás: Indirekt tegyük fel, hogy létezik instabilitási tényező. Azaz f -nek jobban tetszik l' mint l és l' -nek jobban tetszik f mint f' .



Az f előbb szerenádozott l' -nél, mint l -nél (és l' -nél kosarat kapott). Az l' a szerenádozói közül viszont a neki legjobban tetszőnek lesz végül a párja, ellentmondva annak, hogy l' -nek jobban tetszik f , mint f' .

2. előadás (2025. szeptember 16.)

Könyv: ../konyv.pdf#page=108

Ismétlés

Gale-Shapley algoritmus: tetszőleges preferencialisták esetén generál (egy) stabil párosítást.

Megjegyzés: a stabil párosítások száma akár exponenciálisan nagy lehet (n -ben).

Megjegyzés: időnként egész "egzotikus" stabil párosítások is előfordulnak, pl. az összes fiú/lány a preferencialistájának első/utolsó lányát/fiúját kapja.

Optimalitás

Egy adott fiú és adott lány **szóba jönnek egymásnak**, ha van olyan stabil párosítás, amelyben ők egy párt alkotnak.

Egy adott fiú és adott lány **lelki társa** egymásnak, ha minden stabil párosításban ők egy párt alkotnak.

Állítás

- **A:** az algoritmus minden fiúhoz a számára szóba jövő lányok közül a neki legjobban tetszőt párosítja.
- **B:** algoritmus minden lányhoz a számára szóba jövő fiúk közül a neki legkevésbé tetszőt párosítja.

Bizonyítás

A)

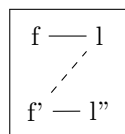
Indirekt tegyük fel, hogy van olyan fiú, akihez a Gale-Shapley algoritmus a számára szóba jövő lányok közül nem a neki legjobban tetszőt párosítja. Ekkor ez a fiú valamelyik nap szerenádozik a számára szóba jövő lányok közül a neki legjobban tetszőnél, akitől kosarat kap. Tekintsük azt a napot, amikor egy ilyen szomorú esemény először fordul elő. Legyen a kosarat kapó fiú f és a kosarat kapó lány l , és f' az a fiú, aki miatt f kosarat kapott.

Ekkor l -nek jobban tetszik f' , mint f . Mivel az első olyan napon vagyunk, amikor egy fiút kikosaraz a számára szóba jövő lányok közül a neki legjobban tetsző, f' -t nem kosarazhatta még ki a számára szóba jövő lányok közül a neki legjobban tetsző l^* . Egy l^* nem lehet l előtt f' preferencialistáján ($l^* = l$ lehetséges).

Ezek után tekintsünk egy olyan M stabil párosítást, amelyben f és l egy párt alkotnak. Ez különbözik a Gale-Shapley algoritmus által meghatározott párosítástól. Jelölje l'' az f' M -beli párját. Mivel l'' szóba jön f' számára, l'' nem lehet l^* előtt f' preferencialistáján ($l'' = l^*$ lehetséges).

Konklúzió: $f' \rightarrow (l, l^*, l'')$, azaz f' -nek jobban tetszik l , mint l''

Illusztráció (M -ben a párok):

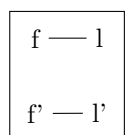


Az eredeti feltevésünk szerint f' és l instabilitás (M -re) \rightarrow **ellentmondás**.

B)

Indirekt tegyük fel, hogy van olyan lány, akihez a Gale-Shapley algoritmus a számára szóba jövő fiúk közül nem a neki legkevésbé tetszőt párosítja. Legyen l egy ilyen lány. Legyen M egy olyan stabil párosítás, amelyben l párja f , kevésbé tetszik l -nek, mint a Gale-Shapley algoritmus szolgáltatja f' párja. Legyen f' M -beli párja l' .

Illusztráció (M -ben a párok):



Most a feltételünk szerint l -nek jobban tetszik f' , mint f . Másrészt a Gale-Shapley algoritmus f' -höz a szóba jövő lányok közül a neki legjobban tetszőt párosítja, következésképpen f' -nek jobban tetszik l (Gale-Shapley-beli párja), mint l' (M -beli párja). Egy f' és l instabilitás (M -re) \rightarrow **ellentmondás**.

Egyértelműség

Futtassuk le a Gale-Shapley algoritmust úgy, hogy a fiúk szerenádognak $\rightarrow M_1$.

Ezután futtassuk le úgy, hogy a lányok szerenádognak $\rightarrow M_2$.

M_1 és M_2 stabil párosítások.

Ha $M_1 \neq M_2$, akkor van legalább két különböző stabil párosítás.

Ha $M_1 = M_2$, akkor ez az egyetlen stabil párosítás. Mivel ez egyben fiú-optimális és fiú-pesszimális, valamint lány-optimális és lány-pesszimális, mindenki számára pontosan egy ellenkező nemű jön szóba.

Stabil szobatárs probléma (unisex változat)

$2n$ fiú van. Mindenkinek van egy preferencialistája a többiekéről. 2 ágyas szobákba szeretnénk beosztani őket.

Instabilitás: $[f_i - f_j]$ és $[f_k - f_l]$

f_i szívesebben lenne f_k -val egy szobában, mint f_j -vel.

f_k szívesebben lenne f_i -vel egy szobában, mint f_l -lel.

Nem feltétlenül van stabil párosítás.

3. előadás (2025. szeptember 23.)

3.1. Stabil házassítás

Gale-Shapley algoritmus: arra optimalizált, hogy a fiúk a lehető legjobban járjanak. Most azt szeretnénk, hogy átlagosan mindenki a lehető legjobban járjon (pl. randiapp).

Állítás (pareto optimalitás): nincs olyan teljes párosítás (nem stabil sem), amelynél minden fiú jobban jár, mint a Gale-Shapley algoritmus által szolgáltatott párosításnál.

További optimalizálási változatok

Jelölés: $r_f(l) \rightarrow$ hányadik az l lány az f fiú preferencialistáján

Jelölés: $r_l(f) \rightarrow$ hányadik az f fiú az l lány preferencialistáján

1. legyen M stabil párosítás és tekintsük minden l lányra és f fiúra, ahol $(f, l) \in M$ az $r_f(l)$ és $r_l(f)$ értékeket.
2. Keressük azt az M stabil párosítást, amelyre:
 - (a) az $r_f(l)$ és $r_l(f)$ értékek közül a legnagyobb a lehető legkisebb (mindenki a lehető legjobban jár)
 - (b) az $r_f(l)$ és $r_l(f)$ összege a lehető legnagyobb (mindenki a lehető legjobban jár)
 - (c) az $r_f(l)$ értékek összegének és az $r_l(f)$ értékek összegének különbségének abszolútértéke a lehető legkisebb (kb. egyenlően jól jár mindkét nem)

Megjegyzés: (a)-(b): polinom időben megoldható, (c): NP-nehéz

3.2. Algoritmustervezési módszerek

1. Oszd meg és uralkodj (pl. összefésüléses rendezés)
2. Randomizálás (pl. gyorsrendezés)
3. Dinamikus programozás (pl. Floyd-Warshall algoritmus)
4. Mohó algoritmusok (pl. Kruskal, Prim, Dijkstra algoritmus)
5. Közelítő (approximációs) algoritmusok (pl. utazó ügynök)

3.3. Oszd meg és uralkodj algoritmusok

Könyv: `../konyv.pdf#page=8`

Séma:

1. A feladatot hasonló, csak méretű részfeladatokra bontjuk.
2. A részfeladatokat rekurzívan megoldjuk (ha a mérete elég kicsi, akkor közvetlenül oldjuk meg).
3. A részfeladatok megoldását összekombináljuk az eredeti feladat megoldásává

Algoritmus elemzése

- helyesség
- hatékonyság

Legegyszerűbb algoritmusok

- $a \geq 1$ részfeladat
- ha n az eredeti feladat mérete, akkor n/b az összes részfeladat mérete, ahol $b > 1$
- a harmadik fázis költsége $f(n) \geq 0$
- ha $f(n)$ jelöli az algoritmus költségét (lényeges lépések száma), akkor $T(n) = a \cdot T(n/b) + f(n)$
- pl. összefésüléses rendezésnél $T(n) = 2T(n/b) + n$

Összehasonlítás más algoritmusok hatékonyságával: zárt képlet. Ilyen egyszerű rekurziókra kézzel levezethető zárt képlet. Az egyszerűség kedvéért tfh. $n = 2^k$ kettő hatvány.

”Önmagába helyettesítés”:

$$\begin{aligned}
 T(n) &= n + \boxed{2T\left(\frac{n}{2}\right)} \\
 &= n + 2\left(\frac{n}{2} + 2T\left(\frac{\frac{n}{2}}{2}\right)\right) \\
 &= n + n + 2^2T\left(\frac{n}{2^2}\right) \\
 &= 2n + 2^2\boxed{T\left(\frac{n}{2^2}\right)} \\
 &= 2n + 2^2\left(\frac{n}{2^2} + 2T\left(\frac{\frac{n}{2^2}}{2}\right)\right) \\
 &= 2n + n + 2^3T\left(\frac{n}{2^3}\right) \\
 &= 3n + 2^3T\left(\frac{n}{2^3}\right) \\
 &\vdots \\
 &= in + 2^iT\left(\frac{n}{2^i}\right) \\
 &\vdots \\
 &= kn + 2^kT\left(\frac{n}{2^k}\right) \\
 &= kn + 2^kT\left(\frac{n}{1}\right) \\
 &= (1) = 0, \text{ hiszen az egyelemű tömbök rendezettek.}
 \end{aligned}$$

Így $T(n) = k = n$. De $n = 2^k \rightarrow k = \log_2 n$, ezért $T(n) = n \log_2 n$. Általánosítva (nem bizonyítjuk): mester tétel.

Mester-tétel

Könyv: ../konyv.pdf#page=10

Zárt formula $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ formájú rekurziókhoz (nem fed le mindent).

$[f(n)$ -t hasonlítjuk össze $n^{\log_b a}$ -val].

3 eset:

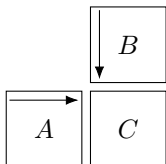
1. ha $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ valamilyen pozitív ϵ -nal, akkor $T(n) = \Theta(n^{\log_b a})$
2. ha $f(n) = \Theta(n^{\log_b a})$, akkor $T(n) = \Theta(n^{\log_b a} \log n)$
3. ha $f(n) = \Omega(n^{\log_b a + \epsilon})$ valamilyen pozitív ϵ -nal ÉS $af\left(\frac{n}{b}\right) \leq cf(n)$ valamilyen $c < 1$ és n elég nagy esetén, akkor $T(n) = \Theta(f(n))$

4. előadás (2025. szeptember 30.)

Gyors mátrixszorzás

Könyv: ../konyv.pdf#page=12

Feladat: A és B kompatibilis mátrixok AB szorzatának kiszámítása.



$$C = A \cdot B$$

$$c_{ij} = \sum_{k=1}^q a_{ik} \cdot b_{kj} \rightarrow 1 \text{ elem kiszámításához } q \text{ db elemi szorzás és } q - 1 \text{ db elemi összeadás kell.}$$

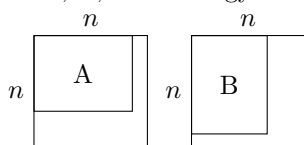
$$p \times r \text{ darab elem} \rightarrow pqr$$

Ha ezek mind $n \times n$ -es mátrixok:

- $M(n) = n^3 \in \Theta(n^3)$ (M az elemi szorzások száma)
- $S(n) = n^3 \in \Theta(n^3)$ (S az elemi összeadások száma)

Cél: ennél hatékonyabb algoritmus.

Ha A, B, C nem négyzetes, 0-ákkal feltöltjük:



Oszd meg és uralkodj ötlet:

$$A = \begin{matrix} \frac{n}{2} & \frac{n}{2} \\ \begin{matrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{matrix} \end{matrix} \quad B = \begin{matrix} \begin{matrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{matrix} \end{matrix} \quad C = \begin{matrix} \begin{matrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{matrix} \end{matrix}$$

Az A és B mátrixok helyett 4-4 db $\frac{n}{2} \times \frac{n}{2}$ -es mátrix.

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{12}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$\text{Elemi szorzások száma: } M(n) = \begin{cases} 1 & n = 1 \\ 8M\left(\frac{n}{2}\right) & n \geq 2 \end{cases}$$

$$\text{Elemi összeadások száma: } S(n) = \begin{cases} 0 & n = 1 \\ 8S\left(\frac{n}{2}\right) + 4\left(\frac{n^2}{2}\right) & n \geq 2 \end{cases}$$

Kérdés: hatékonyabb-e ez a módszer?

Csökkent-e az $M(n)$ és $S(n)$ érték? Használjuk a Mester tételt: $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$

- $M(n) = 8M\left(\frac{n}{2}\right)$
- $a = 8, b = 2, f(n) = 0$
- $n^{\log_2 8} = n^3$

M.T. 1. eset: ha $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ valamilyen $\epsilon > 0$ -ra (pl. $\epsilon = 0.5$), akkor $M(n) \in \Theta(n^3)$.

- $S(n) = 8M\left(\frac{n}{2}\right) + n^2$

- $a = 8, b = 2, f(n) = n^2 \in \mathcal{O}(n^{\log_2 8 - \mathcal{E}})$, pl. $\mathcal{E} = 0.5$
- $n^{\log_2 8} = n^3$

M.T. 1. eset $\rightarrow S(n) \in \Theta(n^3)$.

Sajnos az elemi műveletek száma nem javult, nem nyertünk semmit.

Volker Strassen észrevétele: 7 db $\frac{n}{2} \times \frac{n}{2}$ -es szorzással is megkaphatjuk az eredményt.

$$P_1 = A_{11} \cdot (B_{12} - B_{22})$$

$$P_2 = (A_{11} + A_{12}) \cdot B_{22}$$

$$P_3 = (A_{21} + B_{22}) \cdot B_{11}$$

$$P_4 = A_{22} \cdot (B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$P_6 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$P_7 = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})$$

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

Összesen:

- 7 db szorzás (7 db rekurzív hívás $\frac{n}{2} \times \frac{n}{2}$ méretű mátrixokra)
- 18 db összeadás

$$M(n) = \begin{cases} 1 & n = 1 \\ 7M\left(\frac{n}{2}\right) & n \geq 2 \end{cases} \text{ és } S(n) = \begin{cases} 0 & n = 1 \\ 7S\left(\frac{n}{2}\right) + 18\left(\frac{n^2}{2}\right) & n \geq 2 \end{cases}$$

M.T. ezekre a rekurzív függvényekre:

- $M(n) = 7M\left(\frac{n}{2}\right)$
- $a = 7, b = 2, f(n) = 0$

M.T. 1. eset: mivel $f(n) \in \mathcal{O}(n^{\log_2 7 - \mathcal{E}})$ valamilyen $\mathcal{E} > 0$ -ra (pl. $\mathcal{E} = 0.5$), ezért $M(n) \in \Theta(n^{\log_2 7})$.

- $S(n) = 7S\left(\frac{n}{2}\right) + \frac{9}{2}n^2$
- $a = 7, b = 2, f(n) = 4.5n^2$
- $n^{\log_2 7} = n^{2.81}$

M.T. 1. eset: mivel $f(n) \in \mathcal{O}(n^{\log_2 7 - \mathcal{E}})$ pl. $\mathcal{E} = 0.1$, ezért $S(n) \in \Theta(n^{\log_2 7})$.

$$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11}(B_{12} - B_{22}) + (A_{11} + A_{12})B_{22} \\ &= A_{11}B_{12} - A_{11}B_{22} + A_{11}B_{22} + A_{12}B_{22} \end{aligned}$$

C_{21} hasonlóan

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ &= (A_{11} - A_{22})(B_{11} + B_{22}) + A_{21}(B_{21} - B_{11}) - (A_{11} + A_{12})B_{22} + (A_{12} - A_{22})(B_{21} + B_{22}) \\ &= \dots \end{aligned}$$

HF. kérdés: adott $A, B, C \in \mathcal{R}^{n \times n}$ mátrixok esetén a számolások elvégzése nélkül eldönthető-e, hogy $AB = C$?

4.1. Randomizált algoritmusok

- Las Vegas típusú: sosem téved, de néha lassú, pl. quicksort
- Monte Carlo típusú: gyors, de néha téved, pl. Freivalds algoritmus

Freivalds algoritmus

- Véletlenszerűen választunk egy $\alpha \in \{0, 1\}^n$ bitvektort és kiszámoljuk ($\Theta(n^2)$ költséggel) a $\beta = (AB) \cdot \alpha = A \cdot (B\alpha)$ és $\gamma = C \cdot \alpha$ vektorokat.
- Ha $\beta \neq \gamma$, akkor visszatérünk HAMIS válasszal (itt biztosan nem tévedtünk).
- Ha $\beta = \gamma$, akkor visszatérünk IGAZ válasszal (lehet, hogy tévedtünk).

Észrevétel: az utóbbi eset az n dimenziós α bitvektorok legfeljebb felére teljesülhet. Ha α -t véletlenszerűen választjuk, akkor a tévedés valószínűsége $\leq \frac{1}{2}$. Az algoritmus: végezzük el ezt 100-szor egymás után \rightarrow a tévedés valószínűsége $\leq (\frac{1}{2})^{100}$.

Észrevétel bizonyítása:

Ha $AB \neq C$ és valamilyen α -ra $AB\alpha = C\alpha$, akkor

- $AB - C$ mátrix nem az azonosan 0 mátrix.
- tfl. az i . sor j . eleme $d > 0$.
- legyen α' az α vektor, ha α -ban a j . elemet átváltjuk (bit flip).

Ekkor $(AB - C)\alpha'$ i . koordinátája megváltozott ($-d$ -re, $+d$ -re), tehát $(AB - C)\alpha' \neq 0$. Minden α vektorhoz találhatunk α' vektort, ami kimutatja az $AB \neq C$ tényt, és különböző α vektorokhoz különböző α' -t kapunk \rightarrow a hamis eredményt adó vektorok száma \geq helyes eredményt adó vektorok száma.

5. előadás (2025. október 7.)

Polinomok szorzása

Könyv: ../konyv.pdf#page=16

$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$ és $B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{m-1}x^{m-1}$ $(n-1)$ -ed és $(m-1)$ -edfokú polinomok. Határozzuk meg a szorzatpolinomot: $C(x) = A(x) \cdot B(x)$, ahol $C(x) = c_0 + c_1x + c_2x^2 + \dots + c_{n+m-2}x^{n+m-2}$.

$$c_0 = a_0b_0$$

$$c_1 = a_0b_1 + a_1b_0$$

$$c_2 = a_0b_2 + a_1b_1 + a_2b_0$$

\vdots

$$c_i = a_0b_i + a_1b_{i-1} + a_2b_{i-2} + \dots + a_{i-1}b_1 + a_ib_0$$

$$\begin{bmatrix} a_0b_0 & a_0b_1 & a_0b_2 & a_0b_3 \\ a_1b_0 & a_1b_1 & a_1b_2 & \\ a_2b_0 & a_2b_1 & & \\ a_3b_0 & & & \end{bmatrix}$$

Szorzatpolinom meghatározása: $\rightarrow c_0, c_1, c_2, \dots, c_{n+m-2}$ értékei. Naiv módszer költsége: $\mathcal{O}(nm)$.

Van-e ennél hatékonyabb? Van! Oszd meg és uralkodj algoritmust fejlesztünk (a szokásos egyszerűsítő feltételezésekkel élünk, $n = m$ kettő hatvány). Naiv algoritmus: elemi algebra, szofisztikált algoritmus: analízis.

Interpoláció

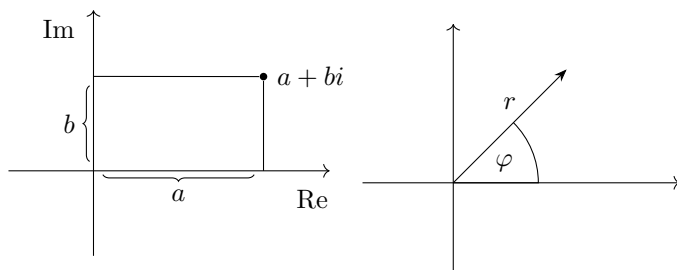
Egy $(n-1)$ -edfokú valós (komplex) polinomot meghatároz n helyettesítési értéke \rightarrow interpolációs formulák (Newton, Lagrange).

Ötlet: határozzuk meg az $A(x)$ és $B(x)$ polinomok helyettesítési értékeit bizonyos z_1, z_2, \dots pontokban. Eből meglesznek $C(x)$ helyettesítési értékei a z_1, z_2, \dots pontokban. Ezekből határozzuk meg $C(x)$ együtthatóit.

Boszorkányos ötlet: ha $A(x)$ és $B(x)$ $(n-1)$ -edfokú polinomok, akkor z_1, z_2, \dots legyenek a $2n$ -edik komplex egységgyökök.

Komplex számok

$a + bi$ alak, ahol $i = \sqrt{-1}$.

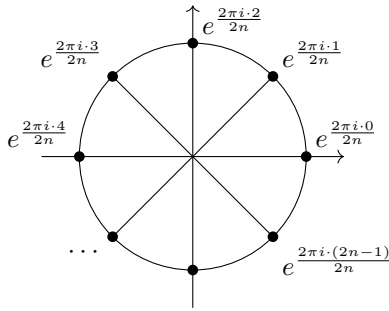


Trigonometrikus alak: $r(\cos \varphi + i \sin \varphi)$

Exponenciális alak: $re^{i\varphi}$

Algebra alaptétele: minden legalább elsőfokú komplex együtthatós polinomnak van komplex gyöke.

2n. komplex egységgyökök: $x^{2n} - 1$ gyökei.



Jelölés: $\omega_{j,2n} = e^{\frac{2\pi i j}{2n}}$

Algoritmus

1. $A(x)$ és $B(x)$ helyettesítési értékei a $2n$. komplex egységgyökökön oszd meg és uralkodj algoritmussal lássuk $A(x)$ -nél a számítást. Trükk: $A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$.

Elkészítünk két feleakkora fokszámú polinomot:

- $A_{\text{páros}} = a_0 + a_2x + a_4x^2 + \dots + a_{2n-2}x^{\frac{n-2}{2}}$
- $A_{\text{páratlan}} = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{\frac{n-2}{2}}$

Ekkor $A(x) = A_{\text{páros}}(x^2) + A_{\text{páratlan}}(x^2)$.

Észrevétel: egy $2n$. komplex egységgyök négyzete egy n . komplex egységgyök: $\left(e^{\frac{2\pi i j}{2n}}\right)^2 = e^{\frac{2\pi i j}{n}}$.

Így $A(x)$ helyettesítési értékének kiszámítását a $2n$. komplex egységgyökökön visszavezettük két feleakkora polinom helyettesítési értékének kiszámítására az n . komplex egységgyökökön. (Persze itt van még némi munka, de az $\mathcal{O}(n)$ költségű.)

Összköltség: $T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n) \rightarrow T(n) = \mathcal{O}(n \log n)$.

2. $C(x) = A(x) \cdot B(x)$ helyettesítési értékeinek kiszámítása a $2n$. komplex egységgyökökön $\rightarrow \mathcal{O}(n)$.

3. $C(x)$ helyettesítési értékeiből a $2n$. komplex egységgyökökön kiszámítjuk $C(x)$ együtthatóit (gyors Fourier-transzformált).

Általánosabban: legyen $C(x)$ legfeljebb $(2n-1)$ -edfokú polinom, amelynek ismertek a helyettesítési értékei a $2n$. komplex egységgyökökön.

Vezessük be a következő $D(x)$ polinomot: $D(x) = d_0 + d_1x + d_2x^2 + \dots + d_{2n-1}x^{2n-1}$ ahol $d_S = C(\omega_{S,2n})$ ($S = 0, 1, \dots, 2n-1$).

Ekkor $D(\omega_{t,2n}) = 2nc_{2n-t}$ (ezt még bizonyítani kell)

Alkalmazva az algoritmus oszd meg és uralkodj algoritmusát, a $D(x)$ polinomra a $D(\omega_{t,2n})$ helyettesítési értékek, így $C(x)$ együtthatói $\mathcal{O}(n \log n)$ lépésben kiszámíthatók $\rightarrow \mathcal{O}(n \log n)$.

6. előadás (2025. október 14.)

6.1. Gyors Fourier transzformált

Legyen $C(x) = c_0 + c_1x + \dots + c_{2n-1}x^{2n-1}$. Tfh. ismerjük $C(x)$ helyettesítési értékeit az $\omega_{1,2n}, \omega_{2,2n}, \dots, \omega_{2n,2n}$ $2n$ -edik komplex egységgyökökön.

Kérdés: $c_0, c_1, \dots, c_{2n-1}$.

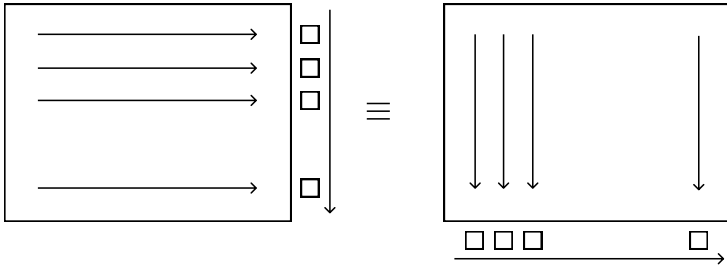
Bevezetünk egy $D(x) = d_0 + d_1x + \dots + d_{2n-1}x^{2n-1}$ polinomot, ahol $d_S = C(\omega_{S,2n})$ ($S = 0, 1, \dots, 2n-1$) (semmi pánik, $\omega_{0,2n} = \omega_{2n,2n}$).

Állítás: $D(\omega_{j,2n}) = 2nc_{2n-j}$ vagyis $c_{2n-j} = \frac{1}{2n}D(\omega_{j,2n})$ ($j = 1, 2, \dots, 2n$).

Bizonyítás:

$$\begin{aligned} D(\omega_{j,2n}) &= \sum_{S=0}^{2n-1} d_S \cdot \omega_{j,2n}^S \\ &= \sum_{S=0}^{2n-1} C(\omega_{S,2n}) \cdot \omega_{j,2n}^S \\ &= \sum_{S=0}^{2n-1} \left(\sum_{t=0}^{2n-1} c_t \cdot \omega_{S,2n}^t \right) \omega_{j,2n}^S \end{aligned}$$

Kettős összegzés



”a sorrend felcserélhető”

Kitérő (C. F. Gauss):

$$1 + 2 + \dots + 49 + 50 = \frac{50 \cdot 51}{2}$$

$$\underbrace{\frac{50}{51} + \frac{49}{51} + \dots + \frac{2}{51} + \frac{1}{51}}_{50 \cdot 51}$$

Visszatérve az eredeti gondolatmenethez:

$$\sum_{S=0}^{2n-1} \left(\sum_{t=0}^{2n-1} c_t + \omega_{S,2n}^t \right) \omega_{j,2n}^S = \sum_{t=0}^{2n-1} c_t \left[\sum_{S=0}^{2n-1} \omega_{S,2n}^t \cdot \omega_{j,2n}^S \right] \text{ ez könnyen számolható:}$$

$$\begin{aligned} \omega_{S,2n}^t \cdot \omega_{j,2n}^S &= (e^{2\pi i \cdot \frac{S}{2n}})^t (e^{2\pi i \cdot \frac{j}{2n}})^S \\ &= e^{2\pi i \cdot \frac{St}{2n}} \cdot e^{2\pi i \cdot \frac{Sj}{2n}} \\ &= e^{2\pi i \cdot \frac{St}{2n} + 2\pi i \cdot \frac{Sj}{2n}} \\ &= e^{\frac{(2\pi i \cdot St + 2\pi i \cdot Sj)}{2n}} \\ &= e^{\frac{S \cdot 2\pi i (t+j)}{2n}} \\ &= \left(e^{\frac{2\pi i (t+j)}{2n}} \right)^S \\ &= \omega_{t+j,2n}^S \end{aligned}$$

Az egész így:

$$\sum_{t=0}^{2n-1} c_t \left[\sum_{S=0}^{2n-1} \omega_{t+j,2n}^S \right]$$

Egy pillanatra bevezetve az $\omega = \omega_{t+j,2n}$ jelölést $\sum_{S=0}^{2n-1} \omega^S = 1 + \omega + \omega^2 + \dots + \omega^{2n-1}$.

Vegyük észre (bizonyítsuk teljes indukcióval): $(\omega - 1)(1 + \omega + \omega^2 + \dots + \omega^{2n-1}) = \omega^{2n} - 1$.

Mivel $\omega = \omega_{t+j,2n}$, egy $2n$ -edik komplex egységgyök, ezért $\omega^{2n} - 1 = 0$.

Ebből következik, hogy ha $\omega \neq 1$, akkor $1 + \omega + \omega^2 + \dots + \omega^{2n-1} = 0$.

Ha $\omega = 1$, akkor $1 + \omega + \omega^2 + \dots + \omega^{2n-1} = 1 + 1 + \dots + 1 = 2n$.

Mikor lesz $\omega_{t+j,2n} = 1$? Ha $t = 2n - j$!

Valóban, $t + j$ többszöröse kell, hogy legyen $2n$ -nek, azonban $1 \leq j \leq 2n$ és $0 \leq t \leq 2n - 1$ miatt ez csak $t + j = 2n$ esetén áll fenn.

$$\text{Így } \sum_{t=0}^{2n-1} c_t \underbrace{\sum_{S=0}^{2n-1} \omega_{t+j,2n}^S}_{\text{csak } t = 2n - j \text{ esetén } \neq 0} = c_{2n-j} \cdot 2n.$$

Nagy egészek szorzása

A, B n db számjegyből áll (tíz-es számrendszerben felírva).

Kérdés: $A \cdot B = ?$

Általános iskola

234 · 425

936

468

1170

99450

$O(n^2)$ "elemi" szorzás

Oszd meg és uralkodj trükk (magunk is kitalálhatjuk):

$$A = A_1 \cdot 10^{\frac{n}{2}} + A_0$$

$$B = B_1 \cdot 10^{\frac{n}{2}} + B_0$$

A_1, A_0, B_1, B_0 $\frac{n}{2}$ db számjegyből áll.

$$AB = (A_1 \cdot 10^{\frac{n}{2}} + A_0)(B_1 \cdot 10^{\frac{n}{2}} + B_0) = A_1 B_1 \cdot 10^n + (A_1 B_0 + A_0 B_1) \cdot 10^{\frac{n}{2}} + A_0 B_0$$

Nyertünk valamit azzal, hogy az eredeti feladatot négy feleakkora méretű feladatra redukáltuk?

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) \rightarrow T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2) : ($$

Azért ilyen olcsón nem várhatunk sokat!

Viszont $A_0 B_0, A_0 B_1, A_1 B_0, A_1 B_1$ helyett igazából $A_0 B_0, A_0 B_1 + A_1 B_0, A_1 B_1$ számolandó.

Piszkos trükk: $(A_1 + A_0)(B_1 + B_0)$ egy szorzás.

$$\text{Eredmény: } A_1 B_1 + \boxed{A_0 B_1 + A_1 B_0} + A_0 B_0.$$

$$\text{Így } A_0 B_1 + A_1 B_0 = (A_1 + A_0)(B_1 + B_0) - A_1 B_1 - A_0 B_0$$

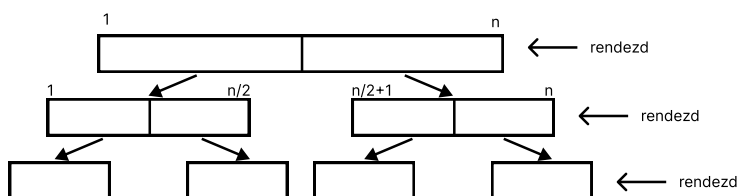
Így a feladat visszavezethető csupán 3 feleakkora méretű feladatra.

$$T(n) = 3T\left(\frac{n}{2}\right) \rightarrow T(n) = \Theta(n^{\log_2 3}) :)$$

6.2. Dinamikus programozás

Alapelv: itt is ugyanaz, mind az oszd meg és uralkodj algoritmusoknál. A feladatot (rekurzívan) visszavezetjük hasonló, csak kisebb méretű feladatok megoldására.

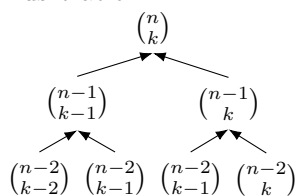
Egy tipikus oszd meg és uralkodj algoritmusnál (összefésüléses rendezés):



Független részproblémák

Nem mindig ez a helyzet!

Illusztráció:



”Átfedő részproblémák”

Összefoglalva

- **Oszd meg és uralkodj** tipikusan fentről lefelé.
- **Dinamikus programozás** tipikusan lentől felfelé és a részproblémák megoldását egy táblázatban jegyezzük fel (létezik fentről lefelé is \rightarrow memoizálás).

Mi mindenre jó?

Elhelyezünk egy sorban különböző címletű érméket, és két játékos felváltva elvehet egy érmét a sor valamelyik végéről.



Milyen stratégiát kövessen Aliz az elvett érmék összértékének maximalizálására, feltéve, hogy Robi is ugyanezt akarja maga számára?

7. előadás (2025. október 21.)

7.1. Dinamikus programozás

Adottak az A_1, A_2, \dots, A_n mátrixok, ahol A_1 egy $q_0 \times q_1$ dimenziós, az A_2 egy $q_1 \times q_2$ dimenziós, \dots , A_n pedig egy $q_{n-1} \times q_n$ dimenziós mátrix. Ekkor képezhető az $A_1 A_2 \dots A_n$ szorzat. A szorzatmátrix a mátrixszorzás asszociativitása miatt nem függ a zárójelezéstől. Viszont az, hogy a szorzat kiszámítása hány elemi műveletet igényel, már általában függ a zárójelezéstől. Az egyszerűség kedvéért tegyük fel, hogy a mátrixok szorzását a hagyományos módon végezzük: $A_{p \times q} \times A_{q \times r} = A_{p \times r}$.

Az elemi lépések most legyenek csupán az elemi szorzások. Ekkor $p \times q \times r$ az elemi szorzások száma a szorzatmátrix kiszámításánál.

Példa

három mátrix:

$$A_1 \rightarrow 50 \times 5$$

$$A_2 \rightarrow 5 \times 20$$

$$A_3 \rightarrow 20 \times 200$$

Kétféleképpen számolhatunk:

- $(A_1 A_2) A_3$
- $A_1 (A_2 A_3)$

Általában: határozzuk meg azt a zárójelezést, amelynek mentén a szorzat számítása a lehető legkevesebb elemi szorzással jár.

A szorzat kiszámítása nem része a feladatnak.

Az algoritmus költségét n függvényében keressük, a dimenziók nem számítanak.

Első gondolat: nézzük meg az összes zárójelezést és válasszuk ki a legkedvezőbbet.

Rossz hír: a mátrixot $\Omega\left(\frac{4^n}{n^{\frac{3}{2}}}\right)$ módon lehet zárójelezni.

$$\Omega\left(\frac{4^n}{n^{\frac{3}{2}}}\right) \rightarrow \text{Catalan számok.}$$

[Számos példa van. Egy mozi pénztáránál $2n$ ember áll sorba, n embernél egy ezres, a többi n embernél egy kétezres van. A mozijegy ezer forint. A pénztárban nyitáskor üres a kassza. Hányféleképpen állhatnak sorba az emberek, hogy mindenkinek lehessen visszaadni.]

Polinomiális algoritmus

Tfh. az $A_1 A_2 \dots A_n$ szorzatot akarjuk kiszámítani. Zárójelezés \Leftrightarrow szorzások sorrendje.

Pl. $(A_1 A_2)(A_3(A_4 A_5))$

1. Első szorzás $A_1 A_2$ vagy $A_4 A_5$.
2. Második szorzás $(A_3(A_4 A_5))$ vagy $(A_1 A_2)$.
3. Harmadik szorzás $A_3(A_4 A_5)$
4. Negyedik (utolsó) szorzás $(A_1 A_2)(A_3(A_4 A_5))$

A zárójelezés nem feltétlenül határozza meg egyértelműen a szorzások sorrendjét, de egyértelműen meghatározza az elemi szorzások számát.

Dinamikus programozás algoritmus tervezése

1. Részproblémák meghatározása

$R[i, j] \rightarrow$ az $A_i A_{i+1} \dots A_j$ szorzat optimális zárójelezésének meghatározása

2. Keressünk kapcsolatot a részproblémák optimális megoldása és a részproblémák optimális megoldása között.

Tegyük fel, hogy megvan $A_i A_{i+1} \dots A_j$ optimális zárójelezése: $\underbrace{(A_1 A_{i+1} \dots A_k)}_{\text{ezek is zárójelezve vannak}} \underbrace{(A_{k+1} \dots A_{j-1} A_j)}_{\text{ezek is}}$

Állítás

Az optimális zárójelezés A_i -től A_k -ig terjedő része, illetve a A_{k+1} -től A_j -ig terjedő része optimális megoldása $R[i, k]$ -nak és $R[k + 1, j]$ -nek.

Bizonyítás

Ha lenne pl. $A_1 A_{i+1} \cdots A_k$ -nak a fenténél kevesebb elemi szorzást használó kiszámítása, erre cserélve $A_{k+1} \cdots A_{j-1} A_j$ optimális zárójelezésében az $A_i \cdots A_k$ részt, egy az "optimálisnál" kevesebb elemi szorzást használó zárójelezéshez jutnánk \rightarrow **ellentmondás**.

3. Aprópénzre váltjuk (2.)-t: rekurzió

Jelölje $l[i, j]$ az $R[i, j]$ optimális megoldásában az elemi szorzások számát.

Alapeset:

$$l[i, i] = 0$$

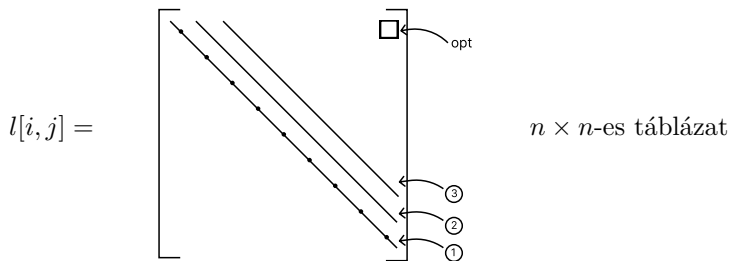
$$l[i, i + 1] = q_{i-1} q_i q_{i+1}$$

$$\text{Általában: } l[i, j] = l[i, k] + l[k + 1, j] + q_{i-1} q_k q_j$$

"Ne akadjunk fenn" \rightarrow honnan tudjuk k -t? Nem tudjuk, de k biztosan $i, i + 1, \dots, j - 1$ közül valamelyik. Piszkos trükk: vizsgáljuk meg az összes lehetőséget, és válasszuk k -nak a legkisebb értéket adót.

$$l[i, j] = \min_{i \leq k \leq j-1} \{l[i, k] + l[k + 1, j] + q_{i-1} q_k q_j\}$$

4. Számolás (3.) mentén



Költség: $\mathcal{O}(n^2)$ cella, cellánként $\mathcal{O}(n)$ számítás $\rightarrow \mathcal{O}(n^3)$

5. Optimális zárójelezés

Minden $l[i, j]$ -nél megjegyezzük a minimumot adó k -t \rightarrow backtracking a jobb felső sarokból.

8. előadás (2025. november 4.)

Ha az $l[i, j]$ értékek mellett feljegyezzük az optimumot adó k értékeket is egy $k[i, j]$ táblázatban, akkor egy $k[1, n]$ fogja megmutatni az utolsó szorzást, pl. 9 mátrix esetén ha $k[1, 9] = 6$, akkor az optimális zárójelezés úgy néz ki, hogy

$$\underbrace{(A_1 A_2 \cdots A_6)}_{\text{ezek is zárójelezve vannak}} \quad \underbrace{(A_7 A_8 A_9)}_{\text{ezek is}}$$

Hogyan? Általánosan:

$$\underbrace{(A_1 \cdots A_{k[1, n]})}_{\text{itt az első szorzás helyét } (k[1, k[1, n]])} \quad \underbrace{(A_7 A_8 A_9)}_{\text{itt pedig } k[k[1, n] + 1, n] \text{ helyét mutatja}}$$

Az előző példát ha nézzük és azt mondjuk $k[1, 6] = 3$ és $k[1, 9] = 8$, akkor a zárójelezés egy szintet lejjebb lépve $((A_1 A_2 A_3)(A_4 A_5 A_6))((A_7 A_8) A_9)$ és így tovább.

8.1. Sztringológia

String-ekkel, string-ek hasonlóságával kapcsolatos kérdésekre keresünk válaszokat (gyorsan).

Az egyik legegyszerűbb ilyen kérdés

Egy adott T karaktersorozat tartalmaz-e egy adott P karaktersorozatot?

Híres algoritmus: (Knuth-Morris-Pratt) $\rightarrow \Theta(|T| + |P|)$ ($|T|$ és $|P|$ a string-ek hossza) Fő ötlet P előfeldolgozása (Dinamikus Programozás).

Egy másik kérdés

Adott egy X és egy Y karaktersorozat. Keressük a leghosszabb közös részsorozatokat.

$$X = (x_1, x_2, \dots, x_m)$$

$$Y = (y_1, y_2, \dots, y_n)$$

Ezeknek $Z = (z_1, z_2, \dots, z_k)$ egy közös részsorozata lesz, ha léteznek olyan

$$1 \leq i_1 < i_2 < \dots < i_k \leq m$$

$$1 \leq j_1 < j_2 < \dots < j_k \leq n$$

részsorozatokat, hogy

$$x_{i_1} = y_{j_1} = z_1$$

$$x_{i_2} = y_{j_2} = z_2$$

$$\dots$$

$$x_{i_k} = y_{j_k} = z_k$$

Kérdés a legnagyobb ilyen k . X és Y hasonló, ha ez a k nagy, és különböző, ha kicsi.

Például:

$$X = \boxed{A} \boxed{C} \boxed{G} \boxed{C} \boxed{T} \boxed{A} \boxed{G} \boxed{C}$$

$$Y = \boxed{A} \boxed{T} \boxed{G} \boxed{C} \boxed{A} \boxed{A} \boxed{T} \boxed{C}$$

$$Z = \text{AGCAC}$$

Egy harmadik

Megint kiindulunk egy X és egy Y karaktorsorozatból, de a hasonlóság definíciója kicsit más lesz.

Legyen $X = (x_1, x_2, \dots, x_m)$ és $Y = (y_1, y_2, \dots, y_n)$. Az X és Y között bevezetjük a szekvenciaáttekintés fogalmát: ez egy párosítás az X -beli pozíciók és az Y -beli pozíciók között: $M \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, n\}$, úgy, hogy minden X -beli és minden Y -beli legfeljebb egy M -beli párban fordul elő (kimaradhat pár pozíció) ÉS ha $(i, j), (i', j') \in M$ és $i \leq i'$, akkor $j \leq j'$.

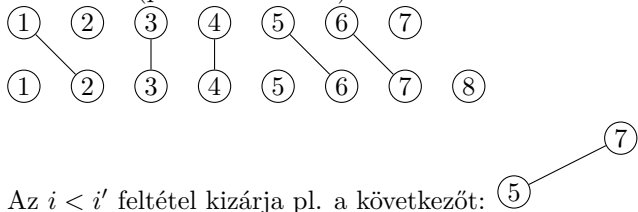
Például:

$X =$ A C G A C G T
 1 2 3 4 5 6 7

$Y =$ C A G A T G G A
 1 2 3 4 5 6 7 8

Az M szekvenciaillesztés itt lehet pl. $M = \{(1, 2), (3, 3), (4, 4), (5, 6), (6, 7)\}$.

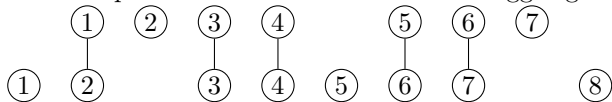
Vizuálisan (pozíciókat nézve):



Az $i < i'$ feltétel kizárja pl. a következőt:

A szekvenciaillesztés megjeleníthető az X karaktorsorozatból az Y karaktorsorozat előállításaként karakterek törlése, beszúrása, cseréje (változatlanul hagyása) egymásutánjaként.

Az előző példában minden ferde vonalat függőlegessé teszünk:



Nézzük a betűket:

-	A	C	G	A	-	C	G	T	-
C	A	-	G	A	T	G	G	-	A

beszúrás törlés csere

Az X transzformációja Y -ná ennek megfelelően:

1. Szúrjunk be egy C-t
2. Másoljuk A-t
3. Töröljük C-t
4. Másoljuk G-t
5. Másoljuk A-t
6. Szúrjunk be egy T-t
7. Cseréljük C-t G-re
8. ...

Persze nem csak ez az egy lehetőség van. Melyiket használjuk a hasonlóság mérésére?

Egy M szekvenciaillesztés minőségét a következőképpen definiáljuk:

1. az M -ben nem szereplő pozíciókra felszámolunk egy fix $y > 0$ értéket (ez része az inputnak)
2. ha $(i, j) \in M$, akkor erre felszámolunk egy $c["p", "q"] \geq 0$ értéket, ahol $x_i = "p"$ és $y_j = "q"$ (ez is része az inputnak)

Ezeket összeadjuk, és keressük a legjobb minőségű szekvenciaillesztést, vagyis egy olyat, ahol ez az összeg minimális.

Az összes szekvenciaillesztés végigbogarászása exponenciálisan hosszú idő: $DP \rightarrow \Theta(mn)$. Erős érvek szólnak amellett, hogy ennél polinomiálisan hatékonyabb algoritmus nem létezik.

[megjegyzés: $\theta(\frac{mn}{\log m+n})$ aszimptotikusan hatékonyabb, de polinomiálisan nem]

DP algoritmus

Miféle részproblémák jöhetnek elő itt? Az eddigi szerény tapasztalataink alapján: optimális szekvenciaillesztés $(x_i, x_{i+1}, \dots, x_j)$, $(y_k, y_{k+1}, \dots, y_l)$ között (infixek).

Kiderül, hogy prefixek is elegek lesznek. $R[i, j]$ részprobléma: $(X_i) = (x_1, x_2, \dots, x_i)$ és $(Y_j) = (y_1, y_2, \dots, y_j)$ optimális szekvenciaillesztés meghatározása.

9. előadás (2025. november 11.)

9.1. Optimális szekvenciaillesztés

(Levenshtein távolság)

Pozíciók:

$$X = (x_1, x_2, \dots, x_m) \rightsquigarrow \{1, 2, \dots, m\}$$

$$Y = (y_1, y_2, \dots, y_n) \rightsquigarrow \{1, 2, \dots, n\}$$

Egy olyan M párosítást (pontosabban szekvenciaillesztést) keresünk $\{1, 2, \dots, m\}$ és $\{1, 2, \dots, n\}$ között, melyre a következő mennyiség minimális:

- egyrészt annyszor egy $g > 0$ valós számot, ahány pozíció nem szerepel M -ben
- másrészt ha $(i, j) \in M$, akkor egy $c[x_i, y_j] \geq 0$ valós számot

DP algoritmus

1. Részproblémák

$R[i, j] \rightarrow$ optimális szekvenciaillesztés meghatározása az $X_i = (x_1, x_2, \dots, x_i)$ és $Y_j = (y_1, y_2, \dots, y_j)$ prefixek között.

2. Optimális részstruktúra tulajdonság

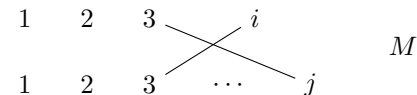
Észrevétel

Legyen M szekvenciaillesztés (akármilyen) X_i és Y_j között. Ekkor három eset lehet csak:

1. $(i, j) \in M$
2. i nem szerepel M -ben első tagként
3. j nem szerepel M -ben második tagként

Ami hiányzik: j szerepel második tagként, i nem szerepel első tagként. $(i, j) \notin M$.

Am ekkor



ami nem szekvenciaillesztés.

Állítás

Legyen M optimális szekvenciaillesztés X_i és Y_j között.

Ekkor

1. Ha $(i, j) \in M$, akkor $M \setminus \{(i, j)\}$ optimális szekvenciaillesztés X_{i-1} és Y_{j-1} között.
2. Ha i nem szerepel M -ben első tagként, akkor M optimális szekvenciaillesztés X_{i-1} és Y_j között.
3. Ha j nem szerepel M -ben második tagként, akkor M optimális szekvenciaillesztés X_i és Y_{j-1} között.

Bizonyítás

1. Ha X_{i-1} és Y_{j-1} között lenne egy $M \setminus \{(i, j)\}$ -nél kedvezőbb M^* szekvenciaillesztés, akkor $M^* \cup \{(i, j)\}$ egy M -nél kedvezőbb szekvenciaillesztés lenne X_i és Y_j között, **ellentmondás**.
2. Ha lenne X_{i-1} és Y_j között egy M -nél kedvezőbb M^* szekvenciaillesztés, akkor itt lenne X_i és Y_j között is.
3. Analóg (2.)-hoz.

3. Rekurzió

Jelölje $l[i, j]$ az $R[i, j]$ megoldásának "értékét".

"Alapeset"

Az egyik (vagy mindkettő) karaktersorozat üres

$$l[i, 0] = i - g$$

$$l[0, j] = j + g$$

($M = \emptyset$ az egyetlen párosítás)

”Általában”

$$(A) \rightarrow l[i, j] = l[i - 1, j - 1] + c[x_i, y_j]$$

$$(B) \rightarrow l[i, j] = l[i - 1, j] + g$$

$$(C) \rightarrow l[i, j] = l[i, j - 1] + g$$

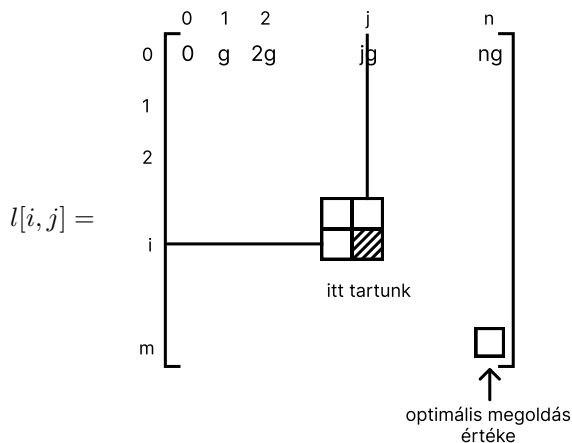
Sajnos nem tudjuk, hogy (A) és (B) és (C) közül melyik teljesül, így mindhármát megvizsgáljuk és a legkedvezőbbet választjuk.

$$l[i, j] = \min \begin{cases} l[i - 1, j - 1] + c[x_i, y_j] \\ l[i - 1, j] + g \\ l[i, j - 1] + g \end{cases}$$

[szakirodalomban sok helyen (bizonyítás nélkül) az áll, hogy ha $x_i = y_j$, akkor $l[i, j] = l[i - 1, j - 1]$ (feltéve, hogy $c[\"p\", \"q\"] \geq 0$ mindig)]

4. Optimális megoldás értéke

Az $(m + 1) \times (n + 1)$ -es $l[i, j]$ táblázatot töltjük ki fentről lefelé, soronként balról jobbra.



Az első sorral kezdünk, az aktuális érték a három szomszédából számítható (amiket már ismerünk)

Számítási bonyolultság: $\mathcal{O}(mn)$ cella, cellánként $\mathcal{O}(1)$ számítás $\rightarrow \mathcal{O}(mn)$.

5. Optimális megoldás

\rightarrow optimális M

$l[i, j]$ számításakor az (A), (B) és (C) közül a legkedvezőbbet adót magát is feljegyezzük.

$\rightarrow \searrow \downarrow$ (vizuálisan)

Ezeket tároljuk egy $k[i, j]$ táblázatban, amelynek elkészülte után a jobb alsó sarokból a nyilak mentén a ”margóig” lépkedünk.

Ha $k[i, j] = \nwarrow$, akkor $M = M \cup \{(i, j)\}$ az üres M -ből indulva.

9.2. Leghosszabb közös részsorozat

$$X = (x_1, x_2, \dots, x_m)$$

$$Y = (y_1, y_2, \dots, y_n)$$

Egy leghosszabb olyan $Z = (z_1, z_2, \dots, z_k)$ -t keresünk, amely megkapható X -ből is és Y -ből is ”néhány” karakter törlésével.

Hasonló DP algoritmus működik itt is, mint az előbb.

1. $R[i, j]$ X_i és Y_j egy leghosszabb közös részsorozatának meghatározása

2. Állítás

Legyen $Z_k = (z_1, z_2, \dots, z_k)$ az $X_m = (x_1, x_2, \dots, x_m)$ és $Y_n = (y_1, y_2, \dots, y_n)$ egy leghosszabb közös részsorozata (LKR).

1. Ha $x_m = y_n$, akkor $z_k = x_m = y_n$ ÉS Z_{k-1} egy LKR X_{m-1} -hez és Y_{n-1} -hez.
2. Ha $x_m \neq z_k$, akkor Z_k egy LKR X_{m-1} -hez és Y_n -hez.
3. Ha $y_n \neq z_k$, akkor Z_k egy LKR X_m -hez és Y_{n-1} -hez.

3. Legyen $l[i, j]$ az $R[i, j]$ optimális megoldásának értéke (hossza).

$$(A) \rightarrow l[i, j] = l[i-1, j-1] + 1$$

$$(B) \rightarrow l[i, j] = l[i-1, j]$$

$$(C) \rightarrow l[i, j] = l[i, j-1]$$

$$l[i, j] = \min \begin{cases} 0 & \text{ha } i = 0 \text{ vagy } j = 0 \text{ (rekurzió "elvarrása")} \\ l[i-1, j-1] + 1 & \text{ha } i, j \geq 1 \text{ és } x_i = y_j \text{ (A)} \\ \min\{l[i-1, j], l[i, j-1]\} & \text{ha } i, j \geq 1 \text{ és } x_i \neq y_j \text{ (B), (C)} \end{cases}$$

10. előadás (2025. november 18.)

10.1. NP-nehéz feladatok és DP

1. Részhalmaz-összeg

(NP-teljes eldöntési feladat)

Adott a_1, a_2, \dots, a_n, B pozitív egészekhez létezik-e olyan $I \in \{1, 2, \dots, n\}$, hogy $\sum_{i \in I} a_i = B$?

2. Pénzváltás

Adott a_1, a_2, \dots, a_n, B pozitív egészekhez léteznek-e olyan $\alpha_1, \alpha_2, \dots, \alpha_n$ nemnegatív egészek, hogy $\sum_{i=1}^n \alpha_i a_i = B$?

Optimalizálási feladat

Ha a válasz igen, mi $\sum_{i=1}^n \alpha_i$ minimális értéke?

”Boltban”

$B \rightarrow$ visszajáró pénz

$a_1, a_2, \dots, a_n \rightarrow$ rendelkezésre álló címletek (pl. érmék)

A legkevesebb érmével akarjuk a B összeget kifizetni.

Az egyszerűség kedvéért tegyük fel, hogy a címletek között szerepel az 1, mondjuk $1 = c_1 < c_2 < c_3 < \dots < c_n$.

Bolti algoritmus: vegyük a legnagyobb olyan c_i -t, amelyre $c_i \leq B$, és adjunk belőle annyit, amennyit lehetséges. Ezután rekurzívan fizetjük ki a maradékot.

Optimális megoldást ad az algoritmus?

A tipikus (ahol a címletek: $1c, 2c, 5c, 10c, 20c, 50c$) boltokban igen.

Sajnos nem minden címlethalmaznál ”működik” ez: $1, 6, 10$.

$12 \rightarrow 10 + 1 + 1$ algoritmus

$12 \rightarrow 6 + 6$ optimális

Tetszőleges c_1, c_2, \dots, c_n címletekre ez egy NP-nehéz feladat.

Meglepetés: polinom időben eldönthető, hogy adott c_1, c_2, \dots, c_n címletekre a bolti algoritmus optimális megoldást ad-e.

3. Hátizsák feladat

Adottak az $a_1, a_2, \dots, a_n, c_1, c_2, \dots, c_n, B$ pozitív egész számok, határozzunk meg egy olyan $I \subseteq \{1, 2, \dots, n\}$ indexhalmazt, amelyre $\sum_{i \in I} a_i \leq B$ és $\sum_{i \in I} c_i$ maximális.

Mitől hátizsák feladat ez? Van n tárgyunk t_1, t_2, \dots, t_n . A t_i tárgy súlya b_i , értéke c_i . Van egy hátizsákunk, amelynek (súly)kapacitása B .

Mely tárgyakat tegyük a hátizsákba, hogy az összérték maximális legyen, a súlykorlátot nem túllépve?

Összefüggések 1-2-3 között

Részhalmaz-összeg és hátizsák feladat

Hogyan lehetne alkalmazni egy hátizsák feladatra adott algoritmust a részhalmaz-összeg feladat megoldására? (visszavezetés, rekurzió)

Tekintsük a részhalmaz-összeg feladat egy példányát: a_1, a_2, \dots, a_n, B

Elkészítünk ehhez egy hátizsák feladat példányt:

n tárgy, az i -edik súlya és értéke is a_i

a hátizsák kapacitása B

Tegyük fel, hogy ennek a hátizsák feladatnak I egy megoldása: $\sum_{i \in I} a_i \leq B$ és $\sum_{i \in I} c_i$ maximális

Két eset van:

$\rightarrow \sum_{i \in I} = B \rightarrow$ igen a részhalmaz-összegeknél

$\rightarrow \sum_{i \in I} < B \rightarrow$ nem (igen esetén I nem lenne optimális megoldás)

Részhalmaz-összeg és pénzváltás

(eldöntési változat)

Pénzváltás példány:

a_1, a_2, \dots, a_n, B a_1 -ből vegyünk $\left\lfloor \frac{B}{a_1} \right\rfloor$ példányt

a_2 -ből vegyünk $\left\lfloor \frac{B}{a_2} \right\rfloor$ példányt

\vdots

a_n -ből vegyünk $\left\lfloor \frac{B}{a_n} \right\rfloor$ példányt

Részhalmaz-összeg példány:

$\underbrace{a_1, \dots, a_1}_{\left\lfloor \frac{B}{a_1} \right\rfloor}, \underbrace{a_2, \dots, a_2}_{\left\lfloor \frac{B}{a_2} \right\rfloor}, \dots, \underbrace{a_n, \dots, a_n}_{\left\lfloor \frac{B}{a_n} \right\rfloor}, B$

Ha itt igen, akkor a pénzváltásnál is igen, ha nem, akkor ott is nem.

Probléma: a részhalmaz-összeg példány mérete nem csak n -től, de B -től is függ, ez utóbbtól lineárisan.

Lineárisan: ez elfogadhatatlan! Ami elfogadható lenne itt: $\Omega(\log B)$.

Elérhető ez? IGEN!

$a_1 \rightarrow \underbrace{a_1, 2a_1, 4a_1, 8a_1, \dots, 2^{\lfloor \log_2 B \rfloor} a_1}_{\text{kettes számrendszerben gondolkodhatunk}}$
 $a_1, 2a_1, 3a_1, \dots, \left\lfloor \frac{B}{a_1} \right\rfloor a_1$ közül minden
 felírható csupán ezeket használva

Ugyanez a_2, a_3, \dots, a_n esetén.

11. előadás (2025. november 25.)

Hátizsák feladat

Legyenek t_1, t_2, \dots, t_n tárgyak. Minden t_i tárgynak van egy w_i súlya és egy v_i értéke. Adott egy hátizsák W kapacitással. Válasszunk ki a tárgyak közül néhányat úgy, hogy a kiválasztott tárgyak összsúlya ne haladja meg a hátizsák W kapacitását és az összértéke a lehető legnagyobb.

Ismert, hogy ez egy NP-nehéz feladat. Ezzel együtt van egy különös tulajdonsága: bármely $\varepsilon > 0$ valós számhoz van olyan polinomiális algoritmus, amely által szolgáltatott megoldásban az összérték legalább $\frac{1}{1+\varepsilon}$ -szorosa az optimális összértéknek. (aprópénzre váltva legalább 99%-a, legalább 99,9%-a, legalább 99,99%-a, és így tovább)

Hátizsák feladat input: $w_1, v_1, w_2, v_2, \dots, w_n, v_n, W$ mind pozitív egészek.

DP algoritmus (általánosan nem polinomiális!)

1. Részproblémák

$R[i, j] \rightarrow$ tárgyak t_1, t_2, \dots, t_i , hátizsák kapacitás: j

2. Optimális részstruktúra tulajdonság

Legyen Opt egy optimális (maximális összértékű) megoldása $R[i, j]$ -nek.

Most $t_i \notin Opt$ (Opt most egy tárgyhalmaz) esetén Opt optimális megoldása $R[i-1, j]$ -nek. Valóban, ha lenne $R[i-1, j]$ -nek Opt -nál nagyobb összértékű megoldása, az Opt -nál nagyobb összértékű megoldása lenne $R[i, j]$ -nek is, **ellentmondás**.

Másképp $t_i \in Opt$ esetén $Opt \setminus \{t_i\}$ egy maximális összértékű megoldása $R[i-1, j-w_i]$ -nek. Valóban, ha lenne $R[i-1, j-w_i]$ -nek $Opt \setminus \{t_i\}$ -nél nagyobb összértékű megoldása, akkor ezt kiegészítve t_i -vel, $R[i, j]$ -nek egy Opt -nál nagyobb összértékű megoldását kapnánk, **ellentmondás**.

3. Rekurzió

Jelölje $h[i, j]$ az $R[i, j]$ optimális megoldásánál az összértéket.

Nyilván $h[i, 0]$ és $h[0, j]$ nulla.

Különben, vagyis amikor $i, j \geq 1$, akkor

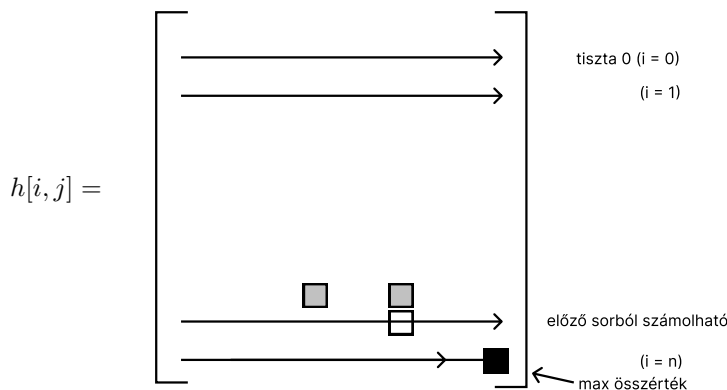
A ha $w_i > j$, akkor t_i nem lehet benne az optimális megoldásban, így $h[i, j] = h[i-1, j]$.

B ha viszont $w_i \leq j$, akkor t_i vagy benne van az optimális megoldásban, vagy nincs, és nem is tudjuk melyik áll fenn a kettő közül. Így szokásos módon mindkét lehetőséget megvizsgáljuk, és a kedvezőbb lesz a befutó:

$$h[i, j] = \max \begin{cases} v_i + h[i-1, j-w_i] & \text{(binnen van)} \\ h[i-1, j] & \text{(nincs benne)} \end{cases}$$

4. Táblázat

Az $(n+1) \times (w+1)$ -dimenziós $h[i, j]$ táblázatot töltjük ki (felülről lefelé)



Oszlopok: $j = 0, 1, \dots, W$

Költség:

egy cella: $\mathcal{O}(1)$

cellák száma: $\mathcal{O}(nW)$

összesen: $\mathcal{O}(nW)$, nem polinomiális!

A táblázatban az oszlopok száma $W + 1$, ez W , mint input input méretében (ami $\log W$) exponenciálisan nagy.

Megjegyzés: kis súlyok esetén az algoritmus polinomiális (pontosítva: $w_1 + w_2 + \dots + w_n \leq W$ esetén a feladat triviális, itt nem ilyenekről van szó)

5. Optimális tárgyhalmoz

$h[i, j]$ számolásakor feljegyezzük, hogy t_i bevétele vagy kihagyása volt kedvezőbb, a jobb alsó sarokból indulva ezek segítségével rekonstruálható egy optimális megoldás.

Lássunk ezek után egy alternatív megközelítést

Hátizsák feladat kicsit másképpen

- input változatlan
- "Célfüggvény" legalább V összértéket szeretnénk elérni az ehhez szükséges tárgyak minimális összsúlya mellett.

DP itt is működik (ezt talán mindenki érzi)

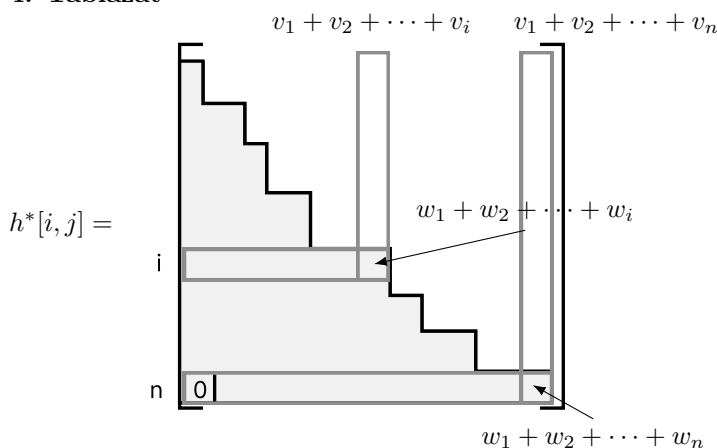
1. Részproblémák

$R[i, j]$ a t_1, t_2, \dots, t_i tárgyakból választhatunk és legalább j összértéket akarunk elérni minimális összsúly mellett.

Jelölje $h^*[i, j]$ a minimális összsúlyt.

2. **3.** nagyon hasonlít az előbbiekhez.

4. Táblázat



Csak az "alsó" részt töltjük ki.

Hogyan használható ez a táblázat az "eredeti" hátizsák feladat megoldására (ahol W a hátizsák kapacitása)? Az utolsó sorban balról jobbra haladva megkeressük az utolsó $\leq W$ értéket, ennek oszlopindexe lesz az elérhető maximális összsúly.

Költség:

egy cella: $\mathcal{O}(1)$

cellák száma: legyen $v^* = \max v_i$

sorok száma: $\mathcal{O}(n)$

oszlopok száma: $\mathcal{O}(nv^*)$

összesen: $\mathcal{O}(n^2 v^*)$, nem polinomiális!

Azonban kis értékek esetén polinomiális! Pl. ha $v^* = \mathcal{O}(n^3)$

12. előadás (2025. december 2.)

Közelítő hátizsák feladat

Tétel

Minden $\varepsilon > 0$ valós számhoz létezik olyan polinomiális algoritmus, amely által visszaadott összérték a hátizsák problémára legalább $\frac{1}{1+\varepsilon}$ -szoros az optimális megoldásnak.

Hátizsák feladatunk

$$t_1 \rightarrow (w_1, v_1)$$

$$t_2 \rightarrow (w_2, v_2)$$

\vdots

$$t_n \rightarrow (w_n, v_n)$$

W

Nyilván azok a tárgyak, amelyek súlya nagyobb, mint W nem lehetnek benne egyetlen megoldásban sem, ezért feltehetjük, hogy $w_i \leq W$ ($1 \leq i \leq n$)

Nyilván elég a tételt olyan ε pozitív számokra igazolni, amelyek pozitív egészek reciprokai.

Definiálhatunk két másik hátizsák feladatot. Legyen $b = \frac{\varepsilon}{2n} v^*$ (ahol $v^* = \max_i v_i$)

A súlyok és a hátizsák kapacitása változatlan.

I. az értékek $\tilde{v}_i = \lceil \frac{v_i}{b} \rceil b$

II. az értékek $\hat{v}_i = \lceil \frac{v_i}{b} \rceil = \frac{\tilde{v}_i}{b}$

Jegyezzük meg, hogy $v_i \leq \tilde{v}_i \leq v_i + b$.

Nyilván **I.** és **II.** optimális megoldása ugyanaz $\rightarrow \mathcal{J}$

Oldjuk meg **II.**-t a múlt végén ismertetett DP algoritmussal.

Mi is volt ez a DP algoritmus?

A részproblémák úgy néztek ki, hogy a t_1, t_2, \dots, t_i tárgyakkal akartunk legalább j értéket elérni minimális összsúly mellett $\rightarrow h^*(i, j)$ volt a min összsúly.

$$[1 \leq i \leq n, 0 \leq j \leq v_1 + v_2 + \dots + v_i]$$

Ekkor:

$$h^*[i, j] = \begin{cases} 0 & \text{ha } j = 0 \\ w_i + h^*[i-1, \max\{0, j - v_i\}] & \text{ha } j > v_1 + v_2 + \dots + v_{i-1} \\ \min \begin{cases} w_i + h^*[i-1, \max\{0, j - v_i\}] \\ h^*[i-1, j] \end{cases} & \text{ha } 1 \leq j \leq v_1 + v_2 + \dots + v_{i-1} \end{cases}$$

Költség: n sor, $\leq nv^*$ oszlop, cellánként $\mathcal{O}(1)$ számolás $\rightarrow \mathcal{O}(n^2 v^*)$

II. megoldásának költsége
 $\mathcal{O}(n^2 \max_i \hat{v}_i)$

Mennyi ez? Legyen $\max_i v_i = v_j$. Ekkor $\max_i \hat{v}_i = \hat{v}_j$.

Igen ám, de $\hat{v}_j = \lceil \frac{v_j}{b} \rceil = \lceil \frac{v^*}{b} \rceil = \lceil \frac{\frac{\varepsilon}{2n} v^*}{\frac{\varepsilon}{2n}} \rceil = \lceil \frac{1}{\varepsilon} \rceil = \lceil \frac{2n}{\varepsilon} \rceil = \frac{2n}{\varepsilon}$ ($\frac{1}{\varepsilon}$ pozitív egész, $2n$ is)

Visszahelyettesítve $\mathcal{O}(n^2 \max_i \hat{v}_i)$ -ba ez $\mathcal{O}(n^3 \varepsilon^{-1})$ ami rögzített ε mellett polinomiális!

Hátra van még az " $\frac{1}{1+\varepsilon}$ rész".

Idézzük fel, hogy \mathcal{J} nem csak a \hat{v}_i értékekkel, de a \tilde{v}_i értékekkel is optimális megoldás.

Mit mondhatunk \mathcal{J} -ről a v_i értékek esetén (eredeti hátizsák feladat)?

Tekintsük az eredeti feladat egy tetszőleges \mathcal{J}^* megengedett megoldását (ebben benne van az optimális is).

Megmutatjuk, hogy $\frac{1}{1+\varepsilon} \sum_{i \in \mathcal{J}^*} v_i \leq \sum_{i \in \mathcal{J}} v_i$. Vagy kicsit átrendezve: $\sum_{i \in \mathcal{J}^*} v_i \leq (1 + \varepsilon) \sum_{i \in \mathcal{J}} v_i$. (Persze \mathcal{J} is egy megengedett megoldása az eredeti feladatnak, mert a súlyok és a kapacitás nem változik.)

Az optimális megoldást adó \mathcal{J}^* -gal ez pont a tétel állítása.

”Bűvészkedünk”

Idézzük fel, hogy $v_i \leq \tilde{v}_i \leq v_i + b$. Most jön egy hosszú egyenlőtlenséglánc:

$$\sum_{i \in \mathcal{J}^*} v_i \leq \sum_{i \in \mathcal{J}^*} \tilde{v}_i \leq \sum_{i \in \mathcal{J}} \tilde{v}_i \leq \sum_{i \in \mathcal{J}} (v_i + b) \leq \sum_{i \in \mathcal{J}} v_i + \sum_{i \in \mathcal{J}} b \leq \left(\sum_{i \in \mathcal{J}} v_i \right) + nb$$

\mathcal{J} az optimális megoldás a \tilde{v}_i értékekre. Arra hajtunk, hogy $nb \leq \varepsilon \sum_{i \in \mathcal{J}} v_i$.

Piszkos trükk: nézzük az egyenlőtlenséglánc következő részét: $\sum_{i \in \mathcal{J}} \tilde{v}_i \leq \left(\sum_{i \in \mathcal{J}} v_i \right) + nb$

Először is a j -edik tárgy (ennek a legnagyobb az értéke) önmaga megengedett megoldása $\boxed{\mathbf{I}}$ -nél, így $\tilde{v}_j \leq \sum_{i \in \mathcal{J}} \tilde{v}_i$.

13. előadás (2025. december 2.)

Ütemezés

Adott munkák $M = \{m_1, m_2, \dots, m_n\}$ halmaza, és minden $m_i \in M$ -hez:

- t_i : ennyi ideig tart elvégezni
- d_i : határidő
- w_i : munkadíj

(≥ 0 egészek)

Mely munkákat vállaljuk el a max. összdíjért?

→ egyszerre csak egy munkát lehet végezni

→ nem lehet félbeszakítani

→ határidőre be kell fejezni

Az $m_{i_1}, m_{i_2}, \dots, m_{i_k}$ kiválasztott munkák egy megengedett ütemezése: megadunk S_{i_1}, f_{i_1} értékeket, ami-re:

$$0 \leq S_{i_1}, f_{i_1} = S_{i_1} + t_{i_1}, f_{i_1} \leq d_{i_1}$$

$$f_{i_1} \leq S_{i_2}, f_{i_2} = S_{i_2} + t_{i_2}, f_{i_2} \leq d_{i_2}$$

$$f_{i_2} \leq S_{i_3}, f_{i_3} = S_{i_3} + t_{i_3}, f_{i_3} \leq d_{i_3}$$

\vdots

$$f_{i_{k-1}} \leq S_{i_k}, f_{i_k} = S_{i_k} + t_{i_k}, f_{i_k} \leq d_{i_k}$$

Szeretnénk az M olyan részhalmazát választani, aminek van megengedett ütemezése és $\sum_{l=1}^k w_{i_k}$ maximális.

Ez NP-nehéz feladat. Létezik DP algoritmus a feladat megoldására.

Speciális eset: $d_1 = d_2 = \dots = d_n = d$. Minden határidő ugyanaz \rightarrow hátizsák probléma t_i súlyokkal, w_i értékkel, d kapacitással.

Egy másik speciális eset: $w_1 = w_2 = \dots = w_n$. Cél: legtöbb munka elvégzése. Erre van $\mathcal{O}(n \log n)$ -es MOHÓ ALGORITMUS.

Mohó algoritmus az ütemezési probléma speciális esetére:

Feltehetjük, hogy $d_1 \leq d_2 \leq \dots \leq d_n$. Egy H halmazba fogjuk gyűjteni a kiválasztott munkákat. Kezdetben legyen $H = \emptyset$. Egymás után vesszük hozzá H -hoz az m_1 munkákat. $i = 1, 2, 3, \dots$

Ha az m_i munka bevitelével $\sum_{m_j \in H} t_j > d_i$ adódik, akkor töröljük H -ból a leghosszabb elvégzési idejűt.

Végül a H -beli feladatokat ütemezzük a 0. időpillanattól szünet nélkül a határidők szerint növekvő sorrendben (kanonikus ütemezés).

Állítás: H -ban minden munka befejeződik legkésőbb a határidőre.

Bizonyítás: triviális.

Állítás: H optimális ütemezés az adott munkákra nézve.

Bizonyítás:

Indukció n -re ($|M| = n$)

$n = 1$: triviális

Most tegyük fel, hogy $n > 1$ és a mohó algoritmus optimális ütemezést ad $\leq n - 1$ munka esetén.

Ha H tartalmazza az összes munkát, akkor optimális \rightarrow kész vagyunk.

Tegyük fel, hogy H nem az összes munkát tartalmazza, és legyen m_k az a munka, amit először törölt a mohó algoritmus, mondjuk az m_i munka hozzávételekor.

Állítás: van olyan \mathcal{O} optimális kanonikus ütemezés, hogy m_k ebben sem szerepel.

Bizonyítás: hamarosan

Hogyan használhatjuk ezt H optimalitásának bizonyítására?

Nyilván: $|H| \leq |\mathcal{O}|^*$ mert \mathcal{O} optimális.

Tekintsük most azt a feladatot, hogy $m_1, m_2, \dots, m_{k-1}, m_{k+1}, \dots, m_n = M'$ (eggyel kevesebb munkánk van) munkákat kell ütemezni.

Ekkor a mohó algoritmus itt is pont a H ütemezést fogja adni.

Az indukciós feltevés miatt itt (M' munkák esetén) a mohó algoritmus optimális ütemezést ad.

\mathcal{O} az M' munkákra megengedett ütemezés (mert nincs benne m_k). Emiatt $\mathcal{O} \leq |H|$ és $*$ (azaz $|H| \leq |\mathcal{O}|$) miatt $|H| = |\mathcal{O}|$, tehát H is optimális ütemezés az M munkákra.

Költségek:

1. kezdeti rendezés: $\mathcal{O}(n \log n)$
2. H manipulációja:
 - tároljuk H elemeit egy (max) kupacban
 - egy segédváltozóban tároljuk a H -beli munkák össz-elvégzési idejét

Műveletek H -val:

- m_i bevétele H -ba
 - bevesszük a kupacban $\leftarrow \mathcal{O}(\log n)$
 - segédváltozó t_i -vel növelve $\leftarrow \mathcal{O}(1)$
 - vizsgálat: segédváltozó $\stackrel{?}{>} d_i \leftarrow \mathcal{O}(1)$
- m_k törlése H -ból
 - remMax $\leftarrow \mathcal{O}(\log n)$
 - segédváltozó csökken t_k -val $\leftarrow \mathcal{O}(1)$

$\Rightarrow n$ darab munkára $\mathcal{O}(n \log n)$ költség

Állítás: van olyan \mathcal{O} optimális kanonikus ütemezés, hogy m_k ebben sem szerepel.

Bizonyítás: tfh. \mathcal{O} optimális kanonikus ütemezés és m_k benne van m_k választásából adódik, hogy az m_i bevételekor távolítottuk el m_k -t H -ból.

$$\sum_{j=1}^i t_j > d_i$$

Emiatt az m_1, m_2, \dots, m_i munkák valamelyike hiányzik \mathcal{O} -ból.

Legyen m_l az a munka, ami nincs \mathcal{O} -ban ($1 \leq l \leq i, l \neq k$).

Cseréljük ki \mathcal{O} -ban az m_k munkát m_l -re, ütemezzük kanonikusan, legyen ez \mathcal{O}^* .

Állítás: \mathcal{O}^* -ban is minden munka befejeződik határidőre.

Bizonyítás: m_k választása miatt $t_k \geq t_l$. $\mathcal{O} \mathcal{O}^*$:

\mathcal{O}^* is optimális, és \mathcal{O}^* nem tartalmazza az m_k munkát. QED