

# Algoritmuskélet (BMEVISZA213) feladatgyűjtemény

Drótos Márton

2012. május 30.

# Tartalomjegyzék

1. Előszó, használati tanácsok	2
2. Nagyságrendek	3
3. Dinamikus programozás	4
4. Szélességi bejárás, legrövidebb utak, párosítások	5
5. Kupacok	6
6. Keresés, rendezés	7
7. Keresőfák, p-f fák, 2-3 fák	9
8. Hash	10
9. Mélységi bejárás, DAG, valamint alkalmazásaik	11
10. Minimális költségű feszítőfák	13
11. Bonyolultságelmélet	14
12. Egészértékű programozás	16
13. Közelítő algoritmusok	16
14. Megoldások	18

## 1. Előszó, használati tanácsok

Jelen feladatgyűjteményben többnyire az elmúlt években a gyakorlataimon feladott feladatok szerepelnek. Sok feladat régebbi ZH-kon, vizsgákon szerepelt.

A megoldások főleg az ötletek leírására helyezik a hangsúlyt, sokszor az egyszerű technikai részletek hiányoznak. Feltételezem, hogy ezeket mindenki képes önállóan is megfogalmazni. Mechanikus feladatból kevés van (nem is szerepel minden típus), hiszen ezeket az előadásokból és a weben található animációkból sokkal könnyebb megtanulni.

A megoldásokat megtanulni, vagy a feladatokat megoldásokkal együtt nézni nem tanácsos, hiszen a számonkéréseken teljesen ismeretlen feladatok lesznek, amiket önállóan kell megoldani. A legcélravezetőbb módszer az, hogy ha valaki önállóan megpróbálja üres papíron megoldani a feladatokat, majd az itt található megoldással összeveti (különös tekintettel az indoklásra).

Minden témakörben az első néhány feladat bevezető és/vagy típusfeladat. Természetesen ez nem jelenti azt, hogy csak ilyenek szerepelnek számonkérésen. A többi feladat többé-kevésbé nehézség szerint van rendezve.

A jegyzet kinyomtatását nem javaslom, egyrészt papírtakarékossági okokból, másrészt elképzelhető, hogy az idők folyamán néhány helyen javítani fogok rajta, esetleg új feladatokkal bővül.

## 2. Nagyságrendek

1. Bizonyítsuk be, hogy

- (a)  $\log_2 f(n) = \Theta(\log_{100} f(n))$  ( $f(n) > 0$ )
- (b)  $f(x) = a_k x^k + a_{k-1} x^{k-1} + \dots + a_0$  ( $a_k \neq 0$ )  $\Rightarrow f(n) = \Theta(n^k)$
- (c)  $2^{n+1} = O(2^n)$ , de  $2^{2n} \neq O(2^n)$
- (d)  $\max(f(n), g(n)) = \Theta(f(n) + g(n))$  ( $f(n), g(n) > 0$ )

2. **[Vizsga: 2007. június 12.]** Egy  $\mathcal{A}$  algoritmusról azt tudjuk, hogy  $n$  hosszú bemeneteken a lépésszáma  $O(n \log n)$ . Lehetséges-e, hogy

- (a) van olyan  $x$  bemenet, amin a lépésszáma  $x^3$ ?
  - (b) minden  $x$  bemeneten legfeljebb  $2007|x|$  lépést használ?
- (Szokás szerint  $|x|$  az  $x$  szó hosszát jelöli.)

3. Jelöljük  $T(n)$ -nel egy algoritmus legnagyobb lehetséges lépésszámát az  $n$  méretű inputokon. Tudjuk, hogy  $T(n) \leq 10$ , ha  $n \leq 5$  és  $T(n) \leq T(n-1) + n/3$ , ha  $n > 5$ . Ekkor mit tudunk mondani  $T(n) = O(n)$ ,  $T(n) = O(n^2)$  és  $T(n) = O(n^3)$  egyenlőségek helyességéről?

4. **[ZH: 2011. március 28.]** Egy problémára két algoritmusunk van.

Az  $\mathcal{A}$  algoritmus az  $n \geq 2$  méretű problémából 10 lépéssel 2 db  $n-1$  méretűt készít és ezeket oldja meg rekurzívan.

A  $\mathcal{B}$  az  $n \geq 2$  problémából 3 lépéssel 4 db  $n-1$  méretűt készít és ezeket oldja meg rekurzívan. Az  $n=1$  esetben mindkét eljárás 1 lépést használ.

Melyik algoritmus lesz nagy  $n$  értékekre gyorsabb?

5. Igaz-e, hogy

- (a) ha  $f = O(g)$  és  $g = O(h)$ , akkor  $f = O(h)$
- (b) ha  $f = \Omega(g)$  és  $g = \Omega(h)$ , akkor  $f = \Omega(h)$

6. Az alábbi függvényeket rendezzük olyan sorozatba, hogy ha  $f_i$  után közvetlenül  $f_j$  következik a sorban, akkor  $f_i(n) = O(f_j(n))$  teljesüljön!

$$f_1(n) = 8n^{2.5}, \quad f_2(n) = 5\sqrt{n} + 1000n, \quad f_3(n) = 2^{\log^2 n}, \quad f_4(n) = 2008n^2 \log n$$

7. Az  $\mathcal{A}$  algoritmusról azt tudjuk, hogy  $n$  hosszú bemeneteken a lépésszáma  $O(n^2)$ . Lehetséges-e, hogy

- (a)  $\forall n$  hosszú bemeneten  $O(n)$  lépést használ?
- (b)  $\exists x$ , hogy az  $x$  bemeneten az algoritmus lépésszáma  $10|x|^2 \log |x| - 800$  (ahol  $|x|$  az  $x$  bemenet hosszát jelöli)?

8. **[Vizsga: 2007. június 19.]** Az alábbi függvényeket rendezze olyan sorozatba, hogy ha  $f_i$  után közvetlenül  $f_j$  következik a sorban, akkor  $f_i(n) = O(f_j(n))$  teljesüljön!

$$f_1(n) = 2^{100n} - 2^{50n}, \quad f_2(n) = 2007n^3, \quad f_3(n) = 3^{3n}$$

9. **[Vizsga: 2007. június 5.]** Jelölje egy algoritmus maximális lépésszámát az  $n$  hosszú bemeneteken  $L(n)$ . Azt tudjuk, hogy minden  $n = 2k > 4$  páros számra  $L(2k) \leq L(2k-2) + 1$  teljesül, és hogy  $L(4) = 10$ . Következik-e ebből, hogy az algoritmus lépésszáma  $O(n)$ ?

10. **[PZH: 2011. április 22.]** Egy problémára két algoritmusunk van.

Az  $\mathcal{A}$  algoritmus az  $n \geq 2$  méretű problémából 5 lépéssel 2 db legfeljebb  $n/2$  méretűt készít és ezeket oldja meg rekurzívan.

A  $\mathcal{B}$  algoritmusról azt tudjuk, hogy lépésszáma az  $n$  méretű problémákon  $O(n^2)$ .

Ha ennyiből lehetséges, határozza meg, melyik algoritmus lesz nagy  $n$  értékekre gyorsabb! Ha ennyi információból még nem következik, hogy  $\mathcal{A}$  vagy  $\mathcal{B}$  lesz a gyorsabb, akkor indokolja meg, miért nem!

11. Tudjuk, hogy  $f(x) = O(h(x))$  és  $g(x) = O(h(x))$ . Igaz-e, hogy
- ha  $h(x) = 3x$ , akkor  $f(g(x)) = O(h(x))$
  - $f(g(x)) = O(h(x)) \forall h$  függvényre

### 3. Dinamikus programozás

- [ZH: 2008. március 28.] Egy  $n \times n$  méretű táblázat minden eleme egy egész szám. A táblázat bal alsó sarkából akarunk eljutni a jobb felső sarkába úgy, hogy egy lépésben a táblázatban vagy felfelé vagy jobbra egyet lépünk. Azt szeretnénk, hogy a lépegetés során látott elemek növekvő sorrendben kövessék egymást. Egy ilyen út értéke a benne szereplő számok összege. Adjon  $O(n^2)$  futási idejű algoritmust, ami meghatározza, hogy az adott táblázatban a szabályok szerinti utak értékei között mekkora a legnagyobb!
- Adott egy fa, melynek csúcsaihoz súlyok vannak rendelve. Adjunk lineáris algoritmust, ami meghatározza a fában található maximális súlyú független ponthalmaz súlyát!
- [Vizsga: 2007. június 12.] Egy  $n$  és egy  $m$  karakterből álló szövegben meg akarjuk találni a legnagyobb azonos darabot, azaz ha az egyik szöveg  $a_1a_2 \cdots a_n$  és a másik  $b_1b_2 \cdots b_m$ , akkor olyan  $1 \leq i \leq n$  és  $1 \leq j \leq m$  indexeket keresünk, hogy

$$a_{i+1} = b_{j+1}, a_{i+2} = b_{j+2}, \dots, a_{i+t} = b_{j+t}$$

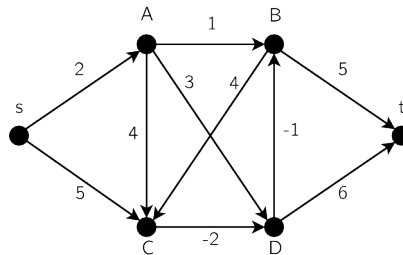
teljesüljön a lehető legnagyobb  $t$  számra. Adjon erre a feladatra  $O(mn)$  lépést használó algoritmust!

- [Vizsga: 2007. május 29.] Legyen  $w = w_1w_2 \cdots w_n$  egy  $n$  betűből álló szó. Hívjuk részsónak  $w$  egy tetszőleges  $w_iw_{i+1} \cdots w_{i+k}$  darabját ( $1 \leq i \leq n-1$ ,  $1 \leq k \leq n-i$ ). Adjunk algoritmust, ami  $O(n)$  lépésben meghatározza az összes  $a$ -val kezdődő és  $b$ -re végződő részsó számát.
- Egy játékban egy  $n \times m$  rács bal felső sarkából kell eljutnunk a jobb alsó sarokba. Egy lépés során a rács mentén vízszintesen vagy függőlegesen tudunk a következő rácspontra lépni. Azonban van néhány kereszteződés, ahova nem szabad lépni. Ezek helyét az  $R$  tömb írja le,  $R[i, j] = 1$ , ha az  $(i, j)$  kereszteződésbe nem léphetünk, egyébként  $R[i, j] = 0$ . Adjunk  $O(nm)$  futási idejű algoritmust annak meghatározására, hogy pontosan  $n + m - 2$  lépést téve a rácson hányféleképpen tudjuk a célt elérni!
- [ZH: 2010. április 19.] Egy  $n \times k$  méretű táblázatban van néhány megjelölt elem. A táblázat bal alsó sarkából akarunk eljutni a jobb felső sarkába úgy, hogy minden lépésben a táblázat egy eleméről vagy a közvetlen felette vagy a tőle jobbra lévő elemre mehetünk (ha van ilyen). Adjon  $O(nk)$  idejű algoritmust, amely a megjelölt elemek helyét ismerve meghatározza, hogy egy ilyen út során maximálisan hány alkalommal tudunk megjelölt elemre lépni!
- Legyenek  $a_1, a_2, \dots, a_n$  tetszőleges egész számok és  $m < n^2$  egész. Adjunk algoritmust, amely a bináris alakjukkal megadott  $a_1, a_2, \dots, a_n$  és  $m$  számokról eldönti polinom időben, hogy az  $a_1, a_2, \dots, a_n$  számok közül kiválasztható-e néhány úgy, hogy az összegük  $m$ -mel osztva egyet adjon maradékul!
- Adjunk algoritmust, ami egy  $n$  csúcú fában lineáris időben meghatározza a fában levő leghosszabb út hosszát!
- [Vizsga: 2009. június 11.] A véges hosszú 0-1 sorozatok egy részét valamilyen szempontból jónak tekintjük, a többi rossznak. Van egy  $A$  algoritmusunk, mely adott  $n$  hosszú 0-1 sorozatról  $O(\log(n!))$  lépésben megmondja, hogy a sorozat jó vagy rossz.  
Adjon olyan eljárást, mely  $A$ -t felhasználva, ha adott egy  $m$  hosszú 0-1 sorozat,  $y = y_1y_2 \cdots y_m$ , akkor eldönti, hogy  $y$  előáll-e néhány jó sorozat egymás után fűzéséből. Az eljárás lépésszáma összesen legyen  $O(m^4)$ .  
Segítség:  $\log(n!) \approx O(n \log n) = O(n^2)$ .

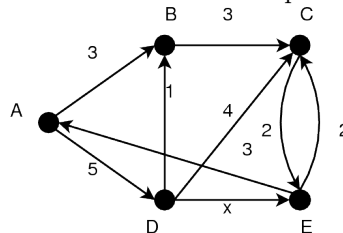
10. Egy  $n$  szóból álló szöveget kell sorokra tördelni. A szöveg  $i$ -edik szava  $\ell_i$  karakterből áll, egy sor  $s$  karakter hosszú. Ha egy sor a szöveg  $i$ -edik szavával kezdődik és a  $j$ -edik szóval végződik, akkor az elválasztó szóközöket is figyelembe véve  $t = s - (\ell_i + \ell_{i+1} + \dots + \ell_j + j - i)$  üres hely marad a sor végén. Egy ilyen sor hibája legyen  $t^2$ . A tördelés hibája a nem üres sorok hibáinak összege. Adjunk  $O(n^2)$  lépéses algoritmust egy legkisebb hibájú tördelés meghatározására! (A szavak sorrendje rögzített.)
11. Az  $1, 2, \dots, n$  számoknak adott két permutációja,  $x_1, \dots, x_n$  és  $y_1, \dots, y_n$ . A két sorozat egy közös részsorozata egy  $1 \leq i_1 < \dots < i_k \leq n$ , és egy  $1 \leq j_1 < \dots < j_k \leq n$  indexsorozattal adható meg, ahol  $x_{i_m} = y_{j_m}$  teljesül minden  $1 \leq m \leq k$  esetén. Adjunk  $O(n^2)$  lépésszámú algoritmust, ami az  $x$  és  $y$  sorozatokban meghatároz egy leghosszabb közös részsorozatot!
12. [MSc felvételi 2009. tavasz] Adott egy  $n$  és egy  $m$  hosszú 0-1 sorozat,  $a_1, a_2, \dots, a_n$ , illetve  $b_1, b_2, \dots, b_m$ . Ezek alapján egy  $T$  tömböt töltöttünk ki a következő módon:  
 Ha  $0 \leq i \leq n$ , akkor  $T[i, 0] = 0$ . Ha  $0 \leq j \leq m$ , akkor  $T[0, j] = 0$ .  
 Ha  $1 \leq i \leq n$  és  $1 \leq j \leq m$ , akkor  $T[i, j] = \begin{cases} T[i-1, j-1] + 1 & \text{ha } a_i = b_j \\ \max(T[i, j-1], T[i-1, j]) & \text{ha } a_i \neq b_j \end{cases}$
- Írja le, hogy mi a jelentése a  $T[i, j]$  értéknek! A két sorozatnak milyen tulajdonságát adja meg a  $T[n, m]$  érték?
13. Adottak  $M_1, M_2, \dots, M_n$  munkák  $H_1, H_2, \dots, H_3$  határidőkkel és  $P_1, P_2, \dots, P_n$  profitokkal. Mind-egyik munka elvégzése pontosan 1 napunkba kerül (egy napon csak egy munkát végezhetünk el). Adjunk  $O(n^2)$  lépésszámú algoritmust, amely meghatározza, hogy mely munkákat vállaljuk el a profit maximalizálása érdekében!

## 4. Szélességi bejárás, legrövidebb utak, párosítások

1. Határozzuk meg a Bellmann-Ford algoritmussal a legrövidebb utat  $s$  és  $t$  között, nyomon követve az algoritmust!

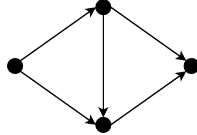


2. [ZH: 2009. április 24.] Dijkstra-algoritmussal határozza meg az alábbi gráfban az  $A$  pontból az összes többi pontba menő legrövidebb utak hosszát az  $x$  pozitív valós paraméter függvényében. Minden lépésnél írja fel a távolságokat tartalmazó  $D$  tömb állapotát és a KÉSZ halmaz elemeit.



3. Hogy néz ki egy irányítatlan teljes gráf szélességi bejárása?
4. [Vizsga: 2012. május 24.] Külföldi ösztöndíjakat szeretnénk megpályázni, ehhez ajánlólevelekre van szükségünk. Összesen  $n$  helyre adjuk be a pályázatot, minden pályázathoz két ajánlólevél szükséges. Ajánlólevelet  $m$  darab embertől tudunk kérni, de nem akarunk senkit sem túlságosan terhelni, ezért egy embertől legfeljebb egy ajánlólevelet akarunk kérni. (Az ajánlólevelek egyediek, egy levelet csak egy pályázatnál tudunk felhasználni.) Sajnos a pályázatok olyanok, hogy nem minden lehetséges ajánló személy jó minden helyre (azt tudjuk, ki hova jó). Adjunk algoritmust, ami  $O((n+m)nm)$  lépésben javasol egy lehetséges megoldást!

5. Éllistával adott a súlyozott élű  $G(V, E)$  gráf. Tegyük fel, hogy az élek súlyai az 1, 2, 3 számok közül valók. Javasoljunk  $O(n + e)$  költségű algoritmust az  $s \in V$  pontból az összes további  $v \in V$  pontokba vivő legrövidebb utak hosszának meghatározására!
6. Legfeljebb hány komponensből állhat egy irányított gráf szélességi bejárása során keletkező erdő?
7. Határozzuk meg a következő gráfban az élsúlyokat úgy, hogy a Dijkstra algoritmus rossz eredményt adjon!



8. Adjuk meg az összes olyan minimális élszámú irányított gráfot (élsúlyokkal együtt), amely(ek)re az alábbi táblázat a Dijkstra algoritmusban szereplő  $D[]$  tömb változásait mutathatja. Adjuk meg a legrövidebb utakat tartalmazó  $P[]$  tömb állapotait is!

$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	$v_6$
0	2	6	$\infty$	$\infty$	7
0	2	5	9	$\infty$	6
0	2	5	6	9	6
0	2	5	6	8	6
0	2	5	6	7	6

9. [ZH: 2009. április 24.] Egy kezdő autóvezető a városban való közlekedése során szeretne gyakorlatának megfelelő útvonalat választani. Az úthálózat egy irányítatlan gráfként van megadva, a csúcsok a kereszteződések, az élek az utak, a csúcsoknál adott, hogy nehéz-e számára a kereszteződés. (Az hogy nehéz, a kereszteződés tulajdonsága, nem azon múlik, merről érkeznek oda és és merre akar rajta áthaladni.) Írjon le egy algoritmust, amivel meg lehet határozni, hogy az autós az egyik adott csúcsnál levő otthonából mely csúcsokba tud autóval úgy eljutni, hogy útja során két nehéz csúcs soha nem jön közvetlenül egymás után. Az algoritmus lépésszáma éllista megadás esetén legyen  $O(n + e)$ , ahol  $n$  a csúcsok és  $e$  az élek száma.
10. Egy  $G$  gráfban pontosan egy él súlya negatív, és nincs a gráfban negatív összsúlyú irányított kör. Adjunk  $O(n^2)$  lépésszámú algoritmust az  $s \in V(G)$  pontból az összes többi pontba vezető legrövidebb utak meghatározására!
11. [ZH: 2007. április 27.] Kutyasétáltatáskor egy parkban egy gazda rögzített, egyenes szakaszokból álló útvonalon halad, aminek töréspontjai  $t_1, \dots, t_n$ , a bejáratot jelölje  $t_0$ , a kijáratot  $t_{n+1}$ . A kutya szabadon szaladgál, de a  $t_i$  pontokban találkozik a gazdájával. A  $t_i$  és  $t_{i+1}$  pontokban való találkozás között a kutya szeretne egy fát is meglátogatni (minden  $i = 0, 1, \dots, n$  esetén legfeljebb egyet-egyét). Legyenek adottak az  $s(t_i, t_{i+1})$  távolságok ( $0 \leq i \leq n$ ), valamint minden fának az összes  $t_i$  ponttól vett távolsága. Tegyük fel, hogy két találkozás között a kutya legfeljebb kétszer akkora távolságot tud megtenni, mint a gazda. Adjunk algoritmust, ami segít a kutyának eldönteni, hogy mikor melyik fát látogassa meg ha a kutya célja, hogy minél több fánál járjon. Az algoritmus lépésszáma legyen  $O(n^2 f + n f^2)$ , ahol  $f$  a parkban levő fák számát jelöli.
12. Legyen  $G = (V, E)$  mátrixszal adott  $n$  pontú, súlyozott élű irányított gráf! Tegyük fel, hogy  $G$  nem tartalmaz negatív összhosszúságú irányított kört, továbbá azt, hogy a  $G$ -beli egyszerű irányított utak legfeljebb 25 élből állnak. Javasoljunk  $O(n^2)$  költségű módszert az  $1 \in V$  pontból az összes további  $v \in V$  pontokba vivő legrövidebb utak hosszának a meghatározására!
13. Adott éllistával egy  $n$  pontú,  $e$  élű  $G$  összefüggő irányítatlan gráf. Adjunk  $O(e)$  költségű algoritmust olyan  $X \subset V(G)$  központi ponthalmaz keresésére, melyre  $|X| \leq n/2$  teljesül! Az  $X \subseteq V(G)$  egy központi ponthalmaz, ha  $G$  minden pontja vagy  $X$ -beli, vagy egyetlen éllel elérhető valamelyik  $X$ -beli pontból.

## 5. Kupacok

1. **[PZH: 2008. május 9.]** A 10 elemű  $A$  tömb első 8 elemére legyen  $A[i] = 2i (1 \leq i \leq 8)$ , és tekintsük ezt, mint egy 8 elemű kupacot. Rajzolja le az ehhez tartozó fát! Hajtsa végre rajta a BESZŰR(3), BESZŰR(1), MINTÖR műveletsort, rajzolja le az egyes műveletek után a kupacot (és jelezze a közben szükséges részlépéseket is)!
2. **[ZH: 2008. március 28.]** Egy orvosi rendelőben a regisztrációnál kell bejelentkezni, ahol az ott dolgozók eldöntik, hogy a beteg az épp rendelő két orvos közül  $A$ -hoz vagy  $B$ -hez kell kerüljön, vagy bármelyikükhöz kerülhet. Ezen kívül, a beutaló ismeretében, a beteghez egy, a sürgősséget kifejező, számot is rendelnek. Amikor valamelyik orvos végzett egy beteggel, akkor azon betegek közül, akiket nem csak a másik orvos láthat el behívja a legnagyobb sürgősségi számút. Tegyük fel, hogy a kiosztott sürgősségi számok egymástól különbözőek. Írjon le egy olyan adatszerkezetet, ami abban az esetben ha  $n$  beteg várakozik, akkor a regisztráción az új beteg beillesztését, illetve az orvosoknak a következő beteg kiválasztását  $O(\log n)$  lépésben lehetővé teszi.

3. **[Vizsga: 2008. május 27.]** Egy kupacba beraktunk egy új  $x$  elemet, majd végrehajtottunk egy MINTÖR műveletet. Mikor fordul elő, hogy végül az eredeti kupacot kapjuk vissza?
4. Adjunk hatékony (*hehe* :)) algoritmust egy kupac tizedik legkisebb elemének meghatározására!
5. Adott egy  $n$  elemet tartalmazó kupac és egy  $k$  kulcs. Keressük meg a kupac  $k$ -nál kisebb elemeit! Ha  $m$  ilyen elem van, akkor az algoritmus  $O(m)$  elemi lépést használhat.
6. Adjunk konstans szorzó erejéig optimális költségű algoritmust az alábbi problémára!  
INPUT: Egy  $A[1 : n]$  tömb, amely eredetileg az  $1, \dots, n$  számokat tartalmazta kupacba rendezve, de öt elem megsérült, és a helyére  $*$  került.  
FELADAT: Találjuk meg a tömb összes olyan kitöltését, ami lehetett az eredeti!
7. Tervezzünk olyan adatszerkezetet, ami egy rendezett halmaz elemeinek tárolására szolgál. A megvalósítandó műveletek: *Felépít*( $n$ ):  $n$  elemből felépíti a struktúrát; *Mintör*, *Maxtör*: a min. illetve max. elem törlése; *Beszúr*( $x$ ): az  $x$  elemet a struktúrába illeszti. Az egyes műveletek költsége ne legyen több, mint *Felépít*:  $O(n)$ ; *Mintör*, *Maxtör*, *Beszúr*:  $O(\log n)$ , ahol  $n$  a tárolt elemek száma.
8. Igazoljuk, hogy egy  $n$  elemből álló kupac felépítése  $\Omega(n)$  összehasonlítást igényel!

## 6. Keresés, rendezés

1. Legalább hány összehasonlítás kell ahhoz, hogy egy  $n$  elemű tömbből egy olyan tagot találjunk meg, ami a tömb 10 legkisebb eleme közé tartozik?
2. **[ZH: 2004. április 8.]** Az  $A[1 : n]$  tömbben levő elemekről tudjuk, hogy  $A[1] \neq A[n]$ . Adjon  $O(\log n)$  összehasonlítást használó algoritmust, amely talál egy olyan  $i$  indexet, hogy  $A[i] \neq A[i + 1]$ !
3. **[ZH: 2004. március 29.]** Az  $A[1 \dots n]$  tömbben egész számokat tárolunk, ugyanaz a szám többször is szerepelhet. Határozzuk meg  $O(n \log n)$  lépésben az összes olyan számot, amelyik egynél többször fordul elő a tömbben.
4. **[Vizsga: 2007. június 19.]** Adott a síkon  $n$  pont, melyek koordinátái  $(a_1, b_1) \dots (a_n, b_n)$ . Olyan  $P = (x, y)$  pontot keresünk a síkon, amire az alábbi összeg minimális.

$$\sum_{i=1}^n (|a_i - x| + |b_i - y|)$$

Adjon algoritmust, ami  $O(n \log n)$  lépésben meghatároz egy ilyen  $P$  pontot.

5. Vázoljunk egy  $O(n)$  időigényű algoritmust (az időkorlát bizonyításával együtt)  $n$  olyan egész számból álló sorozat rendezésére, melynek elemei az
  - (a)  $\{1, \dots, 3n\}$  tartományba esnek!
  - (b)  $\{1, \dots, n^7 - 1\}$  tartományba esnek!



6. [ZH: 2007. április 27.] A valós számokból álló  $a_1^2 \dots a_n^2$  sorozat egy darabig nő, utána csökken. Adjon  $O(n)$  összehasonlítást használó algoritmust, ami rendezzi az  $a_1, \dots, a_n$  sorozatot!
7. Az  $A[1 : n]$  tömbben egy rendezett univerzum  $n$  különböző eleme volt, nagyság szerint növekvő sorrendben. Valaki időközben megkeverte a tömb elemeit, de csak annyira, hogy minden egyes elem új helye az eredetitől legfeljebb 5 távolságra esik. Adjunk  $O(n)$  futásidejű algoritmust az eredeti állapot helyreállítására!
8. [ZH: 2010. április 19.] Az  $A$  tömbben  $n$  különböző számot tárolunk. Tudjuk, hogy  $A[1] > A[2]$  és  $A[n-1] < A[n]$ . Adjunk algoritmust, mely  $O(\log n)$  összehasonlítással megtalál a tömbben egy lokális minimumot (ha van), azaz egy olyan  $1 \leq i \leq n$  indexet, hogy  $A[i]$  tömbbeli szomszédai nagyobbak, mint  $A[i]$ .
9. A (növekvően) rendezett  $A[1 : n]$  tömb  $k$  elemét valaki megváltoztatta. A változtatások helyeit nem ismerjük. Javasoljunk  $O(n + k \log k)$  költségű algoritmust az így módosított tömb rendezésére!
10. Legyen adott egy egészekből álló  $A[1 : n]$  tömb valamint egy  $b$  egész szám. Szeretnénk hatékonyan eldönteni, hogy van-e két olyan  $i, j \in \{1, \dots, n\}$  index, melyekre  $A[i] + A[j] = b$ . Oldjuk meg ezt a feladatot  $O(n \log n)$  időben!
11. [Vizsga: 2009. június 11.] Adott a számegyenesen  $n$  intervallum,  $[a_1, b_1], \dots, [a_n, b_n]$ . Azt akarjuk tudni, hogy együtt milyen hosszú részt fednek le a számegyenesből (azaz, hogy mennyi az  $\cup_{i=1}^n [a_i, b_i]$  összhossza). Adjunk  $O(n \log n)$  lépéses algoritmust ennek a hosszának a meghatározására!
12. Adottak a sík egész koordinátájú  $P_1 = (x_1, y_1), \dots, P_n = (x_n, y_n)$  koordinátájú pontjai. Javasoljunk  $O(n)$  költségű módszert olyan  $P_i \neq P_j$  pontok kiválasztására, amelyekén átmenő egyenes által meghatározott félsíkok közül az egyik tartalmazza az összes pontot!
13. [Vizsga: 2004. június 10.] Az  $n$  méretű (nem feltétlenül rendezett)  $A$  tömb elemei különböző pozitív egész számok. Adjunk algoritmust, amely meghatároz egy  $1 \leq k \leq n$  számot és kiválaszt  $k$  különböző elemet az  $A$  tömbből úgy, hogy a kiválasztott elemek összege nem több, mint  $k^3$ . Ha nincs ilyen  $k$ , akkor az algoritmus jelezze ezt a tényt! Az algoritmus lépésszáma legyen  $O(n \log n)$ ! (Két szám összehasonlítása, összehasonlítása vagy szorzása egy lépésnek számít.)
14. [Vizsga: 2004. június 3.] A  $2^k - 1$  elemű  $A$  tömb elemei mind különbözőek és növekvő sorrendben vannak. Minden elemet egy  $k$  hosszú bitsorozat ír le, tehát tekinthetjük úgy, hogy a  $0, 1, 2, \dots, 2^k - 1$  számokat tároljuk egy kivételével. A feladat ennek a hiányzó számnak a megkeresése. Ehhez egy lépésben valamelyik elem egy bitjére kérdezhetünk rá: a  $BIT(i, j)$  eljárás az  $A[i]$  elem  $j$ -edik bitjét mondja meg. Adjunk olyan algoritmust, amely a  $BIT$  eljárás  $O(k)$ -szori hívásával megtalálja a hiányzó számot (bitsorozatot).
15. Adott egy dobozban  $n$  különböző méretű anyacsavar, valamint egy másik dobozban a hozzájuk illő apacsavarok. Kizárólag a következő összehasonlítási lehetőségünk van: egy apacsavarhoz hozzápróbálunk egy anyacsavart. A próbának háromféle kimenete lehet:  $\text{apa} < \text{anya}$ ,  $\text{apa} = \text{anya}$ ,  $\text{apa} > \text{anya}$ , annak megfelelően, hogy az apacsavar külső átmérője hogyan viszonyul az anyacsavar belső átmérőjéhez. Szeretnénk minden anyacsavarhoz megtalálni a megfelelő apacsavart. Adjunk erre a feladatra átlagosan  $O(n \log n)$  összehasonlítást felhasználó módszert!
16. A 4 elemű  $I$  abc felett adott két szó:  $x = x_1 x_2 \dots x_n$  és  $y = y_1 y_2 \dots y_k$ , ahol  $1 \leq k \leq n$  és  $\forall i, j : x_i, y_j \in I$ . Keressük az  $x$  szóban az olyan részsavakat, amelyek anagrammái  $y$ -nak, azaz az olyan  $i$  indexeket, hogy az  $x_i, x_{i+1}, \dots, x_{i+k-1}$  betűk megfelelő sorrendbe rakva az  $y$  szót adják. Adjunk algoritmust, ami  $x$ -ben az összes ilyen  $i$  helyet  $O(n)$  lépésben meghatározza!
17. [Vizsga: 2003. május 30.] Adott összesen  $2n$  különböző szám két  $n$  elemű halmazban,  $A = \{a_1, \dots, a_n\}$  és  $B = \{b_1, \dots, b_n\}$ . Azt szeretnénk eldönteni minimális számú összehasonlítással, hogy a  $2n$  szám közül a legnagyobb az  $A$  vagy a  $B$  halmazban van-e. (Azaz nem kell feltétlenül meghatározni, melyik elem a legnagyobb, csak azt, hogy melyik halmazba tartozik.) Mutassa meg, hogy ehhez a feladathoz is legalább annyi összehasonlítás kell, amennyi  $2n$  elem közül a maximális meghatározásához szükséges.
18. [ZH: 2002. április 8.] Adottak a  $c_1, c_2, \dots, c_n$  különböző egész számok. Ezeket szeretnénk nagyság szerint rendezni növekvő, vagy csökkenő sorrendbe úgy, hogy a szokásos összehasonlítás helyett, most a következő kérdéseket lehet feltenni: *Három kiválasztott elem közül melyik a középső?* Bizonyítsuk be, hogy a leghatékonyabb algoritmus  $\Theta(n \log_2 n)$  összehasonlítást használ!

## 7. Keresőfák, p-fák, 2-3 fák

- [Vizsga: 2007. június 5.]** Adott egy  $n$  csúcsú és egy  $k$  csúcsú piros-fekete fa. A két fában tárolt összes elemből  $O(n+k)$  lépésben készítsen rendezett tömböt.
  - [ZH: 2004. március 29.]** Egy bináris keresőfában csupa különböző egész számot tárolunk. Lehetséges-e, hogy egy  $KERES(x)$  hívás során a keresési út mentén a 20, 18, 3, 15, 5, 8, 9 kulcsokat látjuk ebben a sorrendben? Ha nem lehetséges, indokolja meg miért nem, ha pedig lehetséges, határozza meg az összes olyan  $x$  egész számot, amire ez megtörténhet.
  - [ZH: 2009. április 24.]** Egy bináris fa csúcsai 0 és 9 közötti egész számokkal vannak megcímkézve. Az inorder bejárás során a címkék sorrendje: 9, 3, 1, 0, 4, 2, 7, 6, 8, 5, a postorder bejárásnál pedig 9, 1, 4, 0, 3,  $x$ , 7, 5,  $y$ , 2. Mi lehet az  $x$  és mi az  $y$ ?
  - [ZH: 2009. április 24.]** Egy 2-3 fa gyökerének három fia van, a benne szereplő két érték 40 és 50. Mennyi lehet a tárolt elemek minimális, illetve maximális száma, ha tudjuk, hogy csak pozitív egész számokat tárol a fa?
  - [Vizsga: 2009. június 17.]** Az MSc-re jelentkezőknek a felvételit alkotó 3 témakör mindegyikéből lesz egy írásbeli pontszámuk  $(P_1, P_2, P_3)$ , és keletkezik egy felvételi pontszámuk is  $(FP)$ . Tegyük fel, hogy a  $P_i$ -k 1 és 30 közötti egészek, míg az  $FP$  tetszőleges pozitív egész szám lehet. Adjon meg egy olyan adatszerkezetet, amivel a következő műveletek az adott időben végrehajthatóak ( $n$  a jelentkezők számát jelöli!)  
BESZŰR( $P_1, P_2, P_3, FP$ ): az adott pontszámok beillesztése –  $O(\log n)$   
KERES( $p$ ): a pontosan  $p$  felvételi ponttal ( $FP = p$ ) rendelkező jelentkezők számát határozza meg –  $O(\log n)$   
KORLÁT( $i, q$ ): az írásbelin az  $i$ -edik témakörből legalább  $q$  pontot elért jelentkezők számát határozza meg –  $O(1)$
  - Az  $[1, 178]$  intervallum összes egészei egy 2-3 fában helyezkednek el. Tudjuk, hogy a gyökérben két kulcs van, és ezek közül az első 17. Mi lehet a második? Miért?
- 
- Lehet-e tetszőleges (adott) kulcshalmaz esetén olyan piros-fekete fát építeni, hogy az azonos szinten lévő elemek azonos színűek legyenek?
    - Van-e olyan piros-fekete fa, ami nem így néz ki?
  - [ZH: 2007. április 27.]** Egy piros-fekete fában lehetséges-e, hogy a piros-fekete tulajdonság megsértése nélkül
    - néhány fekete csúcsot átváltoztathatunk pirosra?
    - valamelyik (csak egy) fekete csúcsot átváltoztathatjuk pirosra?(Mást nem változtatunk a fán.)
  - Az  $S_1$  és  $S_2$  kulcshalmazokat kiegészített 2-3 fában tároljuk. Ezek az eredeti 2-3 fától annyiban különböznek csak, hogy minden csúcsban nyilván van tartva az onnan induló részfa magassága. Tudjuk továbbá, hogy az  $S_1$ -beli kulcsok mind kisebbek, mint az  $S_2$ -beliek. Javasoljunk hatékony algoritmust a két fa egyesítésére!
  - [ZH: 2009. június 4.]** Mutassa meg, hogyan kell a 2-3 fa BESZŰR eljárását módosítani, ha a fa minden  $v$  csúcsában a szokásos dolgokon kívül azt is nyilvántartjuk, hogy hány levél van a  $v$  gyökerű részfában!
  - [PZH: 2008. május 9.]** Válassza a 2-3 fának (és műveleteinek) egy olyan módosítását, amiben továbbra is van KERES, BESZŰR, TÖRÖL, MIN, MAX művelet, és ezeken kívül van még RANG és K-ADIK művelet is, ahol RANG( $x$ ) azt adja vissza, hogy a tárolt elemek között az  $x$  a rendezés szerint hányadik elem, a K-ADIK( $i$ ) pedig, hogy a rendezés szerint a tárolt elemek közül melyik az  $i$ -edik. A módosítás során a felsorolt szokásos műveletek lépésszámának nagyságrendje ne változzon, és mindkét új művelet lépésszáma legyen  $O(\log n)$ , ahol  $n$  a tárolt elemek száma.

- 12. [Vizsga: 2003. március 31.]** Tervezzon adatstruktúrát a következő feltételekkel. Természetes számokat kell tárolni, egy szám többször is szerepelhet. A szükséges műveletek:  
 BESZŰR( $i$ ):  $i$  egy újabb példányát tároljuk  
 TÖRÖL( $i$ ):  $i$  egy példányát töröljük  
 MINDTÖRÖL( $i$ ):  $i$  összes példányát töröljük  
 DARAB( $i$ ): visszaadja, hogy hány példány van  $i$ -ből  
 ELEM( $K$ ): megmondja, a nagyság szerinti rendezésben a  $K$ -adik elem értékét.  
 Az adatstruktúra legyen olyan, hogy ha  $m$ -féle elemet tárolunk, akkor mindegyik művelet lépésigénye  $O(\log m)$ .  
 (Például ha a tárolt elemek 1, 1, 3, 3, 3, 8, akkor DARAB(1) = 2, ELEM(4) = 3 és  $m = 3$ .)
- 13. [ZH: 2011. április 19.]** Adott  $2^k - 1$  különböző szám, mindegyik az  $\{1, 2, \dots, n\}$  halmazból, ezekből kell egy  $O(k)$  mélységű bináris keresőfát készíteni. Adjon olyan algoritmust, amely ezt  $O(n)$  lépésben megcsinálja!
- 14. [Vizsga: 2009. május 28.]** Adott egy  $n$  csúcsú bináris keresőfa. Ennek minden  $v$  csúcsára meg akarjuk határozni, hogy a  $v$  gyökerű részében hány darab  $v$ -nél kisebb elem van tárolva. Adjon algoritmust, ami ezt a feladatot  $O(n)$  lépésben megoldja!
- 15. [ZH: 2009. április 24.]** Egy piros-feketében jelölje  $x$  és  $y$  a gyökér két fiát. Tudjuk, hogy  $fm(x) = fm(y)$ , de az  $x$  csúcs két gyerekének különbözik a fekete magassága. Milyen színű lehet az  $y$  csúcs?
- 16. [ZH: 2003. március 31.]** Egy 2-3 fába egymás után 1000 új elemet illesztettünk be. Mutassa meg, hogy ha ennek során egyszer sem kellett csúcsot szétvágni, akkor a beillesztések sorozata előtt már legalább 2000 elemet tároltunk a fában.
- 17.** Egy bináris keresőfa csúcsait egy, a gyökértől egy levélig menő út szerint három osztályba soroljuk:  $B$  az úttól balra levő,  $U$  az útra eső,  $J$  pedig az úttól jobbra levő csúcsok halmazát jelöli. Igaz-e mindig, hogy minden  $B$ -beli csúcs kulcsa kisebb tetszőleges  $U$ -beli csúcs kulcsánál, és minden  $U$ -beli csúcs kulcsa kisebb tetszőleges  $J$ -beli csúcs kulcsánál?
- 18.** Adott  $n$  pont a síkon, melyek páronként mindkét koordinátájukban különböznek. Bizonyítsuk be, hogy pontosan egy bináris fa létezik, melynek csúcsai az adott  $n$  pont, és az első koordináta szerint a keresőfa tulajdonsággal, a második szerint a kupac tulajdonsággal rendelkezik! (A kupac tulajdonságba most nem értjük bele, hogy a fa teljes bináris fa legyen.)
- 19. [ZH: 2004. március 29.]** Egy kezdetben üres 2-3 fába az  $1, 2, \dots, n$  számokat szúrtuk be ebben a sorrendben. Bizonyítsa be, hogy a keletkezett fában a harmadfokú csúcsok száma  $O(\log n)$ .
- 20.** Adott egy  $n = 2^k - 1$  pontú teljes bináris keresőfa. A fában tárolt elemek egészek az  $I = [1, 2^k]$  intervallumból, és egy szám legfeljebb egyszer fordul elő a fában. Utóbbi feltétel szerint pontosan egy olyan  $i$  egész szám van 1 és  $2^k$  között, amely nincs a fában. Adjunk  $O(\log n)$  lépésszámú algoritmust  $i$  meghatározására!
- 21.** Egy fában az  $x$  csúcs *súlya* legyen  $x$  leszármazottainak száma. Egy bináris fát szigorúan binárisnak mondunk, ha a levelek kivételével minden csúcsonk pontosan 2 fia van. Tegyük fel, hogy egy szigorúan bináris fa minden  $x$  csúcsára fennáll, hogy

$$\frac{1}{2} < \frac{\text{súly}(\text{bal}(x))}{\text{súly}(\text{jobb}(x))} < 2.$$

Bizonyítsuk be, hogy ez csakis egy teljes fa lehet, azaz ha  $k$  szintje van, akkor a csúcsok száma  $2^k - 1$ .  
 (Ez nem kifejezetten keresőfázós feladat, de úgy általában érdekes.)

## 8. Hash

1. Nyitott címzéssel hash-eltünk egy 11 elemű táblába a  $h(k) = k \pmod{11}$  hash-függvény segítségével. A következő kulcsok érkeztek (a megadott sorrendben): 10, 22, 31, 4, 15, 28, 17, 88, 59. Adjuk meg a tábla végső állapotát a következő két próbamódszerre:
- lineáris próba;
  - kvadratis maradék próba!

2. [Vizsga: 2010. január 21.] Kettős hashelést használva szűrje be egy kezdetben üres,  $M = 11$  méretű táblába a következő kulcsokat (ebben a sorrendben): 26, 3, 48, 14, 15, 7. A használt hash függvény legyen

$$h(k) = (k \pmod{M}),$$

a próbasorozat hash függvénye pedig

$$h'(k) = 1 + (k \pmod{M - 5}).$$

Minden beszúrás után rajzolja le a tábla pillanatnyi állapotát!

3. [ZH: 2010. április 19.] Egy  $M$  méretű hash-táblába  $n < M$  elemet raktunk be nyitott címzéssel, kvadratikus próbával, a  $h(x)$  hash-függvényt használva. Ennek során  $t_1$  ütközés történt (ennyiszer kellett tovább próbálkoznunk, egy elem beszúrása során több ütközés is lehetett). Ugyanezt az  $n$  elemet ugyanabban a sorrendben beszúrtuk egy  $M^2$  méretű hash-táblába is, de most lineáris próbával,  $M \cdot h(x) + 1$  hash-függvénnyel, ekkor  $t_2$  ütközés történt. Igazolja, hogy  $t_2 \leq t_1$ .

4. [Vizsga: 2011. június 9.] Az alábbi hash-táblát az üresből kiindulva beszúrások sorozatával kaptuk. Határozzuk meg a beszúrások összes lehetséges sorrendjét, ha a hash-függvény a  $h(x) = 3x \pmod{10}$  volt és a nyitott címzésű hash-elést lineáris próbával alkalmaztuk!

0	1	2	3	4	5	6	7	8	9
					5	19	3	33	23

5. A  $T[0 : M - 1]$  táblában rekordokat tárolunk nyitott címzésű hashelt szervezéssel. Az ütközések feloldására lineáris próbát alkalmazunk. Tegyük fel, hogy a tábla használata során egy hibás törlés történt: egy cellából kitöröltünk egy rekordot a törlés-bit beállítása nélkül.
- Igaz-e, hogy a hibás törlés helye mindig megtalálható?
  - Adjunk hatékony (lineáris) algoritmust a tábla megjavítására! (Módosítsuk a táblát úgy, hogy megszűnjenek a hibás törlés negatív következményei!)
6. A  $T[0 : M - 1]$  táblában  $2n$  elemet helyeztünk el az első  $3n$  helyen egy ismeretlen hash-függvény segítségével. A táblában minden  $3i$  indexű hely üresen maradt ( $0 \leq i \leq n$ ). Legfeljebb hány ütközés lehetett, ha az ütközések feloldására
- lineáris próbát használtunk?
  - kvadratikus próbát használtunk?
7. [ZH: 2005. április 8.] Egy  $m$  méretű hash-táblában már van néhány elem. Adjon  $O(m)$  lépésszámú algoritmust, amely meghatározza, hogy egy újabb elem lineáris próbával történő beszúrásakor maximum hány ütközés történhet.
8. Mi a baja a  $h(k) = k^2 \pmod{7}$  hash-függvénynek, ha a tábla 7 méretű?
9. Mutassuk meg, hogy nyitott címzéses hashelés és lineáris próba esetén már két kulcshoz tartozó hash-függvényérték megegyezése is okozhat tetszőlegesen nagy méretű csomósodást!
10. [Vizsga: 2008. június 3.] Az 1 és 91 közötti összes 3-mal osztható egész számot valamilyen sorrendben egy  $M$  méretű hash-táblába raktuk a  $h(x) = x \pmod{M}$  hash-függvény segítségével, lineáris próbával. Ennek során hány ütközés fordulhatott elő, ha  $M = 35$ , illetve ha  $M = 36$ ?
11. [Vizsga: 2005. május 26.] A kezdetben üres  $M$  méretű hash-táblába sorban beraktuk a  $k_1, k_2, \dots, k_n$  kulcsokat a  $h(x) \equiv x \pmod{M}$  hash-függvénnyel, lineáris próbával. Jelölje  $t_1$  a keletkezett táblában az egymás melletti foglalt mezők maximális számát. Amikor ugyanezt a  $k_1, k_2, \dots, k_n$  sorozatot ugyanabban a sorrendben egy üres  $2M$  méretű táblába rakjuk be a  $h(x) \equiv x \pmod{2M}$  hash-függvénnyel, lineáris próbával, akkor a kapott táblában legyen  $t_2$  az egymás melletti foglalt mezők maximális száma.
- Igazolja, hogy  $t_2 \leq t_1$
  - Igaz-e, hogy  $t_1 \leq 2t_2$ ?

## 9. Mélységi bejárás, DAG, valamint alkalmazásaik

1. A 6 pontú irányított  $G$  gráf csúcsait jelölje  $x, y, z, u, v, w$ . A gráf egy mélységi bejárásánál a mélységi, ill. a befejezési számok a következők:  $x : 1, 6; y : 2, 4; z : 6, 5; u : 3, 3; v : 4, 1; w : 5, 2$ . Adjuk meg a bejáráshoz tartozó mélységi feszítőfa éleit! Rekonstruálható-e  $G$  az előző számok ismeretében?
2. [ZH: 2008. március 28.] Egy  $n \times n$  méretű táblázat minden eleme egy egész szám. A táblázat bal alsó sarkából akarunk eljutni a jobb felső sarkába úgy, hogy egy lépésben a táblázatban vagy felfelé vagy jobbra egyet lépünk. Azt szeretnénk, hogy a lépegetés során látott elemek növekvő sorrendben kövessék egymást. Egy ilyen út értéke a benne szereplő számok összege. Adjon  $O(n^2)$  futási idejű algoritmust, ami meghatározza, hogy az adott táblázatban a szabályok szerinti utak értékei között mekkora a legnagyobb! (*Persze, dinprog is kézenfekvő, de most DAG-gal!*)
3. [ZH: 2011. március 28.] Van  $b$  darab borítékunk, az  $i$ -ediknek a hossza  $h_i$ , a magassága  $m_i$ . Az  $i$ -edik borítékba akkor tudjuk berakni a  $j$ -edik borítékot, ha  $h_j < h_i$  és  $m_j < m_i$  is teljesül (nem forgatjuk és nem is hajtogatjuk a borítékokat). Célunk, hogy minél hosszabb olyan láncot alakítsunk ki, hogy az  $i$ -edikben benne van a  $j$ -edik, abban a  $k$ -adik, stb.  
Legyen adott egy  $L > 0$  egész és a  $h_i$  és  $m_i$  számok. Hogyan lehet  $O(b^2)$  lépésben eldönteni, hogy kialakítható-e a borítékokból egy  $L$  hosszú lánc?

4. [Vizsga: 2008. május 27.] Éllistával adott egy  $n$  pontú  $e$  élű irányított gráf. Azt szeretnénk tudni, hogy van-e benne olyan minden pontot tartalmazó részgráf, ami egy, a gyökerétől a levelek felé irányított fa. Adjon  $O(ne + n^2)$  lépésszámú algoritmust, ami ha van, talál egy ilyen részgráfot.
5. [ZH: 2007. április 27.] Az  $n \times n$  méretű tábla minden mezőjére egy pozitív egész szám van írva, az  $i$ -edik sorának  $j$ -edik elemére  $A[i, j]$ , ahol  $0 \leq i, j < n$ . Feladat, hogy az első oszlopból eljussunk az utolsó oszlopba úgy, hogy egy lépésben mindig a következő oszlopba lépünk, és azon belül, ha az  $i$ -edik sorban voltunk, akkor a következő lépésben vagy az  $(i - 1) \pmod n$ , vagy az  $i$ , vagy az  $(i + 1) \pmod n$  számú sorba kerülhetünk. Adjon  $O(n^2)$  lépésszámú algoritmust, ami meghatározza, hogy az első oszlop melyik eleméből induljunk, ha azt akarjuk, hogy a bejárt mezőkön lévő számok összege minimális legyen (az utolsó oszlop bármelyik mezője lehet az utolsó olyan mező, amire rálépünk).
6. [Vizsga: 2010. június 3.] Egy falutörténet írója  $n$  korábbi lakosról gyűjtött információkat. A kérdésekre kapott válaszok a következő típusúak voltak:

- $S_i$  személy meghalt  $S_j$  születése előtt;
- $S_i$  személy élete során született  $S_j$ ;
- $S_i$  személy korábban született, mint  $S_j$ ;
- $S_i$  korábban halt meg, mint  $S_j$ .

Egy  $S_i, S_j$  párra nem biztos, hogy szerepel minden választípus, és olyan pár is lehet, amely egyetlen válaszban sem szerepel együtt. Mivel az emberek időnként rosszul emlékeznek, nem biztos, hogy minden kapott információ helyes. Adjon algoritmust, amivel  $k$  db fenti típusú válaszról  $O(n + k)$  lépésben eldönthető, hogy van-e közöttük ellentmondás.

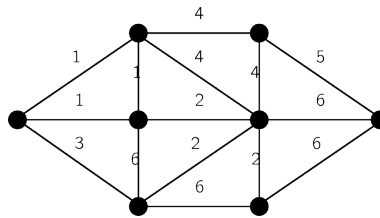
7. [ZH: 2005. április 8.] Cirkuszi akrobaták egymás vállára állva minél nagyobb tornyot szeretnének létrehozni (a toronyban minden szinten csak egy akrobata lesz). Esztétikai és gyakorlati szempontok miatt egy ember vállára csak egy olyan állhat, aki nála alacsonyabb és könnyebb is. A cirkuszban  $n$  akrobata van, adott mindegyikük magassága és súlya. Adjon algoritmust, amely  $O(n^2)$  lépésben megadja a lehetséges legtöbb emberből álló torony összeállítását.
8. [ZH: 2007. április 27.] Tekintsük az olyan  $G$  irányított gráfokat, amelyekben ha eltekintünk az élek irányításától, akkor a kapott irányítatlan  $G'$  gráf összefüggő. A  $G$  gráf egy mélységi bejárásánál maximálisan hány olyan csúcs lehet, amelyre a mélységi és a befejezési szám megegyezik?
9. Adjunk algoritmust, mely egy éllistával megadott irányítatlan gráfban vagy talál egy kört, vagy igazolja a gráf körmentességét  $O(|V|)$  időben (függetlenül attól, hogy  $|E|$  akár sokkal nagyobb is lehet, mint  $|V|$ )!
10. [Vizsga: 2007. június 12.] Egy számítógéphálózatban  $n$  számítógép van. Minden olyan eseményt, hogy az  $i$ -edik gép üzenetet küld a  $j$ -ediknek  $(i, j, t)$  formában feljegyezzük, ahol a  $t$  egész szám az üzenet küldésének időpontját jelöli. Ugyanabban a  $t$  időpontban egy gép több gépnek is küldhet üzenetet. Ha a  $t$  időpontban az  $i$ -edik gép vírusos volt, akkor egy  $(i, j, t)$  üzenet hatására a  $j$ -edik gép megfertőződhet, ami azt jelenti, hogy a  $t + 1$  időponttól kezdve már a  $j$ -edik gép is vírusos lehet.

Legyen adott az  $(i, j, t)$  hármasoknak egy  $m$  hosszú listája, valamint  $x, y$  és  $t_0 < t_1$  egész számok. Azt kell eldöntenünk, hogy ha az  $x$ -edik gép a  $t_0$  időpontban vírusos volt, akkor lehet-e emiatt az  $y$ -edik gép a  $t_1$  időpontban vírusos. Adjon algoritmust, ami ezt a kérdést  $O((t_1 - t_0)n + m)$  lépés után megválaszolja.

11. **[Vizsga: 2007. június 19.]** Egy előre rögzített útvonalon úgy indulunk el, hogy az autó  $L$  literes tankja tele van. Úticélunkhoz úgy akarunk eljutni, hogy legalább egy fél tanknyi benzin maradjon az autóban. Tudjuk, hogy az utunkba eső  $n$  benzinkút közül melyikben mennyibe kerül a benzin, továbbá, hogy két szomszédos benzinkút között, valamint a kiindulóponttól az első benzinkútig, illetve az utolsó benzinkúttól a célunkig mennyi benzint fogyaszt az autó. Az egyszerűség kedvéért ha megállunk egy benzinkútnál, akkor mindig tele tankolunk. Adjon algoritmust, ami  $O(Ln^2)$  lépésben megmondja, hogy hol álljunk meg tankolni ha azt akarjuk, hogy utunk során a benzinköltség minimális legyen. *(Kiegészítés: feltesszük, hogy  $L$  egész, továbbá a fogyasztás mindig egész liter.*
12. A  $G(V, E)$  összefüggő, irányított gráf minden éle az  $1, 2, \dots, k$  számok valamelyikével van súlyozva. Egy út értéke legyen az úton található élek súlyainak maximuma. Adjunk  $O(|E| \log k)$  futásidőjű algoritmust az adott  $x, y \in V$  csúcsok közti legkisebb súlyú út értékének meghatározására!
13. **[Vizsga: 2003. május 30.]** Éllistával adott egy  $G$  gráf, melynek  $n$  csúcsa és  $e$  éle van. A gráf minden csúcsához hozzá van rendelve egy  $1$  és  $k$  közötti egész szám (címke). Találjunk (ha létezik) olyan *tarka* utat a gráfban, amelyben minden  $1 \leq i \leq k$  címke pontosan egyszer fordul elő. Az algoritmus lépésszáma legyen  $O(k!(e + n))$ .
14. Bizonyítsuk be, hogy minden  $G = (V, E)$  irányított gráf felbontható két DAG-ra; pontosabban az élhalmazának van olyan  $E_1, E_2$  partíciója ( $E = E_1 \cup E_2$  és  $E_1 \cap E_2 = \emptyset$ ), hogy a  $G_1 = (V, E_1)$  és a  $G_2 = (V, E_2)$  gráfok DAG-ok!

## 10. Minimális költségű feszítőfák

1. Mennyi az alábbi gráfban a minimális feszítőfa súlya? (Gyakorlásképpen mindhárom módszerrel – Prim, Kruskal, Borůvka – csináljuk meg a feszítőfa-keresést!)



2. Egy teljes gráf pontthalmaza  $x_1, x_2, \dots, x_k, y_1, y_2, \dots, y_l$ . Az  $(x_i, x_j)$  élek költsége (súlya) 1, az  $(y_i, y_j)$  éleké 2, az  $(x_i, y_j)$  éleké 3. Mennyibe kerül a legolcsóbb feszítőfa?
3. **[Vizsga: 2005. június 23.]** Irányítatlan gráf tárolására adjon meg egy adatszerkezetet az alábbi műveletekkel:  
 ÚJCSŰCS( $v$ ): a gráfhoz hozzáad egy új csúcsot;  
 ÚJÉL( $u, v$ ): a már létező  $u$  és  $v$  csúcsok közé felvesz egy élet;  
 VANÚT( $u, v$ ): *igen* értéket ad vissza, ha vezet az  $u$  és  $v$  csúcsok között út, egyébként pedig *nem* értéket.  
 Ha a tárolt gráfnak  $n$  csúcsa van, akkor mindhárom művelet lépésszáma legyen  $O(\log n)$ .
4. **[Vizsga: 2010. május 27.]** Adott egy  $G = (V, E)$  irányítatlan, összefüggő, súlyozott gráf az éllistájával valamint egy  $f \in E$  él. Tegyük fel, hogy a gráfban minden él súlya különböző. Adjon  $O(|V| + |E|)$  lépésszámú algoritmust annak eldöntésére, hogy van-e olyan minimális feszítőfa  $G$ -ben, amely tartalmazza az  $f$  élet!

5. **[Vizsga: 2008. június 3.]** Éllistával adott a  $G = (V, E)$  egyszerű, összefüggő gráf. A gráf élei súlyozottak, a súlyfüggvény  $c : E \rightarrow \{-1, 1\}$ . Adjon algoritmust, ami  $G$ -ben  $O(|V| + |E|)$  lépésben meghatározza, hogy mennyi a minimális súlya egy olyan részgráfnak, ami  $G$  minden pontját tartalmazza és összefüggő.
6. **[Vizsga: 2007. június 5.]** Útépítéskor a környéken sok helyen felszedték a járdát. Az építők 1-től  $n$ -ig megszámozták a fontos pontokat (kapualj, útkereszteződés, stb.). A környék állapotát két  $n \times n$  táblázat írja le. A  $J$  táblázatban  $J[i, j] = 1$ , ha az  $i$  és  $j$  pontok az utcán szomszédosak és megmaradt az ezeket összekötő részen a járda, egyébként az érték 0. A  $P$  táblázat az ideiglenesen elhelyezhető pallókat írja le: ha az  $i$  és  $j$  pontok összeköthetők egy pallóval, akkor  $P[i, j]$  ennek a pallónak a költsége. Amennyiben a két pont nem köthető össze egy pallóval, akkor a táblázatban  $*$  szerepel. (Minden palló pontosan két pontot érint.) Szeretnénk biztosítani, hogy mindenholnan mindenholva el tudjunk jutni (hol járdán hol pallón haladva). Az építők célja, hogy úgy válasszák meg a pallók helyét, hogy minél kevesebb pallót kelljen használniuk, és ezen belül a pallók értékeinek összege minimális legyen. Írjon le egy algoritmust, ami  $O(n^2)$  lépésben javasol egy ilyen elhelyezést. (Egy pontra tetszőlegesen sok palló illeszkedhet, és a gyalogosok az egy pontra illeszkedő pallók bármelyikéről bármelyikére át tudnak lépni.)
7. Mátrixával adott egy  $G$  irányítatlan súlyozott gráf. Adott még a  $G$ -nek egy  $F$  minimális súlyú feszítőfája, és az  $F$ -nek egy  $f$  éle. Adjon  $O(n^2)$  lépésszámú algoritmust, ami meghatározza, hogy az  $f$  él súlyát meddig lehet úgy felemelni, hogy az  $F$  a gráf minimális feszítőfája maradjon.
8. **[Vizsga: 2008. május 27.]** Éllistával adott egy egyszerű, összefüggő, súlyozott irányítatlan  $G = (V, E)$  gráf amiben nincs két egyforma súlyú él. Adjon  $O(|V| \cdot |E|)$  lépésszámú algoritmust, ami megadja a gráfban a második legkisebb súlyú feszítőfát.
9. Legyen adva egy (egyszerű, irányítatlan, összefüggő)  $n$  csúcsú  $G$  gráf éllistával, az élek súlyozásával együtt. Tegyük fel, hogy a  $G$ -ből a  $v_1$  csúcs, valamint a  $v_1$ -re illeszkedő élek elhagyásával keletkező  $G'$  gráf még mindig összefüggő, és adott a  $G'$  egy minimális költségű feszítőfája. Adjunk  $O(n \log n)$  futási idejű algoritmust a  $G$  gráf egy minimális költségű feszítőfájának elkészítésére!
10. Hány éle van az  $n$  pontú egyszerű összefüggő gráfnak, ha pontosan 3 különböző feszítőfája van?
11. Éllistával adott egy összefüggő, egyszerű, irányítatlan,  $n$  csúcsú,  $e$  élű  $G$  gráf csupa különböző élsúlyal. Adjunk egy olyan  $O(e)$  költségű algoritmust, ami a  $G$  gráf egy minimális feszítőfájának legalább  $\frac{2}{3}n$  élét előállítja! (Azaz egy olyan élhalmazt keresünk, ami biztosan része egy minimális költségű feszítőfának.)
12. Bizonyítsuk be, hogy a piros-kék algoritmus akkor is helyesen működik, ha egy feszítőfa költségét az élsúlyok összege helyett az élsúlyok maximumával definiáljuk!

## 11. Bonyolultságelmélet

1. Lássuk be, hogy az alábbi problémák NP-beliek! Melyekről tudjuk megmondani, hogy coNP-ben vannak? Melyekről, hogy P-ben?
  - (a)  $\pi_1 = \{(G, k) \mid G \text{ gráf kiszínezhető } k \text{ színnel}\}$
  - (b)  $\pi_2 = \{G \mid G \text{ irányítatlan gráfban van Euler-kör}\}$
  - (c)  $\pi_3 = \{(G, k) \mid G \text{ irányítatlan gráfban van } k \text{ független pont}\}$
2. Bizonyítsuk be, hogy a következő probléma P-beli:

$$\pi = \left\{ G \mid \begin{array}{l} G \text{ gráf kiszínezhető a piros, zöld, sárga, kék színekkel úgy,} \\ \text{hogy pontosan egy csúcs legyen piros és pontosan két csúcs legyen kék} \end{array} \right\}$$

3. Tegyük fel, hogy van egy olyan  $F$  eljárásunk, ami egy input  $G$  gráfra és  $k$  számra 1 lépés alatt megmondja, hogy van-e  $G$ -ben legalább  $k$  méretű független ponthalmaz. Tervezzünk olyan algoritmust, ami polinomidőben
  - (a) meghatározza  $\alpha(G)$ -t!
  - (b) talál egy  $\alpha(G)$  méretű független ponthalmazt!

4. Adjunk Karp-redukciót a 3-SZÍN nyelvről a 4-SZÍN nyelvre!
5. **[Vizsga: 2007. május 29.]** A  $G$  irányítatlan gráf minden  $x$  pontjához tartozik egy  $s(x)$  súly. Célunk, hogy olyan feszítőfát találjunk a gráfban, amiben a levelekhez tartozó súlyok összege minimális. Fogalmazza meg a feladathoz tartozó nyelvet és vagy lássa be róla, hogy  $P$ -ben van vagy azt, hogy  $NP$ -teljes.
6. Bizonyítsuk be a következő problémáról, hogy  $NP$ -teljes, vagy azt, hogy  $P$ -beli!

$$L = \{G \mid G \text{ irányítatlan teljes gráf, amiben van Hamilton-kör}\}$$

7. **[Vizsga: 2008. június 10.]** Tegyük fel, hogy  $P \neq NP$ . Az alábbi feltételek közül melyikből következik és melyikből nem következik hogy az  $X$  eldöntési probléma nem  $P$ -beli?
- (a) Egy  $NP$ -teljes  $Y$  problémára  $X$  Karp-redukálható.
- (b) Egy  $NP$ -teljes  $Y$  probléma Karp-redukálható  $X$ -re.
- (c) az  $X$  probléma  $NP$ -beli.

8. Lássuk be, hogy az alábbi problémák  $NP$ -beliek! Melyekről tudjuk megmondani, hogy  $coNP$ -ben vannak? Melyekről, hogy  $P$ -ben?

- (a)  $\pi_1 = \{(G, k) \mid G \text{ páros gráfban van teljes párosítás}\}$
- (b)  $\pi_2 = \{(G, k) \mid G \text{ páros gráfban van } k \text{ élből álló párosítás}\}$
- (c)  $\pi_3 = \{G \mid G \text{ irányítatlan gráf, van benne pontosan } 100 \text{ élből álló kör}\}$
- (d)  $\pi_4 = \{(G, k) \mid G \text{ irányítatlan gráf, van benne legalább } k \text{ élből álló kör}\}$
- (e)  $\pi_5 = \left\{ (s_1, s_2, \dots, s_n, b) \mid \forall i \ s_i, b \in \mathbb{Z}^+; \exists j_1, \dots, j_k \ (1 \leq k \leq n) : \sum_{l=1}^k s_{j_l} = b \right\}$

9. A  $G$  irányítatlan gráf minden  $x$  pontjához tartozik egy  $s(x)$  súly. Célunk, hogy olyan feszítőfát találjunk a gráfban, amiben a levelekhez tartozó súlyok összege minimális. Fogalmazzuk meg a feladathoz tartozó eldöntési problémát, és bizonyítsuk be, hogy  $NP$ -beli!
10. Tegyük fel, hogy van egy algoritmusunk, ami polinom időben megmondja, hogy adott  $G$  gráf kiszínezhető-e legfeljebb  $k$  db színnel! (Vagyis input:  $G$  és  $k$ ; output: igen/nem). Hogy tudnánk ennek segítségével polinom időben meghatározni  $\chi(G)$ -t?
11. Tegyük fel, hogy van egy algoritmusunk, ami polinom időben megmondja, hogy adott  $G$  gráf kiszínezhető-e legfeljebb  $k$  db színnel! A fentiek értelmében azt is megtudhatjuk polinom időben, hogy mennyi  $\chi(G)$ . Hogyan tudnánk kiszínezni polinom időben a gráfot  $\chi(G)$  színnel?
12. Legyen a  $\pi$  döntési probléma inputja egy  $G$  gráf, az output pedig pontosan akkor „igen”, ha  $G$  síkbarajzolható. Mutassuk meg, hogy  $\pi \in NP \cap coNP$ .
13. Mi az alábbi problémák bonyolultsága, ha az input egy  $G(V, E)$  gráf ( $|V| = n, |E| = e$ )? Természetesen bizonyítsuk is be!
- (a) Van-e  $G$ -ben egy legalább 15 pontú teljes részgráf?
- (b) Van-e  $G$ -ben legalább  $n/100$  hosszúságú kör?
- (c) Van-e  $G$ -ben olyan feszítőfa, amelyben a maximális foksám legfeljebb 2?
- (d) Van-e  $G$ -ben olyan feszítőfa, amelyben a maximális foksám legfeljebb 3?
14. Bizonyítsuk be, hogy a leghosszabb út meghatározása egy irányított, élsúlyozott  $G$  gráfban  $NP$ -teljes! (Pontosabban az ehhez tartozó eldöntési probléma.) Másképpen: bizonyítsuk be, hogy

$$L = \{(G, k) \mid G \text{ irányított, élsúlyozott gráf, van benne legalább } k \text{ súlyú út}\}$$

$NP$ -teljes!

15. **[Vizsga: 2007. június 12.]** Tudjuk, hogy a síkgráfokból álló nyelv  $P$ -ben van. Legyen a SÍK-MAXKLIKK nyelv a következő:

$$\{(G, k) \mid G \text{ egy síkgráf, amiben van } k \text{ pontú klikk}\}$$

Mutassa meg, hogy ez a nyelv  $NP$ -teljes, vagy mutassa meg, hogy a nyelv  $P$ -ben van.



16. [Vizsga: 2009. május 28.]  $P$ -beli vagy  $NP$ -teljes az alábbi probléma? Adott egy  $G = (V, E)$  irányítatlan gráf és az a kérdés, hogy van-e olyan  $C$  kör a gráfban, melyhez minden  $v \notin C$  csúcsból vezet él.
17. [Vizsga: 2010. május 27.] Az árvíz több helyen fenyegeti a gátakat, tudjuk, hogy  $n$  kritikus hely van. Ezek közül az  $i$ -ediknél a gát megfelelő megerősítéséhez  $h_i$  darab homokzsák kell. Ha az erősítés nem történik meg (vagy csak kevesebb homokzsákkal), akkor az  $i$ -edik helyen  $k_i$  kárt okoz a folyó. Adottak a  $h_i$  és  $k_i$  pozitív számok, továbbá a gátak megerősítéséhez összesen rendelkezésre álló homokzsákok  $Z$  száma ( $Z > 0$  egész). Azt szeretnénk meghatározni, hogy ennyi homokzsákkal hogyan tudjuk a kárt minimalizálni, ha feltesszük, hogy a meg nem erősített pontokon keletkező károk összeadódnak.  
Fogalmazza meg a feladatot eldöntési problémaként és vagy adjon rá polinomiális algoritmust vagy igazolja, hogy a probléma  $NP$ -teljes!
18. [Vizsga: 2010. május 27.] Az  $X$  probléma bemenete egy binárisan felírt  $N > 0$  egész szám, és akkor lesz a válasz igen, ha  $N$  nem 2-hatvány. Az  $Y$  probléma bemenete egy  $G$  egyszerű gráf, és akkor lesz a válasz igen, ha  $G$  csúcsainak színezéséhez 3-nál több szín kell. Ha feltesszük, hogy  $P \neq NP$ , akkor van-e  $X \prec Y$ , illetve  $Y \prec X$  Karp-redukció?
19. [Vizsga: 2010. június 17.] Az  $X$  problémában adott egy  $G$  dag és egy  $k$  pozitív egész szám, a kérdés, hogy van-e  $G$ -ben legalább  $k$  élű út. Igaz-e, hogy  $X \prec 3SZÍN$ , illetve, hogy  $3SZÍN \prec X$ ?
20. [Vizsga: 2007. június 12.] Igazolja, hogy ha a  $3SZÍN$  nyelv benne van  $coNP$ -ben, akkor  $NP = coNP$ .

## 12. Egészértékű programozás

1. [Vizsga: 2009. június 11.] Egy adott egyszerű, irányítatlan gráfban maximális méretű párosítást akarunk találni. Írja le ezt a problémát egy egész értékű programozási feladatként! (A kapott egész értékű programozási feladatot nem kell megoldani.)
- 
2. [Vizsga: 2009. június 4.] Egy adott egyszerű, irányítatlan gráfban maximális méretű teljes részgráfot akarunk találni. Írja le ezt a problémát egy egész értékű programozási feladatként! (A kapott egész értékű programozási feladatot nem kell megoldani.)
3. [Vizsga: 2010. június 3.] Fogalmazza meg egész értékű programozási feladatként az alábbi problémát! Egy adott  $G = (V, E)$  irányítatlan egyszerű gráfban keresünk olyan maximális méretű  $D \subseteq V$  csúcshalmazt, melyre teljesül, hogy minden  $x \in V$  csúcsnak legfeljebb 2 szomszédja van a  $D$  halmazban!
4. [Vizsga: 2010. június 17.] Adott egy  $G = (V, E)$  egyszerű, irányítatlan gráf. Egy olyan  $W \subseteq V$  halmazt keresünk, amely a lehető legtöbb csúcsból áll és teljesül rá, hogy a gráfban bármely 2 független él 4 végpontjából  $W$  legfeljebb 2 pontot tartalmaz.  
Hogyan lehet ezt a problémát egészértékű programozási feladatként felírni? (A kapott EP feladatot nem kell megoldani!)

## 13. Közelítő algoritmusok

1. Bizonyítsuk be, hogy a következő algoritmus legfeljebb  $2\tau(G)$  ponttal lefog egy tetszőleges  $G$  gráfot: Keresünk egy tovább nem bővíthető független élhalmazt, és kiválasztjuk ezen élek végpontjait. Éles ez a korlát?
-

2. **[Vizsga: 2009. június 17.]** A Ládapakolás problémának tekintsük azt a speciális esetét, amikor az  $n$  tárgy mindegyike vagy  $a$  vagy  $b$  méretű, ahol  $0 < a < b < 1$  és  $a + b = 1$ . Adjon ennek megoldására  $O(n)$  lépésszámú algoritmust!
3. **[Vizsga: 2009. május 28.]** Egy csomagküldő szolgálatnál csupa egyforma dobozokba pakolják az elküldendő árut. Céljuk az, hogy a ládában maradó üres helyet kitöltő anyagból minél kevesebbre legyen szükség. Igaz-e, hogy a Ládapakolásra megismert First Fit eljárás erre a problémára is egy 2-közelítő algoritmus?
4. **[Vizsga: 2006. június 12.]** Van  $n$  fájlunk, az  $i$ -edik fájl hosszát jelölje  $h_i$ . Tegyük fel, hogy a fájlok hosszuk szerint nem csökkenő sorrendben követik egymást, azaz  $0 < h_1 \leq h_2 \leq \dots \leq h_n$ . Mentéskor két egyforma méretű lemez áll rendelkezésünkre. A mentésnek sorban kell történnie, előbb az első fájlról kell megmondani, melyik lemezre kerüljön, azután a másodikról, stb. (Fájlokat szétvágni nem szabad, minden fájl teljes egészében kerül az egyik vagy a másik lemezre.) Amikor a soron következő fájl már egyik lemezre sem fér rá, akkor abbahagyjuk az eljárást. Egy ilyen eljárás optimális, ha a lehető legtöbb fájlt lehet segítségével kimenteni.  
Mutassa meg, hogy az a mohó eljárás, amikor a következő fájlt oda tesszük, ahol több hely van, nem feltétlenül optimális. Legfeljebb hány fájllal fogunk kevesebbet kimenteni ezzel a mohó eljárással az optimális (szintén sorrendben mentő) megoldáshoz képest?

## 14. Megoldások

## 2. Nagyságrendek

1. (a)  $\log_2 f(n) = \frac{\log_{100} f(n)}{\log_{100}(2)} = c \cdot \log_{100} f(n)$ . Innen kiadódik mindkettő.  
 (b)  $O$ :  $\sum a_i n^i \leq c \cdot n^k$  kell.  $n^k (> 0)$  osztással  $a_k + \sum a_i \frac{1}{n^{i-k}} \leq c$ , amiből már könnyű látni (analízis).  
 $\Omega$ : teljesen hasonlóan.  
 (c)  $2^{n+1} = 2 \cdot 2^n$ , ahonnan az első triviális.  
 Tfh  $2^{2n} = O(2^n)$ . Ekkor  $n > n_0$  esetén  $2^{2n} \leq c \cdot 2^n$ , ahonnan egy osztással  $2^n \leq c$ . Ez nyilvánvalóan nem lehetséges.  
 (d)  $\max(f(n), g(n)) \leq f(n) + g(n)$ , ezért  $O$  triviálisan adódik.  $\max(f(n), g(n)) \geq \frac{f(n)+g(n)}{2} = c \cdot (f(n) + g(n))$ , ahonnan  $\Omega$  adódik.
2. (a) Természetesen lehet ilyen, pl. ha  $|x| < n_0$ .  
 (b) Lehet, hiszen  $2007|x| = O(|x|) = O(|x| \log |x|)$ .
3. A „favágó” megoldás:

$$\begin{aligned}
 T(n) &\leq T(n-1) + \frac{n}{3} \leq T(n-2) + \frac{n}{3} + \frac{n-1}{3} \\
 &\leq \dots \leq T(5) + \frac{1}{3}(n + (n-1) + \dots + (n - (n-6))) \\
 &= T(5) + \frac{1}{3} \left( \sum_{i=1}^n i - (1+2+3+4+5) \right) = T(5) + \frac{n(n+1)}{6} - 5 \\
 &\leq 10 - 5 + \frac{n(n+1)}{6} = 5 + \frac{n(n+1)}{6}
 \end{aligned}$$

Innen egyszerű bizonyítani  $O(n^2)$ -et, ahonnan  $O(n^3)$  is rögtön adódik.  $O(n)$ -ről nem tudunk semmit, nincs kizárva, de nem is bizonyítható (Vigyázat! Ha  $\leq$  helyett = lenne, akkor már nem ez lenne a helyzet!).

Az „elegáns” megoldás (Vigyázat! Sok veszély!):

$O(n^2)$ -et bizonyítjuk.  $n_0 = 5$ -re és  $c = 1$ -re igaz, mert  $T(5) \leq 10 \leq 1 \cdot 5^2 = 25$ . Indukcióval tfh valamilyen  $n > 5$ -re igaz (azaz  $T(n) \leq c \cdot n^2 = 1 \cdot n^2$ ), belátandó, hogy  $n+1$ -re is.

$$T(n+1) \leq T(n) + \frac{n+1}{3} \leq c \cdot n^2 + \frac{n+1}{3} = 1 \cdot n^2 + \frac{n+1}{3} \leq n^2 + n + 1 \leq n^2 + 2n + 1 = (n+1)^2 = c \cdot (n+1)^2,$$

pont amit bizonyítani akartunk.

**Figyelem!** Itt a konstanst rögzíteni kell előre, nem bánhatunk vele olyan lazán, mint a normál bizonyításoknál! Végig **ugyanaz** a  $c$  szerepel (jelen esetben 1-re rögzítve, más feladatnál lehet más). Az egyszerűséget az adta, hogy felső becsléseket simán adhatunk ( $n$  helyett  $2n$ -t írunk, stb.). Ezzel viszont pl.  $O(n)$ -t akkor sem lehetne ilyen simán kizárni, ha  $T(n)$  definícióban  $\leq$  helyett = lenne.

4.  $\mathcal{A}(n)$ -et és  $\mathcal{B}(n)$ -et felírjuk először rekurzívan, majd átírjuk zárt alakra, ahonnan ki fog jönni (felhasználva, hogy  $1 = n - (n-1)$ , valamint  $\sum_{i=0}^k c^i = \frac{c^{k+1}-1}{c-1}$ ).

$$\begin{aligned}
 \mathcal{A}(n) &= 10 + 2\mathcal{A}(n-1) = 10 + 2(10 + 2\mathcal{A}(n-2)) = 10 + 10 \cdot 2 + 2^2\mathcal{A}(n-2) = \\
 &= 10 + 10 \cdot 2 + 2^2(10 + 2\mathcal{A}(n-3)) = 10 + 10 \cdot 2 + 10 \cdot 2^2 + 2^3\mathcal{A}(n-3) = \\
 &= \dots = 10(2^0 + 2^1 + 2^2 + \dots + 2^{n-2}) + 2^{n-1}\mathcal{A}(1) = 10 \sum_{i=0}^{n-2} 2^i + 2^{n-1} = \\
 &= 10(2^{n-1} - 1) + 2^{n-1} = 11 \cdot 2^{n-1} - 10 = O(2^n) \\
 \mathcal{B}(n) &= 3 + 4\mathcal{B}(n-1) = 3 + 4(3 + 4\mathcal{B}(n-2)) = 3 + 3 \cdot 4 + 4^2\mathcal{B}(n-2) = \\
 &= 3 + 3 \cdot 4 + 4^2(3 + 4\mathcal{B}(n-3)) = 3 + 3 \cdot 4 + 3 \cdot 4^2 + 4^3\mathcal{B}(n-3) = \\
 &= \dots = 3(4^0 + 4^1 + 4^2 + \dots + 4^{n-2}) + 4^{n-1}\mathcal{B}(1) = 3 \sum_{i=0}^{n-2} 4^i + 4^{n-1} = \\
 &= 3\left(\frac{4^{n-1}}{3} - \frac{1}{3}\right) + 4^{n-1} = 2 \cdot 4^{n-1} - 1 = \Omega(4^n)
 \end{aligned}$$

Fentiekből látszik, hogy  $\mathcal{A}$  a gyorsabb (fontos, hogy az egyikre  $O$ -t számoltunk, a másikra  $\Omega$ -t; persze itt mindkettő  $\Theta$ , ezért nem kellett különbözőképpen számolnunk).

5. (a)  $f \leq c_f \cdot g \leq c_f \cdot (c_g \cdot h) = c' \cdot h$  ( $n > \max(n_f, n_g)$  esetén), tehát igaz.  
 (b)  $f \geq c_f \cdot g \geq c_f \cdot (c_g \cdot h) = c' \cdot h$  ( $n > \max(n_f, n_g)$  esetén), tehát igaz.
6.  $f_2 = O(f_4) = O(f_1) = O(f_3)$ . Ez három számolás, nem írom le. Amit bizonyítás nélkül fel lehet hozzá használni:  $\log n \leq c \cdot \sqrt{n}$ .
7. (a) Természetesen, hiszen a definíciónak nem mond ellent.  
 (b) Természetesen, hiszen ha  $|x| < n_0$ , akkor bármi lehet.
8. Megsejtjük, hogy  $f_2(n) = O(f_3(n)) = O(f_1(n))$ . Most bebizonyítjuk, definíció szerint.  
 $f_2(n) = O(f_3(n))$ :  $\exists c > 0, n_0 > 0$ , hogy  $\forall n > n_0$   $2007n^3 \leq c(3^{3n})$ . Ezt átalakítva, konstans hozzávéve  $c$ -hez  $n^3 \leq c \cdot 27^n$ , aminek az igazságát már nem kell bizonyítani, triviálisnak tekinthető.  
 $f_3(n) = O(f_1(n))$ :  $\exists c > 0, n_0 > 0$ , hogy  $\forall n > n_0$   $3^{3n} \leq c(2^{100n} - 2^{50n})$ . Írhatjuk, hogy  $3^{3n} \leq c2^{50n}(2^{50n} - 1)$ , és mondjuk logaritmust vonva  $n \log 27 \leq c' + n \log 2^{50} + \log(2^{50n} - 1) = c' + 50n + \log(2^{50n} - 1)$ . A kifejtetlen tag biztos pozitív, így ez biztosan igaz bármely  $n > 0$  és  $c \geq 0$  esetén.  
*Számolás nélkül, nem definíció szerint nem lehet max. pontot kapni!*
9. Nem következik, hiszen a páratlan  $n$ -ekről semmit sem tudunk, ott lehet akár  $2^n$  is.
10.  $\mathcal{A}$ -ra felső becslésünk van, így kiszámolva  $O(\dots)$  lenne. Az  $O$  felső becslés, így mindkét algoritmus lépésszámára csak felső becslésünk van, vagyis nem tudjuk eldönteni, melyik a gyorsabb (lehet mindkettő akár pl. konstans lépésszámú is).

**Megjegyzés:** itt nem volt rá szükség, de ha a rekurzív képletet fel akarnánk írni és ki akarnánk számolni, akkor (feltéve, hogy  $\mathcal{A}(1) = c$ , valamint néhány egészrész jelet nagyvonalúan elhanyagolva) azt így lehet (felhasználva, hogy  $1 = \frac{n}{2^{\log n}}$ ):

$$\begin{aligned} \mathcal{A}(n) &\leq 5 + 2\mathcal{A}\left(\frac{n}{2}\right) \leq 5 + 2\left(5 + 2\mathcal{A}\left(\frac{n}{4}\right)\right) = 5 + 5 \cdot 2 + 2^2 \mathcal{A}\left(\frac{n}{4}\right) \leq \\ &\leq 5 + 5 \cdot 2 + 2^2\left(5 + 2\mathcal{A}\left(\frac{n}{8}\right)\right) = 5 + 5 \cdot 2 + 5 \cdot 2^2 + 2^3 \mathcal{A}\left(\frac{n}{16}\right) \leq \\ &\leq \dots \leq 5(2^0 + 2^1 + 2^2 + \dots + 2^{\log n}) + 2^{\log n} \mathcal{A}(1) = 5 \sum_{i=0}^{\log n - 1} 2^i + c \cdot n = \\ &= 5(2^{\log n} - 1) + c \cdot n = (5 + c)n - 5 = O(n) \end{aligned}$$

11. (a)  $x > \max(n_f, n_g)$  esetén  $f(g(x)) \leq c_f \cdot 3g(x) \leq c_f \cdot 3 \cdot c_g \cdot 3x \leq c'3x = c'h(x)$ . ( $n_f$  és  $n_g$  az  $f$ -hez és  $g$ -hez tartozó küszöbszámot jelenti.)  
 (b) Ellenpélda:  $f(x) = x^2, g(x) = x^3, h(x) = x^4$ , ebből  $f(g(x)) = (x^3)^2 = x^6$ . Látszik, hogy a feltétel teljesül, viszont  $x^6 \neq O(x^4)$ .

### 3. Dinamikus programozás

1. Felveszünk egy  $T[n \times n]$  táblázatot, ahol  $T[i, j]$  jelentése a következő: a táblázat  $i$ . sor,  $j$ . oszlopába mi a legnagyobb értékű, a szabályokat betartó út értéke? Kitöltés, ha az eredeti táblázat neve  $A$ :

$$T[i, j] = \begin{cases} -\infty & \text{ha } A[i, j] \leq A[i-1, j] \text{ és } A[i, j] \leq A[i, j-1] \\ T[i-1, j] + A[i, j] & \text{ha } A[i, j] > A[i-1, j] \text{ és } A[i, j] \leq A[i, j-1] \\ T[i, j-1] + A[i, j] & \text{ha } A[i, j] \leq A[i-1, j] \text{ és } A[i, j] > A[i, j-1] \\ \max(T[i-1, j], T[i, j-1]) + A[i, j] & \text{ha } A[i, j] > A[i-1, j] \text{ és } A[i, j] > A[i, j-1] \end{cases}$$

$T[1, 1] = A[1, 1]$ , továbbá az esetleges túlindexeléseket értelemszerűen kezeljük. A megoldást  $T[n, n]$  tartalmazza, ami  $-\infty$ , ha nem vezet oda szabályos út. A helyesség bizonyítása indukcióval pont az, amit a kitöltési szabály mond.

Mivel minden egyes mező kitöltése konstans műveletet vesz igénybe, így a teljes lépésszám  $O(n^2)$ .

2. Választunk tetszőlegesen egy gyökeret, így tudunk szintekről beszélni. Észrevesszük, hogy egy tetszőleges pont, mint gyökér által meghatározott részében a maximális súlyú független ponthalmaz kétféleképpen állhat elő: vagy bevesszük az aktuális csúcsot, és a lehető legnagyobb súlyú független ponthalmazt kiválasztjuk az unokáiból induló részfákban, vagy nem vesszük be, és a lehető legnagyobb súlyú független ponthalmazt kiválasztjuk a gyerekeiből induló részfákban. Így lentől felfele haladva minden csúcra meg tudjuk mondani, hogy a belőle induló részében mi a keresett érték, és a teljes fa gyökerében pont az egész fára vonatkozó érték fog szerepelni.

A lépésszámot úgy nehéz lenne kiszámolni, ha azt néznénk, hogy az egyes csúcsokban milyen műveleteket végzünk, hiszen a gyerekek és unokák számáról nem tudunk semmit. Inkább azt célszerű vizsgálni, hogy egy csúcsot hányszor veszünk a kezünkbe: amikor őt vizsgáljuk, vagy a szülőjét, vagy a nagyszülőjét. Vagyis  $n$  csúcs esetén legfeljebb  $O(3n) = O(n)$  lépést teszünk, a bejárás megszervezése pedig szintén lineáris.

3. Felvesszünk egy  $n \times m$  méretű  $A$  mátrixot, ahol  $A[i, j]$  jelentése a következő: az  $x_i$ -n és  $y_j$ -n végződő leghosszabb közös részsorozat hossza. Kitöltés:

$$A[i, j] = \begin{cases} 0 & \text{ha } x_i \neq y_j \\ A[i-1, j-1] + 1 & \text{ha } x_i = y_j \end{cases}$$

A spec esetek (szélek) értelemszerűen. Ekkor a táblázatban a max. szám megadja a legnagyobb közös részsorozat hosszát, onnan átlósan visszalépkedve megkapjuk magát a sort. Táblázat kitöltése  $O(n^2)$ , a megoldás megkeresése is megvan ennyiből (persze ezt nem muszáj utólag, nyilván a kitöltés közben is nyilvántarthatjuk).

4. Észrevesszük, hogy egy tetszőleges  $b$  karakter előtti összes  $a$  karakter meghatároz egy-egy ilyen szót. A szövegben haladva ha  $a$ -t találunk, akkor az eddigi  $a$ -k számát növeljük eggyel, ha  $b$ -t, akkor pedig a végeredményhez hozzáadjuk az  $a$ -számláló aktuális értékét. Az algoritmus egy végigolvasásból megvan, azaz  $O(n)$ , valamint helyes (ezt nagyon precíz indukcióval is bebizonyíthatják).
5. Mivel a két sarok közötti távolság  $n + m - 2$ , ezért ténylegesen csak jobbra vagy lefele léphetünk. Szokásosan egy  $n \times m$  méretű  $A$  mátrixot töltünk, ahol  $A[i, j]$  jelentése: az  $i$ -edik sor  $j$ -edik oszlopára hányféleképpen tudunk szabályosan eljutni. Kitöltés:

$$\begin{aligned} A[1, 1] &= 1 \\ A[i, j] &= \begin{cases} 0 & \text{ha } (i, j) \text{ tiltott} \\ A[i-1, j] + A[i, j-1] & \text{ha } (i, j) \text{ nem tiltott} \end{cases} \end{aligned}$$

Ez azért jó így, mert tiltott mezőre 0-féleképpen mehetünk, egy nem tiltott mezőre pedig csak fentről vagy balról. A megoldás  $A[n, m]$ -ben lesz található. Helyesség indukcióval könnyen belátható, lépésszám  $O(nm)$ , mert egy  $n \times m$  méretű mátrixot töltünk ki egy menetben.

6. Nagy meglepetésre egy  $n \times k$  méretű táblázatot töltünk, ahol  $A[i, j]$  jelentése, hogy a szabályok szerint haladva az  $(i, j)$  mezőre hány megjelölt elemet tartalmaz a legjobb út. Nyilván  $A[n, k]$  fogja tartalmazni a megoldást. Kitöltés (ha  $M[i, j]$  pontosan akkor igaz, ha  $(i, j)$  megjelölt):

$$\begin{aligned} A[1, 1] &= M[1, 1] \\ A[i, j] &= M[i, j] + \max(A[i-1, j], A[i, j-1]) \end{aligned}$$

A lépésszám nyilván  $O(nk)$ .

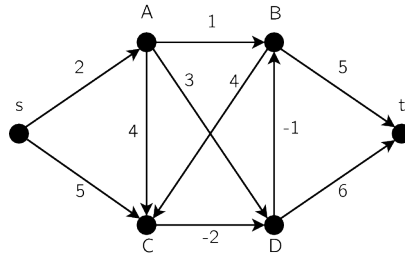
7. Vegyük fel az  $m$ -hez tartozó teljes maradékrendszert egy (mondjuk bool)  $A$  tömbbe! Kezdetben minden hely *false*. Az  $a_i$  szám kézbevételekor  $\forall j : A[j] = \text{true}$  helyre állítsuk  $A[(j + a_i) \pmod m]$ -et *true*-ra, majd  $A[a_i \pmod m]$ -et is. Ha  $A[1]$  *true* lesz, akkor kiválasztható, ha a végére sem, akkor nem. Ezt be is kell bizonyítani! Azt állítom, hogy  $a_i$  kézbevétele után pontosan azok az  $A[j]$ -k vannak *true*-ra állítva, amik valahogy előállnak  $a_1 \dots a_i$  közül néhány összegeként  $(\pmod m)$ . Indukcióval nyilván igaz, hiszen  $a_1$  választásával csak  $a_1 \pmod m$  áll elő. Tfh  $k$ -ra igaz, ekkor nézzük  $k+1$ -re.  $a_1 \dots a_{k+1}$  közül néhányat választva vagy választjuk  $a_{k+1}$ -et, vagy nem. Ha nem, akkor pontosan azok állnak elő, amik  $a_1 \dots a_k$  közül, ha igen, akkor egyrészt előállhat maga  $a_{k+1}$ , másrészt az eddig előálló számok mindegyikéhez hozzávehetjük  $a_{k+1}$ -et is. Ezért a feltétel igaz maradt, hiszen az összes lehetséges esetet lefedtük. A lépésszám polinomiális, hiszen  $O(nm)$  műveletet végzünk (minden számhoz a teljes táblán végigmegyünk), ami ebben a spec. esetben azért polinomiális, mert  $m = O(n^2)$ . (Ha  $m$ -re nem lenne korlát, akkor ez akár exponenciális is lehetne.)
8. Szokásosan tetszőlegesen választhatunk egy gyökeret, így beszélhetünk szintekről és egyebekről. Minden csúcshoz egy számot fogunk rendelni  $(\pmod l(i))$ , lentől felfelé haladva. Ez a szám azt jelenti, hogy a levelekből melyik a leghosszabb út az adott csúcsba. Egy  $v$  csúcs vizsgálatakor meg kell vizsgálni a rajta áthaladó utak közül a leghosszabbat, ez a két legnagyobb számmal rendelkező gyerekén (mondjuk  $i$  és  $j$ ) keresztül megy, a hossza pedig  $l(i) + l(j) + 1$ . Ha ez hosszabb, mint az eddigi leghosszabb, akkor megjegyezzük,  $l(v)$  pedig (ha  $l(i)$  volt a maximális)  $= l(i) + 1$ . Bizonyítás indukcióval;

lépésszám: minden csúcsot csak akkor veszünk kézbe, amikor őt vizsgáljuk, és a szülőjének vizsgálatakor a két max. keresésben, így  $O(3n) = O(n)$ , ebbe a kezdeti gyökeresítés és a bejárás szervezése is belefér.

9. Felveszünk egy  $T$  tömböt  $n$  mérettel.  $T[i] = 1$  azt jelenti, hogy  $y_1 \cdots y_i$  előáll jó sorozatokból.  $T[i]$  kitöltése:  $T[i] = A(1, i) \vee (T[1] \wedge A(2, i)) \vee \cdots \vee (T[j] \wedge A(j+1, i)) \vee \cdots \vee (T[i-1] \wedge A(i, i))$ . Magyarul megnézzük, hogy az eddigi helyes részekhez hozzá tudjuk-e illeszteni az újat. A végeredmény  $T[m]$ -ben található. Ez  $O(n^2)$  db meghívása  $A$ -nak legfeljebb  $n$  méretű inputtal, innen kijön a jó lépésszám.
10. Elég csúnyán hangzik, de nem az. Szokásosan valami olyasmire kell gondolni, hogy egy sor vizsgálatánál már rendelkezésünkre álljon az összes megelőző érdekes adat. Vegyünk fel egy  $D$  tömböt, ahol  $D[i]$  értéke legyen az  $1 \dots i$  szavak legkisebb hibájú tördelésének értéke!  $D[0] = 0$  jelentse azt, hogy 0 szót 0 sorba hiba nélkül be tudunk rakni. Amikor épp az  $i$ -edik szót vizsgáljuk, akkor a következő lehetőségeink vannak:  $i$ -t külön sorba vesszük, az  $i-1$ -ig bezárólag pedig a lehető legjobb tördelést adjuk (jé, ez pont  $D[i-1]$ !);  $i-1$ -et és  $i$ -t vesszük egy sorba, az előttük levőket pedig a legjobban tördeljük (csak nem  $D[i-2]$ ?);  $\dots$ ; 1-től  $i$ -ig csinálunk egy sort. Meg is van a képlet:  $D[i] = \min_{j=1 \dots i} (t_{j,i}^2 + D[j-1])$ , a helyességet a fenti gondolatmenet adja.  $D[n]$  fogja az optimális tördelés értékét megadni. Ha meg is akarjuk kapni a tényleges tördelést, akkor egy  $T$  tömbben nyilvántarthatjuk, hogy adott  $i$ -hez  $D[i]$ -nél hogyan kaptuk a minimumot, azaz hol van a sortörés.  $T[n]$ -től visszafele lépkedve megkapjuk a sortörések helyét. Mivel az  $i$ -edik szónál 1-től  $i$ -ig végignézzük az eddigi tömböt (azaz  $1, 2, \dots, n$  műveletet végzünk), a lépésszám  $O(n^2)$  lesz.
11. Lásd következő feladat.
12. Lásd előző feladat.
13. Alakítsuk a határidőket a mai naptól induló számozásra (ha szükséges). Legyen  $A[i]$  a legfeljebb  $i$  indexű feladatokat használó maximális profitú ütemezés:  $A[0] = 0$ . Ha  $i > 0$ , akkor  $A[i]$  úgy kapható, hogy az  $i$  indexű feladatot megkíséreljük a határidejétől visszafelé betenni a naptárunkba. Ha van üres hely, akkor odatesszük és  $A[i] = A[i-1] + P_i$ . Ha nincs üres hely, de a legkisebb profitú munka profitja ( $P_j$ ) kisebb, mint  $P_i$ , akkor az  $M_j$  munka helyére tesszük be a  $M_i$  munkát. Ez  $O(n^2)$ .

#### 4. Szélességi bejárás, legrövidebb utak, párosítások

1.



Lesz egy  $F$  tömbünk, amit számítógépen tömbként kezelünk, papíron táblázatba írjuk. Az  $s$ -től aktuálisan ismert legrövidebb utak hossza szerepel benne. Kezdetben az  $s$ -ből egy élen elérhető csúcsokhoz tartozik nem  $\infty$  érték.

A	B	C	D	t
2	$\infty$	5	$\infty$	$\infty$

Most végig kell menni minden  $i$  csúcson, és megnézni az összes  $i$ -be befutó élre, hogy ennek segítségével lehet-e javítani. Pl. a  $D$  csúcshoz tartozó érték így számítható:  $\min(F[D], F[A] + 3, F[C] - 2) = \min(\infty, 2 + 3, 5 - 2) = 3$ . Ezt az összes csúcstra végig kell játszani minden lépésben, azaz a  $k$ -edik lépésben a következőt végezzük el:

$$\begin{aligned}
 F_k[A] &= \min(F_{k-1}[A], F_{k-1}[s] + 5) \\
 F_k[B] &= \min(F_{k-1}[B], F_{k-1}[A] + 1, F_{k-1}[D] - 1) \\
 F_k[C] &= \min(F_{k-1}[C], F_{k-1}[s] + 5, F_{k-1}[A] + 4, F_{k-1}[B] + 4) \\
 F_k[D] &= \min(F_{k-1}[D], F_{k-1}[A] + 3, F_{k-1}[C] - 2) \\
 F_k[t] &= \min(F_{k-1}[t], F_{k-1}[B] + 5, F_{k-1}[D] + 6)
 \end{aligned}$$

A táblázat tehát a második lépés után:

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>t</i>
2	$\infty$	5	$\infty$	$\infty$
2	3	5	3	$\infty$

Fontos, hogy mindig az előző sorból vesszük az értékeket, így lehetséges az, hogy  $F[t]$  végtelen maradt (ugyanis  $\min(\infty, F[B] + 5, F[D] + 6) = \min(\infty, \infty, \infty)$ ). A következő sor:

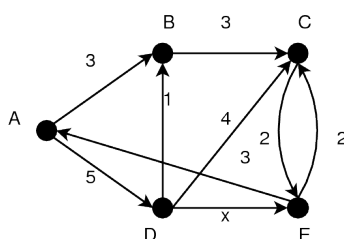
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>t</i>
2	$\infty$	5	$\infty$	$\infty$
2	3	5	3	$\infty$
2	2	5	3	8

Itt is vegyük észre, hogy  $t$ -hez még az előző sorban szereplő értékeket használtuk fel! Tovább:

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>t</i>
2	$\infty$	5	$\infty$	$\infty$
2	3	5	3	$\infty$
2	2	5	3	8
2	2	5	3	7
2	2	5	3	7

Kétféleképpen is észrevehetjük, hogy befejezhetjük. Egyrészt ha két sor megegyezik, akkor biztos nem lesz a továbbiakban változás, másrészt  $n$  csúcs (itt 6) esetén  $n - 1$  sor (itt 5) kitöltése biztos elég.

2.



<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>KÉSZ</b>
<u>0</u>	<b>3</b>	$\infty$	5	$\infty$	A
<u>0</u>	<u>3</u>	6	<b>5</b>	$\infty$	A,B
<u>0</u>	<u>3</u>	6	<u>5</u>	$5+x$	A,B,D

Ha  $0 < x \leq 1$ :

<u>0</u>	<u>3</u>	<b>6</b>	<u>5</u>	$5+x$	A,B,D,E
<u>0</u>	<u>3</u>	<u>6</u>	<u>5</u>	$5+x$	A,B,C,D,E

Ha  $1 < x$ :

<u>0</u>	<u>3</u>	<u>6</u>	<u>5</u>	$\min(8, 5+x)$	A,B,C,D
<u>0</u>	<u>3</u>	<u>6</u>	<u>5</u>	$\min(8, 5+x)$	A,B,C,D,E

3. Egy csillag (1 db  $n - 1$  fokú pont körül  $n - 1$  db elsőfokú pont).

4. A problémát egy páros gráffal fogjuk modellezni, az  $A$  csúcsosztály a lehetséges ajánlók, a  $B$  csúcsosztály pedig az ajánlólevelek (minden pályázott helyre két darab). Egy ajánló és egy ajánlólevél között pontosan akkor menjen él, ha az adott ajánló megírhatja az adott ajánlólevelet.

Ebben a gráfban minden (rész)megoldásnak megfelel egy párosítás, ugyanis ha van egy helyes (rész)megoldásunk, akkor a megfelelő ajánlók és ajánlólevelek által kiválasztott élek függetlenek lesznek (ha nem lennének, az azt jelentené, hogy vagy egy ajánló több levelet is írt, vagy egy ajánlólevelet többen is írtak). Hasonlóan látható, hogy egy párosításhoz mindig tartozik egy helyes részmegoldás. A feladat tehát az, hogy keressünk egy, az ajánlólevelek csúcshalmazát lefedő párosítást. Ezt a magyar módszerrel megtehetjük.

A lépésszám  $O(|V||E|) = O((m + 2n) \cdot m \cdot 2n) = O((n + m)nm)$ , mert  $|V| = m + 2n$ , továbbá  $|E| \leq m \cdot 2n$  hiszen minden ajánlólevél-osztálybeli csúcs fokszáma legfeljebb  $m$  lehet.

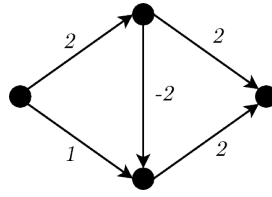
5. A 2 hosszú éleket helyettesítsük 2, a 3 hosszúakat 3 élellet. Az utak hossza így az élszám lesz (minden út annyi élből áll így, amilyen hosszú az eredeti gráfban lenne), így a szélességi bejárás használható legrövidebb út keresésére. A lépésszám  $O(3n + 3e) = O(n + e)$ .

6. Vegyünk egy  $n$  hosszú irányított utat! Ha a szélességi bejárást az utolsó pontból indítjuk, akkor elakad, utána az utolsó előttiből, ott is elakad stb., így pont  $n$  komponenst fog csinálni. Ennél többet



nem is lehet  $n$  pontból.

7. Attól, hogy rakunk bele negatív élsúlyt, még akár működhetne! Úgy kell negatív élsúlyt belerakni, hogy romoljon el, pl:



A bal oldali csúcsból futtatva az alsó csúcsra az algoritmus 1-et mond, pedig a legrövidebb út oda 0 lenne.

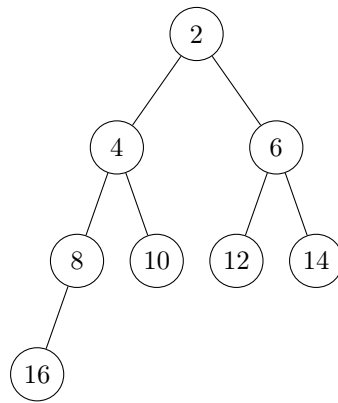
8. A táblázat alapján vizsgáljuk az algoritmus futását. Mivel a legkisebb élszámú gráf kell, ezért csak oda húzunk be éleket, ahol az egyes lépésekben javítottunk (ide pedig kötelező). Látszik, hogy  $v_1$ -ből indulunk. Az első sor miatt innen  $v_2$ -be,  $v_3$ -ba és  $v_6$ -ba vezet él, a megfelelő súlyokkal. A KÉSZ-be  $v_2$ -t választjuk, és a javítások miatt rájövünk, hogy kell lennie egy  $v_2v_3$ ,  $v_2v_4$  és egy  $v_2v_6$  élnek, 3, 7 és 4 súlyokkal. A továbbiakban minden ugyanígy (a gráfot nem rajzolom fel). Csak az utolsó előtti lépésben lesz választási lehetőség, hogy  $v_4$ -et vagy  $v_6$ -ot vegyük be a KÉSZ-be. Ettől függően lesz egy 2 súlyú  $v_4v_5$  és egy 1 súlyú  $v_6v_5$  élünk, vagy pont fordítva. A  $P$  tömb változásai (csak az első esetben):

$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	$v_6$
—	$v_1$	$v_1$	$v_1$	$v_1$	$v_1$
—	$v_1$	$v_2$	$v_2$	$v_1$	$v_2$
—	$v_1$	$v_2$	$v_3$	$v_3$	$v_2$
—	$v_1$	$v_2$	$v_3$	$v_4$	$v_2$
—	$v_1$	$v_2$	$v_3$	$v_6$	$v_2$

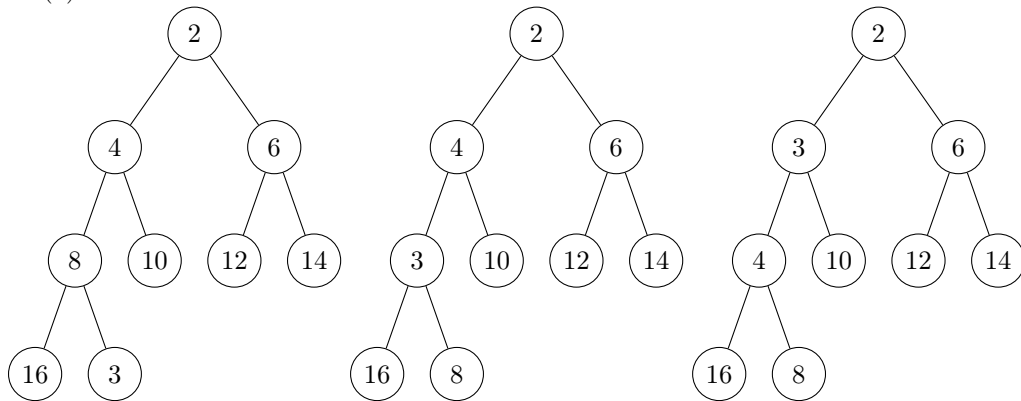
9. A gráfból kihagyjuk a nehezek közti éleket (4 pont). Szélességi bejárással megkeressük, hogy mely csúcsok vannak az otthonnal egy komponensben, ezek lesznek elérhetők (2 pont). Ez jó, mert pontosan azokat az éleket hagytuk el, amiken nem mehetünk át, minden más benne maradt (ezt lehet precízebben is) (2 pont). Lépésszám: éltörlések  $O(e)$ , bejárás  $O(n + e)$ , ami összesen  $O(n + e)$  (2 pont).
10. Egy legrövidebb út kétféle lehet: vagy használja a negatív éleket, vagy nem. Először futtassunk egy Dijkstra-t (a szépség kedvéért hagyjuk ki a negatív éleket), az  $i$  csúcsba vezető legrövidebb út hosszát jelölje  $d_i^+$ . Jelölje  $u$  azt a csúcsot, amiből a negatív él indul. Világos, hogy  $d_u^+$  helyes, hiszen a negatív élen nem mehetünk át az  $o$  eléréséhez az eredeti gráfban. Belőle is indíthatunk egy Dijkstra-t (az eredeti gráfon), hiszen ekkor először a negatív él másik végpontját veszi be a KÉSZ halmazba, ami a definíciónak megfelel. Ezeket a legrövidebb értékeket jelölje  $d_i^-$ . A legrövidebb utak tehát  $\min(d_i^+, d_u^+ + d_i^-)$  képlettel számolhatók. Lépésszám egy keresés és két Dijkstra, azaz  $O(e + n^2 + n^2) = O(n^2)$  (az ugye mindenkinek világos, hogy egy egyszerű gráfban  $e = O(n^2)$ ).
11. Ez csak annyiban trükkös, hogy a várakozásokkal ellentétben ez nem legrövidebb út, hanem párosítás. Ugyanis minden gazda-ponthoz párosítani akarunk egy fát. Egyik pontosztály tehát a gazda pontjai, másik a fák, akkor megy él, ha adott pontból elérhető a fa úgy, hogy utána a kutya vissza tud érní. Itt kell max. párosítás, ami mehet magyar módszerrel, ami  $O(|V||E|) = O((n + f)(nf)) = O(n^2f + nf^2)$ , a felépítés pedig megy  $O(nf)$ -ben, ami nyilván belefér (a távolságszámítás egy pont és fa és pont között konstans idő).
12. A Bellmann-Ford algoritmus során a táblázat  $i$ -edik sorában a kezdőpontból a legfeljebb  $i$  élszámú legrövidebb utak hosszai szerepelnek (ezt nem kell külön bizonyítani!). Mivel tudjuk, hogy  $G$ -ben minden út legfeljebb 25 élből áll, a táblázatot elég a 25. sorig kitölteni. A lépésszám így  $O(25n^2) = O(n^2)$  lesz.
13. Az összefüggőség és irányítatlanság miatt a szélességi bejárás egy szélességi feszítőfát fog adni. A páratlan szinteken lévő pontok száma legyen  $x$ , értelemszerűen a párosoké ekkor  $n - x$ . Ha  $x < n - x$ , akkor válasszuk központi halmaznak a páratlan szinten lévő pontokat, egyébként a párosakat. Könnyen látszik, hogy minden pontra igaz, hogy vagy benne van a kiválasztott halmazban, vagy elérhető belőle egy lépésben (elég csak a szomszéd szintre menni). A kiválasztott halmaz a skatulya-elv miatt biztos legfeljebb  $n/2$  méretű lesz. A lépésszám  $O(n + e)$ , ami az összefüggőség miatt  $O(e)$ .

## 5. Kupacok

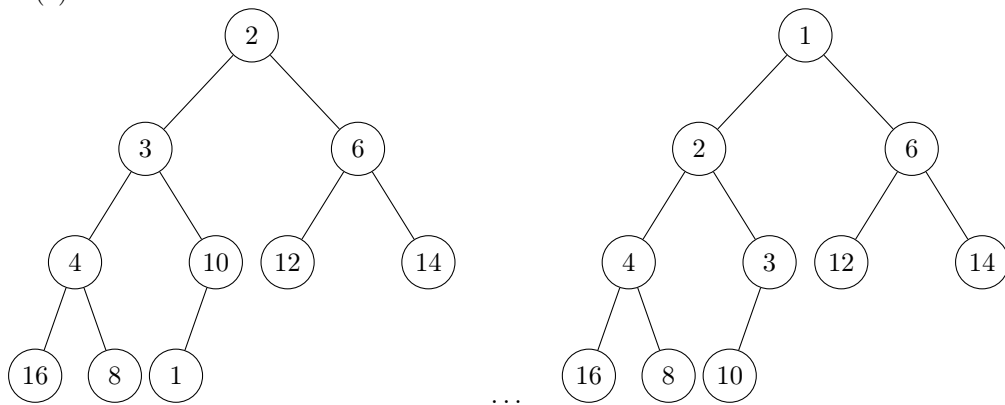
1.



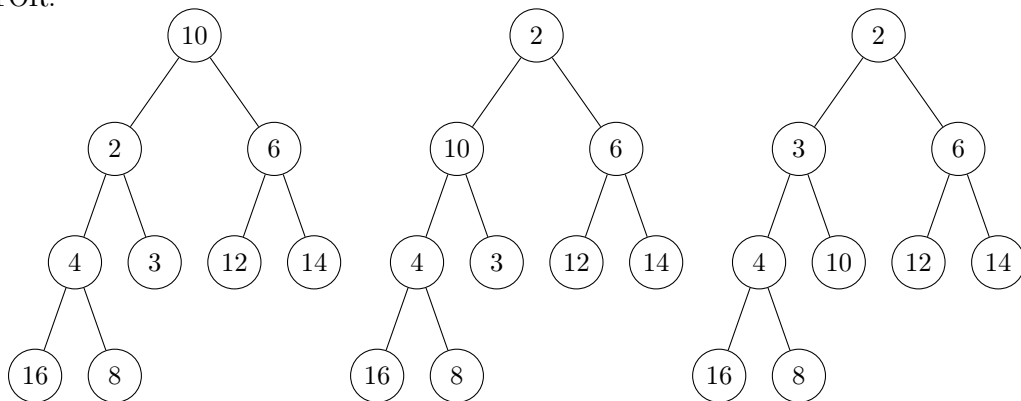
BESZÚR(3):



BESZÚR(1):



MINTŐR:



2. 3 max-kupacot fogunk nyilvántartani, az egyikbe mennek  $A$  betegek, a másikba  $B$  betegek, a harmadikba  $A \vee B$  betegek. Beszúrás: a beteg típusától függő kupacba (ahol jelenleg  $m \leq n$  beteg van),  $O(\log m) = O(\log n)$ . Ha  $A$  orvos végez, akkor a  $\max(\max_A, \max_{A \vee B})$  sorszámú beteget hívja be (mivel a keresett elemek a kupacok gyökerében vannak, ezért ez konstans lépés), majd a kiválasztott beteg kupacában csinálni kell egy MAXTÖR-t, ami, ha  $m < n$  eleme van, akkor  $O(\log m) = O(\log n)$  lépés. A helyesség triviálisan látható.
3. Pontosan akkor, ha az eddig tárolt elemeknél kisebb  $x$ . Ez a feltétel szükséges, hiszen a MINTÖR a legkisebbet fogja visszaadni, és ha az új nem az lenne, akkor mindenképpen bennmaradna. Az elégségesség kicsit macerább. Amit tudunk:  $x$  a gyökéig felment, és legalulról indult. MINTÖRNÉL a legalsó kerül  $x$  helyébe (nevezzük  $y$ -nak), az fog lefele menni. Azt akarjuk bizonyítani, hogy pont az eredeti helyére kerül vissza (és mindenki más is, aki az úton volt). Indukcióval az aktuális szintre: a gyökérben most  $y$  van, egyik gyereke az eddigi legkisebb, és pont azon az oldalon van, ahonnan  $x$  eredetileg „feljött”. Mindenképpen ő fog a gyökérbe kerülni, ezért  $y$  vele fog helyet cserélni, azaz ugyanazon az úton indul lefele, ahonnan  $x$  jött. Ez az érvelés pontosan ugyanígy megy tetszőleges közbenső szinten is, ha  $y$  aktuális pozícióját vesszük a gyökérnek.
4. Konstans (azaz  $O(1)$ ), hiszen az  $i$ -edik legkisebb elem a gyökértől legfeljebb  $i$  távolságra van, így  $2^{10}$  elem közül kell a tizedik legkisebbet megtalálni, ami konstans lépés (a legbutább módon is).
5. Lényegében egy bejárás, elindulunk a gyökérből, és amíg  $k$ -nál kisebb elemet találunk, addig kiírjuk, ha  $\geq k$ -t, akkor az adott ágat hanyagoljuk, hiszen a kupac-tulajdonság miatt arrafele csak még nagyobbak lehetnek. Legfeljebb a nekünk jó elemek mindkét gyereket vizsgáljuk meg feleslegesen, vagyis  $O(2m) = O(m)$  vizsgálatot végzünk.
6.  $O(n)$  lépésből ki lehet találni, hogy melyik 5 szám hiányzik. Ezeknek  $5!$  sorrendje lehet, így mindegyiket végig tudjuk próbálni. Minden egyes elemnek csak a szülőjét és két gyereket kell megvizsgálni, hogy jó-e velük a kupactulajdonság (permutációnként 15 vizsgálat). Ha az adott permutáció jó, akkor van egy lehetséges kitöltésünk. Minden lehetőséget megvizsgáltunk, így az algoritmus biztos helyes. Lépésszám:  $O(n + 5! \cdot 15) = O(n)$ . Ez azért optimális, mert pusztán az 5 sérült elem megtalálása is  $\Omega(n)$  lépés.
7. Használjunk egy min-kupacot és egy max-kupacot, valamint a tárolt elemeket a két kupacban kössük össze pointerekkel (hogy az egyik kupacban lévő, a kezünkben lévő elemről keresés nélkül tudhassuk, hogy hol található a másik kupacban).  
*Felépít:* pointer inicializálása + két kupac építése:  $O(n + n + n) = O(n)$ .  
*Beszúr:* pointer inicializálása + beszúrás egyikbe és másikba:  $O(1 + \log n + \log n) = O(\log n)$ .  
*Mintör:* min-kupacból mintör, max-kupacban pedig a megfelelő hely törlése (annyi különbséggel az eredeti algoritmushoz képest, hogy nem a gyökérből, hanem valahonnan középről indítjuk a törlést, de az algoritmus ugyanúgy fog működni).  $O(\log n + \log n) = O(\log n)$ .  
*Maxtör:* ugyanaz mint előbb, csak szerepcserével.
8. Fejet asztalbaverően egyszerű bizonyítás: mivel a gyökérben a legkisebb elem van, ezért nem lehet hamarabb felépíteni, mint a legkisebb elemet megtalálni, ami pedig önmagában is  $\Omega(n)$ .

## 6. Keresés, rendezés

1.  $n - 10$  elég, mert kidobhatunk tetszőleges 9 elemet, utána közöttük a minimális (amit  $n - 9 - 1$  lépésben megtalálhatunk) biztos benne lesz az eredeti legkisebb 10-ben. De ennyi kell is, hiszen tfh van olyan algoritmus, ami legfeljebb  $n - 11$  összehasonlítással talál egy megfelelő elemet. Ekkor az összehasonlítottsági gráfnak legalább 11 komponense lesz. Így, amennyiben a feltételezett algoritmus mutat nekünk egy elemet megoldásként, ennek az elemnek a komponensén kívül az összes többi elemet egy elég nagy számmal csökkentve az összehasonlított elemek viszonya nem változik, de a kiválasztott elem biztos nem lesz benne a legkisebb 10-ben, vagyis a feltételezett algoritmus rossz választ ad.
2. Egy ilyen tömbben biztosan létezik megfelelő elempár, hiszen a nem létezése azt jelentené, hogy bármely két szomszédos elem megegyezik, így  $A[1] = A[2] = \dots = A[n]$ , ami ellentmond a feltételnek. Így viszont a bináris keresés használható, ugyanis ha megfelezzük a tömböt, akkor a felezésnél lévő elem az első és utolsó közül legalább az egyikkel nem egyenlő, így a megfelelő irányban folytatva az eljárást az eredeti tulajdonsággal rendelkező, de feleakkora tömböt kapunk. Vagyis az algoritmus  $O(\log n)$  lépés alatt végez.
3. A tömböt tudjuk rendezni  $O(n \log n)$  lépésben pl összefésüléssel. Így biztos, hogy a többször előforduló elemek példányai szomszédosak lesznek. Már csak annyi a dolgunk, hogy végigolvas-

suk a tömböt, és minden lépésben a következő dolgokat tartjuk nyilván: az előző elem, a legutolsó többszörösként kiírt elem, és az aktuális elem. Ha az aktuális nem egyezik az utolsóval, akkor továbblépünk és az utolsót aktualizáljuk, ha egyezik, és az utolsó kiírt ugyanez volt, akkor továbblépünk, ha nem ugyanez volt, akkor kiírjuk és az utolsó kiírtat aktualizáljuk. A végigolvasás során minden cellában konstans sok műveletet végzünk, így  $O(n)$  lépésben kész vagyunk. A teljes lépésszám  $O(n \log n) + O(n) = O(n \log n)$ .

4. Észrevesszük, hogy  $x$  és  $y$  koordináta független. Egy dimenzióban: ha csak két pont van, akkor két pont között bárhol lehet, rajtuk kívül viszont rosszabb. Tehát a két szélső pont között kell lennie, de rajtuk kívül a két szélső között is stb, tehát páratlan pont esetén a középsőn, páros esetén a középső kettő közül valamelyiken vagy közöttük (indoklás: ha nem így lenne, akkor biztos, hogy tudnánk jobbat csinálni). Tehát koordinátánként egy rendezés, és középső választása, tehát  $O(n \log n)$ -ben bent vagyunk.
5. (a) Ládarendezés,  $3n$  ládával,  $O(n + 3n) = O(n)$ .  
(b)  $n$ -es számrendszerben felírjuk a számokat (ez darabonként 7 osztás, vagyis konstans), utána számjegyenként radix (az  $n$  alapú számrendszer miatt legfeljebb 7 jegyűek lettek a számok), azaz  $O(7n + 7n) = O(n)$ .
6. Először meg kell keresni a töréspontot, ami  $O(n)$ -ben megvan. Az előtte lévő sorozatban a pozitívok sorozata nő, a negatívoké csökken, az utána lévőben pont fordítva. Így a két pozitív részt (a hátsót természetesen megfordítva)  $O(n)$ -ben össze tudjuk fésülni, hasonlóan a két negatívot is. A végén egyszerűen össze kell őket ragasztani. Költség: pesszimistán is  $4O(n)$ , ami  $O(n)$ .
7. Egy 5 hosszú ablakot tolunk végig a sorozaton. Az első elem biztos, hogy az első 5 elem között van, így közöttük minimumot keresünk, így visszavezettük a feladatot egy  $n - 1$  méretű ugyanilyen feladatra. 5 elem közül 4 összehasonlítással tudunk minimumot keresni, ezt  $n$ -szer kell megtenni (a vége miatt kicsit kevesebbszer, de ez mindegy), így a lépésszám  $O(4n) = O(n)$ .
8. A feltételekből következik, hogy biztos van lokális minimum. Bináris keresést használunk, ha az adott körben  $A[i]$  egy lokális minimum, akkor kész vagyunk, ha  $A[i-1] < A[i]$ , akkor a jobb oldalt dobjuk el, egyébként a balt. Tehát egy lépésben vagy megtaláltuk a kérdéses elemet, vagy kaptunk egy feleakkora tömböt, ami ugyanolyan tulajdonságú, mint az eredeti. Ez így  $O(\log n)$  a bináris keresés elve miatt.
9. Menjünk végig a tömbön, és a hibás párokat (amik rosszul állnak) szedjük külön. Ez azért kell, mert a rossz szomszédok közül legalább az egyiket biztos megváltoztattuk (hiba csak változtatásnál lehet), viszont nem tudjuk, melyiket. Így legfeljebb  $2k$  elemet vettünk ki, ezeket  $O(2k \log 2k) = O(k \log k)$  időben tudjuk rendezni, utána  $O(n + k)$  időben összefésülés. A kezdeti végigolvasás  $O(n)$ , vagyis összességében tényleg  $O(n + k \log k)$  lépésben végzünk.
10. A tömböt  $O(n \log n)$ -ben tudjuk rendezni. Triviális megoldásként tehetjük azt, hogy  $\forall i$ -re megkeresünk bináris kereséssel, hogy szerepel-e a tömbben  $b - A[i]$  érték. Ha igen, kész vagyunk, ha egyszer sem találtunk ilyet, akkor pedig nem. Ez legfeljebb  $n$  db bináris keresés, azaz ez a része is megvan  $O(n \log n)$  lépésből. (Persze egy egyszerű dinamikus programmal a második részt lineáris időben is megoldhatjuk.)
11. Rendezzük az intervallumokat növekvő sorrendbe az  $a_i$  koordináta szerint! A következő dolgokat tartjuk nyilván:  $\min_a$ ,  $\max_b$ ,  $sum$ . Kezdetben  $\min_a = a_1$ ,  $\max_b = b_1$ .  
Ha a sorban  $[a_i, b_i]$  intervallum következik: ha  $a_i < \max_b$ , akkor még a vizsgált intervallumban vagyunk, így  $\max_b = \max(\max_b, b_i)$ . Ha  $a_i > \max_b$ , akkor a következő intervallumig biztos van szünet. Ekkor ezt tesszük:  $sum+ = \max_b - \min_a$ ,  $\min_a = a_i$ ,  $\max_b = b_i$ . A legutolsó intervallum feldolgozása után (az utolsó nyitott intervallum-jelölt lezárása miatt)  $sum+ = \max_b - \min_a$ .  
Bizonyítás pl. indukcióval (lényegében ez egy dinamikus programozásos feladat is egyben). Lépésszám: a tényleges számolás  $O(2n)$ -ben megvan, így a rendezéssel együtt a lépésszám  $O(n \log n)$ .
12. Minimális  $y$  koordinátájú pont lesz az egyik, ami meghatározza, a másik pedig az  $y_{\min}$ -es pontból abszolútértékben legnagyobb meredekségű. Ha ezeken kívül esne egy pont, akkor annak a meredeksége abszolútértékben nagyobb lenne. Meredekséget számolni két pont ismeretében konstans, így a lépésszám két minimumkeresés, azaz  $O(n + n) = O(n)$ . (Persze ugyanezt lehet  $x$  koordináta szerint is, valamint minimum helyett maximummal is.)
13. Észrevétel: ha van ilyen  $k$ , akkor ezeket kicserélhetjük a  $k$  legkisebbre, hiszen az összegük így csak csökkenhet, ami továbbra is jó. Tehát elég mindig a  $k$  legkisebbel foglalkozni. Rendezzük a tömböt, és induljunk el az elejétől. Folyamatosan összegezzük, és az első  $k$ -ra ahol igaz, kész is vagyunk, ha pedig végigértünk, és nem volt ilyen, akkor biztos nincs. Ez így rendezés és egy végigolvasás, azaz

$O(n \log n + n) = O(n \log n)$ .

14. Mivel egy hiányzik, egy helyen elromlik a paritás (aminek ellenőrzéséhez elég az utolsó számjegyre rákérdezni). Ezt a helyet keressük meg bináris kereséssel, ami  $O(\log(2^k - 1)) = O(k)$ .
15. A gyorsrendezés pontosan ilyen műveleteket végez, az pont jó lesz itt is, a lépésszám bizonyításával együtt.
16. Csináljunk egy hisztogramot  $y$ -ből, vagyis egy 4 elemű tömbbe írjuk be, hogy melyik karakterből hányat tartalmaz (magyarul ládarendezés). Ez  $O(k) = O(n)$ . Ezután csináljunk egy hasonló tömböt  $x$  első  $k$  elemére ( $T_x$ ), ami szintén ugyanennyi idő. Ha a két tömb megegyezik (ezt a 4 elem miatt  $O(4) = O(1)$ -ben tudjuk ellenőrizni), akkor találtunk egy anagrammát. Ha az ablakot (ami általánosan az  $x_i$ -nél kezdődik) eggyel jobbra mozgatjuk, akkor  $--T_x[x_i]$  és  $++T_x[x_i + k + 1]$  (azaz az elhagyott karaktert kivesszük, a következőt betesszük). A helyesség könnyen látható, lépésszám: mind az  $n$  pozícióra (igazából kicsit kevesebbre) két tömbművelet és egy 4 hosszú egyenlőségvizsgálat (azaz összességében konstans), tehát  $O(n)$ -ben ez a része is megvan.
17. Tfh van olyan algoritmus, ami legfeljebb  $2n - 2$  öh-t használ. Ekkor adjunk neki  $2n$  elemet, és mi leszünk az ellenség. Jelöljük meg magunknak az összes elemet, mert az eddigi 0 kérdés alapján bármelyik lehet a maximális. Tartsuk továbbá azt is nyilván, hogy melyik halmazban mennyi elem lehet globálisan maximális (kezdetben  $n$  és  $n$ ). Ha a feltételezett algoritmus rákérdez két elemre, akkor a következőképpen mondjuk meg a viszonyukat:
  - $a_i ? b_j$ , az eddigi válaszainkból már kiderült a viszonyuk: megmondjuk az igazságot (a lehetséges maximális elemek halmaza ilyenkor nem változik)
  - $a_i ? a_j$ : ha már kiderült a viszonyuk, akkor megmondjuk, ha nem, akkor tetszőleges viszonyt mondunk rájuk (ezt feljegyezzük), és kihúzzuk a kisebbet a jelöltek közül, meg a számlálót is megfelelően változtatjuk
  - $b_i ? b_j$ : mint előbb
  - $a_i ? b_j$ , az eddigi válaszainkból még nem derül ki a viszonyuk: eldönthetjük, hogy melyik legyen a kisebb. Ebben az esetben pontosan 1 elem maximalitása lesz kizárva. Úgy választjuk meg a relációjukat, hogy abból a halmazból zárjuk ki valakinek a maximalitását, amelyikben a több jelölt van. Ezután megcsináljuk a szükséges adminisztrációt (kihúzzunk egy jelöltet és a számot aktualizáljuk).

Meg kell gondolni, hogy az így definiált relációkhoz mindig tudunk számokat kitalálni (a későbbi anyagban lesz erről részletesebben szó, lásd topologikus rendezés). Ezen túl mivel egy kérdéssel legfeljebb 1 elem maximalitása zárható ki, így  $2n - 2$  kérdés után még legalább 2 olyan elem van, ami lehet maximális ( $\equiv$  megválasztható maximálisnak úgy, hogy az eddigi válaszainkkal konzisztens legyen). Viszont a stratégiánk miatt biztos, hogy mindkét halmaz tartalmaz ilyen, így ennyi kérdés biztos nem elég a feladat megoldásához.

18.  $\Omega(n \log_2 n)$ : ugyanúgy kell bizonyítani, mint az öh. alapú rendezések alsó korlátját, csak itt kettő helyett három lehetőségünk van, így  $\Omega(\log_3 n!)$  fog kijönni. Ez viszont nem baj, mert  $\Omega(\log_3 n!) = \Omega(\log_2 n!) \approx \Omega(n \log_2 n)$ .  
 $O(n \log_2 n)$  (vázlatosan):  $O(n)$  lépésből tudunk szélsőértéket keresni: veszünk 3 elemet véletlenszerűen, a középsőt kidobjuk. Ezt megcsináljuk addig, amíg csak 2 marad, ez lesz a minimális és maximális (nem tudjuk, melyik melyik). Az egyiket válasszuk ki, ha ez  $x$ , akkor a  $kozepso(x, a, b)$  vagy az  $a < b$  kérdésre válaszol, vagy az  $a > b$  kérdésre (attól függ, hogy a minimálisat vagy maximálisat választottuk, és egy adott  $x$ -re ez konzisztens). Azaz az így implementált  $<$  függvény tényleg egy rendezést (vagy növekvő, vagy csökkenő; nem tudjuk) definiál a számainkon:
 
$$<(a, b) \{ \text{return } kozepso(x, a, b) == a \}$$

Ez konstans időben számolható, tehát bármely  $<$  operátort használó rendezés használható lesz ezzel a trükkkel, abból pedig van  $O(n \log_2 n)$  lépésszámú.

## 7. Keresőfák, p-f fák, 2-3 fák

1. Tudjuk, hogy az inorder bejárás egy  $m$  csúcsú keresőfából  $O(m)$  lépésben rendezetten olvassa ki az elemeket, valamint azt is, hogy két rendezett,  $a$  és  $b$  hosszú listát az összefésülés egy rendezett listába tud rakni  $O(a + b)$  lépésben. Így inorder kiolvassuk az elemeket a két p-f fából, közben összefésüljük őket, ami  $O(n) + O(k) + O(n + k) = O(n + k)$  lépés.

2. Lehet, hiszen az  $x < 20$ ,  $x < 18$ ,  $x > 3$ ,  $x < 15$ ,  $x > 8$ ,  $x \neq 9$ -et kielégítő szám lehet, vagyis  $9 < x < 15$  közül bármi jó, valamint a keresőfa tulajdonság sem sérül sehol.
3. 2=gyökér (postorder miatt), 7,6,8,5 a jobb részében van, többi a balban (inorder miatt) (3 pont). Két lehetőségünk van,  $x = 6, y = 8$  és  $x = 8, y = 6$ . Ezeket kipróbálva (az akutális részfa gyökerét meghatározva, mint először) kijön, hogy előbbi lehet (3 pont), utóbbi nem (4 pont) (ezeket végig kell számolni!).
4. A minimális elemszám triviális: mivel a gyökérnek van 3 gyereke, így van legalább 3 levél, de ennyi elég is.  
A maximális elemszámhoz meghatározásához először vizsgáljuk meg, hogy egyáltalán hány levél lehet az egyes részfákban. A lehetséges intervallumok a feladat szerint  $[1, \dots, 39]$ ,  $[40, \dots, 49]$  és  $[50, \dots, \infty]$ , vagyis (mivel a 2-3 fa tulajdonságai miatt mindhárom részfa azonos magasságú), a legfeljebb 10 elemű középső részfa fog felső korlátot jelenteni a magasságra. Mivel  $n$  elem tárolása esetén a szintszám legfeljebb  $\lceil \log n + 1 \rceil$ , így ez a korlát 4. Vagyis a másik két részében legfeljebb  $3^{4-1} = 27$  elemet tárolhatunk (ami nem mond ellent az intervallumoknak), így összesen legfeljebb  $27 + 10 + 27 = 64$  elemünk lehet.
5. Többféle adatszerkezetet is felhasználunk.  $FP$ -t tároljuk egy megfelelő keresőfában (pl. egy 2-3 fában) annyi módosítással, hogy az egyes értékekhez egy számlálót is rendelünk, amit pontazonos beszúrás esetén használunk értelemszerűen. Innen az  $FP$ -re vonatkozó műveletek egyértelműek, a lépésszámok is jók.  
 $P_i$  nyilvántartására a ládarendezéshez hasonlóan tartsunk nyilván egy  $P_i[1, \dots, 30]$  tömböt kezdetben 0-ra inicializálva. Ekkor egy  $P_i$  pontos dolgozat beszúrásakor a megfelelő számlálót megnöveljük eggyel ( $O(1)$  lépés). KORLÁT( $i, q$ ) ekkor így számolható:  $\sum_{j=1}^q P_i[j]$ , és mivel  $q \leq 30$ , ez tényleg  $O(1)$  lépés. A kétféle adatszerkezt lépésszámait összevetve láthatjuk, hogy az előírt műveletek az előírt lépésszámokba beférnek.
6. Ekkor a bal részében legfeljebb 16 elemet tárolhatunk, tehát legfeljebb 5 szintje lehet. Mivel minden levél egy szinten helyezkedik el, ezért a középső és jobb oldali részfa is legfeljebb 5 szintű lehet. Ebben az esetben itt 81 és 81 elemet tárolhatunk legfeljebb. Ezeket összeadva kiderül, hogy ennyit muszáj is, hiszen csak így lehet 178 elemet tárolni. Ekkor viszont a második kulcs a gyökérben  $16 + 81 + 1 = 98$  lesz.
7. (a) Két csúcs esetén pl. nem lehet, hiszen a gyerekek muszáj pirosnak lennie, míg a testvére (aki levél), fekete.  
(b) Mi az hogy, nagyon is! Mondjuk az előző példában szereplő.
8. (a) Van olyan fa, amiben ez igaz (pl. legalább 3 elemet tartalmazó teljes fekete fa), de van olyan is, ahol nem (pl. 2 db értéket tároló fa). És mivel a kérdés úgy értelmezendő, hogy „tetszőlegesre igaz-e”, ezért az ellenpélda bizonyítja a nemleges választ.  
(b) Biztos, hogy nem, hiszen a gyökeret nem pirosíthatjuk be, más helyen viszont ez aszimmetrikusan megváltoztatná a fekete magasságot. (Természetesen ide is elég lenne egy ellenpélda.)
9. Legyenek a magasságok  $m_1$  és  $m_2$ ! Az általánosság megsértése nélkül legyen  $m_1 \leq m_2$ . Ekkor (mivel minden kulcs az egyikben kisebb, mint a másikban) a levelek nem fognak átlapolódni. Ezért egyszerűen az  $m_2 - m_1$ -edik szinten beszúrjuk a szokásos algoritmussal a kisebbik fa gyökerét, ami  $O(m_2 - m_1)$  lépésben menni is fog. Azért könnyű eldönteni, hogy melyik a kisebb, mert a magasságokat ki tudjuk olvasni a gyökérben. E feltétel nélkül meg kéne határozni a magasságokat is.
10. Ha nem kell csúcsvágás, akkor a levéltől a gyökérig végig megnöveljük eggyel a tárolt értéket (ez befér az  $O(\log n)$ -be). Ha kell, akkor a vágott csúcsokra újra kell állítani az értéket (a levelek fölött ez triviális, feljebb pedig a gyerekekből konstans időben kiolvasható és összeadható), egyébként a  $+1$  ugyanúgy terjed felfelé. Ez az eset is befér  $O(\log n)$ -be.
11. Lényegében ugyanaz, mint a következő feladat.
12. A levelekben nem csak a konkrét értéket tároljuk, hanem egy számlálót is, hogy az adott elemből hányat tárolunk (így kicsit módosul a törlés és beszúrás). Azt is nyilvántartjuk továbbá minden csúcsra, hogy a belőle kiinduló részében hány elem van (ez ugyanaz, mint a kettővel ezelőtti feladat, csak a többszörös elemekre is kell egy kicsit figyelni). A  $K$ -adik elem egy keresés, ahol (a részfaméretek alapján) mindig arra megyünk, ahol még épp nem lépjük túl  $K$ -t (értelemszerűen közben folyamatosan összegezzük a „kihagyott” részfák elemszámát). A fának  $m$  levele lesz, a szokásos műveletek nem változnak, így a műveletek  $O(\log m)$ -ben menni fognak. (Természetesen célszerű szépen leírni az egyes műveleteket, ez csak egy vázlat. Továbbá  $p$ -f fával is meg lehetne oldani a feladatot, bár azzal

kicsit macerább.)

13.  $2^k - 1$  számot tudunk egy pontosan  $k$  szintű teljes bináris keresőfában tárolni, így ilyet fogunk csinálni. Mivel a fa alakja adott, csak azt kell kitalálni, hogy melyik szám melyik csúcsba kerüljön. Viszont azt is tudjuk, hogy egy bináris keresőfát inorder bejárva a benne tárolt elemeket növekvő sorrendben kapjuk meg. Így meg tudjuk tenni, hogy felépítünk egy „biankó”  $k$  szintű teljes bináris fát, futtatunk rá egy inorder bejárást, és a növekvően rendezett elemeinkből egy csúcs meglátogatásakor mindig odarakjuk a következőt. A számok különbözősége miatt  $2^k - 1 \leq n$ , így a fa felépítése  $O(2^k - 1) = O(n)$ , a számok ládarendezése (minden feltétel adott az alkalmazhatósághoz)  $O(2^k - 1 + n) = O(n)$ , majd a bejárás  $O(2^k - 1) = O(n)$ . Ezeket összegezve a lépésszám  $O(n)$ .
14. Keresőfát definiáljuk (1 pont). Minden egyes pontban a bal részfa mérete kell (1 pont). Algoritmus (dinprog): a csúcsokban ( $v$ ) két számot tartunk nyilván: a bal részfa méretét ( $b_v$ ), valamint a teljes részfa méretét ( $f_v$ ). Levelekben  $b_v = 0$ ,  $f_v = 1$  (ez nyilván helyes). Lentről felfele töltjük az értékeket, ha  $v$  gyerekei  $i$  és  $j$ , akkor  $b_v = f_i$ ,  $f_v = f_i + f_j + 1$  (ha a lentebbi értékek helyesek, akkor induktívan ez is helyes lesz). (algo: 4 pont, indoklás: 2 pont). Minden csúcsot legfeljebb 2-szer vizsgálunk (amikor őt, és amikor a szülőjét töltjük), valamint a csúcsokon megfelelően végigmenni egy bejárással lineáris idő, így a lépésszám  $O(n + 2n) = O(n)$  (2 pont).
15. Fekete magasság definíciója (1 pont).  $x$  és  $y$  azonos színű, mert ha nem lenne az, akkor a gyökérben a fekete magasság a két ágon különböző lenne (3 pont).  $x$  gyerekei különböző színűek, mert ha egyformák lennének, akkor a két ágon sérülne  $x$ -ben a fekete magasság (1 pont).  $x$ -nek tehát van piros gyereke, így ő kötelezően fekete (4 pont). Ezekből következik, hogy  $y$  is fekete (1 pont).
16. Ha nem volt csúcsvágás, akkor mind az 1000 elemet egy 2 gyerekű csúcsba kellett beszúrni. Ilyen (a triviális, 1 elemet tárolás esetet leszámítva) csak vágás során keletkezhet, így mind az 1000 elemet egy eredetileg is létező 2 gyerekű csúcsba kellett beszúrni, vagyis összesen legalább 2000 elem biztos, hogy már a fában volt.
17. Nem, ellenpéldát lehet rá adni egyszerűen: a gyökérben mondjuk legyen 1, ennek (egyetlen) fia 3, ennek pedig két fia, 2 és 4. Az út:  $\{1, 3, 4\}$ , ettől balra van 2, ami nem kisebb, mint 1.
18. A gyökér mindenképp  $y$  szerint a legkisebb lesz, a kupac tulajdonság miatt. A keresőfa tulajdonság miatt egyértelmű, hogy kik lesznek a bal-, és kik a jobb részfában, de itt megint egyértelmű a kupac tulajdonság miatt, hogy ki lesz az első, és így tovább rekurzívan.
19. Ezt így elég macera lenne bizonyítani, ezért egy ennél erősebb dolgot fogunk. Állítás: az ilyen beszúrások során kizárólag a legjobboldali út mentén vannak harmadfokú csúcsok. Indukcióval  $n = 1 \dots$  kevés esetre könnyen látszik. Tfh  $n$ -re igaz, most beszúrjuk  $n + 1$ -et. Ez nyilván a legjobboldalra megy, ha ott éppen egy másodfokú csúcs van, akkor a feltételnek megfelelően csak ott hoz létre egy harmadfokút, más nem változik. Ha csúcsot vágunk, akkor másodfokúak keletkeznek, ami nem zavar minket, valamint a feljebbi szinten ugyanúgy a legjobboldalon keletkezik egy beszúrási igény. Ez felfele rekurzívan pedig pont ugyanúgy működik, ahogy azt láttuk. Mivel az  $O(\log n)$  magasságú fában csak egy gyökér-level úton lehetnek harmadfokú csúcsok, ezért a számuk is legfeljebb  $O(\log n)$ .
20. A gyökér bal és jobb részfájában pontosan  $2^{k-1} - 1$  elem van, a bal oldaliak nagyság szerint a gyökér előttié, a jobb oldaliak a gyökér utániak. Ha a bal oldalról hiányzik valaki, akkor a gyökér  $2^{k-1} + 1$ , ha a jobb oldalról, akkor  $2^{k-1}$ . Ezzel visszavezettük a feladatot egy eggyel kisebb magasságúra, amit rekurzívan ugyanígy tudunk tovább kezelni, tehát a lépésszám a szintszámmal arányos, azaz  $O(\log n)$ .
21. Indukcióval szintszám szerint.  $l = 1$  szintre biztos igaz. Tfh igaz valamilyen  $l - 1$ -ig, belátjuk  $l$ -re is. Az indukciós feltétel szerint a bal és jobb részfája is teljes bináris fa, a bennük tárolt elemek száma  $2^{l_b} - 1$  és  $2^{l_j} - 1$ , ha a bal- és jobb részfa szintszáma  $l_b$  és  $l_j$ . Tfh  $l_b > l_j$ , ekkor  $\frac{2^{l_b} - 1}{2^{l_j} - 1} > \frac{2 \cdot 2^{l_j} - 1}{2^{l_j} - 1} > \frac{2 \cdot 2^{l_j} - 2}{2^{l_j} - 1} = 2$  (az első egyenlőtlenség azért igaz, mert a bal részfa legalább 1 szinttel magasabb, mint a jobb, a második pedig azért, mert a nevezőből egy nagyobb számot vonunk ki, így a tört értéke csökken), ami ellentmond a feltételnek. Hasonlóan  $l_j > l_b$  is lehetetlen, így  $l_b = l_j$ , pont, amit bizonyítani akartunk.

## 8. Hash

1. (a)

0	1	2	3	4	5	6	7	8	9	10
										10
22										10
22									31	10
22				4					31	10
22			15	4					31	10
22			15	4		28			31	10
22			15	4	17	28			31	10
22			15	4	17	28		88	31	10
22		59	15	4	17	28		88	31	10

(b)

0	1	2	3	4	5	6	7	8	9	10
										10
22										10
22									31	10
22				4					31	10
22				4	15				31	10
22				4	15	28			31	10
22				4	15	28	17		31	10
22	88			4	15	28	17		31	10
22	88		59	4	15	28	17		31	10

2.

0	1	2	3	4	5	6	7	8	9	10
				26						
			3	26						
		48	3	26						
14		48	3	26			15			
14		48	3	26	7		15			

3. Vegyük észre, hogy mivel az eredeti táblában a méretkorlát miatt legfeljebb  $M$  elemet tároltunk, valamint az új hash-függvény minden eredeti hash-értéket  $M$  távolságra helyez egymástól, így az új táblában a különböző hash-értékű elemek nem ütközhetnek lineáris próba esetén (ehhez legalább  $M+1$  eddigi ütközés kellett volna). Így ütközés csak azért adódhat a második esetben, mert az ütköző elemek hash-értéke megegyezik. Ezekben az esetekben viszont ugyanezek az ütközések megtörténtek az eredeti táblában is (a próbasorozatok értékfüggetlensége miatt). Tehát legalább annyi ütközés volt az első, mint a második esetben.

4. Kiszámoljuk a hash-értékeket, hogy tudjuk, ki van a helyén és ki nincs.

$x$	5	19	3	33	23
$h(x)$	5	7	9	9	9

Látszik, hogy 5 és 23 a helyén van, míg senki más nincs. Mivel ütközés esetén a lineáris próba miatt mindig eggyel balra lépünk, ezért 23 hamarabb jött, mint 33, az pedig hasonló okokból hamarabb, mint 3, az pedig hamarabb, mint 19 (indoklás: ha nem így lett volna, akkor az adott hely szabad lett volna, és nem a jelenleg látható pozícióban lenne valaki). Az is látszik, hogy 5 nem zavar senkit, vagyis a beszűrési sorrend:  $23 \rightarrow 33 \rightarrow 3 \rightarrow 19$ , és 5 pedig tetszőleges.

5. (a) Nem, pl egy elem volt a táblában, és azt töröltük.  
 (b) Akkor okozhat ez problémát, ha az utolsó üres hely mondjuk  $i$ , de egy  $i$ -nél kisebb pozíciójú  $x$  elemre  $h(x) \geq i$ . Ha ilyen helyzet nincs, akkor jó a tábla. Arra kell gondolni, hogy a táblán végighaladva mindig az utolsó üres hely érdekel minket. Induljunk el a végéről, és minden elemre számoljuk ki  $h(x)$ -et, és nézzük meg, hogy a nyilvántartott utolsó üresről vagy arról túlról származik-e. Ha igen, akkor máshol nem lehetett hibás törlés, hiszen ha mégsem ott lett volna, akkor a törlés előtt is rossz lett volna a tábla. Bőkjük be az utolsó üresen a törlés-bitet! A tábla jó lesz, hiszen helyes törlés után is így nézne ki a tábla. Egy végigolvasásból megvoltunk, tehát lineáris az algoritmus.



6. (a) Mindkét esetben legfeljebb  $n$ , ugyanis 2 hosszú csomóink vannak, és minden csomóban legfeljebb 1 ütközés történhetett (mindkét esetben több ütközés esetén legalább 3 hosszú lenne a csomó).  $n$  pedig lehetett is, pl lineáris próba esetén minden  $3i + 2$  helyre két elemet beszurva, kvadratikusan próba esetén minden  $3i + 1$  helyre két elemet beszurva.
- (b) Lásd az előzőt.
7. Nyilván akkor történik a legtöbb ütközés, ha a (ciklikus értelemben) leghosszabb folytonosan kitöltött rész legvégére próbáljuk beszurni az új elemet. Így elindulunk a tömb elejéről, az első üres sorozat végéig elmegyünk (a ciklikusság miatt célszerű így), ez  $O(n)$ , utána amíg kitöltött sorozatot találunk, egy számláló értékét növeljük. Ha üres jön, akkor a számláló értékét eltároljuk (feltéve, hogy nagyobb az eddig eltároltnál), majd nullázzuk. Ha körbeértünk, akkor vége. Ez még egy végigolvasás, tehát  $O(n) + O(n) = O(n)$  lépésből megvan az egész.
8. Nem az összes lehetséges értéket veszi fel. Ki kell számolni mind a 7 lehetőséget, és látszik.
9. Konstruktívan: legyen  $h(x_1) = h(x_2)$ , ezután  $h(x_i) = h(x_{i-1}) - 1$ . Így csak  $x_1$ -hez és  $x_2$ -höz tartozó érték egyezik meg, mégis látható, hogy tetszőleges nagy  $n$ -re  $n$  hosszú csomóink lesz (és minden beszurás ütközéses volt).
10.  $M = 35$  esetén a  $3 \dots 33$  számok a saját helyükre kerülnek,  $0 \pmod{3}$  helyekre,  $36 \dots 69$  számok  $1 \pmod{3}$  helyekre,  $72 \dots 90$  számok pedig  $2 \pmod{3}$  helyekre, és mindegyik elfér, tehát nincs ütközés.  $M = 36$ -nál csak  $3 \dots 33$  fér be ütközés nélkül,  $36 \dots 69$  pont ugyanezekre a helyekre menne,  $72 \dots 90$  szintén. Innen már ki lehet számolni.
11. (a) Tfh  $t_2 > t_1$ . Ekkor vegyünk egy  $t_2$  hosszú sorozatot a  $2M$  méretű táblában, és nézzük meg, hogy ezek hogyan helyezkednének el az eredeti táblában (a beszurások sorrendje megegyezik!). Vegyük észre, hogy az eredetibe beszuráskor, ha egyéb ütközés nem volt, ugyanezt a sorozatot kapjuk (ciklikus értelemben, modulo aritmetika miatt), vagy esetleg hosszabbat is (lineáris próba miatt). Ez ellentmond a feltételnek, tehát  $t_2 \leq t_1$ .
- (b) Ellenpélda:  $|0|4|2| \pmod{3}$ , de  $|0| |2| |4| \pmod{6}$ ,  $t_1 = 3$ ,  $t_2 = 1$ , és  $3 \not\leq 2 \cdot 1$ .

## 9. Mélységi bejárás, DAG, valamint alkalmazásai

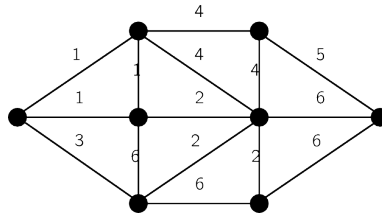
1. A mélységi számok jelentik a sorrendet, ami szerint meglátogattuk a csúcsokat. Tehát  $x$  volt az első, ebből megyél  $y$ -ba,  $y$ -ból  $u$ -ba, majd  $u$ -ból  $v$ -be.  $v$  befejezési száma miatt tudjuk, hogy  $v$ -ből nem tudtunk továbblépni, ezért visszaléptünk  $u$ -ba. Viszont  $u$  még nincs befejezve, így van egy  $u \rightarrow w$  élünk is, ahonnan megint visszaléptünk. Most visszaléptünk  $u$ -ból, majd  $y$ -ból is, és látjuk, hogy  $x$ -ből mentünk  $z$ -be ( $x \rightarrow z$  él). Ezek után visszaléptünk, és elkészültünk. (Ha egy hasonló vizsgafeladatban valaki felrajzolja indoklás nélkül a fát, az legjobb esetben is legfeljebb 4 pontot ér a 10-ből.)
- Természetesen  $G$  nem rekonstruálható, mert csak az adott bejárás szerinti faéleket tudjuk visszaállítani, az előre-, vissza- és keresztélekről nincs információnk (pontosabban néhány él létezését ki tudnánk zárni, de ez nem elég a gráf rekonstruálásához).
2. Felveszünk egy gráfot: csúcsai a táblázat mezői, élei a mezők közötti megengedett lépések (a növekedési feltételt is figyelembe véve), az élsúlyok a mezők értékei. Könnyű látni, hogy egy, a bal alsó saroknak megfelelő csúcsból a jobb felsőig tartó út pont egy megengedett lépegetés, így ebben a gráfban egy leghosszabb utat kell keresni a bal alsó csúcsból indulva. A lépési szabályok miatt a gráf DAG, így  $O(|V| + |E|)$  lépésben ez megtehető, ami jelen esetben  $O(n^2 + 2n^2) = O(n^2)$ , hiszen  $n^2$  csúcsunk van, és minden csúcsból legfeljebb két él indul.
3. Építünk egy gráfot, melynek csúcsai a borítékok, és  $i$  csúcsból pontosan akkor megyél  $j$ -be, ha a  $j$  boríték belerakható  $i$ -be. Ekkor egy irányított út a gráfban megfelel egy helyes borítékláncnak, továbbá egy helyes borítéklánc mindig felírható egy irányított útként a gráfban. Ez a gráf DAG, hiszen ha lenne benne irányított kör, akkor ez azt jelentené, hogy valamely boríték befér saját magába.
- Innen már látszik, hogy ebben a gráfban leghosszabb utat kell keresni, és ha ez legalább  $L$  hosszú, akkor van ilyen lánc (és meg is találtuk), egyébként nincs. Mivel egy DAG-unk van  $b$  csúccsal és  $O(b^2)$  éllel, így a leghosszabb út megkeresése  $O(b + b^2) = O(b^2)$  lépés lesz.
4. Minden csúcsból egy mélységi bejárással ellenőrizzük, hogy ilyen-e. Ha találunk, akkor megállunk, ha nem, akkor pedig nem is lehet (különben valamelyik bejárás megadta volna). Ez összesen  $O(n(n + e)) = O(ne + n^2)$ .
5. A tábla mezőit vesszük egy gráf csúcsainak, az élek a megengedett lépéseket jelölik, súlyuk a mező

- súlya, ahonnan indulnak. Legyen egy forrás, ami az összes első oszlopbelihez hozzá van kötve 0 súllyal, és egy nyelő, amibe az utolsó oszlopból vezetnek élek. Így egy megengedett lépkedés pont egy útnak fog megfelelni a gráfban a forrástól a nyelőig. A gráf DAG, a lépések definíciója miatt.  $|V| = O(n^2)$ ,  $|E| = O(n^2)$  (minden csúcsból legfeljebb 3 él indul, a forrásból pedig  $n$  db), így a legkisebb összegű lépkedés pont egy legrövidebb út a forrásból a nyelőbe, ami  $O(|V| + |E|) = O(n^2)$  lépésben megvan.
6. Vegyünk fel egy  $2n$  csúcsú gráfot, ahol az egyes csúcsok adott személy születésének és halálának felelnek meg ( $S_i$ -re  $v_{i,sz}$  és  $v_{i,h}$ ). A gráf élei jelentik az időbeli megelőzést, azaz  $v_k$ -ból  $v_l$ -be futó él azt jelenti, hogy  $v_k$  esemény  $v_l$  előtt történt. A születési és halál csúcsokat személyenként értelemszerűen összekötjük. A többi él az információkból jön, pl.  $S_i$  személy élete során született  $S_j$  esetén egy  $(v_{i,sz}, v_{j,sz})$  és  $(v_{j,sz}, v_{i,h})$  élet húzunk be (a többi is értelemszerűen). Az információkban pontosan akkor nincs ellentmondás, ha a gráf DAG (egyik irány: ha lenne ellentmondás, akkor az kört jelentene a gráfban; másik irány: ha kör van a gráfban, az egy esemény saját maga előtti bekövetkeztét jelenti, azaz ellentmondás).  $|V| = n$ ,  $|E| = O(n + 2k)$  („élet” élek és állításonként legfeljebb két él), DAG-ság eldöntése mélységi bejárással  $O(|V| + |E|) = O(n + k)$ .
  7. Felveszünk egy irányított gráfot, ahol a csúcsok az akrobaták, két akrobata között akkor fut él (értelemszerű irányítással), ha az egyik ráállhat a másikra. Egy irányított út a gráfban pont egy helyes egymásraállásnak felel meg. A gráf DAG, ha nem lenne az, akkor valaki saját magánál nehezebb és magasabb lenne. A leghosszabb utat keressük, ami mélységi bejárás segítségével  $O(|V| + |E|) = O(n + n^2) = O(n^2)$  ebben az esetben.
  8.  $n$ , ilyen például egy irányított út, ahol pont az iránnyal ellentétesen vesszük a csúcsokat. Több nem lehet, hiszen  $n$  csúcsa van a gráfnak.
  9. Ha egy irányítatlan gráfban  $n - 1$  élnél több van, akkor biztos van benne kör (de ha ennél kevesebb, attól még lehet benne kör!). Így indítunk egy mélységi bejárást, amit  $n$  él vizsgálata után automatikusan leállítunk, egyébként meg magától is a megadott lépéskorlátban megáll.
  10. Vegyünk fel egy gráfot, csúcsai a számítógépek minden időpillanatban  $t_0$  és  $t_1$  között, élei (irányítottak) az üzenetek. Vegyünk fel még éleket az ugyanahhoz a géphez tartozó szomszédos időpontok között is előrefele! Ebben a gráfban az  $x$  gépből pontosan azok érhetők el irányított úton, akik lehetnek vírusosak (ezt kicsit indokolni kell). Így  $x$ -ből egy bejárást indítva megtaláljuk a kérdéses gépeket.  $|V| = O(n(t_1 - t_0))$ ,  $|E| = O(n(t_1 - t_0) + m)$  (saját élek és legfeljebb  $m$  üzenet), így a bejárás lépésszáma  $O((t_1 - t_0)n + m)$ .
  11. Építünk egy gráfot, ami mind az  $n$  közbeeső benzinkúthoz (+ a kezdő- és végponthoz) tartalmaz  $L$  csúcsot;  $v_{i,l}$  jelentése: az  $i$ -edik kúthoz  $l$  liternyi benzinnel érkeztünk. Az élek egyértelműen behúzzhatók a fogyasztás alapján (vagy tankolunk és úgy megyünk tovább, vagy nem tankolunk), így minden csúcsból legfeljebb 2 él indul ki; a feltételek alapján tiltottakat (elfogyó benzin, kevés benzinnel célbaérés) eleve nem húzzuk be. Az élek súlya 0, ha nem tankolunk, és a tankolás költsége (a tankolás mennyiség és az ár ismeretében konstans időben meghatározható), ha tankolunk. Ebben a gráfban egy legrövidebb út pont a legolcsóbb utazást fogja jelenteni. A gráf DAG (csak benzinkutak között, előrefele mennek élek), így a legrövidebb út megkeresésének lépésszáma  $O(|V| + |E|) = O(Ln + 2Ln) = O(Ln)$ , ami pont jó is. (Ha nem csak a szomszédos kutakat néznénk, hanem minden kútból behúznánk az összes élet, ahova eljuthatunk, akkor kutanként és literenként 2 helyett  $O(n)$  él indulhatna ki, így adódna ki  $O(Ln^2)$  – természetesen  $O(Ln) = O(Ln^2)$ , így a leírt megoldás is helyes).
  12. Adott  $i$  számra meg tudjuk nézni, hogy  $x$  és  $y$  csúcs között van-e legfeljebb  $i$  súlyú út, azaz az  $i$ -nél nem nagyobb súlyú élek kihagyásával  $x$ -ből van-e út  $y$ -ba. A legkisebb olyan  $i$  szám, amire van megfelelő út, pont a minimális súlyú út súlya. Mivel csak  $1, \dots, k$  közötti élsúlyaink vannak, bináris kereséssel meg tudjuk keresni a megfelelő  $i$ -t. A szükséges út létezésének eldöntése mehet pl mélységi bejárással, ami  $O(|E| + |V|)$ , viszont itt az összefüggőség miatt  $|V| = O(|E|)$ . Összesen tehát a lépésszám  $O(|E| \log k)$ .
  13. Ha adott egy  $\pi$  permutáció, akkor egy bejárással  $O(e + n)$  lépésben el tudom dönteni, hogy van-e megfelelő,  $k$  hosszú út (csak a  $\pi$  szerinti aktuálisan következő címkéjű éleket vesszük létezőnek a bejárás lépései során). Összesen  $k!$  permutáció lehet, így mindet végig tudjuk nézni  $O(k!(e + n))$  lépésben.
  14. Indukcióval a pontszámra.  $n = 1$ -re triviálisan igaz. Tfh  $n$ -re igaz, kérdés, hogy igaz-e  $n + 1$  pontra? Ha van egy  $n + 1$  pontú irányított gráfunk, akkor véletlenszerűen válasszuk ki egy pontját. A maradék  $n$  az indukciós feltevés szerint felbontható megfelelően két DAG-ra (persze simán lehet, hogy az egyik, másik, vagy mindkettő akár 0 élet tartalmaz, de ez minket nem zavar), ezeknek van egy topologikus

sorrendje. Az  $n + 1$ -edik pont bejövő éleit rendeljük az egyik DAG-hoz, ettől az DAG marad (a topologikus sorrendben az aktuális lesz az utolsó pont, a többbit nem zavarja), a kimenő éleket pedig a másik DAG-hoz (hasonlóan az is DAG marad). Így az állítást igazoltuk (és melleleg a bizonyításunk konstruktív is, azaz ez alapján tényleg tudunk csinálni egy ilyen felbontást).

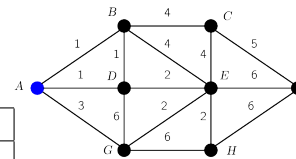
### 10. Minimális költségű feszítőfák

1. A Prim algoritmust fogjuk példaképp futtatni.



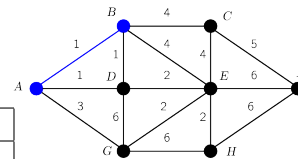
Kezdjük mondjuk  $A$ -val. Kezdeti kitöltés:

	$A$	$B$	$C$	$D$	$E$	$F$	$G$	$H$
KÖZEL	*	$A$	$A$	$A$	$A$	$A$	$A$	$A$
MINSÚLY	*	1	$\infty$	1	$\infty$	$\infty$	3	$\infty$



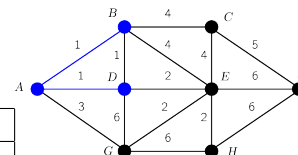
Furcsának tűnhet, hogy mondjuk  $F$ -hez is  $A$  van közel, de a köztük levő súly úgyis végtelen, ezért ez nem probléma. Először mondjuk  $B$ -t választjuk, hozzá  $A$  van közel, tehát  $AB$  élet vesszük be.  $B$  így bekerül az  $U$  halmazba, az ő dolgait kitakarítjuk, és a belőle kiinduló élekkel frissítjük a táblázatot, pl.  $C$ -hez  $B$ -ből 4-gyel jutunk, ez közelebb van, mint  $A$ -ból  $\infty$ -nel.

	$A$	$B$	$C$	$D$	$E$	$F$	$G$	$H$
KÖZEL	*	*	$B$	$A$	$B$	$A$	$A$	$A$
MINSÚLY	*	*	4	1	4	$\infty$	3	$\infty$



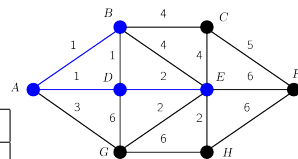
Most csak  $D$ -t vehetjük be, méghozzás az  $AD$  éllel:

	$A$	$B$	$C$	$D$	$E$	$F$	$G$	$H$
KÖZEL	*	*	$B$	*	$D$	$A$	$A$	$A$
MINSÚLY	*	*	4	*	2	$\infty$	3	$\infty$



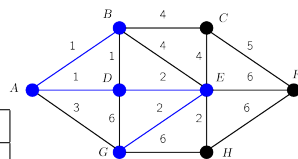
Érdeemes megfigyelni, hogy  $AD$  él nem játszott, hiszen már  $A \in U$ , valamint  $E$ -hez közelebb van  $D$ , mint  $B$ -hez, tehát az frissül. Most  $E$  jön, a  $DE$  éllel:

	$A$	$B$	$C$	$D$	$E$	$F$	$G$	$H$
KÖZEL	*	*	$B$	*	*	$E$	$E$	$E$
MINSÚLY	*	*	4	*	*	6	2	2



Most  $G$ ,  $EG$  éllel:

	$A$	$B$	$C$	$D$	$E$	$F$	$G$	$H$
KÖZEL	*	*	$B$	*	*	$E$	*	$E$
MINSÚLY	*	*	4	*	*	6	*	2



aztán  $H$ ,  $EH$ -val:

	A	B	C	D	E	F	G	H
KÖZEL	*	*	B	*	*	E	*	*
MINSÚLY	*	*	4	*	*	6	*	*

majd  $C$ ,  $BC$ -vel:

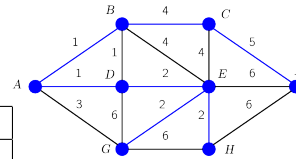
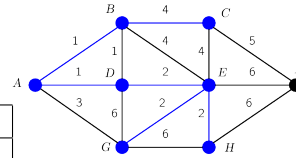
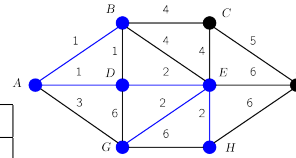
	A	B	C	D	E	F	G	H
KÖZEL	*	*	*	*	*	C	*	*
MINSÚLY	*	*	*	*	*	5	*	*

végül  $F$ , a  $CF$  él felhasználásával:

	A	B	C	D	E	F	G	H
KÖZEL	*	*	*	*	*	*	*	*
MINSÚLY	*	*	*	*	*	*	*	*

Kész vagyunk.

- A Kruskal algoritmus biztosan minimális feszítőfát ad, ezért megnézhetjük, hogy mit csinálna. Először az 1 súlyú élek közül fog választani, belőlük tud építeni egy  $k - 1$  élű fát. Többet nem, hiszen akkor biztos kört csinálna. Utána a 2 súlyúak következnek, belőlük  $l - 1$ -et lehet választani hasonló okok miatt. A végén az  $x_i$  és  $y_j$  pontokon lesz két feszítőfa, ezeket muszáj egy 3 súlyú éllel összekötni. Tehát  $(k - 1) \cdot 1 + (l - 1) \cdot 2 + 3 = k + 2l$ . (Fontos észrevenni, hogy az indoklás azért ilyen egyszerű, mert a választások helyességét nem kell külön bizonyítani, hiszen azt tudjuk, hogy az algoritmus helyesen működik.)
- Gyakorlatilag az UNIÓ-HOLVAN adatszerkezet használható változtatás nélkül, mivel elég a gráf komponenseit nyilvántartani. Van út két csúc között, ha egy komponensben vannak, és nincs ha nincsenek. Egy csúc felvétele létrehoz egy új komponens (halmazt). Egy új él vagy összeköt két komponens (képezi az uniójukat), vagy nem változtat semmin (attól függően, hogy komponensen belül vagy kívül megy). A lépésszámok megfelelősége következik az UNIÓ-HOLVAN adatszerkezet műveleteinek lépésszámaiból (természetesen felsorolandó).
- Először belátjuk, hogy amennyiben minden él súlya különböző, akkor pontosan 1 minimális súlyú feszítőfa van. Tfh ez nem így van, azaz  $\exists F_1, F_2$  feszítőfa, mindkettő súlya minimális, és  $\exists e$  él úgy, hogy  $e \in F_1, e \notin F_2$ . Ekkor  $e$  behúzásával egy kör keletkezik  $F_2$ -ben, ez a kör legyen  $C$ . Ha  $e$  súlya kisebb, mint  $C$  valamely élének súlya, akkor  $C$ -nek ezt az élet  $e$ -re cserélve  $F_2$  súlyát csökkenthetnénk, vagyis  $F_2$  nem lehetett minimális. Ha  $e$  súlya nagyobb  $C$  bármely élének súlyánál, akkor viszont a piros szabály alkalmazható (sőt, alkalmazandó!) rá, vagyis nem lehetett volna  $F_1$  része. Tehát a minimális súlyú feszfa egyértelműsége miatt elég azt ellenőrizni, hogy  $f$  része-e ennek a feszfának. Sajnos magát a feszfát túl drága lenne megkeresni, de nem is kell: nézzük meg, hogy alkalmazható-e  $f$ -re a piros szabály. Azaz keressünk  $f$ -et tartalmazó kört  $G$ -ben (pl bejárással) úgy, hogy  $f$ -en kívül csak a nála kisebb súlyú éleket használhatjuk. Ha van ilyen kör, akkor  $f$ -re alkalmazható a piros szabály, vagyis nem része a minifeszfának, ha pedig nincs, akkor  $f$ -re nem alkalmazható a piros szabály, tehát része a minifeszfának. Mivel csak egy bejárást csináltunk (pl mélységi), a lépésszám  $O(|V| + |E|)$ .
- A keresett részgráfba minden  $-1$  súlyú élet be kell válogatni (ha nem tennénk, akkor egy ilyen él bevitelével jobbat kapnánk). Mostmár csak az összefüggőséget kell biztosítani. Bejárással határozzuk meg a  $-1$  súlyú élek által feszített részgráf komponenseinek számát, ez legyen  $k$ . Ekkor a súly  $\sum_{s(e)=-1} -1 + k - 1$ , hiszen a komponenseket  $k - 1$  darab 1 súlyú éllel kell és elég összekötni (ennyi elég és lehetséges  $G$  öf miatt; ha kevesebb, akkor meg marad külön komponens). A lépésszám a bejárásé:  $O(|V| + |E|)$ .
- Először megmaradt járdákat használva öf komponensek keresése (gráf csúcsai: pontok; élei: megmaradt



járdák), majd a komponensek között min ktg feszfa (gráf csúcsai: előző gráf öf komponensei; élei: lerakható pallók a megfelelő súllyal).

**Helyesség:** Egy, a járdák által meghatározott komponensen belül biztos, hogy mindenholnan mindehova el tudunk jutni palló nélkül. Két komponens között palló nélkül lehetetlen átjárni, hiszen ha lehetne, egy komponensbe tartoznának. Komponensen belül felesleges pallót építeni, hiszen a költsége pozitív, és elhagyásával nem rontjuk el a mindehova eljuthatóságot. Ezek alapján tényleg egy minktg feszfa kell a komponensek között.

**Lépésszám:** Gráfépítés  $O(n^2)$ , bejárás és a komponensek megjelölése  $O(n^2)$  (mx-os megadás!; persze éllistával is csinálhatjuk, ugyanennyi lesz), második gráf csúcsai legfeljebb  $O(n)$ , élei legfeljebb  $O(n^2)$ , tehát felépítése  $O(n^2)$ , minktg feszfakeresés mx-os megadás esetén  $O(n^2)$ . Összesen tehát  $O(n^2)$

7.  $f$  elhagyásával  $F$  két komponensre esik, ezeket egy bejárással meg tudjuk keresni (és minden csúcsonál megjegyezzük, hogy melyik komponensbe tartozik).  $f$ -et pontosan a két komponens között futó élek tudják helyettesíteni, így közülük a minimális súlyú súlyáig tudjuk  $f$ -et növelni. Tehát  $G$  ( $F$ -en kívüli) összes élén végigmegyünk, és a két komponens között futók súlyának minimumát megkeressük. Lépésszám: bejárás és éleken végigmenés:  $O(n^2 + n^2) = O(n^2)$ .
8. Vázlatosan: a legkisebb súlyú feszítőfától a különböző élsúlyok miatt pontosan egy élben fog eltérni (biz. vázlat: tfh legalább kettőben, így az egyiket a minfeszfa megfelelő élére cserélve kapnánk egy kisebb súlyút ami még nem min, tehát a jelöltünk nem lehetett a második legkisebb). Így keresünk egy feszítőfát, és abból egyenként az összes  $(n - 1)$  élet kék helyett pirosra színezzük, és keresünk helyette egy másik kék élet. A kapott  $n - 1$  érték közül a legkisebbet vesszük. Lépésszám:  $O(|E| \log |E| + |V| \cdot |E|) = O(|V| \cdot |E|)$ , feszfakeresés és utána  $n - 1$  kék szabály.
9. A piros-kék algoritmus a  $G'$ -ben pirosra színezett éleket  $G$ -ben is pirosra színeznék (a kékeket nem biztos, hogy kékre! – szorgalmi feladatként mutassunk egy példát, ahol egy  $G'$ -beli kék él  $G$ -ben piros lesz!). Ezért elég, ha  $G'$  feszítőfájához vesszük hozzá  $v_1$ -et, és ebben a gráfban keresünk minktg feszfát. Ez Prim vagy Kruskal algoritmussal éllistás megadásban  $O(|E| \log |E|)$ , és ebben az esetben legfeljebb  $n - 1 + n$  élünk van (feszítőfa és új élek), így a költség  $O(n \log n)$  lesz.
10. Legalább  $n$ , hiszen  $n - 1$  esetén fa lenne, így ekkor csak egy feszítőfája lehetne. Ha van benne egy  $k$  hosszú kör, akkor a kör bármely élet elhagyva a maradék kiegészíthető feszítőfává, így ekkor legalább  $k$  darab van neki. Vagyis csak 3 hosszú kör(ök) lehet(nek) a gráfban. Ha több 3 hosszú kör is van, akkor belőlük függetlenül el lehet hagyni éleket, így a feszítőfák száma 3 egész számú többszöröse lesz. Vagyis a gráfban pontosan egy darab, 3 hosszú kör van (azaz egy háromszög, aminek a csúcsairól fák lóghatnak le), így egy ilyen gráfnak legfeljebb  $n$  éle lehet. Fentieket összerakva (legalább és legfeljebb  $n$ ) pontosan  $n$  éle van.
11. Borúvka algoritmusának első két menetét futtatjuk (első menet után legalább  $\lfloor \frac{n}{2} \rfloor$  él van meg, második után legalább  $\lfloor \frac{3n}{4} \rfloor$ ).
12. A piros-kék algoritmus helyességének bizonyításában csak két helyen foglalkozunk a célfüggvénnyel, és mindkét helyen ilyen értelemben: ha két él közül a nem nagyobb súlyút veszem be a feszfába, akkor nem lehet rosszabb a célfüggvény értéke, mintha a nem kisebb súlyút venném be. Ez az állítás nyilván igaz a mostani költségdefiníciókra is, a bizonyítás többi része pedig természetesen változatlanul igaz marad.

## 11. Bonyolultságelmélet

1. (a) Tanú: megfelelő színezés. Méret:  $kn$ , ami polinomiális az input méretében. Ellenőrzés: páronként a pontokat,  $O(n^2)$ , ami polinomiális.  $P$ -beliségről vagy  $coNP$ -beliségről nem tudunk mit mondani.  
(b) Tanú: egy ilyen kör, ez jó mert blabla. Amúgy  $P$ -beli.  $coNP$ -beliségre szemléletes tanú egy ptlan fokszámú pont (+öf ellenőrzése) (de természetesen a  $P$ -beliség minden további nélkül bizonyítja a  $coNP$ -beliséget).  
(c) Tanú:  $k$  független pont, ez jó mert blabla.  $P$ -beliségről vagy  $coNP$ -beliségről nem tudunk mit mondani.
2. A piros és kék színeket legfeljebb  $n \binom{n-1}{2} = O(n^3)$  féle módon oszthatjuk ki, ami polinomiális. Minden kiosztáshoz a maradék gráfot két színnel kell színeznünk, ami  $P$ -beli feladat (páros gráfság eldöntése). Így polinomszor kell egy polinom költségű algoritmust futtatni, ami polinom időben megy.
3. (a) Van-e benne 1 ftln pont? Ha igen, van-e benne 2? Ha igen, van-e benne 3? Az első NEM válasznál (mondjuk  $k$ -nál tudjuk, hogy  $\alpha(G) = k - 1$ . Lépésszám  $O(n)$  a feltételezés miatt. Lehet bináris kereséssel is, így a lépésszám  $O(\log n)$ .

(b) Meghatározzuk  $\alpha(G)$ -t mondjuk  $O(\log n)$  időben. Választunk egy pontot, kitöröljük a gráfból, és megkérdezzük a maradékról, hogy van-e benne  $\alpha(G)$  független pont. Ha igen, akkor az adott pont nem lehetett egy max. ftn ponthalmaz része, hiszen ekkor létezne a gráfban  $\alpha(G) + 1$  független pont. Ha nem, akkor pedig biztos egy max. ftn halmazbeli pontot töröltünk. Ugyanezt megcsináljuk az összes többi pontra is úgy, hogy a törölt pontokat nem állítjuk vissza. Ha megvan  $\alpha(G)$  pont, akkor kész vagyunk, és legfeljebb  $O(n)$  lépésszámunk van (a feltételezés figyelembevételével).

4. Adott  $G$  gráf, amiről el kell dönteni, hogy színezhető-e 3 színnel. Ezt kell visszavezetni 4 színnel színezésre. Vegyünk fel egy új pontot ( $v$ ), ezt kössük hozzá az összes  $G$ -beli ponthoz. Legyen ez a gráf  $G'$ . Állítás:  $G \in 3SZIN \Leftrightarrow G' \in 4SZIN$ .  $G \in 3SZIN \Rightarrow G' \in 4SZIN$ : vegyük  $G$  egy 3 színnel színezését, és adjuk  $v$ -nek a negyedik színt  $G'$ -ben. Ez a színezés jó lesz.  $G \in 4SZIN \Rightarrow G' \in 3SZIN$ : vegyük  $G'$  egy 4 színnel színezését. Ekkor  $v$  színe különbözik az összes csúcs színétől, így a többi csúcs pont  $G$  egy jó 3 színnel színezése szerint van színezve. Az átalakítás polinomiális.
5. A nyelv:

$$L = \{(G, k) \mid G \text{ irányítatlan csúcssúlyozott teljes gráf, amiben van legfeljebb } k \text{ levélsúlyú feszítőfa}\}$$

Ez  $NP$ -teljes.  $NP$ -beli, mert egy jó tanú egy ilyen feszítőfa (ellenőrzés polinom időben megy, a mérete is polinom). Adunk egy  $H$ -út  $\leftarrow L$  Karp-redukciót. Ha egy  $G$  gráfban Hamilton-utat keresünk, akkor minden csúcsához rendeljük 1 súlyt, és az így keletkezett  $G'$  gráfban keressünk legfeljebb 2 levélsúlyú feszítőfát! Állítás:  $G \in H\text{-út} \Leftrightarrow (G', 2) \in L$ .  $G \in H\text{-út} \Rightarrow (G', 2) \in L$ :  $G$  egy Hamilton-útja pont egy két levélű, 2 súlyú feszítőfa  $G'$ -ben.  $(G', 2) \in L \Rightarrow G \in H\text{-út}$ : egy ilyen feszítőfának pontosan 2 levele kell, hogy legyen (ha több lenne, nagyobb lenne a súlya, egy fában pedig mindig van legalább 2 levél). Ez pedig pont egy olyan utat jelent, ami tartalmazza a gráf összes csúcsát, tehát  $G$  egy Hamilton-útját. A csúcsoknak súlyt adni lehet polinom időben.

6.  $P$ -beli, mert polinom időben ellenőrizhetjük, hogy teljes-e a gráf. Ha nem, akkor a válasz nem, ha pedig igen, akkor mindenképp van benne Hamilton-kör, így a válasz igen.
7. (a) Ez nem mond semmit  $X$ -ről, hiszen  $P \subseteq NP$ , és minden  $NP$ -beli visszavezethető tetszőleges  $NP$ -teljesre.  
 (b) Ekkor  $X$   $NP$ -nehéz, ami  $X \in P$  esetén  $P = NP$ -t jelente, amit feltettünk hogy nem így van. Így  $X$  nem  $P$ -beli.  
 (c) Ettől még lehet  $P$ -ben is, hiszen  $P \subseteq NP$ .
8. (a) Tanú: ilyen párosítás. Méret:  $O(n)$ , ellenőrzés  $O(n)$ , tehát jó. (Egyébként pl magyar módszer polinom idejű, ezért ez  $P$ -ben van, így értelemszerűen  $coNP$ -ben is. A Hall-tétel segítségével szemléletesen lehet  $coNP$ -beliséget bizonyítani: tanú egy olyan  $X \subseteq A$  lesz, hogy  $|X| < |N(X)|$ .  
 (b) Tanú: ilyen párosítás. Méret:  $O(n)$ , ellenőrzés  $O(n)$ , tehát jó. (Egyébként pl magyar módszer polinom idejű, ezért ez  $P$ -ben van, így értelemszerűen  $coNP$ -ben is.)  
 (c) Tanú: egy ilyen kör, ez jó mert blabla. Egyébként  $P$ -beli, mert a kör legfeljebb  $O(\frac{n!}{(n-100)!})$  féleképpen állhat elő, ami  $O(n^{100})$  (ha minden lehetséges pontsorrendet megvizsgálunk). Ebből következően  $coNP$ -beli is.  
 (d) Tanú: egy ilyen kör, ez jó mert blabla.  $P$ -beliségről vagy  $coNP$ -beliségről nem tudunk mit mondani. ( $n^k$  nem polinomiális, ha  $k$  az input része!)  
 (e) Ez a részhalmazösszeg probléma, szépen megfogalmazva. Tanú: egy jó indexhalmaz, azaz egy megfelelő részhalmaz, ez jó mert blabla.  $P$ -beliségről vagy  $coNP$ -beliségről nem tudunk mit mondani.

9. A probléma:

$$L = \{(G, k) \mid G \text{ irányítatlan csúcssúlyozott teljes gráf, amiben van legfeljebb } k \text{ levélsúlyú feszítőfa}\}$$

$NP$ -beli, mert egy jó tanú egy ilyen feszítőfa (ellenőrzés polinom időben megy, a mérete is polinom).

10. Legegyszerűbben úgy, hogy megkérdezzük: kiszínezhető-e 1 színnel? 2-vel? 3-mal?... Az első igen válasznál megvan  $\chi(G)$  értéke. Legfeljebb  $n$ -szer polinom idő, ami összességében polinom. (Ügyesebbek csinálhatják bináris kereséssel is.)
11. Vegyünk fel  $G$  mellé egy  $K_{\chi(G)}$ -t, azaz  $\chi(G)$  pontú teljes gráfot. Az egyes pontjai jelentsék a színeket, amikkel  $G$ -t ki akarjuk színezni. Ha egy darab pont ( $k$ ) kivételével  $K_{\chi(G)}$  minden pontját összekötjük  $G$  egy  $v$  pontjával, az azt jelenti, hogy  $v$ -t  $k$  színűre színeztük (könnyű belátni, hogy más színt nem kaphat). Vegyük sorra  $G$  pontjait, és kezdjük el próbálgatni a színeket. Ha  $v$  pontra  $k$  színt előírunk, akkor megkérdezzük, hogy az aktuális  $G \cup K_{\chi(G)}$  gráfunk színezhető-e  $\chi(G)$  színnel. Ha igen, akkor

találtunk  $v$ -nek egy jó szint és folytathatjuk  $v + 1$ -gyel, ha nem, akkor töröljük a  $v$   $k$ -ra színezéséhez szükséges éleket és próbálkozunk a  $k + 1$  színnel.

A végén minden csúcson lesz egy jó színe. Legfeljebb  $n$  csúcsra  $n$  szint próbálunk ki, egy próba pedig egy  $2n$  csúcsú gráfról kérdez színezhetőséget, vagyis a feltételezés figyelembevételével összességében polinomiális lesz az algoritmus.

12. Ha tudjuk, hogy a síkbarajzolhatóság eldöntése  $P$ -beli, akkor ez minden további nélkül bizonyítja az állítást. Ha esetleg ezt nem tudnánk:

$\pi \in NP$ : tanú: egy konkrét lerajzolás (pl a pontok koordinátaival). Méret:  $2n$ , azaz polinomiális. Ellenőrzés: élpáronként metszéspont-számítás, egyenként polinomiális, összességében  $O(e^2)$  darabot kell, ami még mindig polinomiális.

$\pi \in coNP$ : tanú: egy Kuratowski-gráffal topologikusan izomorf részgráf. Méret: mivel részgráf, nem lehet nagyobb  $G$  méreténél, azaz biztos polinomiális. Ellenőrzés: részgráfság ellenőrzése (polinomiális), valamint a másodfokú pontok kezelése után (polinomiális) Kuratowski-gráfság ellenőrzése (konstans).

13. (a)  $P$ -beli, az összes lehetséges 15 pontú részgráf száma  $\binom{n}{15} = O(n^{15})$ , így mindegyiket meg tudjuk vizsgálni.

(b)  $NP$ -teljes.  $NP$ -beli: tanú egy ilyen kör (poli ell, poli méret).  $NP$ -nehéz:  $H$ -kőről Karp-redukció:  $G'$  legyen  $G$  kiegészítve  $99n$  izolált ponttal, tehát  $n' = 100n$ . Ha  $G$ -ben van  $H$ -kör, akkor  $G'$ -ben ez  $G$ -ben pont egy  $n = n'/100$  hosszú kör. Ha  $G'$ -ben van egy  $n'/100 = n$  hosszú kör, ez (a  $99n$  izolált pont miatt) csak úgy lehet, hogy a kör kizárólag a  $G$ -ben is meglévő pontokat tartalmazza. Ez viszont pont egy  $H$ -kör  $G$ -ben. Az átalakítás polinomiális.

(c) Ez a  $H$ -út probléma szinonímája, amiről tudjuk, hogy  $NP$ -teljes.

(d)  $NP$ -teljes.  $NP$ -beli: tanú egy ilyen feszfa, ez jó, mert  $\dots$   $NP$ -nehéz: az előző problémát vezetjük vissza rá.  $G'$  legyen olyan, hogy  $G$  minden pontjához hozzákötünk egy elsőfokú pontot. Ha  $G$ -ben van olyan feszfa, amiben a maxfok legfeljebb 2, akkor  $G'$ -ben ehhez hozzávéve az új elsőfokú pontokat pont egy jó feszfát kapunk. Ha  $G'$ -ben van olyan feszfa, amiben a maxfok legfeljebb 3, akkor az új elsőfokú pontokhoz vezető élek ebben biztos benne vannak. Ha ezeket elhagyjuk, akkor a feszfa minden egyes pontjának a fokát pontosan eggyel csökkentjük, de az eredeti gráfban ez még mindig feszfa marad, vagyis pont  $G$  egy megfelelő feszfáját kapjuk. Az átalakítás polinomiális, hiszen  $n$  új csúcsot és  $n$  új élet vettünk fel.

14.  $NP$ -beliség: tanú egy ilyen út, méret  $O(n)$ , ellenőrzés  $O(n)$ , tehát polinomiális.  $NP$ -nehézség: a  $H$ -út problémát fogjuk rá visszavezetni ( $H$ -út  $\prec$   $L$ ). A  $G$  gráfból, amiben  $H$ -utat kell keresni, csinálunk egy súlyozott gráfot, minden élsúlyt 1-re állítunk, és legalább  $n - 1$  súlyú út létezését kérdezzük (ez a gráf legyen  $G'$ ). Egyik irány: ha van  $G$ -ben  $H$ -út, akkor  $G'$ -ben van legalább  $n - 1$  súlyú út, mert a  $G$ -beli  $H$ -út pont ilyen  $G'$ -ben. Másik irány: ha van  $G'$ -ben legalább  $n - 1$  súlyú út, akkor ez az 1 súlyok miatt legalább  $n - 1$  élből áll, tehát minden csúcson keresztülmegy, vagyis ez pont egy  $H$ -út  $G$ -ben. Az átalakítás triviálisan polinomiális.

15.  $P$ -beli. Először ellenőrizzük a síkgráfságot polinom lépésben (erről akkor is illik tudni, hogy  $P$ -beli, ha nem írja a feladat). Ha nem, akkor a válasz NEM. Ha igen, és  $k > 4$ , akkor a válasz szintén NEM, hiszen  $K_5$  biztos nincs benne, ha síkgráf (lásd Kuratowski-tétel). Egyébként legfeljebb  $O(\binom{n}{4}) = O(n^4)$  lehetőséget kell végignézni.

16.  $NP$ -teljes.  $NP$ -beliség: tanú egy ilyen kör, mérete  $O(n)$ , ellenőrzés szintén polinomiális.  $NP$ -nehézség:  $H \prec L$  visszavezetést csinálunk.  $G$  gráfról kérdés, hogy van-e benne  $H$ -kör. Csinálunk egy  $G'$  gráfot úgy, hogy  $G$  minden csúcsához felveszünk egy új csúcsot, ami kizárólag a saját párjával van összekötve, és  $G'$ -ről kérdezzük meg, hogy van-e benne ilyen  $C$  kör. Helyesség: egyik irány: ha  $G$ -ben van  $H$ -kör, akkor ez egy jó  $C$  kör  $G'$ -ben, hiszen az extra pontok közül mindegyik kapcsolódik hozzá, az összes többi pedig a körben van. Másik irány: ha van ilyen  $C$  kör  $G'$ -ben, akkor az összes extra elsőfokú pontnak kapcsolódnia kell hozzá, továbbá ezek a pontok nyilván nem lehetnek benne. Vagyis a kör pont az összes  $G$ -beli ponton megy át, azaz ez  $G$ -nek egy  $H$ -köre. Az átalakítás  $n$  él és csúcs felvétele, ami nyilván polinomiális.

- 17.

$$\pi = \{h_1, \dots, h_n, k_1, \dots, k_n, Z, K \mid \forall h_i, k_i > 0; Z \in \mathbb{Z}^+; \exists \text{zsákkiosztás, hogy a megelőzött kár} \geq K\}$$

Érdeemes észrevenni, hogy nem a kár minimalizálása felől közelítjük meg a problémát, hanem a megelőzött kár maximalizálása felől. Ez a probléma  $NP$ -teljes.  $NP$ -beli, mert jó tanú egy zsákkiosztás, mérete, ellenőrzése polinomiális.  $NP$ -nehéz, HÁTIZSÁK  $\prec$   $\pi$  visszavezetést tudunk triviálisan csi-

nálni: ha adott egy hátizsák feladat, akkor  $s_i$  súly legyen az árvizes problémában  $h_i$ ,  $c_i$  érték legyen  $k_i$ , az  $S$  súlykorlát legyen a  $Z$  zsákkorlát, a legalább elérendő  $C$  érték pedig a legalább megelőzendő  $K$  kár. A helyesség és a visszavezetés polinomiálissága triviális.

18.  $X \in P$ , hiszen a bináris felírásban a 2-hatványsságot triviális ellenőrizni (pontosan 1 darab 1-es érték lehet).  $Y$  probléma a 3SZÍN komplementere, így  $coNP$ -teljes.  $X \prec Y$  biztosan van, hiszen egy  $coNP$ -teljes problémára bármely  $coNP$ -beli, így bármely  $P$ -beli visszavezethető ( $P = coP \subseteq coNP$ ). Ha létezne  $Y \prec X$  visszavezetés, akkor  $P = coNP$  lenne, amiből következne  $P = NP$ , ez pedig ellentmond a feltevésnek, tehát ilyen nem lehet.
19.  $X \in P$ , hiszen mint tudjuk, DAG-ban a leghosszabb út keresése polinomiális. 3SZÍN ismert  $NP$ -teljes. Az  $NP$ -teljesség definíciója miatt  $X \prec 3SZÍN$  biztos létezik (bármely  $NP$ -beli visszavezethető bármely  $NP$ -teljesre, valamint  $P \subseteq NP$ ). Ha létezne  $3SZÍN \prec X$ , akkor minden  $NP$ -beli probléma polinom lépésben megoldható lenne, így  $P = NP$  lenne. Tehát egy ilyen visszavezetés létezéséről nem tudunk mit mondani (persze ha valaki mégis bizonyítja vagy a létezését, vagy a nemlétezését, akkor sok jó dolgot kap).
20. A Karp-redukció helyessége nem függ az igen vagy nem választól (acsa feltétel van benne). Először tfh egy probléma  $NP$ -beli. Igazolni szeretnénk, hogy amennyiben egy konkrét kérdésre a válasz nem, akkor létezik erre megfelelő tanú. Mivel 3SZÍN-re vissza tudjuk vezetni, az pedig a feltétel szerint  $\in coNP$ , így a visszavezetés utáni feladatra NEM válasz esetén létezik megfelelő tanú, de a visszavezetés tulajdonságai miatt ez jó tanú a kérdéses feladatra is. Ez azt jelenti, hogy tetszőleges  $NP$ -beli probléma nem válaszára létezik megfelelő tanú, vagyis  $NP \subseteq coNP$ . Ugyanezzel a gondolatmenettel a másik irányba beláthatjuk, hogy  $coNP \subseteq NP$ . A kettőből pedig következik, hogy  $NP = coNP$ .

## 12. Egészértékű programozás

1. Minden  $(v_i, v_j)$  élhez felvesszünk egy  $x_{i,j}$  (bináris) változót (binárisságot biztosító korlátok: (3,4,5)), ami 1, ha  $(v_i, v_j)$  a keresett párosításban van, egyébként 0. A kiválasztott élek számát akarjuk maximalizálni, ami megfelel a változók összegének maximalizálásának (1). Bármely pontra legfeljebb 1, a párosításban szereplő él illeszkedhet (2).

$$\max \sum_{i=1}^n x_{i,j} \quad (1)$$

feltéve, hogy

$$\sum_{(v_i, v_j) \in E} x_{i,j} \leq 1 \quad \forall i = 1 \dots n \quad (2)$$

$$x_{i,j} \leq 1 \quad \forall (v_i, v_j) \in E \quad (3)$$

$$x_{i,j} \geq 0 \quad \forall (v_i, v_j) \in E \quad (4)$$

$$x_{i,j} \in \mathbb{Z} \quad \forall (v_i, v_j) \in E \quad (5)$$

2. Minden ponthoz felvesszünk egy  $x_i$  (bináris) változót (binárisságot biztosító korlátok: (3,4,5)), ami 1, ha a  $v_i$  pont a keresett részgráfban található, egyébként 0. A bevett pontok számát akarjuk maximalizálni, ami megfelel a változók összegének maximalizálásának (1). Két pont nem lehet benne egy teljes részgráfban, így ha összekötetlenek, egyszerre nem lehet őket kiválasztani (2).

$$\max \sum_{i=1}^n x_i \quad (1)$$

feltéve, hogy

$$x_i + x_j \leq 1 \quad \forall i, j : (v_i, v_j) \notin E \quad (2)$$

$$x_i \leq 1 \quad \forall i = 1 \dots n \quad (3)$$

$$x_i \geq 0 \quad \forall i = 1 \dots n \quad (4)$$

$$x_i \in \mathbb{Z} \quad \forall i = 1 \dots n \quad (5)$$



3. Minden ponthoz felvesszünk egy  $x_i$  (bináris) változót (binárisítást biztosító korlátok: (3,4,5)), ami 1, ha a  $v_i$  pont a keresett csúcshalmazban található, egyébként 0. A bevett pontok számát akarjuk maximalizálni, ami megfelel a változók összegének maximalizálásának (1). Minden csúcsra igaz, hogy a szomszédai közül legfeljebb kettő lehet kiválasztva, azaz a szomszédoknak megfelelő változók összege nem lehet 2-nél nagyobb (2).

$$\max \sum_{i=1}^n x_i \quad (1)$$

feltéve, hogy

$$\sum_{\forall j:(v_i,v_j) \in E} x_j \leq 2 \quad \forall i = 1 \dots n \quad (2)$$

$$x_i \leq 1 \quad \forall i = 1 \dots n \quad (3)$$

$$x_i \geq 0 \quad \forall i = 1 \dots n \quad (4)$$

$$x_i \in \mathbb{Z} \quad \forall i = 1 \dots n \quad (5)$$

4. Minden ponthoz felvesszünk egy  $x_i$  (bináris) változót (binárisítást biztosító korlátok: (3,4,5)), ami 1, ha a  $v_i$  pont a keresett csúcshalmazban található, egyébként 0. A bevett pontok számát akarjuk maximalizálni, ami megfelel a változók összegének maximalizálásának (1). Minden független élpárra igaz, hogy a hozzájuk tartozó 4 csúcs közül legfeljebb kettő lehet kiválasztva, azaz minden ilyen 4 csúcsnak megfelelő változók összege nem lehet 2-nél nagyobb (2).

$$\max \sum_{i=1}^n x_i \quad (1)$$

feltéve, hogy

$$x_i + x_j + x_k + x_l \leq 2 \quad \forall e_a, e_b : e_a = (v_i, v_j), e_b = (v_k, v_l), \text{ egymástól ftln élek} \quad (2)$$

$$x_i \leq 1 \quad \forall i = 1 \dots n \quad (3)$$

$$x_i \geq 0 \quad \forall i = 1 \dots n \quad (4)$$

$$x_i \in \mathbb{Z} \quad \forall i = 1 \dots n \quad (5)$$

Megjegyzés: nem tartozik a feladathoz, de érdemes belegondolni, hogy a páronként független élek megkeresése nyilván nem okoz gondot, hiszen végigmegyünk az összes lehetséges élpáron ( $O(e^2)$ ), és a páronként függetlenekre felvesszük a korlátot.

### 13. Közelítő algoritmusok

1. Először belátjuk az algoritmus helyességét, vagyis azt, hogy tényleg egy lefogó ponthalmazt talál. Tfh nem lefogó ponthalmazt kapunk, vagyis maradt olyan  $e$  él, hogy egyik végpontja sincs kiválasztva. Ez viszont azt jelenti, hogy  $e$ -vel bővíthetnénk a független élhalmazunkat, ami ellentmondás. Ezután be kell látni, hogy tényleg 2-közelítő az algoritmus. Jelölje  $k$  a kiválasztott élek számát, ekkor a kiválasztott pontok száma:  $t = 2k$ . Tudjuk, hogy  $\tau(G) \leq \nu(G)$ , valamint nyilvánvalóan legfeljebb annyi élet választhattunk ki, mint a ftln élek maximális száma. Ezeket összerakva:

$$t = 2k \leq 2\nu(G) \leq 2\tau(G).$$

2. **Algoritmus:** ameddig lehet, párosával rakjuk az  $a$  és  $b$  méretű elemeket ládába, a maradékot pedig a FirstFit algoritmussal rendezzük el. Ez ügyesen implementálva ebben a spec esetben triviálisan  $O(n)$ . **Helyesség:** az  $a$  méretű elemek darabszáma legyen  $A$ , a  $b$  méretűeké  $B$ . Ha  $B \geq A$ , akkor az algoritmus  $B$  ládát fog felhasználni ( $OPT \leq B$ ), és mivel  $b > 1/2$ , mindegyik külön ládába kell, hogy kerüljön, így legalább  $B$  ládára szükség is van ( $OPT \geq B$ ). Ha  $A > B$ , akkor két dolgot bizonyítottunk. Az egyik, hogy mindig létezik olyan optimális elrendezés, hogy minden  $b$  méretű elem mellett szerepel egy  $a$  méretű is. Tfh egy olyan optimális elrendezésünk van, ahol legalább az egyik  $b$ -nek nincs párja. Ekkor egy csak  $a$ -t tartalmazó ládából egy  $a$ -t át tudunk rakni ide úgy, hogy a szükséges ládák

számát nem növeljük. Ezt addig csinálhatjuk, amíg a kívánt elrendezéshez nem jutunk, és közben a ládák számát egyszer sem növeltük. Ez alapján elég az olyan elrendezéseket vizsgálni, ahol minden  $b$  párosítva van egy  $a$ -val (azaz amilyen elrendezéseket a mi algoritmusunk is ad), és csak az  $a$ -k elrendezésére kell koncentrálni.

Tfh nem a FirstFit szerinti módon vannak a maradék  $a$ -k bepakolva. Ekkor az általánosság megsértése nélkül feltételezhetjük, hogy a bepakolt elemek darabszáma szerint nemnövekvően vannak rendezve a ládák. A feltételezés szerint az utolsó ládán kívül is van olyan, ahova még beférne egy  $a$  méretű elem (különben pont a FirstFit szerinti elrendezésünk lenne). A legutolsó ládából ekkor egy  $a$ -t átpakolhatunk ide anélkül, hogy a ládák számát növelnénk. Ezt egészen addig csinálhatjuk, amíg nem a kívánt formában van az elrendezés, és a ládák száma egyszer sem nőtt. Azaz mindig létezik olyan optimális megoldás, mint amelyet a mi algoritmusunk ad.

*Ezt a bizonyítást természetesen le lehetne írni sokkal rövidebben is, de a jó érthetőség és a precizitás miatt választottam ezt.*

3. Ellenpélda: dobozméret: 1, elemek: 0.6, 0.3, 0.7, 0.4. Optimális megoldás: [0.6, 0.4], [0.7, 0.3],  $OPT = 0$  (mert 0 hely marad ki). A FirstFit által adott megoldás: [0.6, 0.3], [0.7], [0.4], a kimaradó hely 1. Ha 2-közelítő lenne az algoritmus, akkor  $1 \leq 2OPT = 0$  teljesülne, ami nyilván nincs így. Vagyis a válasz nem.
4. Az, hogy a mohó algoritmus nem optimális, egy ellenpéldával könnyen bizonyítható: a szalag mérete legyen 2, az érkező fájlok mérete 1,1,2. Az optimális megoldásban mind a 3 fájl ki tudjuk írni (az egyik szalagon 1,1, a másikon 2), a mohó csak a két 1-est írja ki, a két szalagra. Némi gondolkodás után megsejtjük, hogy a mohó algoritmus mindig csak legfeljebb 1-gyel kevesebb fájl ír ki, mint amennyit az optimális kiírás szerint lehetne. Ezt bizonyítjuk be. A szalagok mérete legyen  $S$ . Tfh van egy olyan feladatunk, amiben legalább 2 fájllal többet lehet kiírni, mint amit a mohó változat ad. Ezek közül az első kettő hossza legyen  $h_i$  és  $h_{i+1}$ , a feltételből tudjuk, hogy

$$h_i \leq h_{i+1} \tag{1}$$

Az egyik szalagon  $W_1$  méret foglalt, a másikon  $W_2$ . Ha a maradék  $S - W_1$  vagy  $S - W_2$  helyre beférne  $h_i$ , akkor azt a mohó algoritmus berakta volna. Vagyis

$$S - W_1 < h_i \tag{2}$$

$$S - W_2 < h_i \tag{3}$$

Az optimális megoldásban  $W_1$  és  $W_2$  mennyiségű adat biztosan szerepel valamilyen elrendezésben, és ezen kívül  $h_i$  és  $h_{i+1}$  is még befér. Azaz az összes elérhető és felhasznált adatmennyiséget felírva ebben az esetben

$$2S \geq W_1 + W_2 + h_i + h_{i+1} \geq W_1 + W_2 + 2h_i \tag{4}$$

ahol a második egyenlőtlenség (1)-ből következik. Ezt (2) és (3) segítségével tovább írva (figyeljünk, hogy hogy hol van  $\geq$ , és hol van  $>$ ):

$$2S \geq W_1 + W_2 + 2h_i > W_1 + W_2 + (S - W_1) + (S - W_2) = 2S \tag{5}$$

ami nyilvánvalóan lehetetlen. Így nem lehet, hogy a mohó algoritmus által kiírt fájlknál legalább 2-vel több kiírható lenne.