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). Ω: 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 Ω 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:

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

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 $\sum_{i=0}^{k} c^i = \frac{c^{k+1}-1}{c-1}$).

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

Fentiekből látszik, hogy $\mathcal{A}$ a gyorsabb (fontos, hogy az egyikre O-t számoltunk, a másikra Ω-t; persze itt minkettő Θ, 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, konstanst 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 $1 = \frac{n}{2^{\log n}}$):

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

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 részfában 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 lentről felfele haladva minden csúcsra meg tudjuk mondani, hogy a belőle induló részfában 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. Felveszünk 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 sorozatot. 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 precízek 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. Kitöltése:

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

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 $\forall j : A[j] = true$ 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 $a_1 \ldots a_i$ 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. $a_1 \ldots a_{k+1}$ közül néhányat választva vagy választjuk a_{k+1}-et, vagy nem. Ha nem, akkor pontosan azok állnak elő, amik $a_1 \ldots a_k$ 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 (modjuk $l(i)$), lentről felfele 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;

lépészám: 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 $1 \ldots i$ 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 $D[i-2]$?);...; 1-től i-ig csinálunk egy sort. Meg is van a képlet: $D[i] = \min_{j=1\ldots i}(t^2_{j,i} + D[j-1])$, 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 $1, 2, \ldots, n$ műveletet végzünk), a lépésszám $O(n^2)$ lesz.

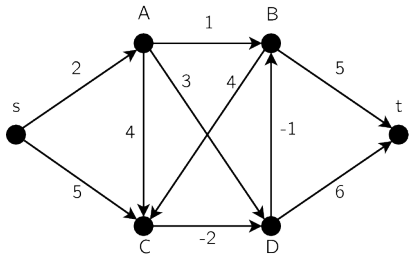
11. Lásd következő feladat.

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 ∞ érték.

A	B	C	D	t
2	∞	5	∞	∞

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 számítódik: $\min(F[D], F[A]+3, F[C]-2) = \min(\infty, 2+3, \mathbf{5-2}) = 3$. Ezt az összes csúcsra végig kell játszani minden lépésben, azaz a k-adik 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:

A	B	C	D	t
2	∞	5	∞	∞
2	3	5	3	∞

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:

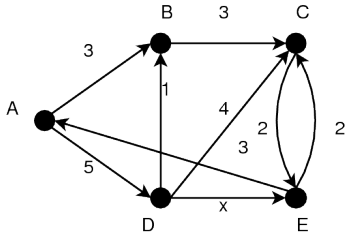
A	B	C	D	t
2	∞	5	∞	∞
2	3	5	3	∞
2	2	5	3	8

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

A	B	C	D	t
2	∞	5	∞	∞
2	3	5	3	∞
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.



A	**B**	**C**	**D**	**E**	**KÉSZ**
$\underline{0}$	**3**	∞	5	∞	A
$\underline{0}$	$\underline{3}$	6	**5**	∞	A,B
$\underline{0}$	$\underline{3}$	6	$\underline{5}$	$5+x$	A,B,D

Ha $0 < x \leq 1$:

$\underline{0}$	$\underline{3}$	**6**	$\underline{5}$	$\underline{5+x}$	A,B,D,E
$\underline{0}$	$\underline{3}$	$\underline{6}$	$\underline{5}$	$\underline{5+x}$	A,B,C,D,E

Ha $1 < x$:

$\underline{0}$	$\underline{3}$	$\underline{6}$	$\underline{5}$	$\min(8, 5+x)$	A,B,C,D
$\underline{0}$	$\underline{3}$	$\underline{6}$	$\underline{5}$	$\underline{\min(8, 5+x)}$	A,B,C,D,E

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

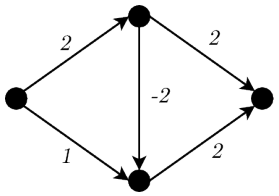
4. A problémát egy páros gráffal fogjuk modellezni, az A csúcsosztály a lehetséges ajánlók, a B csúcsosztály pedig az ajánlólevelek (minden pályázott helyre két darab). Egy ajánló és egy ajánlólevél között pontosan akkkor menjen él, ha az adott ajánló megírhatja az adott ajánlólevelet.
Ebben a gráfban 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 élellel. 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 alogritmus 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 élet, 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 élet, vagy nem. Először futtassunk egy Dijsktrát (a szépség kedvéért hagyjuk ki a negatív élet), 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 ő eléréséhez az eredeti gráfban. Belőle is indíthatunk egy Dijsktrát (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 Dijsktra, azaz $O(e+n^2+n^2) = O(n^2)$ (az ugye mindenkinek világos, hogy egy egyszerű gráfban $e = O(n^2)$).

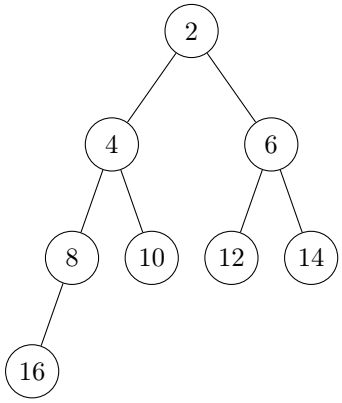
11. Ez csak annyiban trükkös, hogy a várakozásokkal ellentétben ez nem legrövidebb út, hanem párosítás. Ugyanis minden gazda-ponthoz párosítani akarunk egy fát. Egyik pontosztály tehát a gazda pontjai, másik a fák, akkor megy él, ha adott pontból elérhető a fa úgy, hogy utána a kutya vissza tud érni. 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.

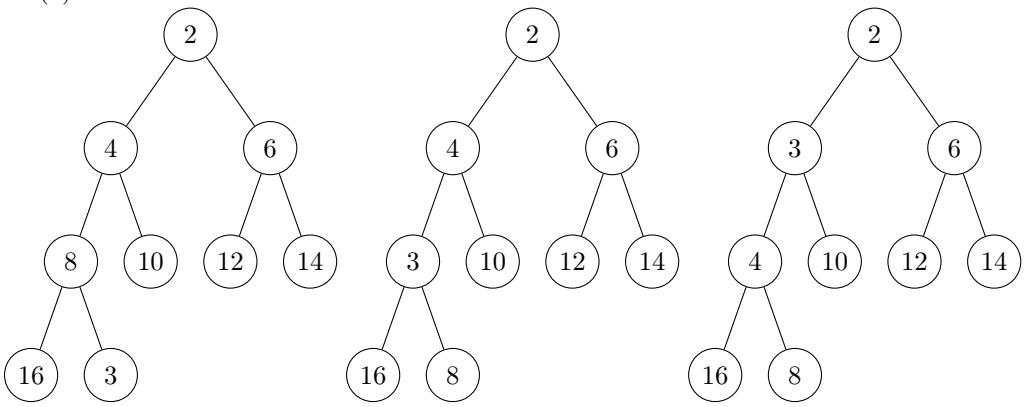
13. Az összefüggőség és irányítatlanság miatt a szélességi bejárás egy szélességi feszítőfát fog adni. A páratlan szinteken lévő pontok száma legyen x, értelemszerűen a párosoké ekkor $n-x$. Ha $x < n-x$, akkor válasszuk központi halmaznak a páratlan szinten lévő pontokat, egyébként a párosakat. Könnyen 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

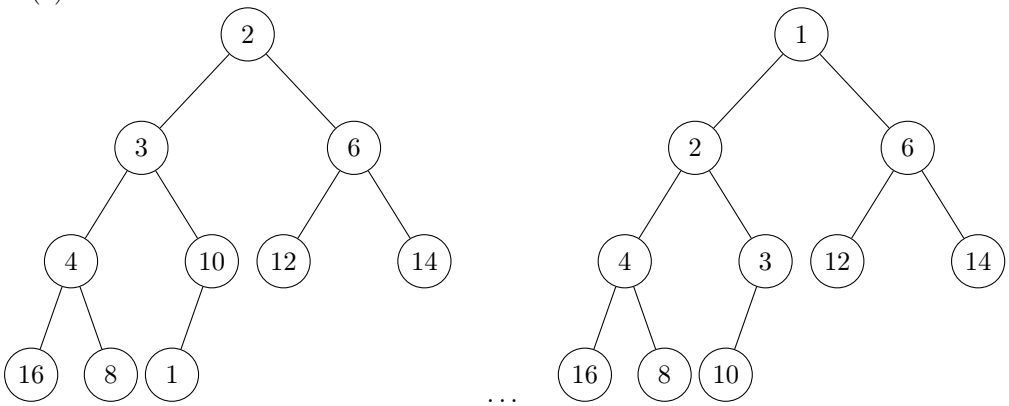
1.



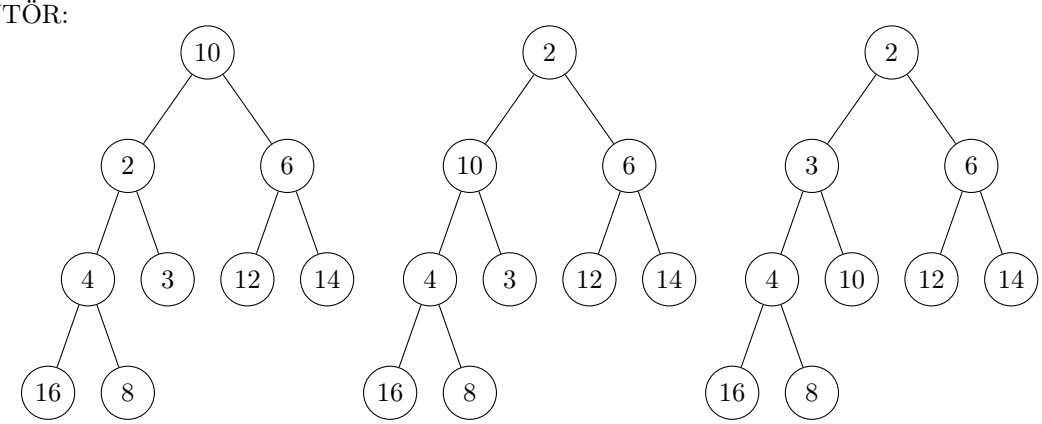
BESZÚR(3):



BESZÚR(1):



MINTÖR:



2. 3 max-kupacot fogunk nyilvántartani, az egyikbe mennek A betegei, a másikba B betegei, a harmadikba $A \vee B$ betegei. 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 bennemaradni. Az elégségesség kicsit macerább. Amit tudunk: x a gyökérig 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 gyerekét 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 gyerekét 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 össz pointerekkel (hogy az egyik kupacban lévő, a kezünkben lévő elemről keresés nélkül tudhassuk, hogy hol található a másik kupacban).
Felépít: pointerek 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 $A[1] = A[2] = \cdots = A[n]$, 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 összefésüléses rendezéssel. 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 kírt 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 megkeressük 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 renezéssel együtt a lépésszám $O(n \log n)$.

12. Minimális y kooridántájú 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-ból, 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 $\texttt{--}T_x[x_i]$ és $\texttt{++}T_x[x_i + k + 1]$ (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 (kihúzunk 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 ilyet, 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:

$$< (a, b)\{\texttt{return } kozepso(x, a, b) == a\}$$

Ez konstans időben számolható, tehát bármely $<$ operátort használó rendezés használható lesz ezzel a trükkel, 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 részfában 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 $[1, \ldots, 39]$, $[40, \ldots, 49]$ és $[50, \ldots, \infty]$, 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 $\lfloor \log n + 1 \rfloor$, így ez a korlát 4. Vagyis a másik két részfában 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 $P_i[1, \ldots, 30]$ 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 belefférnek.

6. Ekkor a bal részfában 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 gyereknek 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 belefér 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 belefér $O(\log n)$-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 kiindulú részfában hány elem van (ez ugyanaz, mint a kettővel ezelőtti feladat, csak a többszörös elemkre 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 részfák 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 lépsszám $O(n+2n) = O(n)$ (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 tárolós 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 $n = 1\ldots$ 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-levél 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 előttiek, 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 beszúrási sorrend: $23 \to 33 \to 3 \to 19$, 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. Bökjük 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 beszúrva, kvadratikus próba esetén minden $3i+1$ helyre két elemet beszúrva.
(b) Lásd az előzőt.

7. Nyilván akkor történik a legtöbb ütközés, ha a (ciklikus értelemben) leghosszabb folytonosan kitöltött rész legvégére próbáljuk beszúrni 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 csomónk lesz (és minden beszúrás ütközéses volt).

10. $M = 35$ esetén a $3 \dots 33$ számok a saját helyükre kerülnek, 0 (mod 3) helyekre, $36 \dots 69$ számok 1 (mod 3) helyekre, $72 \dots 90$ számok pedig 2 (mod 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 beszúrások sorrendje megegyezik!). Vegyük észre, hogy az eredetibe beszúráskor, 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: $|0|4|2|$ (mod 3), de $|0|\ |2|\ |4|\ |$ (mod 6), $t_1 = 3$, $t_2 = 1$, és $3 \not\leq 2 \cdot 1$.

9. Mélységi bejárás, DAG, valamint alkalmazásaik

1. A mélységi számok jelentik a sorrendet, ami szerint meglátogattuk a csúcsokat. Tehát x volt az első, ebből megy él 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 $u \to w$ élünk is, ahonnan megint visszaléptünk. Most visszalépünk u-ból, majd y-ból is, és látjuk, hogy x-ből mentünk z-be ($x \to z$ él). Ezek után visszalépdeltünk, 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 megy él j-be, ha a j boríték belerakható i-be. Ekkor egy irányított út a gráfban megfelel egy helyes borítéklláncnak, 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 belefér 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 behúzhatók 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 tankolandó 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 $1, \ldots, k$ 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 $O(|E| \log k)$.

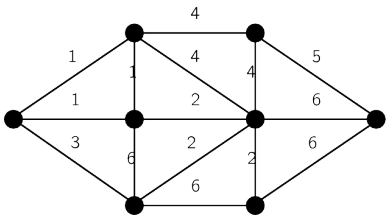
13. Ha adott egy π permutáció, akkor egy bejárással $O(e + n)$ lépésben el tudom dönteni, hogy van-e megfelelő, k hosszú út (csak a π szerinti aktuálisan következő címkéjű éleket vesszük létezőnek a bejárás lépései során). Összesen $k!$ permutáció lehet, így mindet végig tudjuk nézni $O(k!(e + n))$ lépésben.

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 megfelően 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

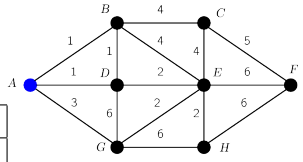
sorrendje. Az $n+1$-edik pont bejövő éleit rendeljük az egyik DAG-hoz, ettől az DAG marad (a topologikus sorrendben az aktuális lesz az utolsó pont, a többit nem zavarja), a kimenő éleket pedig a másik DAG-hoz (hasonlóan az is DAG marad). Így az állítást igazoltuk (és mellesleg a bizonyításunk konstruktív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

10. Minimális költségű feszítőfák

1. A Prim algoritmust fogjuk példaképp futtatni.

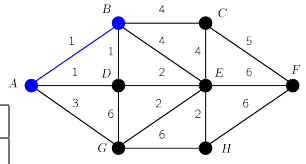


Kezdjünk mondjuk A-val. Kezdeti kitöltés:



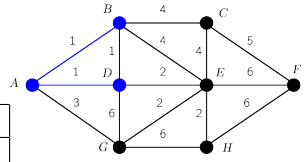
	A	B	C	D	E	F	G	H
KÖZEL	$*$	A	A	A	A	A	A	A
MINSÚLY	$*$	1	∞	1	∞	∞	3	∞

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 ∞-nel.



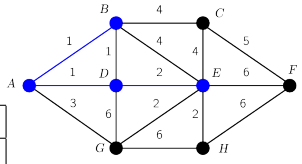
	A	B	C	D	E	F	G	H
KÖZEL	$*$	$*$	B	A	B	A	A	A
MINSÚLY	$*$	$*$	4	1	4	∞	3	∞

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



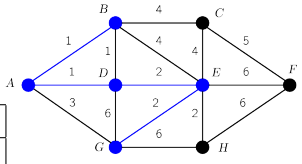
	A	B	C	D	E	F	G	H
KÖZEL	$*$	$*$	B	$*$	D	A	A	A
MINSÚLY	$*$	$*$	4	$*$	2	∞	3	∞

Érdemes 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:



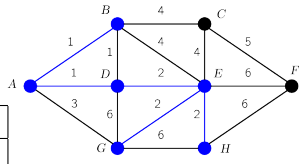
	A	B	C	D	E	F	G	H
KÖZEL	$*$	$*$	B	$*$	$*$	E	E	E
MINSÚLY	$*$	$*$	4	$*$	$*$	6	2	2

Most G, EG éllel:



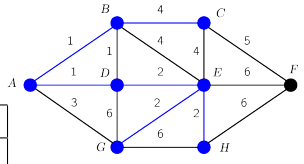
	A	B	C	D	E	F	G	H
KÖZEL	$*$	$*$	B	$*$	$*$	E	$*$	E
MINSÚLY	$*$	$*$	4	$*$	$*$	6	$*$	2

aztán H, EH-val:



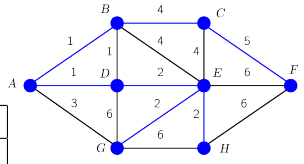
	A	B	C	D	E	F	G	H
KÖZEL	$*$	$*$	B	$*$	$*$	E	$*$	$*$
MINSÚLY	$*$	$*$	4	$*$	$*$	6	$*$	$*$

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.

2. A Kruskal algoritmus biztosan mininális feszítőfát ad, ezért megnézhetjük, hogy mit csinálna. Először az 1 súlyú élek közül fog választani, belőlük tud építeni egy $k-1$ elű 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.)

3. Gyakorlatilag az UNIÓ-HOLVAN adatszerkezet használható változtatás nélkül, mivel elég a gráf komponenseit nyilvántartani. Van út két csúcs között, ha egy komponensben vannak, és nincs ha nincsenek. Egy csúcs felvétele létrehoz egy új komponenst (halmazt). Egy új él vagy összeköt két komponenst (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).

4. 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 élét 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 minfeszfának, ha pedig nincs, akkor f-re nem alkalmazható a piros szabály, tehát része a minfeszfának. Mivel csak egy bejárást csináltunk (pl mélységi), a lépésszám $O(|V|+|E|)$.

5. A keresett részgráfba minden -1 súlyú élet be kell válogatni (ha nem tennénk, akkor egy ilyen él bevételével 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 lépészám a bejárásé: $O(|V|+|E|)$.

6. Először megmaradt járdákat használva öf kompnensek 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 mindenhonnan mindenhova 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 mindenhova 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 csúcsnál 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 színezné (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 élét elhagyva a maradék kiegészíthető feszítőfává, így ekkor legalább k darab van neki. Vagyis csak 3 hoszzú 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. Borůvka algoritmusának első két menetét futtatjuk (első menet után legalább $\left\lfloor \frac{n}{2} \right\rfloor$ él van meg, második után legalább $\left\lfloor \frac{3n}{4} \right\rfloor$).

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 költségdefiníciónkra 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 színezni, 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. ftln ponthalmaz része, hiszen ekkor létezne a gráfban $\alpha(G)+1$ független pont. Ha nem, akkor pedig biztos egy max. ftln 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$\prec L$ Karp-redukciót. Ha egy G gráfban Hamilton-utat keresünk, akkor minden csúcshoz rendeljünk 1 súlyt, és az így keletkezett G' gráfban keressünk legfeljebb 2 levélsúlyú feszítőfát! Állítás: $G \in H$-út $\Leftrightarrow (G',2) \in L$. $G \in H$-út $\Rightarrow (G',2) \in L$: G egy Hamilton-útja pont egy két levelű, 2 súlyú feszítőfa G'-ben. $(G',2) \in L \Rightarrow G \in H$-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 módsze 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 módsze 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 $O(\frac{n!}{(n-100)!})$ félképpen 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 színt 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 csúcsnak lesz egy jó színe. Legfeljebb n csúcsra n színt 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 koordinátáival). 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-körről 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 .... 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.

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

Érdemes 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-hatványságot triviális ellenőrzni (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 ismerten 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 kérdésére 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 következeik, hogy $NP = coNP$.

12. Egészértékű programozás

1. Minden (v_i, v_j) élhez felveszünk 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).

$$\max \sum_{i=1}^{n} x_{i,j} \tag{1}$$

feltéve, hogy

$$\begin{aligned}
\sum_{(v_i,v_j)\in E} x_{i,j} &\leq 1 \quad \forall i = 1 \dots n && (2)\\
x_{i,j} &\leq 1 \quad \forall (v_i, v_j) \in E && (3)\\
x_{i,j} &\geq 0 \quad \forall (v_i, v_j) \in E && (4)\\
x_{i,j} &\in \mathbb{Z} \quad \forall (v_i, v_j) \in E && (5)
\end{aligned}$$

2. Minden ponthoz felveszünk 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).

$$\max \sum_{i=1}^{n} x_i \tag{1}$$

feltéve, hogy

$$\begin{aligned}
x_i + x_j &\leq 1 \quad \forall i, j : (v_i, v_j) \notin E && (2)\\
x_i &\leq 1 \quad \forall i = 1 \dots n && (3)\\
x_i &\geq 0 \quad \forall i = 1 \dots n && (4)\\
x_i &\in \mathbb{Z} \quad \forall i = 1 \dots n && (5)
\end{aligned}$$

3. Minden ponthoz felveszünk egy x_i (bináris) változót (binárisságot 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).

$$\max \sum_{i=1}^{n} x_i \tag{1}$$

feltéve, hogy

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

$$x_i \leq 1 \quad \forall i = 1 \dots n \tag{3}$$

$$x_i \geq 0 \quad \forall i = 1 \dots n \tag{4}$$

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

4. Minden ponthoz felveszünk egy x_i (bináris) változót (binárisságot 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).

$$\max \sum_{i=1}^{n} x_i \tag{1}$$

feltéve, hogy

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

$$x_i \leq 1 \quad \forall i = 1 \dots n \tag{3}$$

$$x_i \geq 0 \quad \forall i = 1 \dots n \tag{4}$$

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

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 bizonyítunk. 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 amilyet 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 fájlt 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 fájlt 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 fájloknál legalább 2-vel több kiírható lenne.