Algoritmusok – gyakorló feladatok

Csákány Rita Csima Judit Friedl Katalin Ivanyos Gábor Küronya Alex
Madas Pál Pintér Márta Rónyai Lajos Sali Attila Simonyi Gábor Szabó Réka

1999. Szeptember 13.

I. Feladatok

1. Vegyes

1. Adjunk hatékony módszert az $I = \{1, 2, \ldots, n\}$ halmaz összes részhalmazának kilistázására. A listán minden részhalmaz pontosan egyszer szerepeljen; a listán szomszédos részhalmazok elemszámának különbsége legfeljebb 1 lehet.

2. Adjunk hatékony módszert az olyan $1 \leq m \leq n$ feltételeknek eleget tevő m természetes számok kilistázására, melyekhez vannak olyan x, y egészek, hogy $m = x^2 + y^2$. Elemezzük a módszer időigényét!

3. A $v \in \Sigma^*$ szó az $u \in \Sigma^*$ *kezdőszelete*, ha van olyan $w \in \Sigma^*$, hogy $u = vw$. Hasonlóan v az u *végszelete*, ha van olyan $y \in \Sigma^*$, hogy $u = yv$. A feladat az, hogy találjuk meg az $A[1:n]$ tömbben tárolt n hosszúságú u szó leghosszabb olyan valódi kezdőszeletét, ami egyben végszelete is. Javasoljunk $O(n)$ uniform költségű módszert.

4. Adott egy $x \in \Sigma^*$, szó, ahol $\Sigma = \{a, b, c\}$. Javasoljunk egy minél kevesebb összehasonlítást igénylő módszert annak az eldöntésére, hogy x tartalmazza-e résszóként az abca szót. Elemezzük a módszer költségigényét!

5. Egy szolgáltató helyen az egyidőben érkezett $A_1, A_2, \ldots A_n$ ügyfeleket kell kiszolgálni. Egyszerre csak egy ügyféllel tudnak foglalkozni és az A_i kiszolgálásához szükséges időmennyiség t_i. Egy adott K kiszolgálási sorrend mellett $T_i(K)$ jelöli az A_i kiszolgálásának befejeztéig eltelt időt. (Pl. ha $n = 2$, $t_1 = 3$, $t_2 = 2$ és $K = A_2, A_1$, akkor $T_1(K) = 5$ időegység.)
Javasoljunk módszert egy olyan K ütemezés a meghatározására, melyre $\sum T_i(K)$ minimális.

6. Adottak $M_1, \ldots, M_n$ munkák $H_1, \ldots, H_n$ határidőkkel és $P_1, \ldots, P_n$ profitokkal. Tegyük fel, hogy minden egyes munka elvégzésére 1 napra van szükségünk. Adjunk algoritmust, mely meghatározza, hogy mely munkákat vállaljuk el, hogy az összprofitunk a lehető legnagyobb legyen.

7. Egy tanteremben fel van szerelve egy $n \times n$-es tábla, melyen n^2 villanykörte helyezkedik el. A tábla minden egyes sorához illetve oszlopához tartozik egy-egy nyomógomb, mellyel a megfelelő sorban (oszlopban) található n darab villanykörte állapotát egyszerre lehet átváltoztatni az ellenkezőjére. (Egy gombnyomásra az adott sorban illetve oszlopban égő körték elalszanak, az alvók pedig kigyulladnak.) A szünet kezdetekor az összes körte leoltott állapotban van. Szünetben a nebulók össze-vissza nyomogatják a gombokat. Hány kapcsolással tudja a tanár visszaállítani az eredeti állapotot? (A gombok egyállapotúak, azaz nem látszik rajtuk, hogy megnyomták-e őket vagy sem.)

8. n dobozban golyók vannak szétosztva úgy, hogy a k-adik dobozba éppen k golyó esik. Adjunk optimális lépésszámú módszert az összes doboz kiürítésére, ha a megengedett lépés a következő: jelöljünk ki tetszőleges számú dobozt, és mindegyikből vegyünk ki ugyanannyi golyót. $\diamondsuit$

9. Adott egy $A[1:8]$ tömb, melyben van *abszolút többségi elem* (olyan elem, amiből legalább 5 van a tömbben). A célunk ennek az elemnek a megtalálása minél kevesebb összehasonlítással. Egy összehasonlítás kétféle eredményt adhat: $=$, vagy $\neq$. Határozzuk meg a szükséges összehasonlítások minimális számát!

10. Adott n chip, melyek képesek egymás tesztelésére a következő módon: ha összekapcsolunk két chipet, mindkét chip nyilatkozik a másikról, hogy hibásnak találta-e. Egy hibátlan chip korrektül felismeri, hogy a másik hibás -e, míg egy hibás chip akármilyen választ adhat. Tegyük fel, hogy a chipek több, mint a fele korrekt. Adjunk algoritmust, mely n-nél kevesebb fenti tesztet használva kikeres egy jó chipet.

11. Egy $2 \times n$-es sakktábla mezőin n piros és $n-1$ kék négyzetet helyezünk el. Ezeket olyan módon akarjuk átrendezni, hogy a felső sorban piros, az alsóban kék négyzetek legyenek, s a bal alsó sarok maradjon üres. Ehhez egy-egy lépés során az üres mezőre tolhatjuk valamelyik szomszédját. Bizonyítsuk be, hogy ehhez

   (a) (*) $O(n^2)$ lépés elégséges és
   (b) (**) $\Omega(n^2)$ lépés szükséges.

12. Rendezzük egy mátrix soraiban az elemeket növekvő sorrendbe, majd ezt követően az oszlopokat rendezzük hasonló módon. Mutassuk meg, hogy a második rendezés nem rontja el a sorok rendezettségét.

13. Adott az $A[1:n, 1:n]$ kétdimenziós Boole (0–1) tömb. Adjunk $O(n^2)$ költségű módszert az A-beli legnagyobb csupa egyesből álló négyzet megkeresésére. Pontosabban: határozzuk meg a legnagyobb olyan $0 \leq k < n$ egészet, melyhez vannak olyan i, j indexek, hogy az $A[i:i+k, j:j+k]$ résztömb minden eleme 1.

14. Adott egy $n \times n$-es mátrix, amelyben csak az első sorban és az első oszlopban vannak 0-tól különböző számadatok. Ennek célszerű tárolásához adjunk meg egy $f: [1..n] \times [1..n] \to [1..2n]$ függvényt, amely $i = 1$ vagy $j = 1$ esetén különböző (i, j) párokhoz különböző $f(i, j)$ értékeket rendel. f előállításához felhasználható a négy alapművelet, az *abs-*, *sgn-*, *mod-* és *div*-függvény, valamint a függvénykompozíció művelete.

15. Egy számkombinációs zár három hengerének mindegyikén 1-től n-ig szerepelnek a számok. Egy lépésben tetszőleges számú és elhelyezkedésű henger összefogható és ezek együtt ugyanabba az irányba egységnyit elforgathatók. Bizonyítsuk be, hogy tetszőleges kiindulási helyzetből a csupa egyes állás elérhető cn lépésben és állapítsuk meg a c konstans optimális értékét!

2. Technika

1. Tegyük fel, hogy van egy számítógépes programunk, ami egy k méretű feladaton a jelenlegi gépünkön 1 nap alatt fut le. Beszereztünk egy százszor gyorsabb számítógépet. Ugyanazon programmal mekkora feladatot lehet az új gépen egy nap alatt megoldani, ha a program lépésszáma n méretű feladat esetén
   (a) n-nel
   (b) n^3-bel
   (c) 2^n-nel arányos?

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

3. Adjuk meg azt a $T(n)$ függvényt zárt alakban, amely a 3-hatványokra van értelmezve és eleget tesz a következő feltételeknek:

   $T(1) = 1$ és $n \geq 2$-re
   $T(n) = 4T(n/3) + n^2$

4. Adjunk algoritmust a *hanoi tornyok* feladatának megoldásához. Mekkora az algoritmus időigénye?

5. Mit csinál a következő c program?

```
typedef double number;
void mitcsinál(number v)
{
number d,r;
int c;
  if ( v < 0 ) { printf(" -"); v = -v; } else printf("  ");
  if ( v < 2 ) { printf(" %0.0lf",v); return; }
  d = 2;
  do {
    c = 0; while ( r=v/d,!fmod(v,d) ) { v = r; c++; }
    if ( c ) { printf(" %0.0lf",d); if ( c > 1 ) printf("^%d",c); }
    if ( d == 2 ) d = 3; else d += 2; if ( d*d > v ) d = v;
  } while ( v != 1 );
}
```

   Elemezzük a program futási idejét!

6. Mit csinál, és mekkora az időigénye a következő algoritmusnak (feltehetjük, hogy n 2-hatvány)?

```
function kitudja(s, t, n : integer) : boolean;
  begin
    if n = 1 then
      if él(s, t) then
        return(igaz)
      else
        return(hamis);
    for i := 1 to V do
      if kitudja(s, i, n div 2) and kitudja(i, t, n div 2) then
        return(igaz);
    return(hamis)
  end;
```

7. Írjunk rekurzív függvényhíváson alapuló algoritmust, ami az n-edik Fibonacci-számot határozza meg! Mennyi a lépésszám?

8. Adott egy n bites M természetes szám. Javasoljunk (n-ben) polinomiális idejű módszert $\lfloor \sqrt[3]{M} \rfloor$ kiszámítására. Adjunk becslést a módszer költségére.

9. Javasoljunk egy hatékony algoritmust $\lfloor \log_3 n \rfloor$ kiszámítására, ahol n egy adott, binárisan ábrázolt pozitív egész! Elemezzük a módszer költségét!

10. Javasoljunk algoritmust az a és b egészek (binárisan ábrázolva) szorzatának kiszámítására az alábbiak szerint:
(i) Az a, b egészeket az $A[1:n]$ illetve $B[1:n]$ Boole tömbök tartalmazzák; ezeket csak olvasni szabad.
(ii) Az eredmény (az ab szorzat) a $C[1:2n]$ Boole tömbbe írandó; ennek minden pozíciójába csak egyszer lehet írni.
(iii) További munkaterületként $O(\log^c n)$ bit tárolására alkalmas helyet használhatunk, ahol c egy pozitív konstans.

3. Rendezés

1. Rendezzük a következő listát kupacos rendezés, gyorsrendezés, és az összefésüléses rendezés segítségével: 4, 11, 9, 10, 5, 6, 8, 1, 2, 16.

2. Rendezzük a következő láncokat a radix rendezés segítségével: *abc*, *acb*, *bca*, *bbc*, *acc*, *bac*, *baa*.

3. Hány összehasonlítással lehet megtalálni n elem közül a legkisebbet?

4. Pontosan hány összehasonlítás kell ahhoz, hogy egy n elemű tömbből egy olyan tagot keressünk, ami a tömb legkisebb 10 eleme közé tartozik? (A tömb egy rendezett univerzum n különböző eleméből áll, de maga nem feltétlenül rendezett. Az eredmény bármelyik lehet a legkisebb tíz közül: tehát pl. az első éppúgy megfelel, mint a tizedik.) ♢

5. Egy csupa különböző egészekből álló sorozat *bitonikus*, ha először nő, utána pedig fogy, vagy fordítva: először fogy, utána nő. Például az $(1, 3, 7, 21, 12, 9, 5)$, $(9, 7, 5, 4, 6, 8)$ és $(1, 2, 3, 4, 5)$ sorozatok bitonikusak. Adjunk $O(n)$ összehasonlítást használó rendező algoritmust n elemű bitonikus sorozatok rendezésére!

6. (a)(**) Össze kell fésülnünk az $A_1 < A_2 < \ldots < A_n$ és a $B_1 < B_2 < \ldots < B_{n+1}$ rendezett halmazokat. Bizonyítsuk be, hogy a szükséges összehasonlítások minimális száma $2n$.
(b) Igaz -e, hogy alkalmas c állandóra minden (n, k) párra az n és a k elemű rendezett halmazok összefésüléséhez kell legalább $c(n + k)$ összehasonlítás? ♢

7. Adjunk konstans szorzó erejéig optimális számú összehasonlítást használó módszert az $A_1, A_2, \ldots, A_m$ egyenként k elemű rendezett tömbök összefésülésére!

8. Célunk az $A[1:n]$ tömb $A[1:k]$ és $A[k+1:n]$ részeinek felcserélése. Oldjuk meg ezt a feladatot $O(n)$ költséggel és konstans munkaterülettel!

9. Adott egy rendezett univerzum n különböző eleméből álló S halmaz. Legyen $l := \lfloor \log_2 n \rfloor$. Felosztandó az S halmaz olyan nem üres $S_1, S_2, \ldots, S_l$ részhalmazokra, hogy tetszőleges $1 \le i < j \le l$ számpárra az S_i halmaz minden eleme kisebb legyen az S_j halmaz minden eleménél. Adjunk meg egy $O(n \log\log n)$ összehasonlítást használó módszert a fenti feladatra!

10. A (növekvően) rendezett $A[1:n]$ tömb k darab elemét valaki megváltoztatta. A változtatások helyeit nem ismerjük. Javasoljunk $O(n + k \log k)$ uniform költségű algoritmust az így módosított tömb rendezésére! ♢

11. Az $A[1:n]$ tömb elemei egy rendezett típusból valók. Tudjuk továbbá, hogy ez a sorozat két monoton növő részsorozat egyesítése. Pontosabban fogalmazva az $\{1, 2, \ldots, n\}$ indexhalmaznak vannak olyan X_1 és X_2 részei, hogy $X_1 \cup X_2 = \{1, 2, \ldots, n\}$ és ha $j, k \in X_i$, $j < k$, akkor $A[j] < A[k]$ $(i = 1, 2)$.
Javasoljunk $O(n)$ összehasonlítást használó módszert az A tömb rendezésére. (A megoldáshoz hasznos lehet a feltételeinknek az a következménye, hogy nem létezhet olyan $j < k < l$ indexhármas, amelyre $A[j] > A[k] > A[l]$.) ♢

12. Adott egy $A[1..n]$ tömb, amelynek elemei egy rendezett halmazból kerülnek ki. Tudjuk, hogy a tömbben nincs $A[i_1] > A[i_2] > A[i_3] > A[i_4]$ részsorozat, ahol $1 \le i_1 < i_2 < i_3 < i_4 \le n$. Adjunk $O(n)$ költségű algoritmust, amely nemcsökkenő sorrendbe rendezi a tömb elemeit. ♢

13. 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 $O(n)$ idejű algoritmust az eredeti állapot helyreállítására! ♢

14. Az egész elemeket tartalmazó $A[1:n]$ tömböt *lassan változónak* nevezzük, ha minden $0 < i < n$ indexre teljesül, hogy $|A[i] - A[i+1]| < 10$. Javasoljunk hatékony módszert lassan változó tömbök rendezésére; elemezzük a módszer költségét!

15. Négy elem rendezéséhez hány összehasonlítás kell? ♢

16. Öt elem rendezéséhez hány összehasonlítás kell? ♢

17. A | 6 | 4 | 8 | 3 | 7 | 2 | 5 | 1 | tömb rendezése során (a rendező algoritmus néhány lépése után) a következő közbülső állapot jött létre: | 4 | 6 | 3 | 8 | 7 | 2 | 5 | 1 | Az alább felsorolt, az előadáson tanult módszerek közül mely(ek) alkalmazásakor fordulhatott ez elő?
a) Beszúrásos rendezés,
b) Buborékrendezés,
c) Összefésüléses rendezés,
d) Gyorsrendezés? ♢

18. Adott egy egész számokat tartalmazó $A[1..n]$ tömb, amelyben legfeljebb n elempár áll inverzióban egymással (két elem akkor áll inverzióban, ha a nagyobb megelőzi a kisebbet). Igaz-e, hogy a buborék-rendezés rendezi az A tömböt
a) legfeljebb n összehasonlítással?
b) legfeljebb n cserével? ♢

19. Egy rendezési algoritmust *konzervatívnak* nevezünk, ha megtartja az egyforma elemeknek az eredeti sorrendjét. Az alábbi rendezési algoritmusok közül melyek konzervatívak:
(a) buborékrendezés; (b) összefésüléses rendezés;
(c) kupacrendezés; (d) gyorsrendezés? ♢

20. Az $A[1:n]$ tömb piros és zöld elemeket tartalmaz. Szeretnénk átrendezni úgy, hogy az egyszínű elemek folytonosan helyezkedjenek el (elöl az összes piros, utána a zöldek vagy fordítva). Egy megengedett *lépés* két szomszédos tömbelem cseréje. Javasoljunk konstans szorzó erejéig optimális lépésszámú algoritmust. ♢

21. Legyen adott egy egészekből álló $A[1:n]$ tömb valamint egy b egész szám. Szeretnénk hatékonyan eldönteni, hogy van-e két olyan $i, j \in \{1, \ldots, n\}$ index, melyekre $A[i] + A[j] = b$. Oldjuk meg ezt a feladatot $O(n \log n)$ időben! ♢

22. Az $A[1:n]$ tömbben egész számokat tárolunk. Tudjuk, hogy van olyan i index, amellyel

$$A[i] < A[i+1] < \ldots < A[n] < A[1] < \ldots < A[i-1]$$

teljesül. Adjunk minél kevesebb összehasonlítást használó módszert ennek az (egyértelműen meghatározott) i indexnek a megkeresésére. Elemezzük a módszer költségét! ♢

23. Adott két n hosszú rendezett lista, $a_1, \ldots, a_n$ és $b_1, \ldots, b_n$, amelyek összesen $2n$ különböző elemet tartalmaznak. Adjunk konstans szorzó erejéig optimális számú összehasonlítást használó algoritmust a $2n$ elem közül az n-edik legkisebb meghatározására! ♢

24. Legyen adott egy rendezett univerzum $2n$ különböző eleméből álló S halmaz. Szeretnénk az S elemeit egy $A[1:2n]$ tömbbe elhelyezni úgy, hogy

$$A[1] < A[2] > A[3] < A[4] > \cdots < A[2n-2] > A[2n-1] < A[2n]$$

teljesüljön. Adjunk meg egy $O(n)$ összehasonlítást használó algoritmust erre a feladatra! ♢

25. Az előadáson az összefésüléses rendezésnek (MSORT) egy rekurziót alkalmazó változatát körvonalaztuk. Javasoljunk egy iteratív (rekurziót, vermet nem használó) implementációt, melynek a költsége a korábbi változatéval egyező nagyságrendű!

26. Egy n elemű sorozat csupa 0-ból és 1-esből áll. Rendezzük a sorozatot $n-1$ összehasonlítással!

27. 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: apa $<$ anya, apa $=$ anya, vagy apa $>$ anya; annak megfelelően, hogy az apacsavar külső átmérője hogyan viszonyul az anyacsavar belső átmérőjéhez. Szeretnénk az anyacsavarokhoz megtalálni a megfelelő apacsavarokat. Adjunk erre a feladatra **átlagosan** $O(n \log n)$ összehasonlítást felhasználó módszert! $\diamondsuit$

28. Tegyük fel, hogy adott a $\Sigma = \{a, b, c, d\}$ abc feletti szavaknak egy S halmaza. Az S-beli szavak összhossza n. Javasoljunk egy $O(n)$ idejű módszert az S elemeinek (lexikografikus) sorbarendezésére! A Σ alaprendezése $a < b < c < d$.

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

30. Adott egy $A[1:n]$ tömb, ami 1 és k közötti egészekből áll. Szeretnénk $O(n+k)$ időben létrehozni egy adatstruktúrát, aminek segítségével konstans költséggel megoldható a következő feladat:
Bemenet: a és b természetes számok;
Meghatározandó: az A tömb azon elemeinek a száma, amelyek az $[a, b]$ intervallumba esnek.
Tervezzük meg az adatstruktúrát és a felépítéséhez szükséges algoritmust! $\diamondsuit$

31. Egy könyvtárban havonta mágneslemezre listázzák a kikölcsönzött könyveket a kölcsönzők neve szerinti sorrendben. A listán minden egyes könyvnek egy rekord felel, melyben a könyv azonosítására szolgáló kulcs egy 1 és $100N$ közé eső természetes szám, ahol N a könyvtári könyvek száma. Adjunk minél rövidebb idejű (konstans szorzó erejéig optimális uniform időigényű) módszert annak eldöntésére, hogy a januárban és februárban kikölcsönzött könyvek összessége megegyezik-e!

32. Egy személyek adatait tartalmazó nyilvántartásban n rekord van. A rekordokban szerepel a személy magassága és testsúlya is. Szeretnénk minél kevesebb munkával eldönteni, hogy vannak-e olyan X és Y személyek a nyilvántartásban, hogy X magasabb Y-nál, de Y nehezebb, mint X. Javasoljunk hatékony módszert a feladatra! Elemezzük a módszer költségét!

33. Tegyük fel, hogy inputként adott egész koordinátájú síkbeli pontok egy $\mathcal{S} = \{P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_{2n} = (x_{2n}, y_{2n})\}$ halmaza. A sík egy Q pontja (az $\mathcal{S}$-re nézve) *középső*, ha az $\mathcal{S}$-nek éppen n pontja van a Q felett, továbbá az $\mathcal{S}$-nek éppen n pontja van a Q-tól jobbra. Javasoljunk hatékony módszert adott $\mathcal{S}$-hez középső pont keresésére! Elemezzük a módszer költségét!

34. Adottak a sík egész koordinátájú $P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_{2n} = (x_{2n}, y_{2n})$ pontjai, melyek közül nincs három egy egyenesen. Javasoljunk $O(n)$ uniform költségű módszert egy olyan egyenes (két különböző pontjának) meghatározására, melynek mindkét oldalán ugyanannyi van a P_i pontok közül.

35. Adottak a sík egész koordinátájú $P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_n = (x_n, y_n)$ pontjai. Javasoljunk $O(n)$ uniform költségű módszert olyan $P_i \neq P_j$ pontok kiválasztására, amelyeken átmenő egyenes által meghatározott félsíkok közül az egyik tartalmazza az összes input pontot.

36. Legyenek $S_1, \ldots, S_k$ olyan nemüres halmazok, melyek elemszámainak összege n, és elemeik 1 és n közé esnek. Mutassunk algoritmust, mely az összes S_i $(1 \leq i \leq k)$ halmazt rendezi $O(n)$ idő alatt!

37. Adjunk hatékony algoritmust egy kupac tizedik legkisebb elemének a megtalálására. Elemezzük a módszer költségét. $\diamondsuit$

38. Igazoljuk, hogy egy n elemből álló bináris kupac felépítése $\Omega(n)$ összehasonlítást igényel! $\diamondsuit$

39. Adott egy n elemet tartalmazó 2-kupac és egy k kulcs. Keressük meg a kupac k-nál kisebb kulcsú elemeit! Ha m ilyen elem van, az algoritmus $O(m)$ elemi lépést használhat. $\diamondsuit$

40. Egy rendezett halmazból n elem kupacban van elhelyezve. Bizonyítsuk be, hogy a legnagyobb elem megkereséséhez $\Omega(n)$ összehasonlítás szükséges! $\diamondsuit$

41. Adjunk konstans szorzó erejéig optimális uniform költségű algoritmust az alábbi problémára:
INPUT: Egy $A[1:n]$ tömb, amely eredetileg az $1, \ldots, n$ természetes számokat tartalmazta kupacba rendezve, de öt elem megsérült és a helyére $*$ került.
FELADAT: Találjuk meg a tömb egy olyan kitöltését, ami lehetett az eredeti! $\diamondsuit$

42. Bizonyítsuk be, hogy ahhoz, hogy egy halmazból a két legnagyobb elemet kiválasszuk, $n + \lceil \log n \rceil - 2$ összehasonlítás szükséges és elégséges.

43. (a) (*) Javasoljunk egy olyan algoritmust, ami $\lceil 1.5n \rceil - 2$ összehasonlítással megtalálja egy n-elemű halmaz legnagyobb és legkisebb elemét!
(b) (**) Igazoljuk, hogy $\lceil 1.5n \rceil - 2$ összehasonlítás szükséges is a fenti feladat megoldására! $\diamondsuit$

44. Adott az $A[1:n]$ csupa különböző egész számot növekvő sorrendben tartalmazó tömb. (A tömbben negatív számok is lehetnek!) Adjunk hatékony algoritmust egy olyan i index meghatározására, melyre $A[i] = i$ (feltéve, hogy van ilyen i): igyekezzünk minél kevesebb elem megvizsgálásával megoldani a feladatot!

45. Egy n elemű rendezett halmaz *középső* elemének a rendezett sorrend szerinti $\lfloor (n+1)/2 \rfloor$-ik elemet nevezzük. Célunk egy n elemű rendezett halmaz elemeit kilistázni úgy, hogy legelöl a középső elem van, utána a maradék középső eleme stb. (pl. ha az elemek 1,2,3,4 akkor a kívánt sorrend 2,3,1,4.) A halmaz elemeit (tetsz. sorrendben) az $A[1:n]$ tömb tartalmazza. Javasoljunk egy olyan megoldást, mely a szükséges összehasonlítások számát tekintve konstans szorzó erejéig optimális.

46. Az egész elemeket tartalmazó $A[1:n]$ tömböt *konvexnek* nevezzük, ha minden i-re $(1 < i < n)$ teljesül, hogy $A[i] \leq 1/2(A[i-1] + A[i+1])$. Javasoljunk olyan algoritmust, mely minél kevesebb összehasonlítással megtalálja egy konvex tömb minimális elemét.

47. Van egy $T[1:n]$ tömbünk, melynek elemei valós számok. Keressük meg T maximális összegű folytonos résztömbjét, vagyis azon $1 \leq i \leq j \leq n$ indexeket, melyre a $T[i] + T[i+1] + \ldots + T[j]$ összeg maximális!

48. Adott három függőleges rúd és n darab ezekre felhúzható korong, melyek 1-től n-ig vannak számozva. Kezdetben a korongok mind az első rúdra vannak felhúzva tetszőleges sorrendben. A cél az, hogy a korongok a számozás szerinti növekvő sorrendben szerepeljenek mind egyazon rúdon. Egy lépésben tetszőleges rúdnak a legfölső korongját áttehetjük egy másik rúdra, a már ott levő korongok tetejére. Mutassuk meg, hogy a kívánt állapot eléréséhez szükséges lépésszám $\theta(n \log n)$. (A korongok "nem csúsznak egymásba": sorrendjük egy-egy rúdon mindig az odahelyezésük sorrendjét tükrözi.) $\diamondsuit$

49. Egy nyilvántartásban a rekordok a kulcsuk szerint (bináris) kupacba vannak rendezve. Feltesszük, hogy a kulcsok egész számok. Szeretnénk implementálni a *kulccsökkentés* műveletét: A bemenet a kupac egy eleme (pl. mutatóval megadva) valamint egy $i > 0$ szám, amivel az elem k kulcsát lecsökkentjük (az új kulcs tehát $k - i$), majd a kupac tulajdonságot helyreállítjuk. Oldjuk meg a feladatot minél hatékonyabban! Elemezzük a módszer költségét! $\diamondsuit$

50. Adott egy n méretű A tömb, amiben 0-k és 1-esek helyezkednek el. Szeretnénk egy B – szintén n méretű – tömböt kitölteni, hogy B i-edik eleme azon legnagyobb $j < i$ indexet tartalmazza, melyre $A[j] = 1$. Pontosabban, legyen

$$B[i] = \begin{cases} \max\{j | 0 < j < i \text{ és } A[j] = 1\}, & \text{ha létezik olyan } 0 < j < i, \text{ hogy } A[j] = 1, \\ 0 & \text{egyébként.} \end{cases}$$

Csináljunk olyan *párhuzamos* algoritmust, ami $O(n)$ processzor felhasználásával $O(\log n)$ időben kitölti a B tömböt!

51. Rendezett $a_1, \ldots a_n$ listában keresünk úgy, hogy egy lépésben egyszerre k darab "kisebb-e mint a_i" típusú kérdést lehet föltenni. Adjunk konstans szorzó erejéig optimális algoritmust!

52. Egy n-szer n-es táblázatban úgy van elhelyezve n^2 db különböző szám, hogy mind a sorok mind az oszlopok monoton növekvőek. Mutassuk meg, hogy ekkor a keresés lépésszáma $\Theta(n)$.

53. Valaki egy összehasonlításon alapuló állítólagos rendezési algoritmusról azt mondja, hogy mivel az eljárás minden olyan sorozatot jól rendez, amelyben a rendezett sorrendhez képest csak egyetlen pár van felcserélve, ezért az algoritmus tetszőleges sorozatot rendez. Helyes-e ez a következtetés? $\diamondsuit$

54. Az egész értékű $A[1:n]$ tömb rendezésére szolgáló program egy *CE-program* (compare-exchange program), ha minden utasítása

if $A[i] < A[j]$ **then** cseréljük fel az$A[i]$ és $A[j]$ értékeket;

alakú. Igazoljuk, hogy ha egy CE-program helyesen működik (az elemeket nem csökkenően rendezi) minden 0,1 értékű input esetén, akkor helyesen működik minden egészértékű input esetén is!

55. Az $I = [0, 2^{100} - 1]$ intervallum egy ismeretlen x egész elemét szeretnénk meghatározni *"Igaz-e, hogy $x < i$?"* alakú kérdésekkel, ahol $i \in I$ egy egész. Tudjuk még, hogy a kapott válaszok közül egy hibás lehet. Javasoljunk egy legfeljebb 150 kérdést felhasználó stratégiát!

4. Keresőfák

1. 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, mint tetszőleges J-beli csúcs kulcsa? ♢

2. A következőket tudjuk egy fa m, n elemeiről: m megelőzi n-t a fa X-order szerinti bejárásában, viszont m az n után jön a fa Y-order szerinti bejárásában (X,Y= $\{pre, post, in\}$). Melyik eset(ek)ben tudjuk eldönteni, hogy m őse-e n-nek?

3. Egy bináris keresőfa "valamely bejárásán" mindig a $\{pre,\ in,\ post\}$-order valamelyikét értjük.
(a) Mely bejárásoknál lehetséges az, hogy a tárolt elemek legnagyobbika megelőzi a legkisebbet?
(b) Tegyük fel, hogy egy bináris keresőfában az $1, 2, \ldots, n$ számok vannak tárolva, továbbá hogy a fa valamely bejárásánál a számok az $n, n-1, \ldots, 1$ sorrendben következnek. Határozzuk meg, melyik lehetett ez a bejárás és milyen lehetett ez a bináris keresőfa!

4. Egy rendezett univerzum n eleme egy A bináris keresőfában, k eleme pedig egy B bináris keresőfában van tárolva. Adjunk minél hatékonyabb módszert a két fában levő elemek nagyság szerinti sorrendben történő kilistázására! (Tehát egy olyan tömböt szeretnénk kapni, amiben az $n + k$ elem rendezetten helyezkedik el.) Elemezzük a módszer költségét! ♢

5. Egy gyökeres fában egy csúcs foka legyen a gyerekeinek a száma. Mutassuk meg, hogy minden bináris fában a levelek száma eggyel több, mint a másodfokú csúcsok száma!

6. 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 fodul elő a fában. Utóbbi feltétel szerint pontosan egy olyan $i \in I$ egész van, amely *nincs* a fában. Adjunk egy hatékony módszert i meghatározására. ♢

7. Illesszük be az alábbi 6 kulcsot egy kezdetben üres (2, 3)-fába a megadott sorrendben: D, B, E, A, C, F. Rajzoljuk le az eredményül kapott fát! ♢

8. Egy 2-3 fában egy rendezett halmaz 10 000 elemét szeretnénk tárolni. Milyen korlátok közé esik a fa magassága?

9. Az $[1, 178]$ intervallum összes egészei egy 2-3 fában helyezkednek el. Tudjuk, hogy a gyökérben két kulcs van, és az első kulcs a 17. Mi lehet a második? Miért? ♢

10. Egy B_{20}-fának (huszadrendű B-fának) 10^9 levele van. Mekkora a fa szintjeinek minimális, illetve maximális száma?

11. Egy 1 000 000 rekordból álló adatállományt B-fában szeretnénk tárolni. A rekordok hossza 200 byte. A kulcs hossza 40 byte. Egy mutató helyigénye 5 byte. Tegyük fel, hogy a lapméret 2000 byte. Adjunk minél pontosabb felső becslést a szükséges lapok (blokkok) számára.

12. Az S_1 és S_2 kulcshalmazokat kiegészített 2-3-fákban tároljuk. Ezek az eredeti 2-3-fától abban különböznek csak, hogy minden csúcsban fel van jegyezve az onnan induló részfa magassága (szintjeinek száma). Tegyük még fel, hogy az S_1-beli kulcsok mind kisebbek az S_2-belieknél. Javasoljunk hatékony algoritmust a két fa egyesítésére. A cél tehát egy olyan kiegészített 2-3-fa, amelyben a kulcsok $S_1 \cup S_2$ elemei. ♢

13. Egy fában az x csúcs *súlya* x leszármazottainak a száma. Egy bináris fát *szigorúan binárisnak* mondunk, ha a levelek kivételével minden csúcsnak 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 $2^k - 1$. ♢

14. A maximum n mélységű szigorúan bináris fából $t(n)$-féle van. Bizonyítsuk be, hogy
(a)$t(n) = t^2(n-1) + 1$
(b)Létezik a valós szám, melyre $t(n) = \lfloor a^{2^n} \rfloor$.

15. Adott n pont a síkon, melyek páronként mindkét koordinátájukban különböznek. Bizonyítsuk be, hogy egy és csak egy bináris fa létezik, melynek szögpontjai az adott n pont, és az első koordináta szerint a keresőfa tulajdonsággal, a második szerint pedig a kupac tulajdonsággal rendelkezik. (Vigyázat: a kupac tulajdonságba nem értendő bele, hogy a fa teljes bináris fa legyen, mint amilyet a tanult "kupacépítő" algoritmus létrehoz.) ♢

16. Egy rendezett halmaz adott n eleméből kupacot szeretnénk építeni, melyre (a kupac tulajdonság mellett) teljesül, hogy minden x csúcs bal részfájának az elemei kisebbek mint az x jobb részfájának az elemei.
Igazoljuk, hogy ehhez legalább $\Omega(n \log n)$ összehasonlítás szükséges!

17. Adottak a sík egész koordinátájú $P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_n = (x_n, y_n)$ pontjai. Javasoljunk $O(n \log n)$ költségű módszert annak eldöntésére, hogy vannak-e olyan P_i, P_j pontok ($i \neq j$), melyek távolsága nem több mint 2. ♢

18. Adjuk meg az $\{1, 2, \ldots, 6\}$ számoknak egy olyan sorrendjét, amikor az AVL-fát építő algoritmus az adott input sorozat esetén alkalmaz dupla forgatást.

19. Építsünk AVL-fát az alábbi input számsorozatból: $4, 3, 2, 1, 7, 6, 5$. ♢

20. Határozzuk meg a nyolc szintből álló AVL-fák minimális, illetve maximális csúcsszámát!

21. Adjunk példát olyan AVL fára, hogy egy alkalmas törlés esetén egyetlen forgatás ne legyen elegendő az AVL-tulajdonság helyreállítására.

22. Adott egy n pontú AVL-tulajdonágú bináris fa. Adjunk meg polinomidejű algoritmust az $1, \ldots, n$ számok egy olyan sorrendjének meghatározására, amely esetén az AVL-fa építő algoritmus a megadott fát hozza létre, mégpedig forgatás nélkül. ♢

23. Egy AVL-fába egy új elemet illesztettünk be az előadáson tanult módszerrel. Az eredményképpen kapott AVL-fa magassága nagyobb, mint az eredetié. Milyen forgatást alkalmazhattunk? ♢

24. Igazoljuk, hogy egy tetszőleges n pontú bináris fa legfeljebb $n-1$ darab egyszeres forgatással átalakítható olyan fává, melyben egyetlen csúcsnak sincs bal fia (szemléletesebben: jobbra menő úttá alakítható). ♢

25. Adjuk meg az $\{1, 2, \ldots, 6\}$ számoknak egy olyan sorrendjét, amikor az AVL-fát építő algoritmus az adott input sorozat esetén alkalmaz dupla forgatást.

26. Építsünk AVL-fát az alábbi input számsorozatból: $4, 3, 2, 1, 7, 6, 5$. ♢

27. Határozzuk meg a nyolc szintből álló AVL-fák minimális, illetve maximális csúcsszámát!

28. Tegyük fel, hogy az AVL fa építő algoritmus inputja az $1, 2, 3, \ldots, 2^n - 1$ sorozat. Mi lesz a felépített AVL fa?

29. Egy l szintű bináris keresőfa csúcsaiban a kulcsokon és a részfák gyökereire mutató pointereken kívül tároljuk a megfelelő részfa súlyát (a csúcs leszármazottainak a számát). Tudjuk, hogy a kulcsok mind különbözőek, valamint hogy a fa minden belső (nem levél) csúcsának pontosan két fia van. Adjunk minél hatékonyabb algoritmust egy olyan levél keresésére, aminek a k kulcsa a lehető legközelebb van a kulcsok rendezése szerinti középső kulcshoz! Másszóval, ha m-mel jelöljük a középső kulcsot, olyan k kulcsú levelet keresünk, hogy ne legyen olyan k' kulcsú levél, hogy $m < k' < k$ vagy $k < k' < m$. Elemezzük a módszer költségét!

30. Adjunk meg olyan adatszerkezetet, amely lehetővé teszi, hogy
(a) az 1 és n közötti egészeket tároljuk;
(b) konstans időben beszúrjunk egy (eddig még nem tartalmazott) elemet;
(c) konstans időben töröljünk egy (közelebbről meg nem határozott, de tartalmazott) elemet!

31. Egy sportklub teniszezői kialakítottak egy erősorrendet. Ezt a következő szabályok szerint tartják karban:

    - új játékos a sorrend végére kerül;
    - a rangsor szerinti i-edik játékos kihívhatja az $i-1$-ediket; ha legyőzi, helyet cserélnek.

    Tervezzünk olyan hatékony adatszerkezetet, mely lehetővé teszi a rangsor számítógépes kezelését! A szükséges funkciók a következők:

    - **Beszúr**(név): az új jövevényt a rangsor végére teszi
    - **Kihív**(név): az $i-1$. helyen álló személy nevét adja, ha a "név"-vel azonosított személy az i. helyen áll és $i > 1$.
    - **Kicserél**(i): kicseréli a rangsor i. és $i-1$. helyén levő személyeket, ha $i > 1$.

32. Tervezzünk olyan adatszerkezetet, ami egy rendezett halmaz elemeinek tárolására szolgál. A megvalósítandó műveletek:

    - **Felépít**(n) n elemből felépiti a struktúrát
    - **Mintör**, **Maxtör** a min. illetve max. elem törlése
    - **Beszúr**(x) az x elemet a struktúrába illeszti.

    Az egyes műveletek uniform költsége ne legyen több, mint **Felépít**: $O(n)$; **Mintör, Maxtör, Beszúr**: $O(\log n)$, ahol n a tárolt elemek száma.

33. Két mezőből álló rekordokat szeretnénk tárolni, melyek mezői: *személynév* valamint *telefonszám*. A következő műveleteket kell hatékonyan megvalósítani: rekord beillesztése, törlése, keresés név, illetve telefonszám alapján, *tól-ig* típusú kérdések a személynévre vonatkozóan. Javasoljunk egy hatékony adatszerkezetet és elemezzük a költségét!

5. Adattömörítés

1. Bizonyítsuk be, hogy adott $p_1 \geq p_2 \geq \ldots \geq p_n$ eloszláshoz van olyan optimális (Huffman-) kód, amire teljesül, hogy
   (a) $p_i \geq p_j \Longrightarrow$ az i-edik kódszó legfeljebb olyan hosszú, mint a j-edik;
   (b) az n-edik és $n-1$-edik kódszavak hossza megegyezik; és
   (c) az n-edik és $n-1$-edik kódszavak csak az utolsó jegyükben különböznek!

2. Bizonyítsuk be, hogy ha adott $p_1 \geq p_2 \geq \ldots \geq p_n$ számok esetén a $p_1, p_2, \ldots, p_{n-2}, p_{n-1}+p_n$ eloszláshoz ismert egy optimális kódolás, akkor a $p_1, p_2, \ldots, p_{n-2}, p_{n-1}, p_n$ eloszláshoz optimális kódolás készíthető oly módon, hogy az előzőz egy 0-t, illetve egy 1-est illesztünk.

3. Legyenek $p_1 \leq \ldots \leq p_n$ egy $n = 2^k \geq 4$ elemű ábécé elemeinek az előfordulási valószínűségei egy szövegben $(\sum p_i = 1)$. Tegyük fel, hogy az összes k hosszú 0-1 sorozatból álló kódolás egy optimális (Huffman) kódot ad. Mekkora lehet p_n legnagyobb értéke?

4. A $p_1 \geq p_2 \geq p_3 \geq p_4$ valószínűség-eloszláshoz ketten készítettek Huffman-kódot. Egyikük kódszavai: 0, 10, 110, 111; másikukéi: 00, 01, 10, 11. Tudva, hogy $p_3 = 1/6$, határozzuk meg p_1, p_2, p_3 és p_4 értékét. $\diamondsuit$

5. Tegyük fel, hogy egy szöveg az A, B, C, D, E betűkből áll. A betűk előfordulási gyakoriságai úgy aránylanak egymáshoz, mint 2:3:4:5:6.
   (a) Adjuk meg a betűk egy Huffman kódolását.
   (b) Mutassuk meg, hogy a kódszavak hossza minden betű esetén egyértelműen meghatározott (független a választott kódolástól). $\diamondsuit$

6. Adjunk meg egy Huffman kódot a MISSISSIPPIT szó tárolására. Mekkora a helymegtakarítás a betűk uniform hosszúságú kódolásához képest?

7. Határozzuk meg a BARBARÁNAK szó egy lehetséges Huffman-kódját és Lempel–Ziv-Welch-kódját. Az utóbbinál tegye fel, hogy a szótár a kezdőhelyzetben az A, Á, B, K, N, R betűket tartalmazza, és ezek kódjai rendre 1, 2, 3, 4, 5, 6. $\diamondsuit$

8. Adjunk meg egy olyan eloszlást, melyhez tartozó bármely optimális kódolásban van hosszabb szó, mint amilyen hosszú szó egy nem feltétlenül optimális kódolásban a maximimális szóhossz.

9. Adott egy k pozitív egész. Adjunk meg egy olyan eloszlást, melyhez tartozó optimális kódban minden kódszó hossza nagyobb lesz k-nál.

10. A $p_1 \leq p_2 \leq \ldots \leq p_n$ eloszlásokhoz Huffman-kódot készítettünk. Mekkora lehet p_1 lehető legnagyobb értéke, ha a kapott kódszavak a következők:

$$1, 01, 001, 0001, 00001, \ldots, \underbrace{000\cdots01}_{n-2}, \text{ valamint } \underbrace{000\cdots00}_{n-1} \quad ?$$

11. Két dobókockánk van, mindegyiken 2 db egyes, 2 db kettes és 2 db hármas található. A két kockával egyszerre dobunk és a dobott számok összegét bináris kóddal kódoljuk. Adjunk meg egy olyan bináris kódot, hogy száz dobás eredményének kódja várhatóan a lehető legrövidebb legyen!

12. Legyen T egy olyan bináris fa, melyben minden nem-levél csúcsnak pontosan 2 fia van. Igaz-e, hogy van olyan eloszlás, amelyhez tartozó Huffman-fa éppen T?

13. Az $\{x, y, z\}$ háromelemű ábécé mellett futtatjuk a Lempel–Ziv–Welch algoritmust egy adott szövegre. A tömörítés egy pontján bekerül a szótárba az xy szó kódja. Amennyiben később szerepel a szövegben xy részszó, biztos-e, hogy együtt fogjuk őket kódolni? $\diamondsuit$

14. A $\Sigma = \{a, b\}$ ábécé feletti szövegeket Huffman és Lempel–Ziv–Welch módszerrel is kódoljuk. A Lempel-Ziv szótárban minden karaktersorozatnak 8 bites kódot adunk. Lesz-e így a szövegek között olyan, amelynek a Lempel-Ziv kódja rövidebb lesz a Huffman-kódjánál? $\diamondsuit$

15. Egy S szöveg tömörítésére a Lempel-Ziv-Welch módszert alkalmaztuk. Azt tapasztaltuk, hogy a kódolás során használt szótárban előfordult 100 betűből álló szó. Adjunk minél jobb alsó becslést az S szöveg hosszára! $\diamondsuit$

16. Egy 1000 betűből álló szöveget kódoltunk a Lempel-Ziv-Welch módszerrel. Legalább hány kódból áll az eredmény, ha a szótár méretére nincs korlát? $\diamondsuit$

17. Legfeljebb milyen hosszú lehet egy n-elemű ábécé feletti szó, amit a Lempel–Ziv–Welch algoritmus nem tömörít (tömörítésen itt az értendő, hogy az algoritmus egynél hosszabb betűsorozatot helyettesít a kódjával)? $\diamondsuit$

6. Hashelés

1. Egy 1 000 000 rekordból álló adatállományt mágneslemezen (vödrös módszerrel) hashelt szervezésben szeretnénk tárolni. A rekordok hossza 200 byte. Egy mutató helyigénye 5 byte. Tegyük fel, hogy a lapméret 2000 byte. Adjunk becslést a szükséges lapok (blokkok) számára azzal a feltevéssel, hogy a vödör-katalógust is a lemezen kell tárolni.

2. Egy u elmű halmaz elemeit egy v méretű vödörkatalógus segítségével tartjuk hashelt állományban. Igazoljuk, hogy akármilyen jó hash-függvény alkalmazása esetén is a keresés $\Omega(u/v)$ időt igényel.

3. Mutassuk meg, hogy (nyitott címzéses hashelés, lin. próbálkozás esetén) már két kulcshoz tartozó hash-függvényérték megegyezése is okozhat "tetszőlegesen" nagy méretű csomósodást.

4. Mi a baja az $f(K) = K^2 \pmod 7$ hash-függvénynek, ahol 7 a táblaméret? $\diamondsuit$

5. A hash-függvény legyen $f(K) = K$, a táblaméret $M = 7$, és $1 \leq K \leq 20$. Helyezzük el a táblában a 3, 4, 7, 11, 14, 17, 20 kulcsokat ebben a sorrendben
   (a)lineáris
   (b)kvadratikus maradék
   próbálást használva az ütközések feloldására.

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

7. A $T[0:M]$ táblában $2n$ elemet helyeztünk el az első $3n$ helyen $(3n < M)$ egy ismeretlen hash-függvény segítségével. A táblában minden $3i$ indexű hely üresen maradt $(0 \leq i < n)$. Legfeljebb hány ütközés lehetett, ha az ütközések feloldására
   a) lineáris próbálást
   b) kvadratikus maradék próbálást használtunk? $\diamondsuit$

8. A $T[0:M-1]$ táblában rekordokat tárolunk nyitott címzésű hashelt szervezéssel. Az ütközések feloldására lineáris próbálást alkalmazunk. Tehát ha a $h(K)$ sorszámú cella foglalt, akkor a K kulcsú rekordot a $h(K)-1, h(K)-2, \ldots$ sorszámú cellák közül az első üresbe tesszük. Tegyük fel, hogy a tábla használata során egy hibás törlés történt: egy cellából kitöröltünk egy rekordot a törlés-bit beállítása nélkül.
   (a) Igaz-e, hogy a hibás törlés helye mindig megtalálható?
   (b) Adjunk hatékony (lineáris időigényű) algoritmust a tábla megjavítására. (Módosítsuk úgy a táblát, hogy megszűnjenek a hibás törlés negatív következményei.) $\diamondsuit$

9. Nyitott címzéssel, lineáris próbálás módszerével akarjuk az $K_1 < K_2 < \ldots < K_n$ kulcsú elemeket egy tömbbe hash-elni a beszúrási algoritmus következő módosításával: Ha egy K kulcsú elem beszúrásának megkísérlésekor a K-nál nagyobb K' kulcsú elem foglalja el a cellát, akkor a K kulcsú elemünket behelyezzük ebbe a cellába, és a beillesztést K helyett a K' kulcsú elemmel folytatjuk a következő cellánál (és ha ott a K'-nél nagyobb K'' kulcsú elemet találjuk, akkor az előbbihez hasonlóan járunk el...). Bizonyítsuk be, hogy az n elem beillesztése után kapott tömb független az elemek beszúrási sorrendjétől!

7. Gráfalgoritmusok

1. Működik-e Dijkstra algoritmusa
   (a) irányítatlan gráfokra?
   (b) olyan gráfra, melyben lehet negatív súlyú él?
   (c) egy olyan gráfra, amelyről annyit tudunk, hogy nincs benne negatív összsúlyú irányított kör? $\diamondsuit$

2. 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. Adja meg a legrövidebb utakat tartalmazó P[ ] tömb állapotait is.

v_1	v_2	v_3	v_4	v_5	v_6
0	2	6	∞	∞	7
0	2	5	9	∞	6
0	2	5	6	9	6
0	2	5	6	8	6
0	2	5	6	7	6

♢

3. Javasoljunk egy $O(|V|^2)$ költségű algoritmust a következő problémára: Adott a $G = (V, E)$ irányított gráf pozitív élsúlyokkal és egy $s \in V$ csúcs. Meghatározandó a legrövidebb s-ből v-be vezető utak száma az összes $v \in V$ csúcsra.

4. Adott egy G egyszerű irányított gráf éllistával, nemnegatív élsúlyokkal. Legyen v_1 egy tetszőleges csúcsa G-nek. Adjunk minél hatékonyabb módszert a v_1-hez legközelebbi 100 csúcs megtalálására! ♢

5. Legyen adott egy $n \times n$ pixelből álló fekete-fehér kép. Szeretnénk a képen a bal felső saroktól a jobb alsó sarokig egy jobbra-lefele haladó határvonalat húzni úgy, hogy a vonaltól jobbra-felfele eső fekete, valamint a vonaltól balra-lefele eső fehér pixelek számának az összege a lehető legkisebb legyen. Oldjuk meg ezt a feladatot $O(n^2)$ időben! ♢

6. Legyen $G = (V, E)$ adjacencia-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 $O(n^2)$ uniform költségű módszert az $1 \in V$ pontból az összes további $v \in V$ pontokba vivő legrövidebb utak hosszának a meghatározására. ♢

7. 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!

8. Legyen G egy DAG (irányított kört nem tartalmazó irányított gráf). Javasoljunk minél hatékonyabb módszert egy G-beli leghosszabb út keresésére. Elemezzük a módszer időigényét!

9. Egy irányított gráfban a csúcsoknak három diszjunkt részhalmaza van kijelölve: A, B és C. Adjunk olyan algoritmust, ami eldönti, hogy van-e a gráfban olyan irányított séta ami A-ból indul, átmegy legalább egy B-beli ponton és C-ben végződik. (A séta nem egyszerű utat jelent, azaz lehetnek benne körök.)

10. Egy $G = (V, E)$ irányított gráf egy *tranzitív redukáltja* egy olyan $G_1 = (V, E_1)$ irányított gráf, amelyre E_1 a lehető legkevesebb élt tartalmazza úgy, hogy G és G_1 tranzitív lezártja ugyanaz legyen.
(1) Bizonyítsuk be, hogy ha a G gráf DAG, akkor a tranzitív redukáltja egyértelmű!
(2) Milyen gráfok lehetnek egy erősen összefüggő gráf tranzitív redukáltjai?
(3) Adjunk hatékony algoritmust egy gráf egy tranzitív redukáltjának előállítására!

11. Legyenek $x_1, \ldots, x_n$ Boole-változók. Legyen adott
"Ha x_i igaz, akkor x_j is igaz."
típusú állítások egy rendszere ("axiómarendszer"). Készítsük el azt az irányított gráfot, melynek a csúcsai $x_1, \ldots, x_n$, éleit pedig úgy kapjuk, hogy minden egyes $x_i \Rightarrow x_j$ "axiómának" megfelelően behúzzuk az $x_i \to x_j$ élt.
(a) Hogyan értelmezhetők a gráf tranzitív lezártjának az élei?
(b) Hogyan értelmezhetők a gráf erős (erősen összefüggő) komponensei? ♢

12. Legyen most olyan rendszerünk, amiben a
"Ha x_i igaz, akkor x_j hamis.", *"Ha x_i hamis, akkor x_j igaz."*, valamint *"Ha x_i hamis, akkor x_j is hamis."*
típusú "axiómák" is meg vannak engedve. Hogyan lehet gráf segítségével megfogalmazni azt, hogy a rendszerünk tartalmaz -e önellentmondást?

13. Legyen adva egy $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_4 \vee \bar{x}_5) \wedge (\ . \ \vee \ . \ ) \wedge \cdots$ típusú Boole-kifejezésünk: kéttagú "vagy"-kifejezések "és"-éből áll; egy "vagy" kifejezés egy tagja egy változó (x_i), vagy egy változó negáltja ($\bar{x}_i$) lehet. Adjunk hatékony algoritmust annak eldöntésére, hogy a formula kielégíthető-e (van -e a változóknak olyan "igaz–hamis" behelyettesítése, hogy a kifejezés értéke "igaz").

14. A 6 pontú 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? ♢

15. Tegyük fel, hogy G egy összefüggő irányítatlan gráf, melynek minden mélységi feszítőfája egy egyszerű irányított út.
(a)Igazoljuk, hogy nincs G-nek artikulációs pontja (olyan csúcsa, melyet az összes belőle induló éllel együtt elhagyva G-ből, a kapott gráf már nem lesz összefüggő)!
(b)Mutassuk meg, hogy G nem szükségképpen teljes gráf!

16. Egy gráf minden szélességi feszítőfája (irányítatlan gráfként nézve) csillag. Mit mondhatunk a gráfról? ♢

17. A szélességi ill. mélységi bejárás közül melyik alkalmazható egy gráf összefüggő komponenseinek meghatározására? ♢

18. Legyen G egy éllistával adott irányítatlan gráf. Adjunk hatékony algoritmust minimális hosszúságú (élszámú) G-beli kör keresésére!

19. Éllistával adott a súlyozott élű $G = (V, E)$ gráf. Tegyük fel, hogy az élek súlyai az 1,2,3 számok közül valók. Javasoljunk $O(n + e)$ uniform költségű algoritmust az $s \in V$ pontból az összes további $v \in V$ pontokba vivő legrövidebb utak hosszának a meghatározására. (Itt n a G gráf csúcsainak, e pedig az éleinek a száma.) ♢

20. 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 $|V|$)! ♢

21. Adott (éllistával) egy G irányítatlan gráf, melynek bizonyos élei zöld színűek. Adjunk hatékony algoritmust olyan G-beli feszítőfa keresésére, melyben pontosan 2 zöld él szerepel! Elemezzük a módszer költségét! ♢

22. Adott éllistával egy összefüggő, egyszerű, irányítatlan, n pontú, e élszámú gráf. Javasoljunk $O(n+e)$ idejű algoritmust egy olyan csúcs keresésére, amely a többi pont bármelyikéből elérhető egy legfeljebb $n/2$ élet tartalmazó úton. ♢

23. Adott éllistával egy n pontú, e élű G összefüggő irányítatlan gráf. Adjunk $O(e)$ uniform 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. ♢

24. Hány éle lehet maximálisan egy olyan irányítatlan gráfnak, melynek van olyan mélységi bejárása, hogy a kapott mélységi feszítő erdő egy $2^n - 1$ szögpontú teljes bináris fa?

25. Egy n pontú egyszerű, irányított gráf egy mélységi bejárása során feljegyeztük az egyes csúcsok mélységi, illetve befejezési számát. Sajnos – a csúcsokon és számaikon kívül – minden egyéb adatunk elveszett, a gráf élei sincsenek meg. Adjunk $O(n)$ idejű algoritmust a mélységi feszítőerdő éllistájának visszaállítására! ♢

26. Legyen G egy k élű összefüggő irányítatlan gráf. Adjunk lineáris idejű algoritmust, amely megcímkézi G éleit az $1, \ldots, k$ számokkal úgy, hogy minden számot pontosan egyszer használjunk fel, továbbá minden egynél nagyobb fokszámú pontra teljesüljön, hogy a rá illeszkedő élek címkéinek a legnagyobb közös osztója 1. ♢

27. G irányítatlan gráf a következő éllistával (zárójelben a költségek, az élek mindkét végpontjukból fel vannak sorolva):
a:b(2),c(3); b:a(2),d(2); c:a(3),d(1); d:b(2),c(1),e(2),f(4); e:d(2),f(1),g(2); f:d(4),e(1),g(2),h(1); g:e(2),f(2),h(3); h:f(1),g(3);
Keressünk G-ben
(a) Prim algoritmusával minimális költségű feszítőfát!
(b) Kruskal algoritmusával minimális költségű feszítőfát!
(c) a-ból kiinduló mélységi feszítőfát!
(d) a-ból kiinduló szélességi feszítőfát!
(e) Határozzuk meg G artikulációs pontjait!

28. Legyen adva egy (egyszerű, irányítatlan, összefüggő) n pontú 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 G' egy minimális költségű feszítőfája. Adjunk minél hatékonyabb algoritmust a G gráf egy minimális költségű feszítőfájának az elkészítésére! (Teljes értékű megoldás: $O(n \log n)$ idejű algoritmus.) ♢

29. Legyen $G(L, U; E)$ a következő páros gráf:
$L = \{1, 2, 3, 4, 5\}$, $U = \{6, 7, 8, 9, 10, \}$; az éllista L-ből:
1:6,7,8; 2:6,9,10; 3:6,7; 4:8,9,10; 5:6;
Keressünk G-ben max. párosítást a magyar módszerrel!

30. Legyen adott éllistával a kétrészes $G = (L, U; E)$ gráf, aminek $2n$ pontja van úgy, hogy $|L| = |U| = n$; éleinek a száma pedig e. Adjunk egy $O(ne)$ uniform költségű algoritmust azon élek meghatározására, amelyek benne vannak egy maximális párosításban! Más szóval, összesen $O(ne)$ időben minden élről döntsük el, hogy szerepel-e maximális párosításban! ♢

31. Egy 20 szobás iroda számítógépeit hálózatba szeretnénk kötni. Az iroda szobái egy 2 méter széles folyosó két oldalán helyezkednek el; mindegyik szoba 3 méter széles (a folyosóval párhuzamos szélességről van szó). A folyosó egy lépcsőházból nyílik. Mindegyik szobában egy számítógép van, éspedig a folyosó felőli falnak a lépcsőház felőli sarkában. Oldjuk meg a lehető legrövidebb összhosszú vezetékkel, hogy bármely számítógépről bármely másik (esetleg közvetve) elérhető legyen a hálózaton. (Bármely két számítógép között vezethetünk egyenesvonalú vezetéket a padlóban. Nem szükséges, hogy egy összeköttetés a falakkal párhuzamos legyen.)

32. A szoftverpiacon n féle grafikus formátum közötti oda-vissza konverzióra használatos programok kaphatók: az i-edik és a j-edik között oda-vissza fordító program ára a_{ij}, futási ideje pedig t_{ij} (ha létezik).
(a) Javasoljunk módszert annak megtervezésére, hogy minden egyes formátumról a saját grafikus terminálunk

által megértett formátumra a lehető leggyorsabban konvertáljunk! (Az ár nem számít.)
(b) Javasoljunk módszert annak eldöntésére, hogy mely programokat vásároljuk meg, ha azt szeretnénk a lehető legolcsóbban megoldani, hogy a megvett programok segítségével bármelyik formátumról bármelyik más formátumra képesek legyünk konvertálni. (Itt a futási idő nem számít).

33. Egy ügyfél elektronikus archívumában n féle grafikus formátum egyikében szeretné tárolni a képeit. Rendelkezésre állnak bizonyos konverterprogramok. Az i-edik formátumról a j-edikre fordító program futási ideje t_{ij} (ha létezik). Javasoljunk módszert annak meghatározására, hogy melyik legyen a tárolási formátum, ha a megrendelő kívánsága az, hogy a különféle formátumokra történő konverziók átlagos futási ideje a lehető legrövidebb legyen!

34. Az ország n különböző $X_1, X_2, \ldots, X_n$ városában ma délután 4-kor futballmérkőzéseket fognak játszani. Egy televiziós társaságnak m operatőre van, akikről tudjuk, hogy délután 2-kor az $Y_1, Y_2, \ldots, Y_m$ városokban lesznek. Adott a városok távolsága is, pontosabban ismert, hogy mennyi idő alatt lehet Y_i-ből X_j-be eljutni. Adjunk hatékony algoritmust arra, hogy a televiziós társaság minél több mérkőzést rögzíteni tudjon (az elejétől a végéig)! Elemezzük a módszer futási idejét!

35. Egy város úthálózata a $c_1, c_2, \ldots, c_n$ csomópontok (kereszteződések) között menő $u_1, u_2, \ldots, u_m$ egyirányú utcákból áll. Tudjuk azt is, hogy nem lehet semmilyen (az utcákat az irányításuknak megfelelően használó) útvonallal sem körbe menni. A városházánál egy kitüntetett csomópont van (c_1). Adjunk minél hatékonyabb módszert azon csomópontok meghatározására, amelyekből egyértelmű útvonalon lehet eljutni a városházához! Elemezzük a módszer költségét!

36. Egy vasúti menetrend segítségével meg akarjuk határozni, hogyan tudunk leggyorsabban eljutni A városból B-be. Azaz adott, hogy melyik városból hova megy közvetlenül vonat, mennyi idő alatt ér oda, ill. ha át akarunk szállni egyik vonatról a másikra, az adott helyen mennyi a várakozási idő. Ezek ismeretében adjunk hatékony algoritmust, amely meghatároz egy legkevesebb ideig tartó utat! ♢

37. Van 12 darab edény, egyenként 100, 97, 27, 47, 55, 201, 258, 1000, 984, 766, 591, 631 liter űrtartalmúak. Kezdetben az 1000 literes edény tele van, az összes többi üres. Egy megengedett lépés a következő: egy edény teljes tartalmát áttöltjük egy másik edénybe vagy pedig egy edényt egy másikból teljesen feltöltünk. Adjunk algoritmust, ami eldönti, hogy elérhető-e megengedett lépések sorozatával az az állapot, amikor a felsorolásban szereplő első nyolc edényben $25-25$ liter, a többiben pedig $200-200$ liter folyadék van, és amennyiben igen, megadja az ehhez szükséges lépések minimális számát. ♢

38. Egy sakkversenyen n versenyző vesz részt. Adott az eddig lejátszott m játszma jegyzőkönyve. Adjunk minél hatékonyabb módszert annak eldöntésére, hogy volt-e körbeverés (olyan $v_1, v_2, \ldots, v_r$ sakkozók, hogy v_1 megverte v_2-t, v_2 megverte v_3-at, v_{r-1} megverte v_r-t, és v_r megverte v_1-et)! Elemezzük a módszer költségét!

39. Adott egy háromszögmentes irányítatlan gráf az adjacenciamátrixával. Adjunk $O(n^3)$ idejű algortimust a gráfban levő négy hosszú körök számának meghatározására! ♢

40. Bizonyítsuk be, bármely algoritmus, mely az inputként adjacencia mátrixával megadott $n > 2$ szögpontú irányítatlan gráfról eldönti, hogy erdő-e, kedvezőtlen esetben meg kell hogy tekintse az adjacencia mátrix $\binom{n}{2}$ elemét. ("Minden egyes pontpárról le kell kérdezni, hogy éle-e a gráfnak.") ♢

41. Igazoljuk, hogy nincs olyan algoritmus, ami egy adjacencia mátrixával adott n pontú irányítatlan gráf összefüggőségét eldönti a mátrix $\binom{n}{2}$-nél kevesebb elemének a beolvasásával! (Másképp fogalmazva, az összefüggőség kérdésére helyest választ adó módszereknél a legrosszabb esetben szükség van mind az $\binom{n}{2}$ lehetséges él lekérdezésére.) ♢

42. Adjunk algoritmust egy összefüggő, éllistával megadott gráf artikulációs pontjainak megkeresésére $O(|E|)$ időben.

43. Egy gráf *kétszeresen összefüggő*, ha nincs artikulációs pontja.
Legyen G egy összefüggő gráf. G élhalmazán, E-n értelmezzük a következő relációt:
$e_1 \sim e_2$, ha $e_1 = e_2$ vagy van egy mindkettőn átmenő kör G-ben.
(a)Biz. be, hogy $\sim$ egy ekvivalenciareláció E-n.
A $\sim$ reláció osztályait hívjuk G *kétszeresen összefüggő komponenseinek.*
(b)Biz. be, hogy G kétszeresen összefüggő komponensei kétszeresen összefüggő gráfok.
(c)Adjunk algoritmust G kétszeresen összefüggő komponenseinek meghatározására.

44. Adott éllistával az n szögpontú (egyszerű, irányítatlan) G gráf, aminek legfeljebb $5n$ éle van. Eldöntendő, hogy a G gráf $\overline{G}$ komplementere összefüggő-e. Oldjuk meg ezt a feladatot egy $O(n)$ uniform költségű algoritmus segítségével! ♢

45. Bizonyítsuk be, hogy a piros-kék algoritmus akkor is korrektül működik (minimális költségű feszítőfát eredményez), ha a feszítőfa költségét az élei súlyának maximumával definiáljuk! ♢

46. Éllistával adott egy összefüggő, egyszerű, irányítatlan G gráf csupa különböző élsúlyokkal. Jelöljük n-nel a csúcsok, e-vel pedig az élek számát. Mutassunk egy lineáris (azaz $O(e)$) uniform költségű algoritmust, ami a G gráf egy minimális feszítőfájának $\lceil 2/3n \rceil$ élét előállítja! (Egy olyan $\lceil 2/3n \rceil$ elemszámú élhalmazt keresünk, ami biztosan része egy minimális költségű feszítőfának.) ♢

47. Adott egy egyszerű irányítatlan $G = (V, E)$ gráf. Szeretnénk a csúcsokat két osztályba sorolni úgy, hogy a gráf éleinek legalább a fele különböző osztályú pontokat kössön össze. (Tehát olyan $V_1, V_2 \subseteq V$ csúcshalmazokat szeretnénk kijelölni, melyekre $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$, és $|\{(u,v) \in E | u \in V_1, v \in V_2\}| \geq |E|/2$.) Adjunk e feladat megoldására $O(|V|^2)$ futási idejű algoritmust! ♢

48. Adjunk példát olyan 5-pontú hálózatra, amelyre az Edmonds–Karp algoritmust lefuttatva az háromszori javítás után találja meg a maximális folyamot, továbbá a javító gráfban s és t távolsága minden lépésben változik. ♢

49. Adott egy $n \times n$-es A tömb, aminek elemei a $[0, 1)$ intervallumba eső racionális számok. Tudjuk, hogy A minden sorának és minden oszlopának az összege egész. Adjunk meg egy polinomidejű algoritmust egy olyan B, $n \times n$-es **0-1** tömb előállítására, aminek mind a sorösszegei, mind az oszlopösszegei azonosak az A tömbéivel, valamint $B[i,j] = 0$ mindazokon az i, j helyeken, ahol $A[i,j] = 0$. (Nem kötelező ajánlás: hálózati folyamok.) ♢

8. Turing gépek

1. Legyen M a következő Turing gép: $M = (Q, T, I, \ddot{u}, \delta, q_0, F)$, ahol
$k = 2$, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, \}$, $T = \{X, 0, 1, \ddot{u}\}$, $I = \{0, 1\}$, $F = \{q_5\}$,

δ :

áll	1.sz.	2.sz.	1.sz		2.sz		új áll
q_0	0	ü	0	H	X	J	q_1
	1	ü	1	H	X	J	q_1
	ü	ü	ü	H	ü	H	q_5
q_1	0	ü	0	J	0	J	q_1
	1	ü	1	J	1	J	q_1
	ü	ü	ü	H	ü	B	q_2
q_2	ü	0	ü	H	0	B	q_2
	ü	1	ü	H	1	B	q_2
	ü	X	ü	B	X	J	q_3
q_3	0	0	0	H	0	J	q_4
	1	1	1	H	1	J	q_4
q_4	0	0	0	B	0	H	q_3
	0	1	0	B	1	H	q_3
	1	0	1	B	0	H	q_3
	1	1	1	B	1	H	q_3
	0	ü	0	H	ü	H	q_5
	1	ü	1	H	ü	H	q_5

(a) Mi lesz a 2. szalagon a q_2 állapotban?
(b) Mi L_M, azaz milyen 0–1 -sorozatokat ismer fel M?
(c) Hogyan módosítható M, hogy a megfelelő 0–1 értékű függvényt számolja ki?

2. Legyen f és g a teljes I^*-on értelmezett függvények. T.f. f-et kiszámítja M, g-t pedig M'. Készítsünk egy olyan TG-t ezen M és M' felhasználásával, ami az $f \circ g$ összetett függvényt számítja ki! ♢

3. Készítsünk olyan (többszalagos) TG-t, ami az inputján megadott két binárisan ábrázolt számot $(x\#y)$ lineáris időben összeadja!

4. Hogyan modellezhető a címezhető memória egy TG-ben?

5. Rekurzív-e a prímszámokból (pl. bin. ábrázolás) álló nyelv? ♢

6. Bizonyítsuk be, hogy: L_1, L_2 rekurzív (rekurzíve felsorolható) $\Longrightarrow$ $L_1 \cup L_2$, $L_1 \cap L_2$ és az $L := \{xy \in I^* : x \in L_1,\ y \in L_2\}$ (egymás után írás) nyelv rekurzív (r.f.). ♢

7. Bizonyítsuk be, hogy az L_d diagonális nyelv komplementer nyelve rekurzíve felsorolható, azaz $I^* \setminus L_d \in R\mathcal{E}$!

8. Legyenek L_1 és L_2 rekurzíve felsorolható nyelvek. Igaz-e, hogy $L_1 \setminus L_2$ (vagyis azon szavak összessége, melyek L_1-ben vannak, de nincsenek L_2-ben) rekurzíve felsorolható? ♢

9. Ha A, B két halmaz, akkor a *szimmetrikus differenciájuk* alatt az $A \oplus B = (A \backslash B) \cup (B \backslash A)$ halmazt értjük.
a) Rekurzív-e $L_1 \oplus L_2$, ha L_1 és L_2 rekurzív?
b) Ha L_1 és L_2 két rekurzíve felsorolható nyelv, akkor igaz lesz-e ez $L_1 \oplus L_2$-re is? ♢

10. Álljon az L nyelv azon $x\#y$ párokból, ahol M_x (az x kódú TG) egszalagos, az y inputtal nem ír a szalagjára semmit, és ha $\ddot{u}$ jelet talál, azon nem lép túl. Rekurzív-e L? $\diamondsuit$

11. Álljon az L nyelv azon $x\#y$ párokból, ahol M_x (az x kódú TG) egszalagos, és az y inputtal nem ír a szalagjára semmit. Rekurzív-e L? $\diamondsuit$

12. Igazoljuk, hogy az L_h megállási nyelvnek van végtelen sok szót tartalmazó rekurzív részнyelve! $\diamondsuit$

13. Igazoljuk, hogy az alábbi L nyelv rekurzív (azaz a megállási probléma a kizárólag olvasó TG-k esetén eldönthető):
$L = \{x * y$; az M_x egyszalagos TG létezik és sohasem ír a szalagra; továbbá $y \in I^*$ és M_x megáll az y inputtal $\} \in R$!

14. Tekintsük a következő L nyelvet:
$L = \{x \in I^*$; az M_x egyszalagos TG létezik és semelyik $y \in I^*$ szóval mint inputtal elindítva sem mozdul el a fej a szalag első mezejéről $\}$.
Igazoljuk, hogy $L \in R$!

15. Rekurzív-e az a nyelv, ami azon $x\#y \in I^*$ szavakból áll, melyekre létezik M_x (az x szóval kódolt Turing gép), M_x egyszalagos, és a feje az y input esetén a szalagján csak "jobbra" lépéseket tesz (sohasem teszi meg a "helyben" illetve "balra" lépéseket)?

16. Rekurzív-e az azon $x \in I^*$ szavakból álló nyelv, melyekre az M_x Turing-gép létezik, és van olyan $y \in I^*$ szó, hogy M_x az y input esetén 5 lépésen belül megáll? $\diamondsuit$

17. Rekurzív-e az azon $x\#y$ párokból álló nyelv, ahol M_x az y input esetén legfeljebb egy szalagjára ír?

18. Álljon L szópárokból: $L \subseteq \{x\#y | x, y \in \{0,1\}^*\}$. Tegyük fel, hogy L, mint egy $\{0, 1, \#\}$ feletti nyelv rekurzíve felsorolható. Bizonyítsuk be, hogy az
$$L_1 = \{x \in \{0,1\}^* | \text{ létezik olyan } y \in \{0,1\}^*, \text{ hogy } x\#y \in L\}$$
nyelv is rekurzíve felsorolható!

19. Legyen $f : \{0,1\}^* \to \{0,1\}^*$ egy függvény, és tegyük fel, hogy a párokból álló $\{x\#f(x), \quad x \in \{0,1\}^*\}$ nyelv rekurzíve felsorolható. Igaz-e, hogy ekkor f egy rekurzív függvény? $\diamondsuit$

20. Legyen M egy olyan kétszalagos Turing gép, aminek egy kizárólag olvasható input szalagja, továbbá egy írható-olvasható munkaszalagja van, és M második feje mindig (bármely futásának bármely lépésében) a munkaszalag első 25 cellájának valamelyike fölött helyezkedik el. Adjunk meg egy M' egyszalagos, a szalagját csak olvasó Turing gépet, ami M-et szimulálja (ugyanazokon az input szavakon áll meg, és ugyanazt a nyelvet ismeri fel, mint M)!

21. Egy $L \subseteq I^*$ nyelv *páratlan nyelve*, L_{ptlan} az a nyelv, melynek szavai L szavainak páratlan sorszámú karaktereinek összeolvasásából kaphatók:
$L_{ptlan} = \{x_1 x_3 \dots x_{2n-1} \in I^* | \text{létezik } x_2, x_4, \dots, x_{2n} \in I, \text{hogy } x_1 x_2 \dots x_{2n-1} \in L \text{ vagy } x_1 x_2 \dots x_{2n-1} x_{2n} \in L\}$.
Bizonyítsuk be, hogy ha L rekuzíve felsorolható, akkor L_{ptlan} is rekurzíve felsorolható!

22. Álljon az L nyelv azon $x\#y \in I^*$ párokból, melyekre az M_x kétszalagos Turing gép létezik, és M_x az y input esetén nem változtatja meg az első szalagjának a tartalmát (azaz M_x futása során az első szalagján végig az $y\ddot{u}\ddot{u}\dots$ sorozat található)! Bizonyítsuk be, hogy az L nyelv nem rekurzív!

23. Mutassuk meg, hogy van olyan $f : I^* \to N$ függvény, ami nem rekurzív, de amire a következő $g : I^* \to \{0,1\}$ függvény rekurzív: $g(y) = 1$, ha $f(y)$ páratlan és $g(y) = 0$, ha $f(y)$ páros! $\diamondsuit$

24. Igazoljuk, hogy a következő $L \subseteq I^*$ nyelv rekurzíve felsorolható!
$L = \{x \in I^*, \ M_x$ létezik és $L(M_x)$ nem üres $\}$.

25. Jelölje $|x|$, ill. $C(x)$ az $x \in \{0,1\}^*$ szó hosszát, ill. Kolmogorov-bonyolultságát. Bizonyítsuk be, hogy:
(a) $x \in \{0,1\}^*$, $|x| = n$ és x-ben pontosan 20 db 1-es van $\Longrightarrow C(x) \leq c \log n$ alkalmas c konstansra.
(b) $\nexists$ TG, mely n inputtal olyan n hosszú $x \in \{0,1\}^*$ szót számol ki, melyre $C(x) > 2 \log n$.
(c) $\exists x \in \{0,1\}^*$, $|x| = n$, $C(x) \geq n - 1$, és x 0-val kezdődik.
(d) $\exists x, y \in \{0,1\}^*$, $C(x) > C(xy)$. "A Kolmogorov-bonyolultság nem monoton a prefixeken." $\diamondsuit$

26. Legyen $x \in \{0,1\}^*$ olyan n hosszúságú szó, melyben pontosan 2 nulla van. Igazoljuk, hogy $C(x) < n/10$, feltéve, hogy n elég nagy! $\diamondsuit$

27. Igazoljuk, hogy van olyan n hosszúságú $x \in \{0,1\}^*$ szó, ami 2 db 1-essel kezdődik és $C(x) \geq n - 2$. $\diamondsuit$

28. Az $x \in I^*$ szó x^f *fordítottja* az x betűinek fordított sorrendben való leírásával kapott szó. Például $11010^f = 01011$. Mutassuk meg, hogy van olyan c állandó, hogy $|C(x) - C(x^f)| < c$ teljesül minden $x \in I^*$ szóra!

29. Igazoljuk, hogy a Kolmogorov-bonyolultságot nem lehet Turing géppel 2-es szorzótényező erejéig közelítőleg sem kiszámítani! Más szóval, nem létezik olyan $f : \{0,1\}^* \to N$ rekurzív függvény, amire tetszőleges $x \in \{0,1\}^*$ esetén $\frac{1}{2} C(x) \leq f(x) \leq 2C(x)$ teljesül. $\diamondsuit$

30. Mutassuk meg, hogy nincs olyan M 2^n-időkorlátos NTG, amire $\forall x \in I^*$ esetén $C_M(x) \leq C(x) + 100$.

31. Igazoljuk, hogy
(a) tetszőleges n-re $C(0^n 1^n 0^n) \leq \log_2 n + c$, ahol c egy konstans;
(b) van olyan n, hogy $C(0^n 1^n 0^n) > 0.5 \log_2 n$.

32. Igazoljuk, hogy a következő $f : I^* \to \{0,1\}$ függvény nem rekurzív! $f(x) = 1$ ha az x szó összenyomhatatlan, és $f(x) = 0$ különben.

33. Legyen $I = \{0,1\}$. A $H : I^* \to Z^+$ függvény az $x \in I^*$ szóhoz azon $y \in I^*$ összenyomhatatlan szavak számát rendeli, amelyek a kanonikus sorrendben megelőzik x-et. Igaz-e, hogy H rekurzív függvény? $\diamondsuit$

34. Igazoljuk, hogy tetszőleges n természetes számhoz megadható olyan $x \in \{0,1\}^*$ szó, melyre teljesül, hogy $| x | > n$ és $C(x) < 0,5 \log_2 | x |$.

35. Kolmogorov-véletlen-e az az $\{x_i\}$ végtelen 0-1-sorozat, amelyben $x_i = 1$ pontosan akkor, ha i Fibonacci-szám? $\diamondsuit$

36. Legyen $\underline{y} = y_0 y_1 y_2 y_3 \dots$ egy olyan végtelen bitsorozat, amire tetszőleges k természetes szám esetén $y_{2k+1} = y_{2k}$. Lehet-e az $\underline{y}$ sorozat Kolmogorov-értelemben véletlen? $\diamondsuit$

37. Tegyük fel, hogy az $x_1 x_2 x_3 \dots$ végtelen 0-1 sorozat Kolmogorov értelemben véletlen. Igazoljuk, hogy ekkor az az $y_1 y_2 y_3 \dots$ sorozat is véletlen, amit úgy képzünk, hogy az $x_1 x_2 x_3 \dots$ sorozat 2-hatvány indexű bitjetit átírjuk 0-ra. ($y_i = x_i$, ha i nem 2-hatvány; $y_i = 0$, ha i 2-hatvány.) $\diamondsuit$

38. Egy M Turing gép és $x \in I^*$ szó esetén legyen
$$CT_M(x) := \min\{|y|; \quad M(y) = x, \text{ és } M \text{ az } y \text{ inputon legfeljebb } |x|^2 \text{ lépés után megáll}\},$$
és legyen $CT_M(x) = \infty$, ha nincs a fentieket kielégítő $y \in I^*$ szó. Mutassuk meg, hogy nem létezik olyan M Turing gép, melyre $CT_M(x) = C(x)$ teljesülne minden $x \in I^*$ input szó esetén. $\diamondsuit$

39. Legyenek $L_1, L_2 \subseteq N$ diofantikus halmazok. Igazoljuk (Matyijaszevics tételének felhasználása nélkül), hogy $L_1 \cup L_2$ és $L_1 \cap L_2$ is diofantikus halmazok. (A *diofantikus halmaz* definícióját ld. a jegyzetben.)

40. Igazoljuk (Matyijaszevics tételének felhasználása nélkül), hogy az összetett természetes számok halmaza diofantikus.
(Segítség: először mutassuk meg, hogy az 1-nél nagyobb egészek halmaza diofantikus).

41. Bizonyítsuk be, hogy Post megfeleltetési problémájának az egyelemű abc feletti változata (amikor a párok elemei az $\{a\}^*$-beli szavak) eldönthető! $\diamondsuit$

9. Bonyolultsági osztályok

1. Igazoljuk, hogy ha $L \subseteq I^*$ egy véges nyelv, akkor L felismerhető egy $O(n)$ időkorlátos TG-vel. $\diamondsuit$

2. Polinomidejű-e az az algoritmus, ami a bemenetként bináris ábrázolással adott n és m természetes számok szorzatát úgy számítja ki, hogy az n számhoz $(m-1)$-szer hozzáadja önmagát?
$s := n$;
`for` $i := 1$ `to` $m - 1$ `do`
$s := s + n$;
`return` s; $\diamondsuit$

3. Valaki egy n pontú $G = (V, E)$ gráfban $\lfloor log^2 n \rfloor$ elemű független csúcshalmazt szeretne találni. Azt a meglehetősen egyszerű módszert választja, hogy sorra veszi a V csúcshalmaz $\lfloor log^2 n \rfloor$ elemű S részhalmazait, és minden ilyen S-ről megvizsgálja, hogy független-e. Igaz-e, hogy a részletek megfelelő kidolgozásával ez a módszer polinom idejű algoritmust eredményez? $\diamondsuit$

4. Bizonyítsuk be, hogy az egészek között végzett négy alapművelet elvégezhető polinomidőben, míg a hatványozás nem. $\diamondsuit$

5. Mutassuk meg, hogy a modulo m tekintett hatványozás elvégezhető polinomidőben. $\diamondsuit$

6. Egy $L \subseteq I^*$ nyelv esetén az L^* nyelv definíciója a következő:
$L^* = \{x \in I^*$; és van olyan $k \geq 0$ egész és $x_1, \dots, x_k \in L$ hogy $x = x_1 x_2 \cdots x_k\}$.
Igazoljuk, hogy ha $L \in NP$, akkor $L^* \in NP$. $\diamondsuit$

7. (*) Egy $L \subseteq I^*$ nyelv esetén az L^* nyelv definíciója a következő:
$L^* = \{x \in I^*$; és van olyan $k \geq 0$ egész és $x_1, \ldots, x_k \in L$ hogy $x = x_1 x_2 \cdots x_k\}$.
Igazoljuk, hogy ha $L \in P$, akkor $L^* \in P$!

8. Javasoljunk egy determinisztikus polinom idejű módszert, mely egy adott $x \in I^*$ input szóhoz talál egy leghosszabb $y \neq x$ szót, mely legalább kétszer fordul elő x-ben résszóként!

9. Az L nyelv felismerhető egy $3\log n$-tárkorlátos TG-vel. Igazoljuk, hogy ekkor $L \in P$. ♢

10. Legyen $T : N^+ \to N^+$ egy rekurzív függvény és
$L = \{x \in I^*$, M_x létezik és M_x nem fogadja el x-et legfeljebb $T(\mid x \mid)$ lépés megtétele után $\}$.
Mutassuk meg, hogy az L nyelv rekurzív, de nincs olyan TG, mely $T(n)$ időben felismeri. ♢

11. Tegyük fel, hogy van egy polinom idejű M eljárásunk, mely a (binárisan ábrázolt) m, n pozitív egészekből álló bementre megadja $m!(mod\ n)$ értékét. Javasoljunk egy polinom idejű M-et használó módszert az n prímtényezőinek kiszámítására. ♢

12. Tegyük fel, hogy van egy olyan K nevű eljárás, mely (egyetlen lépésben) megmondja, hogy az inputjaként magadott irányítatlan gráf színezhető-e három színnel. Csináljunk egy a K eljárást használó algoritmust, mely polinomidőben megadja az input G gráf egy 3-színezését, ha $G \in 3 - SZÍN$.

13. Tegyük fel, hogy van egy olyan X eljárásunk, ami az $X3C$ (hármasokkal való pontos lefedhetőség) problémát 1 költséggel eldönti. Adjunk meg egy, a fenti X eljárást használó polinomidejű algoritmust arra a problémára, melynek bemenete ugyanaz, mint az $X3C$ problémáé, de a feladat az, hogy *találjunk* is egy pontos lefedést az adott hármasokból, ha létezik!

14. Tegyük fel, hogy van egy P eljárásunk, mely egy input gráfra 1 költséggel megmondja a gráfban levő maximális klikk(ek) méretét. Tervezzünk egy olyan, a P eljárást használó algoritmust, mely polinom időben talál egy maximális klikket az input gráfban! ♢

15. Tegyük fel, hogy van egy eljárásunk, mely egy tetszőleges G gráfra megmondja, hogy G-ben hány Hamilton kör van. Az eljárás egy hívásának költsége $\mid V(G) \mid$. Adjunk egy az előbbi eljárást használó polinom idejű módszert, mely tetszőleges gráfban talál egy Hamilton kört (ha létezik egyáltalán).

16. Tegyük fel, hogy a 3SAT probléma eldöntésére (a 3SAT nyelvbe tartozás eldöntésére) van egy polinomidejű algoritmusunk. Igazoljuk, hogy ekkor a következő keresési feladat is megoldható polinomidőben:
BEMENET: Egy ϕ 3-CNF.
KERESENDŐ: A változók egy olyan kiértékelése, amire a ϕ formula igaz lesz, ha egyáltalán van ilyen.

17. Igazoljuk, hogy az alábbi L nyelv NP-ben van:
$L = \{f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$; $a_i \in Z$ és az $f(x) = 0$ egyenletnek van egész megoldása $\}$.
Megjegyzés: az a_i számokat az inputban binárisan ábrázoljuk. ♢

18. Igazoljuk, hogy az alábbi eldöntési probléma (vagyis a megfelelő nyelv) a PSPACE osztályban van:
Input: $s_1, \ldots, s_m \in N^+$ súlyok valamint további két természetes szám k és e.
Kérdés: Van-e k darab olyan I részhalmaza az $\{1, \ldots, m\}$ halmaznak, amelyekre $\sum_{i \in I} s_i \leq e$? ♢

19. Tekinstük a következő L nyelvet:
$L = \{(G, k)$; G egy páros (kétrészes gráf), k egy pozitív egész és G-ben pontosan k teljes párosítás van $\}$.
Igazoljuk, hogy $L \in$ PSPACE.

20. Igazoljuk, hogy az alábbi nyelv a PSPACE osztályban van:
$L = \{(G, k)$; G egy irányítatlan gráf, $k \in N^+$ és van G-ben k darab Hamilton kör $\}$.

21. A valós egyenes korlátos zárt intervallumainak egy V véges halmaza egy intervallum gráfot határoz meg, melynek csúcsai a V elemei, továbbá $I, J \in V$ esetén (I, J) él pontosan akkor ha $I \cap J \neq \emptyset$.
Igazoljuk, hogy a MAXKLIKK probléma polinom időben megoldható intervallum gráfokra (az intervallumok végpontjai egészek és az intervallumok a végpontok bináris ábrázolásával szerepelnek az inputban).

22. Mutassuk meg, hogy az alábbi nyelv $NP-$teljes:
$L = \{G$; G irányítatlan gráf és van G-ben $\mid V(G) \mid -2$ élszámú egyszerű út $\}$.

23. Egy G irányítatlan gráfot **2-lebonthatónak** nevezünk, ha csúcshalmaza az üres halmazzá redukálható az alábbi lépés ismételt végrehajtásával:
G-ből egy tetszőleges legfeljebb másodfokú pont a csatlakozó élekkel együtt törölhető.
Mutassuk meg, hogy a 2-lebontható gráfok polinom időben felismerhetők. Javasoljunk ezen felül hatékony algoritmust az ilyen gráfok 3 színnel való színezésére.

24. Álljon az L nyelv azon $(G; U, L)$ páros gáfokból, melyeknek pontosan egy teljes párosítása van. Mutassuk meg, hogy $L \in \mathbf{P}$.

25. Igazoljuk, hogy a 2 színnel színezhető gráfok nyelve P-beli. ♢

26. Álljon az L nyelv azon $G = G(V, E)$ irányítatlan gráfokból, melyeknek a V csúcshalmaza felosztható két olyan nemüres V_1 ill. V_2 részre ($V = V_1 \cup V_2$ és $V_1 \cap V_2 = \emptyset$), hogy G-nek legfeljebb két V_1 és V_2 között menő éle (olyan, aminek az egyik végpontja V_1-be, a másik V_2-be esik) van! Bizonyítsuk be, hogy $L \in$ P! ♢

27. Mutassuk meg, hogy a $MAXKLIKK$ probléma polinom időben megoldható az olyan G irányítatlan gráfok esetén, melyekben bármely két különböző pontnak legfeljebb $2\log_2 \mid V(G) \mid$ közös szomszédja van. ♢

28. A G gráfban minden csúcs foka legfeljebb 3. Jelölje k a G-beli maximális független csúcshalmazok elemszámát. Adjunk polinom idejű módszert $\lfloor k/4 \rfloor$ elemű G-beli független ponthalmaz keresésére. ♢

29. Adott egy irányítatlan, egyszerű G gráf. Szeretnénk G-ben egy olyan éllefogó csúcshalmazt keresni, aminek a mérete legfeljebb kétszerese a lehető legkisebb éllefogó ponthalmazénak. Ismertessünk egy módszert, ami polinom időben megoldja ezt a feladatot! ♢

30. a) Mutassuk meg, hogy a Hamilton-kör keresésének feladata poliom időben megoldható azon irányítatlan gráfok esetén, amelyeknek legfeljebb $n + 10$ élük van (n a csúcsok száma).
b) Adjunk $O(n)$ lépésszámú algoritmust (éllistával adott bemeneten). ♢

31. Legyen G egy n-pontú összefüggő irányítatlan gráf, melynek $n + 10$ éle van. Mutassuk meg, hogy a G 3 színnel való színezhetősége eldönthető polinom időben (a 3-SZÍN feladatnak ez a megszorítása **P**-ben van)!

32. Adjunk polinomidejű algoritmust a következő feladatra: Adott n darab pozitív egész súly: $s_1, \ldots, s_n$. Keresendő az adott súlyokból az n legkisebb részhalmazösszeg. Más szóval, n darab olyan $I \subseteq \{1, \ldots, n\}$ halmazt szeretnénk kiválasztani, amelyek mindegyikére igaz az, hogy legfeljebb $n - 1$ olyan $J \subseteq \{1, \ldots, n\}$ részhalmaz van, amire

$$\sum_{j \in J} s_j < \sum_{i \in I} s_i.$$

33. Az alábbi $f : N \to N$ függvények közül melyek tartoznak FP-be és melyek nem?
(a) $f(n) = n!$
(b) $f(n) =$ az n legkisebb prímosztója
(c) $f(n) = \lfloor \log_2 n \rfloor$. ♢

34. Az alábbi, gráfokon értelmezett függvények közül melyek tartoznak az FP osztályba ?
(a) $f(G) =$ a G 3-színezéseinek száma;
(b) $f(G) =$ a G-beli egyszerű utak listája;
(c) $f(G) =$ a G-beli háromszögek (3 elemű klikkek) listája.

35. Legyen $f : \{0,1\}^* \to \{0,1\}^*$ egy függvény, és tegyük fel, hogy a párokból álló $\{x \# f(x),\ x \in \{0,1\}^*\}$ nyelv polinom időben felismerhető. Igaz-e, hogy az f függvény polinom időben kiszámítható? ♢

36. Igazoljuk, hogy a 4 színnel színezhető irányítatlan gráfok nyelve NP-teljes!

37. Álljon az L nyelv azon irányítatlan gráfokból, melyekben van $\lfloor \sqrt{n} \rfloor$ méretű kör, ahol n a gráf csúcsainak a száma! Bizonyítsuk be, hogy L NP-teljes!

38. Legyen SAT_5 azon kielégíthető Boole formulák nyelve, melyekben a változóknak legfeljebb 5 előfordulása van (egy változó, illetve a negáltja együttesen legfeljebb ötször szerepel a formulában). Mutassuk meg, hogy a SAT_5 nyelv **NP**-teljes! (Ajánlás: adjunk meg egy $SAT \prec SAT_5$ Karp-redukciót.)

39. Jelölje RH_1 a részhalmaz-összeg probléma azon speciális esetét, melyben az első súly 1 (a tanult jelöléssel: $s_1 = 1$). Igazoljuk, hogy az RH_1 nyelv **NP**-teljes. (Ajánlás: adjunk meg egy $RH \prec RH_1$ Karp-redukciót.)

40. Jelölje RH' a Részhalmazösszeg feladatnak azt a speciális esetét, amelyben az s_i súlyok mind páros számok. Karp-redukció megadásával igazoljuk, hogy az RH' feladat NP-teljes. ♢

41. Alkalmas Karp-redukció megadásával igazoljuk, hogy a *Ládapakolás* feladatnak az a speciális esete is **NP**-teljes, ahol az első súly értéke 1/2!

42. Igazoljuk, hogy polinom időben megoldható a *Ládapakolás* feladatnak az a speciális esete, amelyben minden súly nagyobb, mint $\frac{1}{3}$. ♢

43. Igazoljuk, hogy az alábbi L nyelv P-ben van:
$L = \{(v_1, v_2, \ldots, v_m, b)$; $v_i, b \in Z^+$, $v_i \leq m$ és van olyan $I \subseteq \{1, 2, \ldots, m\}$, hogy $\sum_{i \in I} v_i = b\}$. ♢

44. Igazoljuk, hogy a **Hátizsák** problémának az a speciális esete, melyben minden súly ugyanolyan értékű ($s_1 = s_2 = \cdots = s_m$), megoldható polinom időben.

45. Jelölje R a részhalmazösszeg problémának azt a speciális esetét, amikor a súlyok 2^m és $2^m + m^2$ közé eső egészek, ahol m a súlyok száma:
$R = \{(s_1, s_2, \ldots, s_m; b) | s_i, b$ egészek, $2^m \leq s_i \leq 2^m + m^2$, és van olyan $I \subseteq \{1, 2, \ldots, m\}$, hogy $\sum_{i \in I} s_i = b\}$.
Igazoljuk, hogy $R \in \mathbf{P}$!

46. Igazoljuk, hogy polinom időben megoldható a *Ládapakolás* feladatnak az a speciális esete, amelyben minden súly $\frac{1}{2^j}$ alakú. ♢

47. Jelölje H a következő hipotézist: Az NP-teljes feladatok nem oldhatók meg $O(1, 2^n)$-nél kisebb lépésszámú algoritmussal, ahol n az input mérete. Mutassuk meg, hogy ha H igaz, akkor a következő feladat nem lehet NP-teljes:
Input: Egy n csúcsú irányítatlan gráf G.
Kérdés: Van-e G -ben legalább $\log n$ független pont? ♢

48. Lehet-e NP-teljes az alábbi L nyelv?
$L = \{(G; k)$; G irányítatlan gráf melyben minden pont foka legfeljebb $\log_2 | V(G) |$, k pozitív egész és G-ben van k pontú teljes részgráf $\}$. ♢

49. Igazoljuk, hogy $2 - SAT \in P$! ♢

50. Tegyük fel, hogy adott egy G gráf, és minden $v \in V(G)$ csúcsához egy $tilos(v) \in \{piros, fehér, zöld\}$ szín. Szeretnénk eldönteni, hogy G csúcsai kiszínezhetők-e a *piros, fehér, zöld* színekkel úgy, hogy a szomszédos (éllel összekötött) csúcsok színei különbözők, és minden v csúcs színe különbözik $tilos(v)$-től.
Mutassuk meg, hogy ez a feladat megoldható ($|V(G)|$-ben) polinom időben! (Segítség: mutassuk meg, hogy a követelmények leírhatók egy $2-$CNF formulával!)

51. Mutassuk meg, hogy az $X3C$-nek az a speciális esete, melyben az alaphalmaz $3n$ elemű és legfeljebb $n + \log_2 n$ db hármasunk van, megoldható polinom időben. ♢

52. Mutassuk meg, hogy polinom időben megoldható az $X3C$ probléma azon speciális esete, melyben az U alaphalmaz minden elemét pontosan 2 hármas tartalmazza.

53. Adjunk közvetlen bizonyítást arra, hogy X3C$\prec$MAXFÜGGETLEN, azaz adjunk meg egy Karp redukciót, ami a hármasokkal való pontos lefedhetőség problémáját vezeti vissza arra a problémára, hogy létezik-e egy gráfban adott méretű független pontrendszer!

54. Bizonyítsuk be, hogy a következő probléma NP-teljes:
Input: G irányítatlan gráf, $k \in \mathbf{N}$,
Kérdés: van-e G-nek legalább k-pontú 2 színnel színezhető feszített részgráfja?
(Segítség: Karp-redukáljuk rá a $MAXFTLEN$ problémát.) ♢

55. Igazoljuk, hogy az alábbi (*Csúcsfedés* nevű) probléma NP-teljes:

Input: Egy irányítatlan gráf G és egy pozitív egész k.

Kérdés: Van-e olyan $W \subseteq V(G), | W | \leq k$, hogy minden $e \in E(G)$ él illeszkedik legalább egy W-beli csúcshoz?

56. Igazoljuk, hogy az alábbi L nyelv NP-teljes:
$L = \{(G_1; G_2)$; G_1, G_2 irányítatlan gráfok, G_1 fa és van olyan $f : V(G_1) \to V(G_2)$ leképezés, melyre $x \neq y$ esetén $f(x) \neq f(y)$; továbbá $(x, y) \in E(G_1) \Leftrightarrow (f(x), f(y)) \in E(G_2)\}$.

57. Adott az $X3C$ probléma egy példánya. Az alaphalmaz mérete $3n$, továbbá m db hármasunk van. Javasoljunk egy $O(m^2 n)$ méretű (azaz összesen $O(m^2 n)$ változóból és műveleti jelből álló) konjunktív normálformát, mely akkor és csak akkor kielégíthető, ha a pontos fedés megvalósítható. ♢

58. Igazoljuk, hogy a Hamilton-út probléma NP-teljes! (Van-e G-ben $| V(G) | - 1$ élszámú egyszerű út?) ♢

59. Jelentse L azon irányítatlan G gráfok nyelvét, melyek egy alkalmas él hozzáadása után tartalmaznak Hamilton kört. Igazoljuk, hogy L NP-teljes.

60. Egy $G = (V, E)$ egyszerű irányítatlan gráfnak egy $G' = (V, E')$ részgráfját ($E' \subseteq E$) *izmosnak* nevezzük, ha G'-nek egy tetszőleges élét elhagyva is összefüggő gráfot kapunk. (Szokásos elnevezés: G' kétszeresen élösszefüggő.)
Tekintsük a következő, *MINIZMOS* nevű feladatot:
BEMENET: A G gráf pozitív élsúlyokkal. Az e él súlyát jelöljük $c(e)$-vel!
KIMENET: $\min\{\sum_{e \in E'} c(e) \mid G' = (V, E')$ izmos részgráf$\}$, ha létezik G-ben izmos részgráf; ∞ egyébként.
(Más szóval, kiszámítandó G-ben a minimális élsúlyösszegű izmos részgráf élsúlyösszege — feltéve, hogy van ilyen.) Igazoljuk, hogy a *MINIZMOS* feladat **NP**-nehéz!

61. Igazoljuk, hogy a következő feladat P-beli:
Adott egy G gráf és egy $S \subseteq V(G)$ halmaz. Van-e G-nek olyan feszítőfája, melynek végpontjai (elsőfokú pontjai) között S pontjai mind szerepelnek?

62. Igazoljuk, hogy a következő feladat NP-nehéz:
Adott egy G gráf és egy $S \subseteq V(G)$ ponthalmaz. Van-e G-nek olyan feszítőfája, melynek végpontjai pontosan az S pontjai?

63. Tekintsük a következő algoritmikus problémát: MAXKÖR
Input: Egy G irányítatlan gráf, éllistával.
Feladat: Maximális hosszúságú (élszámú) kör keresése G-ben.
Igazoljuk, hogy ha MAXKÖR polinom időben megoldható, akkor $P = NP$!

64. Igazoljuk, hogy az *Egész Értékű Lineáris Programozás* (EP) feladatának az a változata is NP-teljes, amelyben a szokásos feltételeken túl azt is megköveteljük, hogy a megoldásvektor komponensei a $\{0, 1\}$ halmazba essenek. ♢

65. Mutassuk meg, hogy az alábbi probléma NP-teljes: EGYENLETEK
Input: $f_1, \ldots, f_k \in Z[x_1, \ldots, x_n]$ polinomok.
Kérdés: Van-e az $f_1(\mathbf{x}) = \ldots = f_k(\mathbf{x}) = 0$ egyenletrendszernek olyan $\mathbf{x}_0$ megoldása, melynek minden komponense 0 vagy 1 ?
(Nem kötelező ajánlat: mutassuk meg, hogy $X3C \prec$EGYENLETEK!)

66. Az $X4C$ feladat bemenete egy U véges halmaz és az U-nak bizonyos $X_1, X_2, \ldots, X_k$ négyelemű részhalmazai. Eldöntendő, hogy az X_i-k közül kiválaszthatók-e páronként diszjunkt halmazok, amelyek lefedik U-t. Igazoljuk, hogy $X4C$ egy NP-teljes feladat. ♢

67. Álljon a $PONTY$ nyelv azokból a konjunktív normálformájú ϕ Boole-formulákból, amelyekhez van a változóknak olyan kiértékelése, ami szerint a ϕ formula minden tagjában pontosan egy literál lesz igaz! Bizonyítsuk be, hogy a $PONTY$ nyelv NP-teljes! ♢

10. Aritmetika

1. Álljon az L nyelv azon $a_0, a_1, \ldots, a_{n-1}, \beta_1, \ldots, \beta_n$ $2n$ egész számból álló sorozatokból, melyekre az $f(x) = x^n + a_{n-1} x^{n-1} + \ldots + a_0$ egyváltozós polinomnak a gyökei éppen a csupa különböző $\beta_1, \ldots, \beta_n$ számok. Adjunk egy $O(n)$ aritmetikai műveletet használó randomizált algoritmust az L nyelv felismerésére! (Olyan, $O(n)$ aritmetikai műveletet használó randomizált algoritmus szükséges, ami mindig elfogad, ha a polinom gyökei éppen az adott számok, és legalább 50%-os valószínűséggel elutasít, ha nem ez a helyzet. Hogy elejét vegyük egy csábító, de rossz irányba menő elindulásnak: az $(x - \beta_1)(x - \beta_2) \cdots (x - \beta_n)$ polinom összes együtthatóját *bizonyítottan* nem lehet kiszámolni $O(n)$ aritmetikai művelettel.)

2. Legyen $n = pq$, ahol p és q prímszámok.
(a) Mutassuk meg, hogy ha m egy egész és $lnko(m, n) = 1$, akkor az $x^2 \equiv m (mod\, n)$ kongruenciának 0 vagy 4 egész megoldása van. (Segítség: Kínai Maradéktétel)
(b) Igazoljuk, hogy ha van olyan polinom idejű algoritmus, mely a fenti alakú n számokra tetszőleges m input esetén megadja az $x^2 \equiv m (mod\, n)$ egy egész megoldását (feltéve természetesen, hogy van megoldás), akkor p és q megkapható egy randomizált polinom idejű módszerrel.

3. A feladat egy tesztelési problémával kapcsolatos. Adottak az A, B, C n-szer n-es egészelemű mátrixok. Szeretnénk ellenőrizni, hogy teljesül-e az $AB = C$ egyenlőség.
Valaki a következő módszert javasolja: válasszunk egy $\mathbf{x}$ véletlen 0-1 vektort, majd számítsuk ki az $X\mathbf{x}$ vektort, ahol $X = AB - C$. Ha az eredmény nem $\mathbf{0}$, akkor az egyenlőség nem áll fenn, különben *valószínűleg* igen.
Igazoljuk, hogy
(a) az $X\mathbf{x}$ vektor $O(n^2)$ szorzással illetve összeadással kiszámítható;
(b) ha $X \neq 0$, akkor $P(X\mathbf{x} \neq 0) \geq 1/2$ (vagyis a hibás konklúzió valószínűsége legfeljebb $1/2$).

4. Az előadáson megismert prekondícionáló algoritmust alkalmazva fejezzük ki az
(a) $f(x) = x^7 + x^6 + 2x^5 + 2x^3 + 5x - 1$,
(b) $f(x) = x^7 + 2x^4 + 3x^3 + 4x + 5$,
(c) $f(x) = x^7 + x^5 - x^2 + 3x + 9$
polinomokat $x^{2^j} + a$ alakú polinomok segítségével. Hány szorzást igényel az így prekondícionált $f(x)$ egy kiértékelése?

II. Megoldások

1. Vegyes

8. Azt vegyük észre, hogy ha menet közben létrejött két vagy több olyan doboz, amelyek ugyannyi golyót tartalmaznak, akkor a továbbiakban ezek a dobozok együtt kezelhetők. Az első lépésben mindazon dobozokból, amelyeknek a mérete legalább $\lceil n/2 \rceil$, vegyünk ki $\lceil n/2 \rceil$ golyót. Az eztután maradó nemüres dobozméretek $1, 2, \ldots, \lfloor n/2 \rfloor$, azaz egy $\lfloor n/2 \rfloor$-es feladatra vezettük vissza az eredetit. Így folytatva a lépésszám $\lfloor \log_2(n+1) \rfloor$. Állítjuk, hogy ez az optimális. Tekintsünk ugyanis egy tetszőleges módszert. Tegyük fel, hogy visszafele a k-adik lépésben a különböző nemüres dobozméretek száma l_k-ról l_{k-1}-re változik. Tegyük továbbá fel, hogy azon dobozoknak, amelyekből kiveszünk golyókat, s féle különböző mérete van. Ezek közül legalább $s-1$ nem lesz üres, és a méretük mind különböző lesz. A változatlanul hagyott dobozok pedig legalább $l_k - s$ különböző méretet adnak. Tehát az $l_{k-1} \geq s-1$ és az $l_{k-1} \geq l_k - s$ egyentlőtlenségek egyaránt teljesülnek. Ezeket összeadva és átrendezve nyerjük az $l_k \leq 2l_{k-1}+1$ egyenlőtlenséget. Világos, hogy $l_1 \leq 1$. A fentebb bizonyított egyenlőtlenség kínálta rekurziót megoldva azt kapjuk, hogy $l_k \leq 2^k - 1$, tehát k lépésben legfeljebb $2^k - 1$ különböző dobozméretre tudjuk a feladatot megoldani. Tehát ha k az optimális lépésszám, akkor $n \leq 2^k - 1$, azaz $k \geq \log_2(n+1)$.

2. Technika

3. Rendezés

4. $n-10$ összehasonlítás elegendő: kidobunk 9 elemet és a maradékban megkeressük a legkisebbet (például a buborékrendezés külső ciklusát egyszer végrehajtjuk). Ennyi szükséges is (ha $n > 11$), mert $n-10$-nél kevesebb összehasonlítás esetén az összehasonlítottsági gráfnak – két elemet akkor kötünk össze, ha az algoritmus során összehasonlítottuk őket – több mint 10 komponense van. (Ez abból az ismert tényből következik, hogy egy n pontú k összefüggő komponensből álló gráfnak legalább $n-k$ éle van.) Ebben az esetben egy tetszőlegesen választott elemhez a rendelkezésre álló információ nem zárja ki, hogy az illető elem nagyobb legyen, mint az őt nem tartalmazó komponensekbe eső elemek bármelyike. Ezekből pedig legalább 10 van.

6. (a) Az előadáson tanult összefésülési módszer legfeljebb $n + (n+1) - 1 = 2n$ összehasonlítást igényel. Tegyük fel, hogy van egy módszer, ami $2n$-nél kevesebb összehasonlítást használ. Az ellenség-stratégiát használjuk ezzel a módszerrel szemben: Egy $A[i]?B[j]$ összehasonlításra legyen a válasz

$$\begin{array}{ll} A[i] < B[j], & \text{ha } i < j; \\ A[i] > B[j], & \text{ha } i \geq j. \end{array}$$

Mivel az összehasonlítások száma kevesebb, mint $2n$, létezik olyan i index, hogy vagy az $A[i]?B[i]$ vagy az $A[i]?B[i+1]$ összehasonlítás nem történt meg. Tegyük fel, hogy nem volt $A[i]?B[i]$ összehasonlítás. Ekkor az elvégzett összehasonlítások eredményei sem a $B[1] < A[1] < B[2] < A[2] < \ldots < A[i-1] < B[i] < A[i] < B[i+1] < A[i+1] \ldots$ sorrendet, sem a $B[1] < A[1] < B[2] < A[2] < \ldots < A[i-1] < A[i] < B[i] < B[i+1] < A[i+1] \ldots$ sorrendet nem zárják ki. Hasonlóan, az $A[i]?B[i+1]$ összehasonlítás hiányában sem lehet egyértelmű sorrendet megállapítani.
(b) Nem, mert a k elem beillesztető bináris kereséssel összesen $O(k \log n)$ összehasonlítással, és ha $k = o(n/\log n)$ akkor $k \log n = o(n+k)$, azaz $k \log n/(n+k) \to 0$. (Pl. $k = \log n$, $k = \sqrt{n}$, vagy $k = n/\log^2 n$, stb.)

10. Vegyünk fel két segédtömböt, R-et és N-et. Az R-be rendezetten gyűjtünk minél több elemet, míg N a maradék tárolására szolgál. Az A tömb $A[i]$ elemeire, az elejétől kezdve csináljuk a következőt: Ha az R tömb üres vagy $A[i]$ nem kisebb, mint az R tömbbe utoljára betett elem, akkor rakjuk $A[i]$-t az R tömb végére. Egyébként mind az R tömb utolsó elemét, mind $A[i]$ tegyük át az N tömbbe. Ez nyilván lineáris időben lefut. Világos továbbá, hogy az R tömb rendezett lesz. N-be csak akkor teszünk elemeket, ha $A[i]$ nagyobb, mint az R tömb tetején levő $A[j]$ ($j < i$) elem. Ez csak úgy lehet, hogy vagy $A[i]$-t vagy $A[j]$-t megváltoztatták, ezért N-be legfőljebb $2k$ elem kerül. Így az N tömb rendezése $O(k \log k)$ lépésben megvalósítható. Ezek után az A tömb rendezése $O(n)$ időben elvégezhető az R és N tömbök összefésülésével.

11. A segítségül felajánlott állítás azért igaz, mert $A[j]$, $A[k]$ és $A[l]$ közül legalább kettő azonos X_i részsorozatba esik és így a nagyság szerenti sorrendjük nem lehet fordított. Az algoritmus érdemi része az A tömb két monoton sorozatra osztása lineáris időben. Ezután ugyanis a rendezés összefésüléssel lineáris időben befejezhető. Vegyünk föl két segédtömböt: B_1-et és B_2-t. Az A tömb $A[l]$ elemeit (l = 1-től n-ig sorban) a B_1 és B_2 tömbök közül a kisebbik indexű végére illesztjük úgy, hogy ezen tömbök rendezettsége megmaradjon. Állítjuk, hogy ez a feltétel miatt megtehető. Ugyanis ha nem, legyen l az az első index, melyre $A[l]$ nem illeszthető egyik tömb végére sem. Ekkor a B_2 tömb végén egy olyan A_k elem ($k < l$) helyezkedik el, melyre $A[k] > A[l]$. Tekintsük most azt a pillanatot, amikor az $A[k]$ elemet a B_2 tömb végére tettük. Ekkor a B_1 tömb végén egy olyan $A[j]$ elem ($j < k$) kell hogy elhelyezkedjen, melye $A[k] < A[j]$, hiszen különben $A[k]$-t a B_1 tömbbe raktuk volna. Ilyen $A[j]$, $A[k]$, $A[l]$ hármas pedig nem létezhet a feltétel miatt. Világos, hogy a módszer költsége $O(n)$. Megjegyezzük, hogy a vázolt algoritmus egyben bizonyítja azt is, hogy a mondott következmény valójában elégséges is ahhoz, hogy az A tömb felbontható legyen két monoton nem csökkenő részsorozatra.

12. Felveszünk három segédtömböt, legyenek ezek B_1, B_2, B_3. Vegyük sorra az A tömb elemeit. Ha az $A[i]$ elem kerül sorra, keressük meg azt a legkisebb indexű B_j tömböt, mely vagy üres, vagy a végén levő elem kisebb $A[i]$-nél, és fűzzük $A[i]$-t B_j végére. Állítjuk, hogy mindig van ilyen B_j. Tegyük fel indirekte, hogy nincs. Legyen i_4 a legelső index, amelyre $A[i_4]$ nem fér be egyik B_j tömbbe sem. Ekkor a B_3 tömb végén már van egy elem: $A[i_3] > A[i_4]$, ahol $i_3 < i_4$, különben $A[i_4]$-et betehettük volna B_1, B_2, B_3 valamelyikébe. Tekintsük most azt a pillanatot, amikor $A[i_3]$ a helyére került. Ekkor a B_2 tömb végén egy olyan $A[i_2]$ elem volt ($i_2 < i_3$), ami nagyobb $A[i_3]$-nál, különben az $A[i_3]$ elemet a B_1 vagy B_2 tömb végére raktuk volna. Hasonlóan, $A[i_2]$ bekerülésekor pedig B_1 végén egy olyan $A[i_1]$ elem volt ($i_1 < i_2$), ami nagyobb $A[i_2]$-nél. A feltevésnek ellentmondó $A[i_1] > A[i_2] > A[i_3] > A[i_4]$ részsorozatot kaptunk. Tehát a módszer működik. Világos, hogy az időigény lineáris és a B_i tömbök rendezettek lesznek. Az algoritmus tehát a B_i tömbök összefésülésével fejezhető be, ami szintén megoldható lineáris időben.

13. Vegyünk fel egy 6 hosszú segédtömböt, töltsük fel az A tömb első hat elemével, majd a tömb további elemeire sorban csináljuk a következőt: írjuk ki a segédtömbben levő legkisebb elemet, majd (ha van) vegyük be a tömb soron következő elemét. Mivel 6 elem közül a legkisebb kiválasztása konstans időt igényel, az összköltség nyilván $O(n)$. Indukcióval igazoljuk, hogy az első k lépésben a tömb k legkisebb eleme íródik ki növekvő sorrenben. A kezdőlépés: a legkisebb elem a feltétel miatt az első hat hely valamelyikén helyezkedik el, tehát az első lépésben tényleg a legkisebb elem íródik ki. Az indukciós lépés: tegyük fel, hogy k lépés után a k legkisebb elem íródott ki (sorrendben). Azt kell csak igazolni, hogy a $k+1$-edik lépésben a $k+1$-edik legkisebb elemet írjuk ki. A $k+1$-edik lépésig látott elemek a segédtömbben jelenlevőkkel együtt a tömb első $\min(n, k+6)$ helyén levő elem. A feltétel miatt ezek között van a nagyság szerinti $k+1$-edik. Az indukciós feltevés szerint még nem írtuk ki, tehát mindenképpen a segédtömbben van, szükségszerűen annak a legkisebb eleme. Tehát tényleg őt írjuk ki $k+1$-edikként.

15. $\lceil \log_2 4! \rceil = 5$ összehasonlítás szükséges. Összefésüléses módszerrel ennyi elegendő is.

16. $\lceil \log_2 5! \rceil = 7$ összehasonlítás szükséges. Ennyi elegendő is: Legyen az öt elem a, b, c, d, e. Először rendezzünk "kieséses versenyt" az a, b, c, d elemek között. Ez három összehasonlítást igényel. Az általánosság megszorítása nélkül feltehető, hogy az eredmény $a \leq b \leq c \geq d$. Az e elemet az $a \leq b \leq c$ sorozatba két lépésben beszúrhatjuk. Ezzel elérjük, hogy az $\{a, b, c, e\}$ halmaz rendezett. Mivel már tudjuk, hogy $d \leq c$, a d elemet már csak az $\{a, b\}$ halmazba (ha $e \geq c$) vagy az $\{a, b, e\}$ halmazba (ha $e < c$) kell beszúrni, amihez mindkét esetben elegendő két összehasonlítás.

17. a) Nem. Az első három lépés: $6\cdots$, $46\cdots$, $346\cdots$. Ezek után a 3 sose kerülhet a 4 mögé, tehát a kérdéses részeredmény nem jöhet létre.
b) Igen, az első három lépés után: $6 \leftrightarrow 4$, $6 \not\leftrightarrow 8$, $8 \leftrightarrow 3$.
c) Igen, a tömb első felében levő két negyed rendezése után, ezek összefésülése előtt.
d) Nem. Ha az első partícionáló elem nem 1, akkor az első mozgatás elviszi az 1-et a sor végéről, és soha nem kerülhet vissza oda. Ez tehát nem lehet. Ha pedig az első partícionáló elem 1, akkor az első partícionálásnál végéig nem történik mozgatás: az alsó rész üres, a felső pedig a teljes. Az előírás szerint viszont ezután az 1-et a tömb elejére kell vinni. Onnan pedig már nem vihető el. Tehát ez sem lehetséges.

18. a) Csak $n \leq 3$-ra, mert a buborékrendezés összehasonlításainak a száma minden bemenetre $n(n-1)/2$. Még arra a kicsit takarékosabb buborékrendezés-változatra sem igaz, amikor abbahagyjuk a buborékoltatást egy cserét nem végrehajtó menet (külső ciklus) után: ha n elég nagy és megcseréljük az $\lfloor n/3 \rfloor$-adik és a $\lceil 2n/3 \rceil$-adik elemet, kb. $2n/3$ inverzió jön létre, ugyanakkor a buborékrendezésnek kb. $n/3$ menete szükséges, ami $c \cdot n^2$ összehasonlítást igényel.
b) Igen, mert minden egyes cserénél eggyel csökken az inverziók száma: a megcserélt pár esetén ez megszűnik, és mivel szomszédos elemeket cserélünk, a többi pár viszonya nem változik.

19. (a) A buborékrendezés nyilván konzervatív. (b) Az összefésüléses rendezés egy olyan implementációja konzervatív, ami egyforma elemek felbukkanása esetén azt az elemet írja az eredménytömbbe, amelyik a két összefésülendő tömb közül "előbb levőben" szerepel. (c) A kupacrendezés az 1,2,2 input esetén a két kettes sorrendjét felcseréli a MINTÖR művelet során. (d) A gyorsrendezés jegyzetbeli implementációja megcseréli a két egyes sorrendjét a 2,1,1 tömbben, ha a particionáló elem véletlenül a 2.

20. Világos, hogy csak különböző színű elemeket érdemes cserélni. Ha a kezdetben az első és az utolsó $n/4$ elem piros, akkor akármelyik végére is akarjuk rendezni a pirosakat, valamelyik $n/4$ elem helyet kell cseréljen az összes zölddel, tehát szükséges lehet $n^2/8$ lépés. $O(n^2)$ lépéssel meg meg is lehet csinálni, hisz tetszőleges sorrendet el lehet érni n^2-nél kevesebb szomszéd cseréjével például buborékrendezéssel (piros < zöld, ha a pirosakat akarjuk a tömb elejére tenni).

Megjegyzés: annak eldöntésére sem olyan nehéz eljárást adni, hogy mikor melyik színt érdemes előre gyűjteni.

21. Rendezzük először a tömböt $O(n \log n)$ időben. Ezek után bináris kereséssel minden egyes i indexre $O(\log n)$ időben nézzük meg, hogy a $b - A[i]$ szám benne van-e a (rendezett) tömbben. Ez összesen megintcsak $O(n \log n)$ időt igényel. Megjegyezzük, hogy a rendezés utáni keresés elvégezhető lineáris időben is: az $A[i]$ tömb és a "fordított" $b - A[i]$ tömb ($i = n, n-1, \ldots, 1$) összefésülésével.

22. Vegyük észre, hogy $A[1] < A[j]$ esetén $i > j$, ha pedig $A[1] > A[j]$, akkor $i \leq j$. Az i index ezért megtalálható bináris kereséssel: kezdetben legyen $j = \lceil n/2 \rceil$, az $A[1]$ és $A[j]$ összehasonlítása után a keresést a megfelelő részlistában folytathatjuk. Az összehasonlítások száma így $O(\log n)$ lesz. (Ennyi kell is, hisz az i felvehet n különböző értéket, és ahhoz hogy ennyi lehetséges kimenetel legyen, szükség van legalább $\log_2 n$ összehasonlításra.)

23. Mivel a $2n$ elem bármelyike lehet a keresett, az algoritmusnak $2n$ féle kimenete kell legyen. Ahhoz hogy az összehasonlítások fájának legyen legalább ennyi levele, szükséges $\log(2n)$ elágazás, azaz legalább $1 + \log n$ összehasonlítás.

    Megmutatjuk, hogy a feladat egyetlen összehasonlítás árán visszavezethető kb. feleakkora méretűre. Ebből már következik, hogy $O(\log n)$ összehasonlítás elegendő. A visszavezetés: legyen $m = \lfloor \frac{n}{2} \rfloor$. Hasonlítsuk össze a két lista m-edik ("középső") elemeit, mondjuk legyen $a_m < b_m$. Ekker az $a_i \leq a_m$ elemeknél van $n+1$ nagyobb a két listában, mégpedig $\lceil \frac{n}{2} \rceil$ darab a_j és $\lceil \frac{n}{2} \rceil + 1$ darab b_j. Az első m a_j tehát mind kisebb a keresett elemnél. Hasonlóan látható, hogy az utolsó m elem a b_i-k közül mind nagyobb a keresettnél (a b_j-k között $\lceil \frac{n}{2} \rceil$ és az a_j-k közül $\lfloor \frac{n}{2} \rfloor$, azaz összesen n elem kisebb b_i-nél). Az m darab a_i és ugyanennyi b_i tehát elhagyható, a megmaradt listák hossza $n - m = \lceil \frac{n}{2} \rceil$, és ezekben kell az $n-m$-edik elemet meglelni.

24. Tegyük $i = 1, 2, \ldots, 2n-1$-re (ebben a sorrendben) a következőt: ha $A[i]$ és $A[i+1]$ nem a kívánt viszonyban vannak, akkor hajtsuk végre az $A[i] \leftrightarrow A[i+1]$ cserét.

27. A gyorsrendezéshez hasonlóan járunk el: kiválasztunk egy véletlen apát, az anyákat ezzel az apával összevetve megkeressük a párját, egyben a maradékot két részre osztjuk: kisebbekre és nagyobbakra. A megtalált párral most az apákat is két részre osztjuk. Ezzel visszavezettük a feladatot két kisebbre. A módszer átlagos költségére az $O(n \log n)$ korlát a gyorsrendezés elemzésével megegyező módon kapható.

29. (a) Végezzünk ládarendezést $3n$ láda felhasználásával.
    (b) Ábrázoljuk a számokat n alapú számrendszerben úgy, hogy minden szám pontosan 7 jegyű legyen (kezdő 0-k). Ez minden egyes számra megtehető legfőljebb 6 maradékos osztás segítségével. Hajtsunk végre radix rendezést a számjegyek alapján. A költség $O(7(n+n)) = O(n)$.

30. Az adatsruktúra egy $kisebb[1:k+1]$ tömb lesz, aminek i-edik rekeszébe az A tömb i-nél kisebb elemeinek a száma kerül. Adott a, b párra a keresett szám $kisebb[\min(b+1, k+1)] - kisebb[a]$. A $kisebb$ tömböt az A ládarendezése ($O(n+k)$) után az elejéről kezdve dinamikusan, $O(n+k)$ időben töltjük ki: $kisebb[i+1] = kisebb[i] + l[i]$, ahol $l[i]$ az i-vel egyenlő elemek száma (ld. dinamikus programozás).

37. A tizedik elem távolsága a gyökértől legfőljebb 9. Az ilyen elemek száma legfőljebb $2^{10} - 1 = 1023$ és ezek a kupacot reprezentáló tömb első 1023 helyén vannak. A konstans sok elem közül a tizedik legkisebb kikeresése konstans időt igényel.

38. Ha már fel van építve a kupac, a legkisebb elem további összehasonlítások nélkül megtalálható (a kupac gyökerében). Tehát a kupacépítéshez legalább annyi összehasonlítás szükséges, mint a legkisebb elem kiválasztásához, ami $n-1$.

39. A fa preorder bejárásához hasonló módszert adunk, azzal kiegészítve, hogy megfelelő körültekintéssel egész részfák bejárását megtakarítjuk (vö. *branch and bound* technika). Amikor a kupac egy elemét meglátogatjuk, hasonlítsuk össze k-val. Ha kisebb, írjuk ki ezt az elemet, és folytassuk a bejárást az esetleges gyermekeire. Ha nem, ne folytassuk, mert a kupac-tulajdonság miatt semelyik elem nem jöhet szóba a részfából. A költség $O(m)$, hiszen egy k-nál kisebb elemnek legfeljebb a két fiát látogathatjuk meg "fölöslegesen".

40. A legnagyobb elem nyilván a levelek között helyezkedik el. Ahhoz, hogy rábizonyítsuk egy levélre, hogy a többi levélnél nagyobb, minden további levélnek legalább egy másik kupacbeli elemnél kisebbnek kell bizonyulnia. Mivel a kupac-tulajdonság semelyik levélre nem garantál ilyesmit, legalább *levelek száma* $- 1 = \Omega(n)$ új összehasonlítás szükséges.

41. Lineáris időben megkereshetjük, hogy mely számok hiányoznak és mely helyekről. Az üres helyekre az összes (5!, tehát konstans sok) lehetőséget kipróbáljuk. A kupac-tulajdonságot lokálisan ellenőrizhetjük az 5 kérdéses hely mindegyikére: csak az esetleges apával illetve gyermekekkel kell az elemet összehasonlítani. Tehát a tesztek költsége konstans, az összköltség $O(n)$. Ennél kevesebb időben nyilván nem tudjuk az üres helyeket sem megkeresni.

43. (a) Hasonlítsunk össze $\lfloor n/2 \rfloor$ független párt. A nagyobbak közül keressük meg a legnagyobbat $\lfloor n/2 \rfloor - 1$ összehasonlítással, a kisebbek közül pedig a legkisebbet, ugyanennyi összehasonlítással. Ha n páros volt, akkor megtaláltuk a keresett két elemet, ha pedig páratlan, akkor a kimaradt elemmel még legrosszabb esetben 2 összehasonlítást kell megtenni. Összesen $\lceil 1.5n \rceil - 2$ összehasonlítást végeztünk.
    (b) Tegyük fel, hogy van s összehasonlítást használó algoritmusunk a két elem megkeresésére. Az ellenségmódszert fogjuk használni. Jelölje $K(l)$ illetve $N(l)$ azon elemek halmazát, amelyek az algoritmus l-edik összehasonlító lépése után semely más elemnél nem bizonyultak még nagyobbnak illetve semely más elemnél nem bizonyultak még kisebbnek. Az ellenség stratégiájára csak az a megkötés van, hogy ha egy $K(l)$-beli elemet egy $N(l)$ belivel kell összehasonlítani, akkor a $K(l)$-beli legyen a kisebb. Legyen $M(l) = K(l) \cap N(l)$. Kezdetben $|K(0)| = |N(0)| = |M(0)| = n$, és végül $|K(s)| = |N(s)| = 1$. Vizsgáljuk meg, mit tud változtatni az $l+1$-edik összehasonlítás a $|K(l)| + |N(l)|$ mennyiségen! Ha két $M(l)$-belit hasonlítunk össze, akkor $|K(l+1)| + |N(l+1)| = |K(l)| + |N(l)| - 2$. Minden más esetben a $K(l)$ illetve $N(l)$ halmaznak legfeljebb az egyike csökken, azaz $|K(l+1)| + |N(l+1)| \geq |K(l)| + |N(l)| - 1$. Legyen t azon összehasonlítások száma, amikor $|K(l)| + |N(l)|$ kettővel csökken. Ilyet csak akkor lehet csinálni, amikor $M(l)$ legalább kételemű, viszont cserébe $|M(l+1)| = |M(l)| - 2|$. Innen azonnal látható, hogy $t \leq \lfloor n/2 \rfloor$. Mivel a $|K(l)| + |N(l)|$ $2n$-ről 2-re kell lecsökkennie, $2n - 2 \leq 2t + (s-t) = s + t \leq s + \lfloor n/2 \rfloor$. Innen pedig $s \geq 2n - 2 - \lfloor n/2 \rfloor = \lceil 1.5n \rceil - 2$.

48. Az összefésüléses rendezés kínál egy $O(n \log n)$ idejű módszert. Az alsó korlát a rendezéshez szükséges összehasonlítások számára adott alsó becsléshez hasonlóan látható be: $n!$ permutációt kell tudni kezelni, ugyanakkor egy lépésben legfeljebb 6 féle dolgot csinálhatunk (aszerint, hogy melyik rúdról melyik másik rúdra helyezünk át). Tehát $\log_6(n!) = \Omega(n \log n)$ lépés szükséges.

49. Ld. a FOGYASZT eljárást a jegyzetben.

53. Nem. Hajtsuk végre a buborékrendezés első menetét (külső ciklusát), majd csináljuk meg ugyanezt a tömb végéről lefele (azaz mintha a fordított tömbre a fordított rendezéssel csinálnánk egy buborékrendezés-menetet). Ez a párcserével adódó tömböket rendezi. Tegyük fel ugyanis, hogy az elemek $a_1 < a_2 < \ldots < a_n$ és az $a_i \leftrightarrow a_j$ csere történt ($i < j$). Az első menetben az $a_j \leftrightarrow a_{i+1}, a_j \leftrightarrow a_{i+2}, \ldots a_j \leftrightarrow a_{j-1}, a_j \leftrightarrow a_i$ cserékkel az a_j elem vándorol a helyére: $\ldots a_{i-1} a_{i+1} \ldots a_{j-1} a_i a_j a j + 1 \ldots$ sorrend jön létre. A második menetben az a_i vándorol a helyére és így visszaáll a rendezett állapot. Ugyanakkor ez a módszer a 4321 bemenetből az 1324 sorrendet hozza létre, ami nem rendezett.

4. Keresőfák

1. Nem. Tekintsük a következő fát: a gyökérben legyen 1, ennek egyetlen fia legyen a 3, ennek két fia pedig a 2 és a 4. Legyen $U = \{1, 3, 4\}$. Ekkor $B = \{2\}$, $J = \emptyset$. Nem igaz, hogy $2 < 1$.

4. Mindkét fa elemeit listázzuk ki az inorder bejárás szerint, majd a két listát fésüljük össze! A részlépések költsége $O(n)$, $O(k)$, illetve $O(n+k)$, tehát az összköltség $O(n+k)$.

6. A bináris kereséshez hasonlóan járunk el. Megnézzük a gyökérben levő elemet, legyen ez x. A bal részfában $2^{k-1} - 1$ különböző x-nél kisebb szám, a jobb részfában pedig $2^{k-1} - 1$ különböző x-nél nagyobb, de $2^k + 1$-nél kisebb szám van. Ezért $2^{k-1} \leq x$ és $2^{k-1} \leq 2^k + 1 - x$. Így x biztosan a $\{2^{k-1}, 2^{k-1}+1\}$ elemek valamelyike. Ha $x = 2^{k-1}$, akkor a bal részfában az $1, \ldots, 2^{k-1} - 1$ elemek mindegyike kell hogy szerepeljen, tehát a hiányzó szám a $[2^{k-1} + 1, 2^k]$ intervallumba esik. Ezzel visszavezethető az eredeti feladat a jobb részfára vonatkozó hasonló feladatra. Ha viszont $x = 2^{k-1} + 1$, akkor a hiányzó szám az $[1, 2^{k-1}]$ intervallumba esik, és a bal részfával kell folytatni. Az algoritmus költsége nyilván $O(k) = O(\log_2 n)$.

7. A gyökérnek két fia lesz. Az egyiknek a fiai az A, B, C levelek, a másikéi D, E, F. A gyökérben szereplő kulcs a D. A két közbülső csúcsban 2-2 kulcs lesz: B, C illetve E, F. Megjegyezzük, hogy csúcsvágás az A elem beillesztésekor történik.

9. Jelölje l a fa szintjeinek a számát! A gyökér baloldali részfájában 16 levél van, így ott a szintszám legfeljebb $1 + \log_2 16 = 5$. Tehát $l \leq 5 + 1 = 6$. A középső részfában legfeljebb $3^{l-2} \leq 3^4 = 81$ levél lehet. Ugyanez igaz a jobboldali részfára. Összesen $178 = 16 + 81 + 81$ levél van, tehát mindkét utóbbi részfában pontosan 81 levélnek kell lennie. Ennek megfelelően a második kulcs a $16 + 81 + 1 = 98$.

12. Jelölje $m(S)$ az S részfa magasságát és tegyük fel, hogy mondjuk $m(S_1) \leq m(S_2)$. (Fordított esetben hasonlóan kell eljárni.) Keressük meg S_2 azon legbaloldalibb x csúcsát, melyre $m(x) = m(S_1)$. Az x csúcs mellé szúrjuk be az S_1 gyökerét a 2-3 fáknál szokásos eljárással. (Tehát először beillesztjük x apja alá, és ha annak háromnál több gyermeke támad, akkor csúcsvágást alkalmazunk, ami esetleg felfele terjed.) Ha új gyökeret kell létrehozni, az ahhoz tartozó magasságot (eggyel nagyobb az alatta levők magasságánál) is írjuk fel. Lépésszám: $O(|m(S_1) - m(S_2)| + 1)$

13. Indukcióval a fa l magassága (szintszáma) szerint. Mivel az egypontú fa teljes bináris fa, $l = 1$-re igaz az állítás. Tegyük fel, hogy $l > 1$ és az l-nél alacsonyabb fákra már beláttuk az állítást. Vegyünk most egy, a feltevésnek megfelelő l magasságú fát. Legyen a (gyökértől vett) bal részfa magassága k_b és a jobb részfáé k_j. Az indukciós feltevés szerint a bal részfa egy k_b szintű, tehát $2^{k_b} - 1$ súlyú a jobb részfa pedig egy $2^{k_j} - 1$ súlyú teljes bináris fa. Ha $k_b > k_j$, akkor $(2^{k_b} - 1)/(2^{k_j} - 1) > (2 \cdot 2^{k_j} - 1)/(2^{k_j} - 1) > (2 \cdot 2^{k_j} - 2)/(2^{k_j} - 1) = 2$, ami ellentmond

a feltevésnek. Hasonlóan, a $k_b < k_j$ eset sem fordulhat elő. Tehát a két részfa egyforma teljes bináris fa, ami éppen azt jelenti, hogy az eredeti fánk is teljes.

15. A kupac-tulajdonság miatt a gyökérben az a P pont van, amelynek a legkisebb a második koordinátája. A keresőfa-tulajdonság miatt a bal részfában azok a pontok vannak, amelyeknek az első koordinátája kisebb P-énél, a jobb részfában pedig azok, amelyek első koordinátája nagyobb P-énél. Ez alapján az állítás indukcióval egyszerűen adódik.

17. A négyzetrács minden egyes pontjához 12 "közeli" (2-nél nem messzeb levő) további rácspont van. Minden egyes P_i pontra szeretnénk $O(\log n)$ időben eldönteni, hogy a 12 P_i-hez közeli rácspont valamelyike szerepel-e a $P_1, \ldots, P_n$ között. Ez meg is tehető, ha a pontokból előzetesen $O(n \log n)$ költséggel (például a koordinátapárok lexikografikus rendezése alapján) $O(logn)$ szintű keresőfát (pl. AVL-fát vagy 2-3 fát) építünk.

19. A fa élei: $(3,2), (3,6), (2,1), (6,4), (6,7), (4,5)$. A 2 beillesztésekor egyszeres, a 6 beillesztésekor pedig dupla forgatás történik.

22. A fa csúcsainak inorder sorszáma határozza meg, mely számok mely csúcsokban helyezkednek el. A fa csúcsaiban levő kulcsok tehát bejárással lineáris időben meghatározhatók. Ezután a szélességi bejárás szerinti sorrendben szúrjuk be a kulcsokat (üres fából kiindulva). Ez is lineáris időbe telik. Mivel a szélességi bejárás szintenként történik, az AVL-tulajdonság teljesülni fog a kapott sorozatra minden egyes beillesztési lépésben.

23. Ha egyáltalán forgattunk, akkor a forgatást a beillesztett elem keresési útja mentén legmélyebben levő azon x csúcs "körül" hajtottuk végre, ahol a keresőfa-tulajdonság megsérült a beillesztés során. Ez csak úgy történhetett, hogy az eredeti fában x egyik részfája eggyel magasabb volt, mint a másik, és a beszúrás után még eggyel magasabb lett. Megfigyelhetjük, hogy a megfelelő forgatás (akár dupla, akár szimpla) a két részfa magasságát egyenlővé teszi, nevezetesen akkorává, amekkora a magasabb részfa eredetileg volt. Ezért az új fa magassága megyegyezik az eredetiével. Tehát semmilyen forgatást nem alkalmaztunk.

24. A következő iteratív módszerrel járuk el. Minden egyes lépésben, ha a fa még nem egy jobbra menő út, akkor a jobb szélén haladó út valamelyik pontjának van bal fia. Válasszunk ki egy ilyen csúcsot, és forgassunk "körülötte" jobbra. Vegyük észre, hogy a forgatástól a jobb szélen menő út hossza eggyel megnő. Ezért az eljárás $n - l$ forgatás után véget ér, ahol l a eredeti jobb szélső út pontjainak a száma.

26. A fa élei: $(3,2), (3,6), (2,1), (6,4), (6,7), (4,5)$. A 2 beillesztésekor egyszeres, a 6 beillesztésekor pedig dupla forgatás történik.

5. Adattömörítés

4. Felrajzoljuk a kódoknak megfelelő Huffman-fákat. A Huffman-fa felépítési szabályát alkalmazzuk az első fára. Többek között azt kapjuk, hogy a $p_1, p_2, p_3 + p_4$ mennyiségek körében p_2 és $p_3 + p_4$ két legkisebb. Következésképpen $p_3 + p_4 \leq p_1$. A másik fára alkalmazva a szabályt azt nyerjük, hogy p_1 és p_2 is két legkisebb, tehát $p_1 \leq p_3 + p_4$. A két egyentlőtenséget összvetve azt nyerjük, hogy $p_1 = p_3 + p_4$. Tudjuk, hogy $p_4 \leq p_3 = 1/6$. Tegyük fel, hogy $p_4 < 1/6$. Ekkor $p_1 = p_3 + p_4 < 1/3$. Továbbá ($p_2 \leq p_1$ miatt) $p_2 < 1/3$ is igaz. Ekkor viszont $p_1 + p_2 + p_3 + p_4 < 1/3 + 1/3 + 1/6 + 1/6 = 1$, ami ellentmond annak, hogy a valószínűségek összege 1 kell hogy legyen. Tehát $p_4 = 1/6$ és ebből $p_1 = p_2 = 1/3$ adódik.

5. (a) Feltehető, hogy a gyakoriságok 2,3,4,5,6. Alkalmazzuk a Huffman-algoritmust!
(b) A Huffman-algoritmussal kétféle fát kaphatunk. Mindkettőre igaz, hogy a három gyakori betű kódja 2 bitből áll, a maradék kettőé pedig 3-ból.

7. A betűk egy lehetséges Huffman-kódja: A − 11, Á − 100, B − 01, K − 001, N − 000, R − 101
a szóé ezek szerint: 01111010111110110000011001

Az LZW szótár: A − 1, Á − 2, B − 3, K − 4, N − 5, R − 6, BA − 7, AR − 8, RB − 9, BAR − 10, RÁ − 11, ÁN − 12, NA − 13, AK − 14
a szó kódja: 316762514

13. Nem. Pl. legyen a szó $yxyxy$. Ekkor a szótár: $x - 1$, $y - 2$, $z - 3$, $yx - 4$, $xy - 5$, $yxy - 6$, a kód: 2142

14. A Huffman-kód esetén mindkét karakter 1–1 bittel lesz elkódolva, egy n betűs szöveg kódja is n hosszú.

Vegyünk egy jó hosszú csupa a-ból álló szót, a^n. Ha $n = k(k+1)/2$, akkor a szótárba az a^2, a^3, a^k szavak kerülnek be, a kódszó $c(a)\,c(a^2)\,\ldots\,c(a^k)$ lesz. Hogy mindez (és a meg b is) beférjen a szótárba szükséges, hogy $k+1 \leq 2^8$ teljesüljön. A kódszó hossza $8k$ bit, ez kisebb mint n, ha $8 < (k+1)/2$, azaz $k > 15$, $n \geq 8 \cdot 17 = 136$. (Ha kihasználjuk a szótárméret nyújtotta lehetőségeket, azaz $k = 2^8 - 1 = 255$, akkor $n = 255 \cdot 2^7$ lehet, a LZW kódszó hossza pedig csak $8k = 255 \cdot 8$, tizenhatodrésze a Huffman-kódénak.)

15. Legyen s egy százbetűs szó a szótárból. Minden $1 \leq k \leq 100$-ra jelöljük s_k-val az s szó első k betűből álló kezdőszeletét és S_k-val az S szöveg azon kezdőszeletét, aminek beolvasásakor a szótárba bekerül az s_k szó. A kezdőeseteket tekintve látjuk, hogy $|S_1| = 0$ és $|S_2| \geq 2$. A $k > 2$ esetre az $|S_k| \geq |S_{k-1}| + (k-1)$ becslést alkalmazhatjuk. Ez onnan következik, hogy S_k-nak a legutóbb szótárba írt szó utolsó betűjétől kezdődő végszelete éppen s_k, valamint, hogy s_{k-1}-nek benn kell lenni a szótárban, amikor s_k építése elkezdődik. Innen $|S| \geq |S_2| + \sum_{k=3}^{100}(k-1) = 1 + \sum_{l=1}^{99} l = 1 + 99 \cdot 100/2 = 4451$. A gondolatmenetből az is látható, hogy ez az alsó korlát pontosan akkor érhető el, ha S csupa egyforma betűből álló szöveg.

16. A legelőször kiírt kód egy 1 hosszú részsorozatot kódol. Mivel egy szó szótárba kerülésekor a prefixei már a szótárban vannak, a másodikként kiírt kód egy legföljebb 2 hosszú részsorozatot kódol. Hasonlóan, a k-adik kód egy legföljebb k hosszú szót kódolhat. Összesen tehát k kóddal legföljebb $1 + 2 + \ldots + k = k(k+1)/2$ hosszú szöveget lehet kódolni. Ez a korlát a csupa azonos betűből alló $k(k+1)/2$ hosszú szóval el is érhető: $l \leq k$-ra a szöveg $l(l+1)/2$-edik betűjéig tartó l hosszú részsorozata helyett éppen az l hosszú szó kódját írjuk ki. Ha a csupa egyforma betűs szöveg hossza $(k-1)k/2$ és $k(k+1)/2$ közé esik, az utolsónak kiírt kód értelemszerűen egy rövidebb szó kódja. Mivel $44 \cdot 45/2 < 1000 < 45 \cdot 46/2$, egy 1000 hosszú szöveg kódolásához legalább 45 kód kell, és van olyan 1000 hosszú szöveg, amihez ez elegendő is.

17. Ha egy betűsorozatot tömörít az algortimus, akkor már az összes prefixe a szótárban van. Tehát azt kell vizsgálni, mikor nincsen 2 hosszú tömörített sorozat. Ehhez az kell, hogy a szövegben (pontosabban a szótár beteléséig előforduló részében) ne ismétlődjék 2 hosszú szó. Ugyanis amíg nincs ismétlődés az összes előforduló kettő hosszú szó bekerül a szótárba, és az első ismétlődéskor a kód ki is íródik. (Ismétlődésnek számít az "aaa"-ban az "aa" kétszeri előfordulása is.) Mivel egy k hosszú szónak ($k > 1$ esetén) $k-1$ darab 2 hosszú részsorozata van, valamint a lehetséges betűpárok száma n^2, a szöveg maximális hossza legfeljebb $n^2 + 1$. Indukcióval belátjuk, hogy létezik is ilyen hosszú ismétlődésmentes szöveg, ami az első betűvel kétszeri ismétlődésével kezdődik és az első betűvel is végződik. $n = 1$-re a konstrukció nyilvánvaló: 11. Tegyük fel, hogy s egy megfelelő $(n-1)^2 + 1$ hosszú, az $1, \ldots, n-1$ betűkből álló szöveg. Legyen $s' = snn(n-1)n(n-2)\ldots n2n1$. Könnyen ellenőrizhető, hogy s' hossza $(n-1)^2 + 1 + 2n - 1 = n^2 + 1$ és s'-ben sem ismétlődnek a 2 hosszú szavak. Tehát a maximális hossz $n^2 + 1$, feltéve, hogy a szótár elég nagy ahhoz, hogy elférjen benne az összes betűpár. Ha nem, akkor az előbb vázolt konstrukció elejéről az 1-et törölve és a végére akárhányszor odafűzve tetszőleges hosszú tömörítetlen szöveget kapunk.

6. Hashelés

4. Az, hogy csak négy értéket vesz fel az összes lehetséges 7 közül.

6. (a) 22, üres, 59, 15, 4, 17, 28, üres, 88, 31, 10.
(b) 22, 88, üres, 59, 4, 15, 28, 17, üres, 31, 10.

7. a) Olyan elemek (kulcsok), amelyek között a táblában üres hely áll, nem ütközhettek, azaz legfeljebb az egymásmelleti párok másodiknak érkezett eleme ütközhetett az elsővel, számuk max n.
b) A kvadratikus próbánál a próbasorozat így indul: 0,1,-1, azaz 3 egymás melletti helyet néz meg. Mivel a táblában minden 3. hely üres, egy elem csak egyszer ütközhetett, így megint legfeljebb n ütközés lehetett.

Ha a hash-függvény a $2k-1$-edik és a $2k$-adik elemhez ($k = 1, \ldots, n$) egyaránt a $3k+2$ számot rendeli, a lineáris próbánál tényleg létre is jön n darab ütközés. Hasonló a helyzet a kvadratikus maradék próbánál, csak ott a $3k+1$ értéket kell venni.

8. (a) Nem. Például ha a rekordot közvetlenül a beszúrása után törültük, olyan eredmény kapunk, mintha sem a beszúrás, sem a törlés nem történt volna meg.
(b) Baj akkor van, ha létezik olyan K kulcs, ami benn van a táblában, de $h(K)$ és K tényleges helye között (ciklikusan értelmezve) kitöröltünk egy cellát. Először keressük meg a táblában az első elemtől kezdve az első üres helyet. Ezután, az első elemtől kezdve járjuk be a táblát "visszafele" haladva. Minden egyes lépésben csináljuk a következőt: ha a cella üres, jegyezzük föl, hogy ő az utoljára látott üres hely. Ha nem, és a K kulcs helyezkedik el benne, számoljuk ki $h(K)$-t és hasonlítsuk össze az utoljára látott üres hellyel. Ha az üres hely $h(K)$ és K tényleges helye között van (ciklikus értelemben), megtaláltuk a hibás törlés helyét, és az első olyan K kulcsot, amely "rossz" helyen van. Ez eddig nyilván lineáris időt vesz igénybe. Ezután K-t mozgassuk át az üres helyre, az új üres hely szerepét vegye át K eredeti helye. Ha (visszafele) a következő cella üres, készen vagyunk. Ha nem, vizsgáljuk meg a benne levő K' kulcsot, hogy a mostani üres hely $h(K')$ és K' helye között van. Ha igen, megint mozgassuk át K'-t, ha nem, lépjünk a megelőző cellára. Ezt az eljárást ismételjük az első üres cella feltalálásáig vagy "körbeérésig" (az első átmozgatás helyére). Ez a menet is lineáris idejű, hiszen a táblát most is legfeljebb egyszer járjuk "körbe".

7. Gráfalgoritmusok

1. (a) Az éleket mindkétfelé irányítottnak véve igen.
(b) Nem.
(c) Nem.

2. A minimális élszámú gráfokban csak azok az élek szerepelnek, amelyek mentén valamelyik lépésben javítás történt. A harmadik lépésben választási lehetőségünk van, v_4 illetve v_6 közül melyik csúcsot tegyük a KÉSZ halmazba, két minimális gráf lesz. A szomszédsági mátrixok élsúlyokkal és a P

$$\begin{pmatrix} 0 & 2 & 6 & * & * & 7 \\ * & 0 & 3 & 7 & * & 4 \\ * & * & 0 & 1 & 4 & * \\ * & * & * & 0 & 2 & * \\ * & * & * & * & 0 & * \\ * & * & * & * & 1 & 0 \end{pmatrix} \qquad P: \begin{array}{|c|c|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ \hline v_1 & v_1 & v_1 & v_1 & v_1 & v_1 \\ \hline v_1 & v_1 & v_2 & v_2 & v_1 & v_2 \\ \hline v_1 & v_1 & v_2 & v_3 & v_3 & v_2 \\ \hline v_1 & v_1 & v_2 & v_3 & v_4 & v_2 \\ \hline v_1 & v_1 & v_2 & v_3 & v_6 & v_2 \\ \hline \end{array}$$

$$\begin{pmatrix} 0 & 2 & 6 & * & * & 7 \\ * & 0 & 3 & 7 & * & 4 \\ * & * & 0 & 1 & 4 & * \\ * & * & * & 0 & 1 & * \\ * & * & * & * & 0 & * \\ * & * & * & * & 2 & 0 \end{pmatrix} \qquad P: \begin{array}{|c|c|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ \hline v_1 & v_1 & v_1 & v_1 & v_1 & v_1 \\ \hline v_1 & v_1 & v_2 & v_2 & v_1 & v_2 \\ \hline v_1 & v_1 & v_2 & v_3 & v_3 & v_2 \\ \hline v_1 & v_1 & v_2 & v_3 & v_6 & v_2 \\ \hline v_1 & v_1 & v_2 & v_3 & v_4 & v_2 \\ \hline \end{array}$$

4. A Dijkstra-algoritmus kupacot nem használó változatának első 100 menetét hajtsuk végre! Ekkor a KÉSZ halmazba "a" 100 legközelebbi csúcs kerül. A módszer fontosabb összetevőinek a költsége: kezedti táblázatkitöltés: $O(n)$, 100 minimumkeresés: $100O(n) = O(n)$, a táblázat módosítása 100-szor: $100O(n) = O(n)$. Az összköltség tehát $O(n)$. (Ez kisebb, mint az input mérete: nem kell az egész állistát végigolvasni.) Ha a 2-kupacos változatot használjuk, rosszabbul járunk, mert az $O(n)$ kulcsmódosítás összköltsége $O(n \log n)$ lesz. Meggondolható, hogy $d = \lfloor\sqrt{n}\rfloor$-nel d-kupacot használva szintén $O(n)$ összköltségű algoritmust kapunk.

5. Legyen G a következő irányított gráf: G csúcsai legyenek a pixelek, G élei pedig menjenek a szomszédos pixelek között jobbra, illetve lefele. G nyilván egy DAG, aminek n^2 csúcsa van és minden csúcs befoka legfeljebb 2. Az éleknek súlyokat is feleltessünk meg: egy függőleges élnek a tőle jobbra eső fekete, egy vízszintesnek pedig az alatta levő fehér pixelek számát. Világos, hogy G éllistás ábrázolása $O(n^2)$ időben elkészíthető. Az élsúlyok is megkaphatók $O(n^2)$ időben a függőleges élekre soronként jobbról balra haladva, a vízszintesekre pedig oszloponként lentről fölfele. Ebben a DAG-ban kell legrövidebb utat találni a bal felsőből a jobb alsó sarokba. Ez lineáris, azaz esetünkben $O(n^2)$ időben megtehető.

6. A legrövidebb utak megtalálására alkalmazhatjuk a Bellman-Ford módszert. Vegyük azonban észre, hogy most elegendő a távolságokat tartalmazó $T[i,j]$ értékeket az $1 \le i \le 25$ esetekben meghatározni (25-nél több élből álló egyszerű út úgysincs a gráfban). Ez pedig $25O(n^2) = O(n^2)$ lépés lesz.

11. (a) Az axiómarendszerből levezethető $x_k \Rightarrow x_l$ implikációk.
(b) Egy erős komponensbe az ekvivalens változók kerülnek: x_k és x_l pontosab akkor egy komponensbe esik egy komponensbe, ha $x_k \Leftrightarrow x_l$ levezethető az axiómarendszerből.

14. A bejárás nyilván x-ből indul. A másodikként meglátogatott y csúcs befejezési száma kisebb, mint x-é, tehát y fia x-nek. Hasonlóan, a harmadikkét látott u csúcs fia y-nak, v pedig fia u-nak. Ez utóbbi csúcs levél, mert nincs nála kisebb befejezési szám. Hasonlóan, az ötödik w csúcs is levél. Ez pedig fia u-nak, mert meglátogatása később kezdődött, ugyanakkor eggyel előbb fejeződött be. Az utolsó z csúcs viszont nem lehet leszármazottja y-nak, mert befejezési száma nagyobb. Viszont leszármazottja x-nek. Ez csak úgy lehet, hogy fia x-nek. A fa élei tehát: $(x,y),(y,u),(u,v),(u,w),(x,z)$. Mivel például a (v,x) visszaél léte vagy nemléte nem befolyásolja a bejárást, a gráf nem rekonstruálható.

16. Ha egy csúcs foka legalább 2, akkor az onnan indított szélességi bejárás csak úgy adhat csillagot, ha a csúcs minden egyes további ponttal össze van kötve. Nyilván van legalább egy ilyen pont. Ha csak egy van, akkor a gráfunk csillag. Ha pedig van legalább kettő, akkor minden csúcs foka legalább 2, és így a gráf teljes.

17. Mindkettő lineáris idejű megoldást ad.

19. Húzzuk szét a 2 súlyú éleket 2 éllé (egy-egy új pont felvételével), a 3 súlyúakat pedig 3 éllé. Az így kapott $G' = (V', E')$ gráfra teljesül, hogy $|V'| \le |V| + 2|E|$ és $|E'| \le 3|E|$. A G-beli legrövidebb utak helyett elegendő G'-ben legkevesebb élből álló utakat keresni. Ez pedig a v-ből induló szélességi bejárással megtehető. Lépésszám $= O(|V'| + |E'|) = O(n + e)$.

20. Módosítsuk a mélységi bejáró algoritmust úgy, hogy álljunk meg, ha a soron következő él mentén már bejárt csúcsba jutunk. Ha ez nem következik be, akkor a gráf körmentes. Ha igen, akkor az él visszaél és a meglevő feszítő erdőben a végpontjai között menő úttal együtt kört alkot, ami $O(n)$ időben megtalálható. Bármely eset is következik be, a bejárás legfeljebb n él megtekintése után véget ér, hiszen egy erdőnk és esetleg még egy élünk van. A szélességi bejárás hasonlóan módosítható.

21. Építsünk föl egy feszítő erdőt a gráf nem zöld éleiből $O(n + e)$ időben (pl. mélységi bejárással), és egyben számozzuk meg a csúcsokat aszerint, hogy hanyadik komponensbe esnek. Ha a kapott komponensek száma nagyobb, mint három, akkor nem oldható meg a feladat. Ha három komponens van, két olyan zöld élet kell keresni, amelyek különböző komponens-párokat kötnek össze (pl. egy az első és második komponens közt menő él, valamint egy a második és harmadik komponenseket összekötő él megteszi). Ha két komponens van, egy olyan zöld él kell, ami a két komponenst köti össze, és egy olyan, ami valamelyik komponensen belül halad. Az utóbbi él egyik végpontjából menő feszítőerdő-élt el kell kagyni. Az egyetlen komponens esete hasonlóan kezelendő. Mindhárom esetben a két megfelelő él keresése a zöld élek végignézésével lineáris időben elvégezhető. Az összköltség tehát $O(n + e)$.

22. Készítsünk egy feszítőfát $O(n+e)$ költséggel mélységi (vagy szélességi) bejárással. Közben jegyezzük fel az egyes csúcsoknak a fa élei szerinti távolságát a gyökértől. Ez is belefér az $O(n + e)$ időbe. Keressük meg a gyökértől legtávolabbi (egyik) levelet ($O(n)$ idő). Ha a legnagyobb távolság legfeljebb $n/2$, akkor a gyökér megfelelő. Ha nagyobb, vegyük a kiválasztott l levéltől a gyökér felé a feszítőfában vezető úton (ez a bejárás közbeni fiú-apa mutatókkal adminisztrálható) levő $\lfloor n/2 \rfloor$-edik u csúcsot. Állítjuk, hogy u egy megfelelő választás. Legyen ugyanis v egy másik csúcs. Legyen w az u és v legmélyebben levő közös őse a feszítőfában. Ha $w = u$, azaz v az u részfájában van, akkor mivel az l levél a lehető legtávolabb van a gyökértől, u részfájában a lehető legtávolabb van u-tól is, tehát $\lfloor n/2 \rfloor = d(u,l) \ge d(u,v)$. Ellenkező eseteben (ha $w \ne u$) tekintsük a $w - l$ és a $w - v$ utak egyesítését. Ezen utak w kiválasztása miatt csak a w ponban találkoznak, tehát az egyesítés szintén egy út, ami l-ből v-be vezet. Ennek az útnak az $\lfloor n/2 \rfloor$-edik csúcsa u. Mivel az út hossza legfeljebb $n-1$, u távolsága a másik végponttól legfeljebb $n - 1 - \lfloor n/2 \rfloor \le n/2$ lehet.

23. Készítsük el G egy feszítőfáját (pl. mélységi bejárással) $O(e)$ időben! Vegyük a feszítőfa páros vagy páratlan szintjein levő csúcsokat aszerint, hogy melyek vannak kevesebben!

25. **I.** A visszaállításnál megpróbáljuk a tankönyben megadott rekurziót használó bejáró algoritmust szimulálni, pontosabban az mb() rutin rekurzív hívási hierarchiáját felderíteni. A hívások időbeli sorrendjét a mélységi számozás adja. Ezért előzetesen rendezzük a csúcsokat a mélységi számok szerinti növekvő sorrendben. Ez ládarendezéssel $O(n)$ időben megtehető. A hívási hierarchia pillanatnyi állapotát egy verem segítségével ábrázoljuk, a rekurzív hívások szokásos számítógépes megvalósításának megfelelő módon: A verem sorrendben azokat a v csúcsokat tartalmazza, amelyekre az adott pillanatban élő mb(v) hívás van, más szóval, amelyek meglátogatása éppen folyamatban van. Egy v csúcsnak a verembe kerülése az mb(v) hívásnak, ez megfelel annak, hogy mikor történik az mb(v) hívás, azaz mikor kezdjük a v gyökerű részfa bejárását, míg a verem tetejéről való levétele a visszatérésnek, azaz a részfa-bejárás befejezésének felel meg. Ennek megfelelően a csúcsok a befejezési szám szerinti sorrenben kerülnek ki a veremből. Karbantartunk egy befejezezési sorszámot. Ez kezdetben 1 és minden "befejezéskor", azaz a veremből kivételkor léptetjük. Azt, hogy egy adott pillanatban a verem tetején levő v csúcsot levegyük-e vagy egy új csúcs felrakásával folytassuk, aszerint döntjük el, hogy v befejezési száma megegyezik-e vagy sem a befejezési sorszámmal A feszítő erdő élei a veremben valaha is előforduló egymást követő (u,v) csúcspárok. Egy ilyen élet feljegyezhetünk akkor, amikor v a verembe kerül. Az algoritmus futása (érvényes bemenet esetén) az mb() hívásokat (és visszatéréseket) követi, és egy-egy ilyenre konstans mennyiségű munka jut, tehát az összes időigény $O(n)$.

II. Megadunk egy elvében hasonló, de vermet nem használó algoritmust is. Az egyszerűség kedvéért először egy olyan módszert tárgyalunk, amely arra a speciális esetre működik, amikor a mélységi feszítőerdő egyetlen fából áll. A csúcsokat itt is a mélységi sorszám szerinti sorrendben vesszük sorra. Az éleket is a meglátogatásuk sorrendjében vesszük fel. A már elkészült fadarabot apa $\leftrightarrow$ fiú mutatókkal tartjuk karban.

Amikor egy a gyökértől különböző v csúcs sorra kerül, meg kell keresni az apját. Legyen ez w. Legyen u a v előtt közvetlenül a fába felvett csúcs. Állítjuk, hogy a w csúcs az u-nak és v-nek közös őse a fában. Tekintsük a bejárásnak azt a szakaszát, amikor a w gyökerű részfát járjuk be. Ha w-nek az első meglátogatott fia éppen a v, akkor az állítás $u = w$-vel teljesül. Ellenkező esetben v előtt w-nek valamelyik leszármazottját látogattuk meg: ez kell, hogy u legyen. Tehát w tényleg közös őse u-nak és v-nek, szükségszerűen a gyökérből az u-ba vezető úton u-hoz legközelebb eső közös ős. A v csúcs apját ez alapján a v-csúcsból a gyökér fele visszafelé ballagva találhatjuk meg. Ehhez szükség van annak ellenőrzésére, hogy egy ilyen w' csúcs őse-e v-nek is. Ennek feltétele a következő: egy olyan w', amely őse u-nak akkor és csak akkor őse v-nek, ha bszám$[v]$ < bszám$[w']$, azaz ha v meglátogatása előbb fejeződik be, mint w'-é. Az állítás "csak akkor" része nyilvánvaló. A fordított irányú következtetéshez tudjuk, hogy w és ősei úgyis ősei v-nek, tehát csak azt az esetet kell vizsgálni, amikor w' egy valódi leszármazottja w-nek. Ekkor w' és v két különböző részfájában van w-nek. Az egyik részfa bejárása előbb fejeződik be, mint a másiké megkezdődik. Mivel u és w' ugyanabban a részfában van és u-t előbb láttuk,

mint v-t, a korábban bejárt részfa csakis a w'-t tartalmazó lehet. Ezért w' meglátogatását előbb fejeztük be: bszám$[w']$ < bszám$[v]$. Tehát a kritérium tényleg használható.

A lépésszám a fenti tesztek számában lineáris. Hány ilyen tesztet alkalmazunk? Ennek megbecsléséhez fogjuk meg a dolgot a másik végénél. Tehát azt számoljuk meg, hogy rögzített w' csúcshoz hány v-re alkalmazzuk a bszám$[v]$ < bszám$[w']$ tesztet. Pozitív eredménnyel éppen annyiszor, ahány fia van w'-nek. Hányszor fordul elő negatív kimenettel? Az előző fejtegetésből azt látjuk, hogy egy ilyen esetben w' részfájának bejárása előbb fejeződik be, mint v-ének a megkezdése. Igaz ez v helyett minden olyan további csúcsra is, melynek mélységi száma legalább annyi, mint v-é. Tehát ilyen csúcsnak nem lehet őse w'. Ha egy v' csúcsra alkalmazzuk a bszám$[v']$ < bszám$[w']$ tesztet, akkor a v'-t mélységi számban közvetlenül megelőző u' csúcsnak őse kell legyen w'. Ez nem fordulhat elő mszám$[v']$ > mszám$[v]$ esetén, mert akkor mszám$[u']$ = mszám$[v']$ − 1 ≥ mszám$[v]$ tejesül. Tehát rögzített w'-höz legfeljebb egy negatív teszt tartozik. Összesen tehát $n-1$ pozitív teszt (ennyi a feszítőfa éleinek a száma) és legfeljebb n sikertelen teszt van. Tehát a módszer költsége $O(n)$.

Mi van, ha a feszítő erdőnek több komponense van? A módszer korrektül működik egészen addig, amíg egy v csúcsnak sehogyan sem találunk apát. Ebben az esetben új fát kell kezdeni v gyökérrel és arra folytatni az algoritmust. A már bejárt komponensek csúcsait soha többé nem fogjuk látni, tehát a komponensenkénti költségek összeadódnak.

26. Feltesszük, hogy G éllistával adott. Indítsunk mélységi bejárást egy tetszőleges csúcsból. Az élek számozása lényegében a bejárás szerinti felhasználás sorrendjében történik: kapja egy él azt a sorszámot, ahányadikként a bejárás során először (azaz fa- vagy visszaélként) felbukkan. A kiindulási pontra teljesül a feltétel, mert valamelyik belőle induló él kapja az 1-es sorszámot. Egy tetszőleges ettől különböző legalább másodfokú u is, mert az az él amely mentén u-ba érkeztünk és u éllistájának az első (a beérkezőtől különböző) éle két szomszédos sorszámot fog kapni. (Ehhez azt kell észrevenni, hogy amikor u meglátogatása megkezdődik, az u-ra illeszkedő élek közül csak egyetlen egy, a mélységi feszítőfa u-ba vezető éle kapott sorszámot.) Világos, hogy a módszer költsége a bejárásával arányos, tehát az élszámban lineáris.

28. Először a piros szabály ismételt alkalmazásával belátjuk, hogy G-nek van olyan minimális költségű feszítőfája, aminek az élei G' adott F feszítőfájának az élei, valamint a v_1 csúcsból kiinduló élek közül kerülnek ki, azaz, takaros a színezésünk, ha a maradék (azaz G'-beli pontok között menő nem F-beli) éleket pirosra festjük. Legyen ugyanis g egy tetszőleges még nem piros maradék él. Ekkor g az F bizonyos éleivel együtt kört alkot. A körnek az egyik legnagyobb súlyú éle g, mert különben a nála nagyobb súlyú élt g-re kicserélve egy F-nél olcsóbb feszítőfát kapnánk. Tehát g pirosra festhető, és még mindig takaros színezésünk van. Ezt ismételve adódik az állítás. Töröljük most a maradék éleket! Így egy $e'' = O(n)$ élű gráfban kell minimális feszítőfát keresni, ami pl. Prim módszerével $O(e'' \log e'') = O(n \log n)$ időben megtehető.

30. Készítsük el G-nek egy E' maximális párosítását a magyar módszerrel $O(ne)$ időben! Az E'-be tartozó élek, továbbá azok az élek, amelyeknek egyetlen végpontja párosított, nyilván benne vannak egy max. párosításban. A további élek felderítésére csináljuk a következőt: L minden egyes párosított v pontjára töröljük E'-ből azt az élet, ami v-ből indul, majd építsük föl a maradék párosításhoz tartozó alternáló erdőt $O(e)$ időben! (Ne álljunk meg az első javító út megtalálásakor! Valójában elég lesz csak a v csúcsból induló alternáló bejárást elvégezni.) Egy v-ből induló e él pontosan akkor van benne egy max. párosításban, ha van olyan javító út, aminek első éle éppen e. Ha az alternáló bejárás során minden egyes csúcsra nyilvántartjuk a fában a v-ből odavezető út legelső élét, akkor a javító utak végpontjaiból ezek az élek kiderülnek. A költség csúcsonként $O(e)$, összesen tehát $O(ne)$.

36. Készítsük el a következő gráfot: A menetrendben megadott minden egyes "vonathoz", azaz rendezett (U, V) állomás–állomás párhoz vegyünk fel két végpontot és kössük össze őket egy olyan súlyú irányított éllel, amennyi az U-ból V-be menő vonat menetideje. Tehát egy városnak annyi "kimenő" csúcs fog megfelelni, ahány városba onnan közvetlen vonat indul és annyi "bemenő" csúcs, ahány helyről vonat érkezik. Kössük továbbá össze az (U, V) vonathoz rendelt végpontot a (V, W) vonat kezdőpontjával egy akkora súlyú irányított éllel, amennyi a várakozási idő. A kitüntetett A városhoz vegyünk föl még egy külön – szintén A-val jelölt – csúcsot és indítsunk belőle 0 súlyú éleket az A-ból induló vonatokhoz. Hasonló módon vegyük fel a B csúcsot és a B-be menő éleket. A feladat ezek után az így kapott gráfban egy legrövidebb az A-ból B-be vezető út megtalálása, ami megoldható Dijkstra algoritmusával. Ha k vonat van megadva és l várakozási idő, akkor a gráf csúcsainak a száma $O(k)$ míg az élek száma $O(k + l)$. Dijkstra algoritmusának az éllitás változata tehát $O((k + l) \log k)$ időben oldja meg a feladatot.

37. Definiálunk egy gráfot, melynek csúcsai a lehetséges helyzetek (melyik edényben mennyi folyadék van), egy csúcsból irányított él vezet egy másikba, ha egy megengedett lépéssel elérhető belőle. A feladat azt kérdezi, hogy az adott kiindulási helyzetből elérhető-e egy adott helyzet, azaz hogy a gráfban vezet-e oda út és még a legrövidebb út hosszát is meg kell határoznunk. Ezt megtehetjük a szélességi bejárás segítségével. Vegyük észre, hogy nem feltétlenül kell előbb megadnunk az egész gráfot (a helyzetek száma nagyon sok lehet), hanem elegendő, hogy amikor egy pont szomszédaira vagyunk kíváncsiak, azokat előállítjuk. (Ez azért lehet hasznos, mert általában lehetnek olyan helyzetek, amelyek nem érhetőek el a kiindulási helyzetből, másrészt ha a cél kevés lépésen belül elérhető, nincs szükség a gráf további részének felírására.)

39. Minden csúcsból indítsunk egy szélességi bejárást. Egy v csúcson átmenő 4 hosszú körök száma megegyezik a szélességi feszítőfa második és harmadik szintjei közt menő keresztélek számával. Nyilvántartva a szintszámot, ezek az élek csúcsonként $O(e) = O(n^2)$ időben leszámolhatók. Az összeget el kell osztani 4-gyel.

40. Tegyük fel, hogy van olyan algoritmus, ami $\binom{n}{2}$-nél kevesebb lehetséges élet kérdez le. Ekkor persze van olyan is, ami éppen $\binom{n}{2} - 1$ élet kérdez. Ezzel az algoritmussal szemben az ellenség alkalmazza a következő ("mohó") stratégiát: Az "(u, v) él-e" kérdésre legyen a válasz "igen", ha az eddig "bevallott" élekkel együtt nem keletkezik kör, egyébként a válasz legyen az, hogy "nem". A feltételezés szerint $\binom{n}{2} - 1$ kérdés után a probléma eldől: függetlenül az $\binom{n}{2}$-edik (u', v') él létezésétől vagy az bizonyosodik be, hogy a gráfban van kör, vagy az, hogy nincs. Előbbi nyilván csak úgy következhet be, ha a bevallott élek között néhány kört alkot. Ezt a lehetőséget a stratégia kizárja. Tehát az a helyzet, hogy az utolsó éltől függetlenül már biztosan körmentes a gráf. Ez csak úgy lehet, hogy az eddig bevallott élekből álló gráf egy nem összefüggő erdő. Mivel a pontok száma nagyobb mint 2, egynél több olyan rendezetlen $\{u, v\}$ pár van, hogy u és v különböző komponensbe esik. Speciálisan van ezek között olyan $\{u, v\}$ pár, hogy $\{u, v\} \neq \{u', v'\}$. Világos, hogy az (u, v) él nem csinál kört, és nyilván a lekérdezése pillanatában sem csinálhatott. Ezért viszont a kérdésre "igen" kellett hogy legyen a válasz. Ez ellentmondás.

41. Tegyük fel, hogy van olyan algoritmus, ami $\binom{n}{2}$-nél kevesebb lehetséges élet kérdez le. Ekkor persze van olyan is, ami éppen $\binom{n}{2} - 1$ élet kérdez. Ezzel az algoritmussal szemben az ellenség alkalmazza a következő – a piros szabályhoz hasonló – stratégiát: Az "(u, v) él-e" kérdésre legyen a válasz "nem", ha a bevallott élek és a még hátralevő élek tartalmaznak egy feszítőfát, egyébként a válasz legyen az, hogy "igen". A feltételezés szerint $\binom{n}{2} - 1$ kérdés után – függetlenül az $\binom{n}{2}$-edik – (u', v') él létezésétől eldől, hogy a gráf összefüggő-e vagy sem. A nem összefüggőség a stratégia miatt nem fordulhat elő, hiszen a bevallott élek a hátralevőkkel együtt mindig tartalmaznak egy feszítőfát. Az összefüggőség bizonyítványa az, hogy a bevallott élek tartalmaznak egy feszítőfát. Ekkor az (u', v') éllel együtt keletkezik egy kör, aminek legyen egy bevallott éle (u, v). Az (u, v) élet a lekérdezésekor le lehetett volna tagadni, hiszen nélküle is létezik feszítőfa. Ez is ellentmond a stratégiának.

44. Az éllisták végignézésével fokszám szerint osztályozzuk a csúcsokat $O(n)$ időben! Az első osztályba azok tartozzanak, amelyeknek kevesebb mint $n/2$ szomszédjuk van, a másodikba pedig azok, amelyeknek a foka legalább $n/2$. A komplementer gráfban az első típusú csúcsok mindegyikének több szomszédja van mint $n/2 - 1$, így az összefüggő komponensük mérete nagyobb, mint $n/2$. Ilyenből nem fér el egynél több, tehát az első osztályba eső csúcsok mind egy komponensben vannak. Egyszerű számolás mutatja, hogy a második osztályba legfeljebb 10 csúcs tartozhat. Az első osztályt összehúzva egy ponttá a feladatot egy legfeljebb 11 pontú gráf összefüggőségének a vizsgálatára vezethetjük vissza.

45. Figyeljük meg, hogy a piros-kék algoritmus helyességének a bizonyításában a költség egyetlen helyen került említésre: amikor azt láttuk be, hogy egy F minimális költségű feszítőfa és egy $g = (u, v)$ él esetén az F-beli $u \rightsquigarrow v$ út egy tetszőleges olyan g' élét, amire $c(g) \leq c(g')$ g-re cserélve ismét minimális költségű feszítőfát kapunk (a fákról szóló Állítás 5. pontja). Minden további érvelés ezt az észrevételt használta. Márpedig ezen tény nyilvánvalóan érvényes a feladatban előírt költségre is.
Másik bizonyítás: Legyen G a gráf, n a csúcsok száma. Alkalmas eltolással nyilván feltehető, hogy az élsúlyok mind nemnegatívak. Minden p természetes számra legyen egy F feszítőfa L_p-költsége $L_p(F) := \sum_{f \in F} c(f)^p)^{\frac{1}{p}}$. Legyen továbbá $L_\infty(F) := \max_{f \in F} c(f)$. Az F és F' feszítőfákra nyilván $L_p(F) \leq L_p(F') \Leftrightarrow \sum_{f \in F} c(f)^p \leq \sum_{f' \in F'} c(f')^p$. Tehát F akkor és csak akkor lesz olyan feszítőfa, amire $L_p(F)$ minimális, ha F hagyományos értelemben minimális költségű feszítőfa arra az élsúlyozásra, ahol az e él súlya $c(e)^p$. Ezért a piros-kék algortimus korrekt az L_p-ben minimális költségű feszítőfa keresésére. Tudjuk, hogy tetszőleges $a_1, \ldots, a_{n-1}$ nemnegatív számokra $\lim_{p \to \infty} (\sum_{i=1}^{n-1} a_i^p)^{\frac{1}{p}} = \max_{i=1}^{n} a_i$. Ez alapján G-hez választható olyan p, amire G-nek az L_p-ben minimális költségű feszítőfái egyben L_∞-ben minimális költségű feszítőfák is. Mivel az adott p-re a piros-kék algoritmus L_p értelemben korrekt, az lesz L_∞-ben is. (A megfelelő p létezéséhez: Legyen ϵ a lehető legkisebb olyan pozitív szám, ami fellép mint két különböző G-beli élsúly különbsége. (Ha nincs ilyen, azaz minden él azonos súlyú, akkor legyen $\epsilon = 100$. Ekkor ugyanis minden feszítőfa optimális mind L_∞ mind L_p értelemben tetszőleges p-re.) G minden egyes feszítőfájára létezik olyan N_F természetes szám, hogy $p \geq N_F$ esetén $|L_p(F) - L_\infty(F)| < \epsilon/2$. Legyen $p = \max_F N_F$. Ekkor $L_\infty(F') < L_\infty(F) \Rightarrow L_p(F') < L_p(F)$, tehát ha $L_p(F)$ minimális, akkor $L_\infty(F)$ is az. Vigyázat, ha több L_∞-ben minimális költségű feszítőfa van, általában nem lesz mindegyik L_p-ben minimális.)

46. A Borůvka-módszer első két menetét hajtjuk végre kellő körültekintéssel. Az első menetben minden csúcsból a legkisebb kimenő élet kell választani. Az összköltség a fokszámok összegével arányos, ami $O(e)$. A második menetben minden kék fára kell megtenni ugyanezt. A kék fákat bejárva meg tudjuk címkézni a csúcsokat a fák sorszáma szerint, majd minden egyes kék fát megint bejárva el tudjuk készíteni a fából kiinduló élek listáját és a minimumot is ki tudjuk választani fánként $O(\sum_{v \in \text{fa}} \text{fokszám}(v))$ időben. Az összköltség $O(\sum_{v \in V} \text{fokszám}(v)) = O(2e) = O(e)$. Ha $n > 4$, akkor az első menet után minden kék fa legalább 2 pontú, míg a második menet után már legalább 4 pontú. Ezért a kék élek száma legalább $3/4n$.

47. Legyen először $V_1 = V_2 = \emptyset$. Vegyük sorra G csúcsait. Amikor v sorra kerül, tegyük V_1 illetve V_2 közül abba az osztályba, ahol eddig a kevesebb szomszédja van. Az n szerinti indukcióval egyszerűen látható, hogy ez az

eljárás jó lesz.

48. Legyenek a csúcsok s, t, a, b, c. Az élek pl. a következők: $(s,t), (s,a), (s,b), (a,t), (b,c), (c,t)$, mindegyik kapacitása 1. A 0 folyamból kiindulva az első lépésben az (s,t) éllel javítunk. A javítógráfban nem marad 1 hosszú út, most egy kettő hosszú következik $(s,a), (a,t)$. Az újabb javítógráfban már csak 3 hosszú út van s-ből t-be, ez lesz a harmadik lépés.

49. Vegyük fel az $u_1, \ldots, u_n$, valamint a $v_1, \ldots, v_n$ pontokat, valamint egy s forrást és egy t nyelőt. Vezessünk u_i-ből v_j-be egy 1 kapacitású élet, ha $A[i,j] \neq 0$. Az s forrásból vezessünk minden egyes u_i-be olyan kapacitású élet, amennyi az i-edik sorösszeg. Hasonlóan, a (v_j, t) él kapacitása legyen a j-edik oszlop összege. Ez egy csupa egész kapacitású hálózat, tehát létezik egy egész értékű maximális folyam, továbbá a Ford–Fulkerson algoritmus (pl. az Edmonds–Karp növelő heurisztikával polinom időben) egy ilyet talál. A kapacitások miatt az (u_i, v_j) éleken a folyamérték tehát 0 vagy 1. Legyen $B[i,j]$ ez a folyamérték. Ha (u_i, v_j) nem volt él, akkor természetesen legyen $B[i,j] = 0$. Azt állítjuk, hogy az így kapott B mátrix jó lesz. Ehhez határozzuk meg a maximális folyamértéket. Legyen x az A mátrix elemeinek az összege. Mivel ez a sorösszegek összege, (és egyben az oszlopösszegek összege), nem lehet x-nél nagyobb folyam (vannak ekkora vágások: a forrásból induló élek, vagy a nyelőbe menők). De ekkora folyam létezik is: telítsük a forrás illetve a nyelő éleit és legyen a folyam értéke az (u_i, v_j) éleken $A[i,j]$. Mindebből az következik, hogy a talált folyam értéke is x. Ez csak úgy lehet, ha a forrásból induló élek és a nyelőbe menők is mind telítettek, ami éppen a sor- illetve oszlopösszeg feltételekkel egyenértékű.

8. Turing gépek

2. Legyen M'' az a gép, amely először szimulálja M'-t, majd M' outputján szimulálja M et (M inputja legyen M' outputja, M'' outputja legyen M outputjával azonos. Tetszőleges $x \in I^*$ input szó esetén az M' megáll és a $g(x)$ függvényétéket adja, a $g(x)$ inputtal elindítva pedig M megáll és az $f(g(x))$ értéket számítja ki.

5. Tekintsük azt az algoritmust, ami adott n-re $[\sqrt{n}]$-ig végignézi a számokat, hogy azok között van-e valódi osztója n-nek. Az algoritmus Turing-géppel megvalósítható, és minden n inputra megáll (legfeljebb $[\sqrt{n}]$ osztás után).

6. A rekurzivitásra vonatkozó állítások könnyűek: Legyen M_1 illetve M_2 olyan Turing-gép, mely minden inputra megáll, és az L_1 illetve L_2 nyelvet ismerik fel ($L_{M_1} = L_1, L_{M_2} = L_2$). Az $L_1 \cup L_2$ nyelv felismerésére tekintsük a következő M gépet: az $x \in I^*$ inputra először szimulálja M_1-et, és álljon meg elfogadó állapotban, ha M_1 elfogadta x-et. Egyébként futtassa le x-re M_2-t is, és aszerint fogadja el x-et végül, hogy M_2 elfogadta-e x-et vagy sem. Világos, hogy M mindig megáll és $L_M = L_1 \cup L_2$. Hasonló módon igazolható $L_1 \cap L_2$ rekurzivitása. Az L nyelv felismeréséhez egy ciklust kell szervezni, melyben a z input szó összes lehetséges xy szétvágására lefuttajuk M_1-et x-re, M_2-t pedig y-ra.
A rekurzív felsorolhatóságokhoz megintcsak legyenek legyenek M_1 és M_2 olyan Turing-gépek, hogy $L_{M_1} = L_1$ és $L_{M_2} = L_2$. Most sajnos nem tételezhetjük fel, hogy ezek minden input esetén megállnak. Az $L_1 \cap L_2$ nyelv felismerése mégis végezhető az előző módszerrel, hiszen ha x-re M_1 nem áll meg, akkor $x \notin L_1 \Rightarrow x \notin L_1 \cap L_2$, tehát nem baj, hogy az összetett gép "végtelen ciklusba" esik. A hasonló megközelítés nyilván nem működik $L_1 \cup L_2$-re, hiszen M_2-nek is meg kell adni a lehetőséget x felismerésére. A megoldás M_1 és M_2 "párhuzamos" futtatása: szimuláljuk felváltva M_1 és M_2 lépéseit! Értelemszerűen, ha valamelyik megáll, csak a másikat futtassuk tovább.
Az L nyelv rekurzív felsorolhatóságának igazolásához ezt az ötletet finomíthatjuk. Egy n szerinti "külső" ciklusban futtasunk egy "belső" ciklust, amely a z input összes lehetséges $z = xy$ szétvágására lefuttatja M_1 illetve M_2 első n lépését az x illetve y inputra. Akkor állunk meg elfogadó állapotban, ha valamely n-re létezik olyan $z = xy$ feldarabolás, hogy M_1 x-et, M_2 pedig y-t elfogadja legfeljebb n lépésben. Egyébként végtelen ciklusba kerülünk. Nyilvánvaló, hogy $z \in L \Leftrightarrow$ létezik megfelelő $z = xy$ felosztás és alkalmas n, tehát éppen L-et ismerjük fel.

8. Nem. Legyen $L_1 = I^*$ és L_2 egy rekurzíve felsorolható, de nem rekurzív nyelv (pl. L_h). Ekkor $L_1 \setminus L_2 = \overline{L_2}$ nem rekurzíve felsorolható (ld. a $\mathcal{RE} \cap \text{co}\mathcal{RE} = \mathcal{E}$ összefüggést!)

9. a) Igen. Ha L_1, L_2 rekurzív, akkor komplementerük, metszetük, uniójuk is rekurzív, így $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$ is és $L_1 \oplus L_2$ is.
b) Nem. Legyen $L_1 = I^*$ és L_2 egy rekurzíve felsorolható, de nem rekurzív nyelv. Ekkor $L_1 \oplus L_2 = \overline{L_2}$, ami nem rekurzíve felsorolható.

10. Igen. Adott $x \in I^*$ szóra a következő lehetőségek vannak:
(o) M_x nem létezik;
(i) M_x valamikor megsérti a feltételt: ráír a szalagjára vagy túlfut az üresjelen;
(ii) M_x megáll a feltételek betartásával;
(iii) M_x végtelen ciklusba esik a feltételek betartásával.
A (o), (i) és (ii) eseteket könnyű felismerni M_x szimulációjával. Az (iii) eset felismerésére figyeljük a pillanatnyi helyzetleírás (PHL) ismétlődését! M_x ugyanis pontosan akkor esik végtelen ciklusba, ha egy pillanatnyi helyzet ismétlődik. A feltételek betartása esetén a szalag állapota állandó, M_x feje a szalag első $|y|+1$ cellájának valamelyikén helyezkedik el. Tehát a pillanatnyi helyzet egy pozíció–belső állapot párral jellemezhető, ahol a pozíció nem haladja meg $|y|+1$-et. Így legfeljebb $(|y|+1) \cdot$ belső állapotok száma lépésben vagy az (i),(ii) események valamelyike bekövetkezik, vagy pedig ismétlődik egy pillanatnyi helyzet, az (iii) estenek megfeleően. Utóbbi a pillanatnyi helyzetek tárolása segítségével (vagy a tár-idő tétel bizonyításához hasonló technikával) ismerhető fel.

11. Az előző feladathoz képest az a nehézség, hogy az olvasófej tetszőlegesen távol lehet az első üresjeltől. Azt vegyük azonban észre, hogy ha egyszer ez a távolság nagyobb mint $|Q|$ (a belső állapotok száma), akkor olyan "végtelen ciklusba" esett M_x, amikor az olvasófeje a végtelenbe szalad ki. Legyen ugyanis az l-edik lépés az az első olyan lépés, amikor M_x feje először túllépi a mondott távolságot! Ekkor van olyan $k < l$ lépésszám, hogy a k-adik lépésben M_x belső állapota ugyanaz, mint az l-edik lépésben, M_x feje az input végéhez közelebb van, továbbá a k-adik, $k+1$-edik, stb. l-edik lépések mindegyikében a fej az input utáni üres szalagrészen mozog. Látható, hogy az l-edik lépés és a következő $l-k$ lépésben M_x helyzete lényegében ugyanaz, mint a k-adik és az azt követő lépésben, a különbség csak annyi, hogy a feje az üres szalagrészen még jobbra eltolt pozíciókban mozog. Világos, hogy ugyanez fog ismétlődni a végtelenségig. Tehát a következő eseteket lehet megkülönbözteni:
(o) M_x nem létezik;
(i) M_x megsérti a feltételeket;
(ii) M_x megáll a feltételeket végig betartva;
(iii) M_x a feltételek betartásával túllép a $2|y|+1$-edik cellán;
(iv) M_x a feltételeket betartja végtelen ciklusba esik úgy, hogy feje mindig az első $|y|+|Q|+1$ cella valamelyikén van.
Ezen esetek megkülönböztetése immár az előző feladathoz hasonlóan végezhető.

12. Legyen $x \in I^*$ a jegyzetben levő $n \mapsto n+1$ függvényt kiszámító TG kódja. Ekkor az $L = \{x\#1^n | n \in Z^+\}$ nyelv rekurzív és $L \subseteq L_h$.

16. Vegyük észre, hogy öt lépésben az input szalagon legfőljebb ötöt haladhatunk előre. Tehát ha van alkalmas y szó, akkor annak első (legfeljebb) 5 betűje is megteszi. Ezért a probléma eldöntéséhez elegendő M_x első 5 lépését végigpróbálni a legfeljebb ötbetűs szavak véges halmazán. Tehát a nyelv rekurzív.

19. Igaz. Legyen M egy olyan Turing-gép, amely az $x\#f(x)$ párokat ismeri fel (azaz $L_M = \{x\#f(x),\ \ x \in \{0,1\}^*\}$). Célunk egy olyan M' gép konstruálása, amely az f függvényt számítja ki. Az alapötlet adott x-hez sorra kipróbálni az $y \in \{0,1\}^*$ szavakra, hogy $x\#y \in L_M$ teljesül-e. Nyilvánvaló, hogy $x\#y \in L_M \Leftrightarrow y = f(x)$. A kipróbálás történhet a $\{0,1\}^*$ halmaz valamely rendezése (pl. első az üres szó, majd a többi bináris számként értelmezve nagyság szerint) alapján. Ez így sajnos nem működik, hiszen előfordulhat, hogy valamely, az $f(x)$ értéket megelőző y-ra M nem áll meg. A probléma a következőképpen kezelhető: generáljuk sorra az (n,y) párokat, ahol n egy nemnegatív egész és $y \in \{0,1\}^*$. Ez megtehető például $\{0,1\}^*$ fent említett rendezése építve a szokásos párgenerátor segítségével. Adott (n,y) párra futtasuk le M-nek első (legfőljebb) n lépését az $x\#y$ inputon. Ha n lépésen belül M nem áll meg elfogadó állapotban, vegyük a következő párt, egyébként pedig írjuk ki y-t és álljunk meg. Világos, hogy a fent vázolt algoritmust megvalósító Turing-gép minden x-re megáll (amikor az $(n, f(x))$ pár kerül sorra, ahol az n szám M-nek az $x\#f(x)$ páron vett lépéseinek a száma) és éppen $f(x)$-et számítja ki.

23. Legyen $f(x) := 2h(x)$, ahol h egy tetszőleges nem rekurzív $I^* \to N$ (teljes) függvény (pl. $h(x) = C(x)$, az x szó Kolmogorov-bonyolultsága). Ekkor a g függvény azonosan 1, tehát rekurzív.

25. (a) Legyen M az a Turing-gép, melynek legális inputja 21 binárisan ábrázolt természtes számból áll: $n\#n_1\#\ldots\#n_{20}$, ahol $n_1, \ldots, n_{20}$ húsz csupa különböző szám az $[1,n]$ intervallumból; és M írjon az output szalagjára először n darab 0-t, majd az $n_1, \ldots, n_{20}$-adik helyeket írja át 1-esre. A $\{0,1,\#\}$ abc alkalmas bináris kódolásával $C_M(x) \leq 42\log_2 n + 20$ és az invariancia-tétel miatt $C(x) \leq C_M(x) + c_1 \leq 42\log_2 n + 20 + c_1$, ahol c_1 egy állandó. Alkalmas c állandóra ez kisebb mint $c\log n$.
(b) Tegyük fel, hogy mégis van ilyen M Turing-gép. Ekkor $C_M(x) = n$ hossza $+ c_1 = [\log_2 n + 1] + c_1$, így az invariancia-tétel miatt $C(x) \leq \log_2 n + c_2$ alkalmas c_2 állandóval, ami nem lehet nagyobb $2\log n$-nél ha n elég nagy. Ez ellentmondás.
(c) A 0-val kezdődő n hosszú szavak száma 2^{n-1}, ugyanakkor az $n-1$-nél kisebb Kolmogorov-bonyolultságú szavak száma (a lehetséges inputhossz szerint leszámolva) legfőljebb $1+2+4+\ldots+2^{n-2} = 2^{n-1}-1$.
(d) Adott k-ra jelölje 1^k az k darab 1-esből álló szót. Figyeljük meg, hogy $C(1^k) = C(k)$ egy additív állandó erejéig. Legyen k egy olyan n bináris jegyű szám, amire $C(k) \geq n$. Ilyen az előző részfeladat értelmében létezik. Legyen $x = 1^k$ és $y = 1^{2^{n+1}-k}$. Ekkor $C(x) \geq n + c_1$ és $C(xy) = C(1^{2^{n+1}}) \leq \log_2 n + c_2$ alkalmas c_1, c_2 állandókkal. Válasszuk n-et olyan nagynak, hogy $n + c_1 > \log_2 n + c_2$ teljesüljön!

26. Lásd az előző feladat (a) pontját.

27. Lásd az előző feladat (c) pontjánál alkalmazott ötletet.

29. Tegyük fel indirekte, hogy létezik ilyen f rekurzív függvény! Eszerint létezik egy M Turing-gép, ami minden x-re megáll és kiszámít egy olyan $f(x)$ számot, amelyre $\frac{1}{2}C(x) \le f(x) \le 2C(x)$. Legyen M' az a Turing-gép, amely az n input esetén kiírja azt az első x szót, amire $f(x) \ge [n/2]$ (pl. lexikografikus sorrendben sorra generálva az n hosszú x szavakat M-et használva kiszámítja $f(x)$-et). Mivel van összenyomhatatlan szó, ilyen x tényleg létezik. Nyilvánvaló, hogy $C_{M'}(x) \le \log_2 n + c'$ alkalmas c' állandóval. Az invariancia-tétel miatt $C(x) \le \log_2 n + c$ egy másik c állandóval. Ugyanakkor a feltevésünk miatt $C(x) \ge f(x)/2 \ge (n-1)/4$. A két egyenlőtlenséget összeolvasva kapjuk, hogy $\log_2 n + c \ge (n-1)/4$, ami képtelenség elég nagy n-re. Ellentmondásra jutottunk. (Lényegében a $C(x)$ kiszámíthatatlanságát igazoló jegyzetbeli érvelést használtuk.)

33. Nem. Legyen ugyanis $f(n)$ a kanonikus rendezés szerinti első n hosszú összenyomhatatlan szó. Ha H rekurzív, akkor f is, hiszen ha egymás után sorban generáljuk az n hosszú szavakat és mindegyikre meghatározzuk a H értékét, akkor amelyik x utáni y szónál a H érték változik, az az x összenyomhatatlan. Viszont, ha $f(n) = x$, akkor az invariancia-tétel szerint $C(x) \le \log_2(n+1) +$ állandó, ami ellentmond x összenyomhatatlanságának.

35. Nem. Van olyan algoritmus (Turing-gép), amely az n bemenetre kiadja az $x_0x_1 \dots x_n$ szót: generálja az $F_0, F_1, \dots, F_j$ Fibonacci-számokat, amíg túl nem jut az n-en, és közben sorra kiírja az x_i biteket: $x_0 = 1$, és ezután $x_i = 0$, ha $F_{j-1} < i < F_j$, és $x_{F_j} = 1$. Ezek szerint $C(x_n) \le \log n + c$, azaz $\lim \frac{C(x_n)}{n} \ne 1$.

36. Nem. Az $y_0y_1 \dots y_{2n-1}$ részsorozat előállítható a páros indexű bitek duplázásával, így $C(y_0y_1 \dots y_{2n-1}) \le n + c$ alkalmas c állandóval, és $\frac{1}{2n}C(y_0y_1 \dots y_{2n-1}) \le \frac{1}{2} + c/n < \frac{3}{4}$ minden eléggé nagy n-re.

37. Az $x_1x_2 \dots x_n$ szó megkapható az $y_1y_2 \dots y_n$ szóból a "hiányzó" bitek, azaz az $x_1x_2x_4 \dots x_{2^{\lfloor \log_2 n \rfloor}}$ sorozat hozzáadásával. Ebből azt nyerjük, hogy $C(x_1x_2 \dots x_n) \le C(y_1y_2 \dots y_n) + c + \log_2 n$, ahol c egy alkalmas állandó. (Legyen $Z_n \in \{0,1\}^*$ olyan szó, amire $|Z_n| = C(y_1 \dots y_n)$ és az U univerzális Turing-gép Z_n input esetén az $y_1 \dots y_n$ szót számolja ki! Legyen M az a gép, ami a kétrészes $Z\#W$ inputon először Z-re lefuttatja U-t, majd az eredmény 2-hatvány indexű bitjeit átírja W bitjeivel. Ekkor $C_M(x_1 \dots x_n) \le C(y_1 \dots y_n) + \log_2 n$ egy additív konstans erejéig. Az invariancia-tételt alkalmazva nyerjük az állítást.) Így $1 + \frac{\text{állandó}}{n} \ge \frac{1}{n}C(y_1 \dots y_n) \ge \frac{1}{n}C(x_1 \dots x_n) - \frac{c+\log n}{n}$. A mindkét szélső oldal határértéke 1, ezért a beszorított mennyiség is konvergens és 1-hez tart.

38. Tetszőleges M Turing-gépre a CT_M függvény rekurzív: adott $x \in I^*$ szóra elég a legfőljebb $|x|^2$ hosszú $y \in I^*$ bemenetekkel futtatni M-et $\le |x|^2$ lépésig. Tudva, hogy CT_M rekurzív, $CT_M = C$ lehetetlen, mert C nem rekurzív.

41. Az input párok legyenek $(a^{u_1}, a^{v_1}), \dots, (a^{u_n}, a^{v_n})$, ahol a^u azt a szót jelöli, amely u darab a betűből áll. A megfeleltetési feladat az adott bemenettel nyilván pontosan akkor oldható meg, ha léteznek olyan $x_1, \dots, x_n$ nem mind 0 nemnegatív egész számok, melyekre $x_1u_1 + \dots + x_nu_n = x_1v_1 + \dots, x_nv_n$, azaz $x_1(u_1 - v_1) + \dots, x_n(u_n - v_n) = 0$. Ha az $u_i - v_i$ különbségek mind pozitívak vagy mind negatívak, akkor ez nyilván nem oldható meg. Ellenkező esetben két lehetőség van. Vagy valamelyik i indexre $u_i = v_i$: ekkor egy lehetséges megoldás az, hogy $x_i = 1$ és $x_k = 0$ $k \ne i$ esetén. Vagy pedig van pozitív és negatív különbség is: $u_i > v_i$ és $u_j < v_j$ valamely i, j indexekre. Ebben az esetben $x_i = (v_j - u_j)$, $x_j = (u_i - v_i)$, $x_k = 0$ $k \notin \{i, j\}$-re egy megoldást ad. Tehát azt kell ellenőrizni, hogy a különbségek mind azonos előjelűek-e. Ez nyilván megtehető akár lineáris időben is.

9. Bonyolultsági osztályok

1. Legyen l az L-beli szavak max. hossza. Az l-nél hosszabb input szavakat eleve elutasítva, a max. l hosszú szavak esetén pedig keresést alkalmazva L valójában konstans időben felismerhető.

2. Nem: az $m-1$ a bemenet hosszához (kb. $\log_2 n + \log_2 m$) képest exponenciálisan nagy is lehet.

3. Nem. Ugyanis $\binom{n}{\lfloor \log^2 n \rfloor}$ lehetőséget vizsgál meg a módszer, tehát legalább ennyi a lépésszám. Ugyanakkor bármely c állandóra $\binom{n}{\lfloor \log^2 n \rfloor}/n^c \to \infty$. Ez utóbbi állítás például a következő elemi becslés felhasználásával igazolható: $\binom{n}{\lfloor \log^2 n \rfloor} = \frac{n}{\lfloor \log^2 n \rfloor} \cdots \frac{n - \lfloor \log^2 n \rfloor + 1}{1} \ge (\frac{n - \lfloor \log^2 n \rfloor}{\lfloor \log^2 n \rfloor})^{\log^2 n - 1} \ge n^{\frac{1}{2}(\log^2 n - 1)}$, ha n elegendően nagy.

4. Az alapműveletkre ismert algoritmusok polinomidejűek. A hatványozás esetén azonban pl. a 2^n szám mérete n, ami exponenciális az operandusok $O(\log_2 n)$ méretében.

5. Az $x^n \bmod m$ feladat mérete $\theta(\log x + \log n + \log m)$. Az x^n hatványozás elvégezhető polinom sok szorzás segítségével a gyors hatványozás (iterált négyzetreemelés) módszerével: Ha $n = \sum_{i=0}^{k} a_i 2^i$ alakú ($k \le \log_2 n$, $a_i \in \{0,1\}$), akkor kiszámítva x^1-t, x^2-t, …, x^{2^k}-t összesen k négyzetreemeléssel, majd azokat az x^{2^i} hatványokat (max. k szorzással) összeszorozva, amelyekre $a_i = 1$, megkapjuk x^n-et, max. $2k$ szorzás felhasználásával. Mivel a szorzásokat modulo m kell elvégezni, sem a részeredmények, sem a végeredmény mérete nem haladja meg a $2\log_2 m$ korlátot. Egy szorzás (két m-nél nem nagyobb számnak az összeszorzása és a szorzat maradékos osztása m-mel) elvégezhető $\log m$-ben polinomidőben.

6. Ha $y_1, y_2, \dots, y_k$ az NP definíciójának megfelelő (polinomméretű és polinomidőben ellenőrizhető) tanúk arra, hogy $x_1, x_2, \dots, x_k \in L$, akkor a k szám, az elvágási poziciók sorozata, valamint az $y_1, y_2, \dots, y_k$ tanúk sorozata együttesen egy megfelelő tanút alkotnak $x \in L$-re.

9. A tár-idő tétel alapján létezik olyan c konstans, hogy $L \in TIME(c^{3\log_2 n}) = TIME(2^{\log_2 c\, 3\log_2 n}) = TIME(n^{3\log_2 c}) \subseteq P$.

10. Legyen M az a gép, mely x input esetén kiszámítja $T(|x|)$-et, majd szimulálja M_x-nek az első $T(|x|)$ lépését az x inputon. M mindig megáll és a segítségével el lehet már dönteni az L-be tartozás problémáját.
Tegyük fel, hogy M egy olyan TG, amely L-et $T(n)$ időben felismeri. Legyen $M = M_x$ valamely $x \in I^*$ szóra. Futtasssuk M-et az x inputon. A feltevés miatt az $x \in L$ esetben $M = M_x$ max. $T(|x|)$ lépésben elfogadja x-et, ellentmondva L definíciójának. Az $x \notin L$ esetben viszont L definíciója szerint $M = M_x$ elfogadja x-et, amit nem tehet, hiszen az L nyelvet ismeri fel. Ellentmondás.

11. Bináris kereséssel $O(\log n)$ lépésben, M-et használva keressük meg azt a legkisebb k számot, melyre $n | k!$. Ekkor $k = n$ esetén n prímszám. Különben $1 \ne \text{lnko}(k,n) \ne n$, egy valódi osztója n-nek. Az euklideszi algoritmussal ($O(\log n)$ aritmetikai művelet) számítsuk ki tehát $\text{lnko}(k,n)$-et, majd alkalmazzuk rekurzíve az eddigi eljárást $\text{lnko}(k,n)$-re, valamint $n/\text{lnko}(k,n)$-re. Nyilván összesen max. annyiszor kell ezt az eljárást meghívni, ahány prímtényezője van n-nek, ez pedig $O(\log n)$. Összesen tehát $O(\log^2 n)$-szer van szükség M hívására, valamint $O(\log^2 n)$ aritmetikai műveletet végzünk el $O(\log n)$ méretű számokon.

14. Vegyük sorra a G gráf $v \in V$ csúcsait. Ha a v csúcs és a hozzá illeszkedő élek elhagyásával nyert gráfban a max. klikkméret kisebb, mint az eredeti gráfban (ezt P segítségével állapíthatjuk meg), akkor v a G gráf minden max. méretű klikkjében benne van. Különben hagyjuk el v-t, és ismételjük az eljárást. Ha már nincs elhagyható csúcs, kaptunk egy max. méretű klikket. Max. $|V|$ iterációra (és P-hívásra) van szükség.

17. Egy megoldás lesz a tanú. Mivel az f-nek egy egész gyöke osztója az a_0 konstans tagnak, ezért a mérete (az abszolút értékének a számjegyeinek a száma) legfeljebb a_0 mérete lehet, tehát polinomméretű, és az f-be való behelyettesítés polinomidőben el is végezhető.

18. Generáljuk valamilyen sorrendben az $\{1, \dots, m\}$ halmaz részhalmazait, és számoljuk meg azokat, amelyekre a fenti összegezési feltétel teljesül. Nyilvánvaló, hogy egy rögzített részhalmazra a feltétel ellenőrzése polinom tárban elvégezhető. A végigpróbálás tárigénye lényegében az egyes próbák tárigényének a maximuma.

25. A következő észrevétel alapján: Ha G összefüggő, akkor G színezhető két színnel $\iff$ G-nek egy tetszőleges pontjából kiinduló utak mentén a páros-páratlan színezés egy jó színezés.

26. A G éppen akkor van L-ben, ha el tudunk belőle hagyni legfeljebb 2 élet úgy, hogy a kapott gráf nem összefüggő. Ez minden E-beli párra lineáris időben vizsgálható (pl. mélységi bejárással).

27. Nézzük végig G minden egyes élére a végpontokat tartalmazó max. klikkek méretét, mégpedig egyszerűen úgy, hogy megvizsgáljuk a végpontok szomszédainak az összes részhalmazát, hogy klikket alkotnak-e. A feltevés miatt ezen halmazok száma élenként legfeljebb $2^{2\log_2 |V(G)|} = |V(G)|^2$, így összesen $O(|E(G)||V(G)|^2)$ esetet kell megvizsgálni, és a legkedvezőbbet (a legnagyobbat, ami klikket alkot) kiválasztani.

28. Induljunk ki egy tetszőleges csúcsból, és dobjuk ki a szomszédait. A maradékból válasszunk egy tetszőleges csúcsot, és dobjuk ki annak a szomszédait is, és így tovább. Mivel egy csúcs bevételével max. 3 másik csúcsot dobunk ki, végül egy legalább $|V(G)|/4 \ge k/4$ méretű független ponthalmazt kapunk.

29. Válasszunk egy tovább nem bővíthető független élrendszert (párosítást). Ez például a tankönyvben a sok független élet kereső mohó módszerrel az élek számával arányos lépésben megtehető. Tegyük fel, hogy k darab független élet kaptunk. Az élrendszer végpontjai ($2k$ ilyen van) nyilván lefogják az összes élet, valamint az is igaz lesz, a gráf összes élét nem lehet k-nál kevesebb csúccsal lefogni, hiszen már a kiválasztott élrendszerünket sem lehet.

30. a) A Hamilton-kör n élet tartalmaz az $e \le n+10$ közül. Az összes lehetséges n élű részgráf száma $\binom{e}{n} \le \binom{n+10}{n} = \binom{n+10}{10} \le n^{10}$, azaz polinomiális. Egy ilyen részgráfról polinom lépésben ellenőrizhető, hogy Hamilton-kört alkot vagy sem. Ha mindegyiket végigpróbáljuk az is polinom lépés lesz.

b) Ha van elsőfokú pontja a gráfnak, akkor nincs Hamilton-kör. A másodfokú pontokat elhagyhatjuk közvetlenül összekötve a két szomszédot. Egy ilyen lépés után mind az élek, mind a csúcsok száma 1-gyel csökken, így megmarad az $e \le n + 10$ egyenlőtlenség. Az eredmény egy olyan n' pontú G' gráf, amelyben $e' \le n' + 10$ és minden pont foka legalább 3. Ez viszont azt jelenti, hogy $e' \ge 3n'/2$, amiből $n' \le 20$. Erről a konstans méretű G'-ről kell eldönteni, van-e benne Hamilton-kör. Ez pedig konstans lépésben megoldható. A redukció lineáris lépést igényel, összesen tehát $O(n)$ az idő.

33. (a) $\notin FP$: Már az output leírása is $\Omega(\log n!) = \Omega(n \log n)$ időbe kerül, ami exponenciális az input $O(\log n)$ méretében.
(b) nem tudjuk (egyenértékű a törzsténeyezős felbontás problémájával); azt sejtik, hogy nem FP-beli.
(c) $\in FP$.

35. Nem. Legyen pl. $f(x) = 1^{2^{|x|}}$, tehát f az x szóhoz a $2^{|x|}$ darab egyesből álló szót rendeli. Ha adott $x\#y$, akkor meg kell nézni, hogy y csak 1-est tartalmaz-e. (Ez $|x| + |y|$ lépés.) Ha igen, akkor a hosszát is ellenőrizzük 2 szalag segítségével: az egyiken egyetlen 1-esből kiindulva mindig megduplázzuk az 1-ek számát, a másik szalagon számláljuk, hányszor dupláztunk eddig. Ha valamikor az 1-ek sorozata hosszabb, mint y, abbahagyjuk, ellenkező esetben addig folytatjuk, míg $|x|$ számú duplázás nem történt. A k hosszú sorozat duplázása $O(k^2)$ lépés. Mivel $k < |y|$ ez legfeljebb $O(|x||y|^2)$ lépést igényel, ami a bemenet hosszának polinomja.

Másrészt világos, hogy $f(x)$ nem számolható ki $|x|$-ben polinomiálisan, hiszen már csak az eredmény kiírásához is szükséges $2^{|x|}$ lépés.

40. Az világos, hogy $RH' \in$ NP, hisz egy megfelelő I részhalmaz jó lesz tanunak. A teljességhez megmutajuk hogyan lehet az RH nyelvet Karp-redukálni az RH' nyelvre. Legyen az f a következő leképezés $f : (s_1, \ldots, s_m; b) \mapsto (2s_1, \ldots, 2s_m; 2b)$. Világos, hogy ez polinomidőben számolható. Továbbá $\sum_I s_i = b$ akkor és csak akkor, ha $\sum_I 2s_i = 2b$.

42. I. *ágyúval verébre*
Az $s \geq 1/3$ súlyokból legfeljebb kettő kerülhet egy ládába. Készítsünk el egy gráfot, melynek csúcsai a súlyoknak felelnek meg. Közülük kettő akkor legyen összekötve, ha összegük ≤ 1. A pakolások megfelelnek a gráfbeli párosításoknak. Akkor használunk minimális számú ládát, ha a megfelelő párosítás maximális. Maximális párosítást gráfokban (nem csak párosokban) lehet polinomidőben találni, így a feladat is megoldható polinom lépésben. (Ennél azért jobbat is lehet mondani, ha kihasználjuk a gráf sajátosságait. Az hogy hogyan, látszik majd a következő megoldásból.)

II. *Gondolkozzunk*
Rendezzük a súlyokat csökkenő sorrendbe, $s_1 \geq \ldots \geq s_n$. Legyen $i = 1$, $j = n$. Ha $s_i + s_j \leq 1$, rakjuk ezeket egy ládába, i-t növeljük, j-t csökkentsük eggyel. Ha $s_i + s_j > 1$, az i. súly egyedül lesz egy ládában, növeljük i-t eggyel. Ha $i \geq j$, az eljárás véget ért, megadtunk egy pakolást (lineáris időben!).

Most belátjuk, hogy ez optimális: elég megmutatni, hogy van olyan optimális pakolás, amelynél az s_1-et tartalmazó ládának ugyanaz a tartalma, mint az általunk javasolt elhelyezésnél. Ebből $-$ s_1-et és esetleg s_n-et elhagyva $-$ kézenfekvő indukcióval következik az állítás.

Ha s_1 a mi módszerünknél egyedül van a ládájában, akkor minden más bepakolásnál is egyedül kell lennie, ekkor tehát készen vagyunk. Ha egy pakolásnál s_1 társa s_j $(1 < j < n)$, akkor világos, hogy s_j és s_n helyet cserélhet anélkül, hogy túllépnénk a korlátot. Legyen ugyanis s_n társa s. Ekkor $1 \geq s_1 + s_j \geq s_1 + s_n$ és $1 \geq s_1 + s_j \geq s + s_j$. Még egyszerűbben tudunk elbánni egy olyan pakolással, amelyben s_1 és/vagy s_n egyedül van.

III. *vizsgáljuk meg a tanultakat*
Az FFD ebben az esetben optimális eredményt ad. Hasonló godolatmenettel, mint az előbb meg lehet mutatni, hogy az FFD-től legkevésbé eltérő optimális megoldás megegyezik vele.

43. Ez a feladat a részhalmazösszeg (RH) probléma kicsi súlyokra, a hátizsák probléma egy speciális esete. A súlyok kicsisége a következőképpen használható ki: Ha $b > m^2$, akkor nincs megoldás. $b \leq m^2$ esetén viszont a jegyzetben ismertetett dinamikus programozási algoritmus logaritmikus költsége $O(bl) = O(m^2 l) = O(l^3)$, ahol l az input hossza (a számadatok hosszának összege).

46. Alkalmazzuk az FFD (First Fit Decreasing) módszert. Ez nyilvánvalóan polinom idejű. Állítjuk, hogy az algoritmus minden egyes lépésében az összes nemüres láda tele lesz, kivéve esetleg az utolsót. Ezt indukcióval igazoljuk. A kezdőeset triviális. Tegyük fel, hogy az első k súly bepakolásánál teljesül az állítás és a $k+1$-edik súly 2^{-j}. Ha minden láda tele van, akkor az indukciós lépés triviális. Tegyük tehát fel, hogy az utolsó ládában még $m > 0$ hely marad. Tudjuk, hogy $m = 1 - 2^{-i_1} - 2^{-i_2} - \ldots - 2^{-i_k}$, ahol $i_l \leq j$ $(l = 1, \ldots, k)$. Ezért $2^j m$ egy pozitív egész szám, és így $m \geq 2^{-j}$, tehát a $k+1$-edik súly befér az utolsó ládába. Az állítás igaz akkor is, amikor az összes súly bepakoltuk. Világos, hogy kevesebb ládába ennyi összsúly nem fér bele.

47. $V(G)$ minden egyes $\lceil \log_2 n \rceil$ elemű részhalmazára $O(\log_2 n)^2$ vizsgálattal megállapítható, hogy független pontrendszert alkot-e. A részhalmazok száma $\binom{n}{\lceil \log_2 n \rceil} = O(n^{\lceil \log_2 n \rceil})$, így egy $n^{O(\log n)} = 2^{O(\log^2 n)} = 1.2^{o(n)}$ költségű algoritmusunk van, tehát a probléma a feltevés miatt nem lehet NP-teljes.

48. $L \in P$: vizsgáljuk végig G összes csúcsára, hogy a szomszédaik között van-e $k-1$ méretű klikk. Egy rögzített csúcsra max. $2^{\log_2 |V(G)|} = |V(G)|$ részhalmazt kell megvizsgálni, hogy $\geq k$ méretű teljes részgráfot alkotnak-e. Mindezt $|V(G)|$ db. csúcsra kell megcsinálni. Tehát L csak akkor lehet NP-teljes, ha $P = NP$.

49. Útmutatás a jegyzetben!

51. Először vizsgáljuk meg azt a feltételt, hogy a hármasok egyáltalán lefedik-e az összes pontot. Ha nem, készen vagyunk. Különben a hármasok legfeljebb $3\log_2 n$ pontot fedhetnek le többszörösen. Az egyszeresen lefedett pontokat tartalmazó hármasokat mindenképpen be kell venni a rendszerbe. Ha ezek a hármasok nem diszjunktak páronként, megint csak készen vagyunk. A fennmaradó hármasokból vegyük azokat, amelyek diszjunktak az eddig bevettektől. Azt kell már csak eldönteni, hogy az így maradt hármasokkal, melyek száma legfeljebb $2\log_2 n$, megvalósítható-e az eddig le nem fedett pontok pontos fedése. Mivel ezekből a hármasokból legfeljebb $2^{2\log_2 n} = n^2$ részhalmaz választható ki, ez megvalósítható max. n^2 eset megvizsgálásával.

54. Egy megfelelő részgráf jó tanúnak, mert polinomiális méretű és annak ellenőrzése, hogy feszített részgráf és két színnel színezhető, polinom lépésben megoldható. Tehát a feladat benne van NP-ben.

A visszavezetés: legyen H egy n csúcsú gráf és $m \in \mathbf{N}$ a $MAXFTLEN$ probléma egy bemenete. A G gráf tartalmazza H-t és még további n csúcsot, melyeket összekötünk H valamennyi pontjával. A feladatbeli k legyen $n+m$. Ha H-ban van m független pont, akkor ezek az új n ponttal együtt egy $m+n$ pontú páros gráfot adnak. A másik irányban pedig, ha G-ben van $n+m$ pontú páros részgráf, akkor ennek pontjai között kell legyen legalább egy az új pontokból, legyen egy ilyen csúcs v és legalább m pont a H-ból. Ha a H-beli m pont között menne él, akkor ez v-vel együtt egy háromszöget adna, ami nem lehet egy két színnel színezhető gráfban.

57. Legyenek $x_1, \ldots, x_m$ Boole-változók. Az $x_i = igaz$ helyettesítés annak feleljen meg, hogy a megfelelő hármas tagja-e a fedésnek. Minden egyes pontnak feleltessünk meg egy olyan formulát, ami azt fejezi ki, hogy a pont pontosan egyszer van lefedve: Ha egy pontot az $i_1, \ldots, i_k$-adik hármasok tartalmazzák, akkor a $(O(k^2)$ méretű) formula:
$(x_{i_1} \wedge \bar{x}_{i_2} \wedge \ldots \wedge \bar{x}_{i_k}) \vee (\bar{x}_{i_1} \wedge x_{i_2} \wedge \ldots \wedge \bar{x}_{i_k}) \vee \ldots \vee (\bar{x}_{i_1} \wedge \bar{x}_{i_2} \wedge \ldots \wedge x_{i_k})$. Képezzük az ilyen formuláknak az $\wedge$-ét. Világos, hogy a formula pontosan akkor elégíthető ki, ha a megfelelő pontos fedés megvalósítható. Látható az is, hogy a formulánknak a mérete (a benne szereplő literálok, illetve műveleti jeleknek a száma) $O(m^2 n)$. A jegyzetben szereplő módszerrel készítsünk egy olyan 3-CNF formulát, ami ugyanakkor elégíthető ki, mint ez a formula. Mivel a konsrtukció a kifejezésfában minden csomóponthoz egy változót és egy 3 változós formulát rendel, a méret $O(m^2 n)$ marad.

58. A probléma nyilván NP-beli. Belátjuk, hogy a Hamilton-kör probléma Karp értelemben redukálható a Hamilton-út problémára. Legyen G egy irányítatlan gráf. Adjunk G-hez még három csúcsot: u, v_0, v_1-et, kössük össze éllel v_0-t v_1-gyel, u-t G egy rögzített w csúcsával, v_1-et pedig w szomszédaival. Egy G-beli Hamilton-kör ből úgy csinálható Hamilton-út az új gráfban, hogy az egyik w-re illeszkedő $w - x$ élét kitöröljük, és helyette bevesszük a $u - w$, $x - v_1$, $v_1 - v_0$ éleket. Az új gráfban pedig egy Hamilton-út két végső éle csak $u - w$, illtve $v_1 - v_0$ lehet, közben pedig bejárja G csúcsait. Mivel v_1 összes szomszédja szomszédja w-nek is, az út G-beli része Hamilton-körré zárható.

64. A tankönyben szereplő RH $\prec$ EP Karp-redukció valójában erre a speciális esetre vezeti vissza a Részhalmazösszeg (RH) problémát. (Az egyenlőtlenségek között szerepelnek azok, amelyek a megoldások komponenseit $\{0, 1\}$-be szorítják.)

66. Világos, hogy a probléma NP-ben van: egy megadott kiválasztásról hatékonyan ellenőrizhető, hogy jó-e. A nehézség igazolásához megadunk egy $X3C \prec X4C$ Karp redukciót. Legyen $V, Y_1, \ldots, Y_s$ az $X3C$ feladat egy példánya. Ebből megkonstruáljuk az $X4C$ feledat egy példányát: Legyen $t = \lfloor |V|/3 \rfloor$, $U = V \cup \{v_1\} \cup \ldots \cup \{v_t\}$ (diszjunkt unió, azaz a lefedendő V alaphalmazhoz még hozzáveszünk t pontot). Legyenek továbbá $X_{ij} = Y_i \cup \{v_j\}$ $(i = 1, \ldots, s, j = 1 \ldots, t)$. Világos, hogy ez a konstrukció az $X3C$ feladat méretében polinom időben végrehajtható (a méret meghatározó tagja a négyesek száma: st). Az is nyilvánvaló, hogy V pontosan akkor fedhető le diszjunktan néhány X_i hármassal, ha U diszjunktan lefedhető néhány Y_{ij} négyessel.

67. A tanú-tételből egyszerűen adódik, hogy $PONTY \in$ NP: egy megfelelő kiértékelés választható tanúnak. Az NP-teljesség nehezebb részének bizonyításához megadunk egy $X3C \prec PONTY$ Karp-redukciót. Az $X3C$ egy $V, \mathcal{H}$ példányához minden egyes $H \in \mathcal{H}$ hármasnak feleltessünk meg egy x_H Boole-változót, és képezzük a $\Psi = \bigvee_{v \in V} (\bigwedge_{H \ni v} x_H)$ konjunktív normálformájú Boole-kifejezést. Ψ polinomméretű: $O(|V||\mathcal{H}|$; és Ψ-nek egy a feledatban előírtaknak eleget tevő kiértékelése éppen V egy pontos fedésének feleltethető meg és viszont: $x_H =$ = "igaz" $\Leftrightarrow H \in$ fedés. Tehát a megadott leképezés tényleg egy Karp-redukció.

10. Aritmetika