Eötvös Loránd Tudományegyetem
Informatikai Kar

Szabó László

Algoritmusok

Feladatgyűjtemény

Budapest, 2016

Lektorálta: Fekete István

A jegyzet az ELTE 2016. évi jegyzetpályázatának támogatásával készült.

Tartalomjegyzék

Bevezetés

Ez a példatár 134 feladatot tartalmaz az algoritmusok világából. Mindegyik feladatra részletes megoldást adunk. A feladatok döntő többsége az informatikus BSc és MSc hallgatók számára tartott algoritmuselmélet kurzusok anyagához kapcsolódik. Ezek a kurzusok rendszerint érinti az alábbi témákat, ezek ismeretét feltételezzük az olvasóról.

- Aszimptotikus jelölések.
- Beillesztéses rendezés. Összefésüléses rendezés.
- Láncolt listák. Bináris keresőfák. Piros-fekete fák.
- Dinamikus programozás.
- Mohó módszer.
- Gráfok ábrázolása szomszédsági listákkal és szomszédsági mátrixszal.
- Szélességi bejárás. Mélységi bejárás.
- Legrövidebb utak. Dijkstra algoritmus. Bellmann-Ford algoritmus. Floyd-Warshall algoritmus.
- Minimális költségű feszítőfák. Prim algoritmus. Kruskal algoritmus.
- Hálózatok és folyamok. Ford-Fulkerson algoritmus. A maximális folyam – minimális vágás tétel. Egészértékűség.
- NP-teljesség. A SAT és a 3SAT nyelv.

A feladatokat hat téma köré csoportosítottuk: keresés és rendezés, adatszerkezetek, dinamikus programozás és mohó módszer, gráfalgoritmusok, folyamok és párosítások, NP-teljesség és közelítő algoritmusok. A feladatokat, illetve azok ötleteit az irodalomjegyzékben felsorolt tankönyvekből és az internetről gyűjtöttük.

A példatárat leginkább informatikus és matematikus egyetemi hallgatóknak ajánljuk, de érdeklődő középiskolások is haszonnal forgathatják, különösen ha programozás versenyekre készülnek.

Szeretnék köszönetet mondani Fekete Istvánnak, amiért elvállalta a kézirat lektorálását, és értékes tanácsaival jelentősen hozzájárult annak jobbá tételéhez. Szeretnék még köszönetet mondani Bene Katalinnak a hasznos beszélgetésekért, amelyeket a feladatokról folytattunk.

Keresés és rendezés

Bináris keresés

Feladat.
Adott valós számok egy monoton növekvően rendezett $A[1:n]$ tömbje. Feladatunk annak eldöntése, hogy egy v szám előfordul-e A elemei között. Igenlő válasz esetén a v elem A-beli helye is érdekel bennünket. Egy lehetséges megoldás a következő.

Először A középső elemével, legyen ez $A[k]$, hasonlítjuk össze v-t. Ha $v = A[k]$, akkor kész vagyunk. Ha viszont $v \neq A[k]$, akkor a v elemet elég az $A[1], \ldots, A[k-1]$ vagy az $A[k+1], \ldots, A[n]$ elemek között keresni attól függően, hogy $v < A[k]$ vagy $v > A[k]$. Tehát egyetlen kérdéssel vagy megtaláljuk v-t, vagy visszavezetjük a feladatot egy feleakkora méretű feladatra. Az eljárást addig ismételjük, amíg a felezés lehetséges. A módszer neve bináris keresés.

(A) Írjuk meg a bináris keresés algoritmust rekurzív és iteratív formában is!

(B) Mutassuk meg, hogy az algoritmus költsége $O(\log n)$.

Megoldás.
(A) Lássuk először a rekurzív változatot. Az eljárást az $(A, v, 1, n)$ paraméterekkel kell meghívni.

```
RekurzívBinárisKeresés(A,v,i,j)
if i>j then return nil
középső=[(i+j)/2]
if A[középső]=v
  then
    return középső
  else
    if A[középső]>v
      then return RekurzívBinárisKeresés(A,v,i,középső-1)
      else return RekurzívBinárisKeresés(A,v,középső+1,j)
```

Itt $[(i+j)/2]$ szokásos módon az $(i+j)/2$ kifejezés értékének egészrészét jelöli. Jöjjön ezután az iteratív változat.

```
IteratívBinárisKeresés(A,n,v)
i=1
j=n
while i<=j do
  középső=[(i+j)/2]
  if A[középső]=v
    then
      return középső
    else
      if A[középső]>v
        then j=középső-1
        else i=középső+1
return nil
```

(B) Jelölje a keresés során egyre csökkenő méretű résztömböket rendre $A_1, A_2, \ldots, A_j$. Itt A_1 maga az $A[1:n]$ tömb, az A_j résztömbről pedig feltesszük, hogy nem üres és a középső elemével történő összehasonlítás már megválaszolja a "v benne van-e az A tömbben" kérdést. A felhasznált összehasonlítások száma így összesen j. Az $|A_{i+1}| \leqslant |A_i|/2$ egyenlőtlenségek összefűzésével kapjuk, hogy

$$|A_j| \leqslant |A_{j-1}|/2 \leqslant |A_{j-2}|/2^2 \leqslant \cdots \leqslant |A_2|/2^{j-2} \leqslant |A_1|/2^{j-1} = n/2^{j-1}$$

(természetesen $|A_i|$ az A_i résztömb elemszámát jelöli). Ezt összevetve az $|A_j| \geqslant 1$ egyenlőtlenséggel $n/2^{j-1} \geqslant 1$, illetve átrendezve $j \leqslant \log_2 n + 1$ adódik.

Két rendezett tömb középső eleme

Feladat.
Legyenek $A[1:n]$ és $B[1:n]$ monoton növekvően rendezett tömbök. Tegyük fel, hogy A és B összesen $2n$ eleme mind különböző. Adjunk $O(\log n)$ költségű algoritmust, amely A és B összesen $2n$ eleme közül meghatározza az n-ediket!

Megoldás.
Tegyük fel, hogy az n-edik elem az A tömbben van, mondjuk annak k-adik eleme.

Legyen először $k < n$. Ekkor az A tömb $k-1$ eleme kisebb, mint $A[k]$, és $n-k$ eleme nagyobb, mint $A[k]$. Tudjuk továbbá, hogy A és B összesen $2n$ eleme közül $n-1$ kisebb, mint $A[k]$, és n nagyobb, mint $A[k]$. Ezért ebben az esetben a B tömb $(n-1)-(k-1) = n-k$ eleme kisebb, mint $A[k]$, és $n-(n-k) = k$ eleme nagyobb, mint $A[k]$. A B tömb rendezettsége miatt így $B[n-k] < A[k] < B[n-k+1]$. Vegyük észre azt is, hogy ha $1 \leqslant k' < k$, akkor $A[k'] < A[k]$ és $n-k < n-k'$, így $A[k'] < A[k] < B[n-k']$. Hasonlóan, ha $k < k'' \leqslant n$, akkor $A[k] < A[k'']$ és $n-k''+1 < n-k+1$, így $B[n-k''+1] < A[k] < A[k'']$.

Hátra van még a $k = n$ eset. Ekkor világos módon $A[k] < B[1]$, továbbá minden $1 \leqslant k' < k$ esetén $A[k'] < B[n-k']$.

Mindezek alapján bináris kereséssel eldönthetjük, hogy az A tömbben van-e olyan $A[k]$ elem, amelyre vagy $k < n$ és $B[n-k] < A[k] < B[n-k+1]$, vagy pedig $k = n$ és $A[k] < B[1]$; ha találunk ilyen $A[k]$-t, akkor az lesz az n-edik elem. Ha nincs ilyen $A[k]$, akkor az n-edik elem szükségképpen a B tömbben van. A B tömbben ugyancsak bináris kereséssel találhatunk olyan $B[k]$ elemet, amelyre vagy $k < n$ és $A[n-k] < B[k] < A[n-k+1]$, vagy pedig $k = n$ és $B[k] < A[1]$; ekkor $B[k]$ lesz az n-edik elem.

```
KözépsőElem(X,Y,l,h)
if l>h
  then
    return nil
  else
    k=[(l+h)/2]
    if k=n
      then
        if X[k]<Y[1]
          then return X[k]
          else return nil
      else
        if Y[n-k]<X[k] AND X[k]<Y[n-k+1]
          then
            return X[k]
          else
            if X[k]<Y[n-k]
              then return KözépsőElem(X,Y,k+1,h)
              else return KözépsőElem(X,Y,l,k-1)
```

```
KétTömbKözépsőEleme(A,B,n)
középső=KözépsőElem(A,B,1,n)
if középső=nil then középső=KözépsőElem(B,A,1,n)
return középső
```

A bináris keresések költsége $O(\log n)$, így az algoritmus teljes költsége $O(\log n)$.

Barkochba

Feladat.
Az 1, 2, 3, ..., 16 számok közül kell kitalálni egyet barkochba kérdésekkel. Egy ilyen kérdés a következő:

"A gondolt szám a H halmazban van?"

(A) Legalább hány kérdésre van szükség ahhoz, hogy kitaláljuk a gondolt számot?

(B) Tegyük fel, hogy a kérdéseinket előre le kell írni, és nincs befolyásunk arra, hogy a gondoló milyen sorrendben nézi és válaszolja meg azokat. Legalább hány kérdésre van szükség ekkor ahhoz, hogy kitaláljuk a gondolt számot?

(C) Tegyük fel, hogy a kérdéseinket előre le kell írni, és nincs befolyásunk arra, hogy a gondoló milyen sorrendben nézi és válaszolja meg azokat. Hány kérdéssel tudjuk biztosan kitalálni a gondolt számot, ha előfordulhat, hogy az egyik válasz elvész, mielőtt eljutna hozzánk?

(D) Tegyük fel, hogy a kérdéseinket előre le kell írni, és nincs befolyásunk arra, hogy a gondoló milyen sorrendben nézi és válaszolja meg azokat. Hány kérdéssel tudjuk biztosan kitalálni a gondolt számot, ha előfordulhat, hogy egyszer téves választ kapunk?

Megoldás.
(A) Egy-egy kérdés a szóba jövő számok halmazát két részhalmazra bontja: az egyikben azok a számok lesznek, amelyek esetén a kérdésre "igen" választ kapunk (amennyiben az a gondolt szám), a másikban pedig azok, amelyek esetén a válasz "nem". "Rossz esetben" olyan választ kapunk, amely azt mutatja, hogy a gondolt szám abban a részhalmazban van, amelyik elemszáma legalább akkora, mint a másiké. Ezért ha minimális számú kérdéssel szeretnénk kitalálni a gondolt számot, akkor nem tehetünk jobbat, minthogy kérdéseinkkel egymás után megfelezzük a lehetőségeket. Négy kérdés mindenképp szükséges a kitaláláshoz, és ennyi elég is (vö. bináris keresés).

(B) Most is szükség van legalább négy kérdésre. Megmutatjuk, hogy ennyi elég is. Ehhez azt kell elérnünk, hogy a következő kérdés az előzőre adott bármely válasz esetén megfelezze a lehetőségeket. Az alábbi megoldás négy kérdésében a halmazok a következők:

1. halmaz									9	10	11	12	13	14	15	16
2. halmaz					5	6	7	8					13	14	15	16
3. halmaz			3	4			7	8			11	12			15	16
4. halmaz		2		4		6		8		10		12		14		16

Ehhez a megoldáshoz például a következő gondolatmenettel juthatunk el. Írjuk fel a gondolt számnál eggyel kisebb számot kettes számrendszerben négy jeggyel (pótoljuk az elejét nullákkal, ha szükséges). Ezek a felírások a következő táblázat oszlopaiban láthatók.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$n-1$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1. sor	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
2. sor	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
3. sor	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
4. sor	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Kérdéseink ezek után a következők:

1. Az így kapott szám balról (a táblázatban felülről) számított első jegye 1?
2. Az így kapott szám balról (a táblázatban felülről) számított második jegye 1?
3. Az így kapott szám balról (a táblázatban felülről) számított harmadik jegye 1?
4. Az így kapott szám balról (a táblázatban felülről) számított negyedik jegye 1?

A négy válasz alapján megkapjuk a gondolt számnál eggyel kisebb szám kettes számrendszerbeli alakját, így ki tudjuk találni a gondolt számot.

Vegyük észre, hogy táblázatunk (alsó) négy sora megfelel az első konstrukcióbeli négy kérdésnek. Az adott sorban az $n-1$ számnak megfelelő

oszlopban pontosan akkor áll 1, ha n benne van az első konstrukció megfelelő kérdésében szereplő halmazban.

(C) Öt kérdés nyilván szükséges. Megmutatjuk, hogy ennyi elég is. Öt megfelelő kérdés a következőképpen adható meg. Most 16 db olyan 5 hosszúságú $0-1$ sorozatot keresünk, amelyek közül bármelyik kettő legalább két helyen eltér egymástól. Ha ugyanis az egyik kérdésre nem érkezik válasz, azaz a sorozatokban az egyik helyen álló elem eltűnik, akkor továbbra is 16 különböző sorozatot kell kapnunk. Ez felel meg annak, hogy a négy megmaradt kérdésre adott válaszból minden esetben kitalálható a gondolt szám. Az alábbi táblázat oszlopaiba 16 megfelelő sorozatot írtunk be.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1. sor	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
2. sor	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
3. sor	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
4. sor	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
5. sor	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Vegyük észre, hogy az utolsó sor az előzőekhez hozzátett paritásjelző-bit. Az 5 kérdés az 5 sorból olvasható ki. Egy kérdés azokra a számokra kérdez rá, amelyek oszlopában az adott sorban 1 áll. Az öt kérdésben a halmazok a következők:

1. halmaz									9	10	11	12	13	14	15	16
2. halmaz					5	6	7	8					13	14	15	16
3. halmaz			3	4			7	8			11	12			15	16
4. halmaz		2		4		6		8		10		12		14		16
5. halmaz		2	3		5			8	9			12		14	15	

Az öt kérdés közül az első négy megegyezik a feladat előző változatában konstruált négy kérdéssel, az ötödik pedig azokra a számokra kérdez rá, amelyekre addig páratlan sokszor kérdeztünk rá. Így összességében minden számra páros sokszor kérdezünk rá, tehát ha mind az öt kérdésre kapnánk választ, akkor páros sok "igen"-t hallanánk. Ezért, ha egy válasz hiányzik, akkor az kitalálható a többi négyből.

(D) Először vizsgáljuk meg, hogy legalább hány kérdésre van szükség! Tegyük fel, hogy k előre leírt kérdéssel ki lehet találni a gondolt számot. Tekintsük ezt a k kérdést, és válasszuk ki a 16 szám egyikét, mintha az lenne a gondolt

szám. Válaszoljuk meg a k db kérdést hazugság nélkül. Írjunk 1-est az "igen", 0-t a "nem" válasz helyett. Így egy k hosszúságú $0-1$ sorozatot kapunk, amit a kiválasztott szám kódjának fogunk nevezni.

Ha a válaszoló egyszer hazudik, akkor nem a szám kódját kapjuk, hanem egy olyan sorozatot, amely egy helyen különbözik a szám kódjától. Az ilyen sorozatokat a szám álkódjának fogjuk nevezni. Álkódból éppen k darab van, mert a k kérdésből egyre kaphatunk rossz választ, és mindegyik kérdésre egyféle rossz válasz jöhet.

A 16 számhoz 16 kód és $16k$ álkód tartozik. Ezeknek mind különbözőeknek kell lenni, mert ha volna köztük két egyforma, akkor az annak a $0-1$ sorozatnak megfelelő válaszok esetén nem tudnánk kitalálni a gondolt számot. Összesen 2^k darab k hosszú $0-1$ sorozat van, így biztosan teljesül a $16(k+1) \leqslant 2^k$ egyenlőtlenség. A két oldal értékét $k = 1, 2, \ldots$ esetén táblázatban írtuk fel:

k	1	2	3	4	5	6	7	8
$16(k+1)$	32	48	64	80	96	112	128	144
2^k	2	4	8	16	32	64	128	256

Látható, hogy az egyenlőtlenség $k = 7$ esetén teljesül először. Tehát 7 előre leírt kérdésre mindenképp szükség van.

Megmutatjuk, hogy 7 előre leírt kérdéssel megoldható a feladat. Ehhez 16 db olyan 7 hosszúságú $0-1$ sorozatot kell megadni, amelyek közül bármelyik kettő legalább három helyen eltér egymástól. Ekkor a 16 kód és 16×7 álkód mind különböző lesz, hiszen ha két különböző kódhoz tartozó álkód megegyezik, akkor a két kód csak két helyen térhet el egymástól.

A következőképpen járunk el. A 0000000 sorozattól kezdve mindig a lexikografikusan legkisebb olyan sorozatot keressük, amelyik az összes korábban már kiválasztott sorozattól legalább 3 helyen különbözik. Az eredményül kapott 16 db $0-1$ sorozat az alábbi táblázat oszlopaiban látható:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1. sor	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
2. sor	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
3. sor	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
4. sor	0	0	1	1	1	1	0	0	1	1	0	0	0	0	1	1
5. sor	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
6. sor	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1
7. sor	0	1	1	0	0	1	1	0	1	0	0	1	1	0	0	1

A 7 kérdés a 7 sorból olvasható ki. Egy kérdés azokra a számokra kérdez rá, amelyek oszlopában az adott sorban 1 áll. A hét kérdésben a halmazok a következők:

1. halmaz									9	10	11	12	13	14	15	16
2. halmaz					5	6	7	8					13	14	15	16
3. halmaz			3	4			7	8			11	12			15	16
4. halmaz			3	4	5	6			9	10					15	16
5. halmaz		2		4		6		8		10		12		14		16
6. halmaz		2		4	5		7		9		11			14		16
7. halmaz		2	3			6	7		9			12	13			16

Hamis érme

Feladat.
Van 27 érménk, közülük egy hamis, könnyebb a többinél. Feladat egy kétkarú mérleget használva minél kevesebb méréssel meghatározni, hogy melyik a hamis érme.

Megoldás.
Először osszuk három darab 9 elemű csoportba az érméket. Két csoportot hasonlítsunk össze. Ha az egyik csoport könnyebb a másiknál, akkor abban van a hamis érme, ha egyenlők, akkor a harmadikban. Ezek után 9 érme közül kell kiválasztani a hamisat. Osszuk három darab 3 elemű csoportba az érméket. Két csoportot hasonlítsunk össze. Ha az egyik csoport könnyebb a másiknál, akkor abban van a hamis érme, ha egyenlők, akkor a harmadikban. Végül 3 érme közül kell kiválasztani a hamisat. Hasonlítsunk össze két érmét. Ha az egyik könnyebb, akkor az a hamis, ha egyenlők, akkor a harmadik. Így három mérés elegendő a hamis érme meghatározásához.

Ceruzaelemek

Feladat.
Micimackó mézdetektora két ceruzaelemmel működik. A fiókban 8 egyforma típusú elem van, ezek közül 4 vadonatúj, 4 viszont már lemerült. Az elemek sajnos összekeveredtek. Micimackó éppen használni szeretné a mézdetektort. Jobb lehetőség híján kiválaszt két elemet, beteszi a mézdetektorba, és megnézi, hogy a készülék bekapcsolódik-e. Legalább hány próbálkozásra van szükség a mézdetektor működésbe hozásához?

Megoldás.
Az egyszerűség kedvéért legyen az elemek halmaza $\{1, 2, 3, 4, 5, 6, 7, 8\}$. Először vegyük az $1, 2, 3$ elemeket, és próbáljuk ki mindegyiket mindegyikkel. Ha a mézdetektor nem kapcsolódik be egyik próbálkozásra sem, akkor az $1, 2, 3$ elemek közül legalább kettő lemerült. Ezután vegyük a $4, 5, 6$ elemeket, és próbáljuk ki mindegyiket mindegyikkel. Ha a mézdetektor nem kapcsolódik be egyik próbálkozásra sem, akkor a $4, 5, 6$ elemek közül is legalább kettő lemerült. Ha tehát eddig nem kapcsolódott be a mézdetektor, akkor $1, 2, 3, 4, 5, 6$ elemek közül legalább négy lemerült, következésképpen a $7, 8$ elemek újak, így ezeket betéve a készülékbe az szükségképpen bekapcsolódik. Ez a legrosszabb esetben is hét próbálkozás.

Megmutatjuk, hogy hétnél kevesebb próbálkozás általában nem elég. Indirekt tegyük fel, hogy van olyan eljárás, amely minden esetben legfeljebb hat próbálkozás után működésbe hozza a mézdetektort. Tekintsük azt a gráfot, amelynek csúcsai az elemek, két csúcs között pedig akkor megy él, ha az adott elempárt az eljárás során valamikor kipróbáljuk. Ennek a gráfnak nyolc csúcsa és legfeljebb hat éle van, továbbá nyilván nem tartalmaz négy olyan csúcsot, amelyek közül semelyik kettő nincs összekötve (négy csúcsú üres gráfot), mert ha történetesen ezek az új elemek, akkor ezekből semelyik kettőt nem próbáltuk ki, így a készülék az eljárás során nem kapcsolódott be. Hogy néz ki ennek a gráfnak a komplementere? A csúcsok száma nyolc, az élek száma legalább 22, és a gráf nem tartalmaz négy csúcsú teljes gráfot. Ez viszont ellentmond Turán tételének (ld. az utolsó fejezetben): ha egy nyolc csúcsú gráf nem tartalmaz négy csúcsú teljes gráfot, akkor éleinek száma legfeljebb 21.

Csuprok

Feladat.
Micimackó 63 fiókba osztotta szét a mézescsuprait úgy, hogy a k-adik fiókba éppen k csupor van ($k = 1, 2, \ldots, 63$). Adjunk optimális lépésszámú módszert az összes fiók kiürítésére, ha egy megengedett lépés a következő: jelöljünk ki tetszőleges számú fiókot, és mindegyikből vegyünk ki ugyanannyi mézescsuprot!

Megoldás.
Vegyük észre, hogy ha menet közben létrejön két vagy több olyan fiók, amelyek ugyanannyi csuprot tartalmaznak, akkor a továbbiakban ezek a fiókok együtt kezelhetők.

Az első lépésben minden olyan fiókból, amelyben legalább 32 csupor

van, vegyünk ki 32 csuprot. Ezután a nem üres fiókokban a csuprok száma $1, 2, \ldots, 31$. A második lépésben minden olyan fiókból, amelyben legalább 16 csupor van, vegyünk ki 16 csuprot. Ezután a nem üres fiókokban a csuprok száma $1, 2, \ldots, 15$. A harmadik lépésben minden olyan fiókból, amelyben legalább 8 csupor van, vegyünk ki 8 csuprot. Ezután a nem üres fiókokban a csuprok száma $1, 2, \ldots, 7$. Így folytatva 6 lépésben kiürül az összes fiók.

Megmutatjuk, hogy 6-nál kevesebb lépés nem elég. Tekintsünk egy tetszőleges módszert, és tegyük fel, hogy ez t lépésben üríti ki a fiókokat. Minden $0 \leqslant i \leqslant t$ esetén jelölje s_i az i-edik lépés után a nem üres fiókokban előforduló különböző csuporszámok számát. Most $s_0 = 63$ és $s_t = 0$.

Legyen $1 \leqslant i \leqslant t$, és tegyük fel, hogy az i-edik lépésben kiválasztott fiókokban előforduló különböző csuporszámok száma s. Ezekben a fiókokban az i-edik lépés után is s különböző csuporszám fog előfordulni, igaz közöttük a 0 is szerepelhet. Ebből következik, hogy

$$s_i \geqslant s - 1.$$

Másrészt azokban a fiókokban, amelyeket nem választottunk ki az i-edik lépésben, az előforduló különböző csuporszámok száma legalább $s_{i-1} - s$. Így

$$s_i \geqslant s_{i-1} - s.$$

Összeadva a két egyenlőtlenséget

$$2s_i \geqslant s_{i-1} - 1,$$

illetve átrendezve

$$s_{i-1} \leqslant 2s_i + 1$$

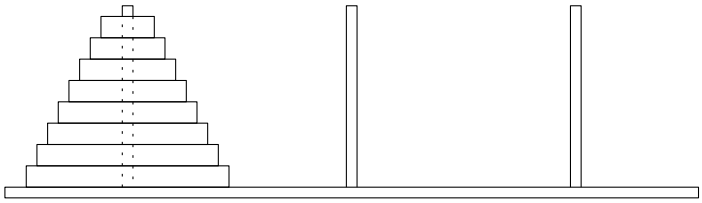
adódik. Egy kis számolás következik:

$$\begin{aligned} 63 = s_0 &\leqslant 1 + 2s_1 \\ &\leqslant 1 + 2(1 + 2s_2) = 1 + 2 + 2^2 s_2 \\ &\leqslant 1 + 2 + 2^2(1 + 2s_3) = 1 + 2 + 2^2 + 2^3 s_3 \leqslant \cdots \\ &\leqslant 1 + 2 + 2^2 + \cdots + 2^{t-1} + 2^t s_t \\ &= 1 + 2 + 2^2 + \cdots + 2^{t-1} \\ &= 2^t - 1. \end{aligned}$$

Ez azt jelenti, hogy $2^t \geqslant 64$, ennélfogva $t \geqslant 6$.

Hanoi tornyai

Feladat.
Egy talapzatba három függőleges rúd van rögzítve. A rudak egyikén n különböző átmérőjű, középen kifúrt korong helyezkedik el, alulról felfelé csökkenő sorrendben.



Egy lépésben valamelyik rúd legfelső korongját áttehetjük egy üres rúdra vagy egy másik rúdon levő korongok tetejére, ha ezzel nagyobb átmérőjű korong nem kerül kisebb átmérőjű fölé. A feladat az összes korong áthelyezése egy másik rúdra a fenti szabály betartásával; a korongok a másik rúdon is alulról felfelé az átmérőjük szerint csökkenő sorrendben vannak. Legalább hány lépés szükséges ehhez?

Megoldás.
A rudakra első, második és harmadik sorszámmal fogunk hivatkozni. Az általánosság megszorítása nélkül feltehető, hogy az első rúdról a második rúdra akarjuk áthelyezni a korongokat. Egy lehetséges megoldás a következő. Ha $n = 1$, akkor egyszerűen tegyük át a korongot az első rúdról a másodikra. Ha pedig $n \geqslant 2$, akkor járjunk el az alábbi módon.

(1) Helyezzük át az első rúd felső $n-1$ korongját (rekurzívan) a harmadik rúdra.

(2) Tegyük át a legnagyobb korongot az elsőről a második rúdra.

(3) Helyezzük át a harmadik rúdon levő $n-1$ korongot (rekurzívan) a második rúdra, a legnagyobb korong fölé.

Lássuk az algoritmust ezek után. Az algoritmust az $(n, 1, 2, 3)$ paraméterekkel kell meghívni.

```
HanoiTornyai(i,honnan,hova,felhasználásával)
if i=1
  then
    print honnan & '-->' & hova
```

```
else
  HanoiTornyai(i-1,honnan,felhasználásával,hova)
  print honnan & '-->' & hova
  HanoiTornyai(i-1,felhasználásával,hova,honnan)
```

Jelölje $T(n)$ az algoritmus lépésszámát n korong áthelyezése esetén. Nyilván $T(1) = 1$. Legyen ezután $n \geqslant 2$. Az első fázis, amikor az első rúd felső $n-1$ korongját áthelyezzük a harmadik rúdra, $T(n-1)$ lépés. A második fázis, amikor a legnagyobb korongot az elsőről a második rúdra helyezzük át, egy lépés. A harmadik fázis, amikor a harmadik rúdon levő $n-1$ korongot áthelyezzük a második rúdra, ismét $T(n-1)$ lépés. Ebből következik, hogy $T(n) = 2T(n-1)+1$ minden $n \geqslant 2$ egész számra. Összefoglalva

$$T(n) = \begin{cases} 1 & \text{ha } n = 1, \\ 2T(n-1)+1 & \text{ha } n \geqslant 2. \end{cases}$$

A rekurzív formulát kifejtve

$$\begin{aligned} T(n) &= 1 + 2T(n-1) \\ &= 1 + 2(1 + 2T(n-2)) \\ &= 1 + 2 + 4T(n-2) \\ &= 1 + 2 + 4(1 + 2T(n-3)) \\ &= 1 + 2 + 4 + 8T(n-3) \\ &= 1 + 2 + 4 + 8(1 + 2T(n-4)) \\ &= 1 + 2 + 4 + 8 + 16T(n-4) \\ &\vdots \\ &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1}T(1) \\ &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1} \\ &= \frac{2^n - 1}{2 - 1} = 2^n - 1. \end{aligned}$$

Megmutatjuk, hogy az általunk talált algoritmusnál nincs hatékonyabb eljárás. Jelölje $T^*(n)$ a minimális lépésszámot, amely n korong egyik rúdról egy másikra történő áthelyezéséhez szükséges. Nyilván $T^*(1) = 1$. Tegyük fel ezután, hogy $n \geqslant 2$ korongot szeretnénk áthelyezni az első rúdról a másodikra. Valamelyik lépésben nyilván át kell helyeznünk a legnagyobb korongot az első rúdról egy másikra, mondjuk a másodikra. Hogy ezt megtehessük, nyilván az összes többi korongnak a harmadik rúdon kell lenni. Ekkor viszont ezt az $n-1$ korongot előzőleg át kellett helyezni az első rúdról a harmadikra, ami legalább $T^*(n-1)$ lépés. Hasonlóan, miután a legnagyobb korong a végső

helyére kerül, ismét legalább $T^*(n-1)$ lépés szükséges ahhoz, hogy a többi korongot fölé helyezzük. Ebből következik, hogy $T^*(n) \geqslant 2T^*(n-1)+1$ minden $n \geqslant 2$ egész számra. Innen rögtön adódik, hogy $T^*(n) \geqslant T(n)$ minden n pozitív egész számra.

Csere

Feladat.
Célunk az $A[1:n]$ tömb $A[1:k]$ és $A[k+1:n]$ részeinek felcserélése. Oldjuk meg ezt a feladatot csupán konstans méretű segédmemóriát használva $O(n)$ költséggel!

Megoldás.
Egy lehetséges rekurzív megoldás a következő.

- Ha az $A[1:k]$ és az $A[k+1:n]$ részek ugyanolyan méretűek, akkor egyszerűen cseréljük fel a két részt.
- Ha az $A[1:k]$ rész mérete kisebb, mint az $A[k+1:n]$ részé, akkor cseréljük fel az $A[1:k]$ részt az azzal megegyező méretű $A[k+1:2k]$ résszel, majd (rekurzívan) cseréljük fel az új $A[k+1:2k]$ részt az $A[2k+1:n]$ résszel.
- Ha az $A[1:k]$ rész mérete nagyobb, mint az $A[k+1:n]$ részé, akkor cseréljük fel az $A[2k-n+1:k]$ részt az azzal megegyező méretű $A[k+1:n]$ résszel, majd (rekurzívan) cseréljük fel az $A[1:2k-n]$ részt az új $A[2k-n+1:k]$ résszel.

```
EgyenlőRészeketFelcserél(A,i,l,j)
for s=1 to l-i+1 do
  Csere(A[i+s-1],A[l+s])

Felcserél(A,i,l,j)
if l-i+1=j-l
  then
    EgyenlőRészeketFelcserél(A,i,l,j)
  else
    if l-i+1<j-l
      then
        EgyenlőRészeketFelcserél(A,i,l,2l-i+1)
        Felcserél(A,l+1,2l-i+1,j)
```

```
      else
        EgyenlőRészeketFelcserél(A,2l-j+1,l,j)
        Felcserél(A,i,2l-j,l)

Felcserél(A,1,k,n)
```

Mennyi az eljárás során végrehajtott elemcserék száma? Vegyük észre, hogy minden cserével a tömb legalább egy eleme a helyére kerül. Ebből következik, hogy a cserék száma legfeljebb n.

A feladat meglepően egyszerű módon is megoldható. Először fordítsuk meg az elemek sorrendjét az $A[1 : k]$ résztömbben, majd az $A[k + 1 : n]$ résztömbben, végül az egész $A[1 : n]$ tömbben.

```
SorrendetFordít(A,i,j)
if i<j then
  for l=1 to [(j-i+1)/2] do
    csere(A[i+l-1],A[j-l+1])

Felcserél(A,n,k)
SorrendetFordít(A,1,k)
SorrendetFordít(A,k+1,n)
SorrendetFordít(A,1,n)
```

Öt szám rendezése

Feladat.
Mutassuk meg, hogy öt szám rendezhető 7 összehasonlítással!

Megoldás.
Jelölje a rendezendő elemeket a, b, c, d, e. Először hasonlítsuk össze az a és b, illetve a c és d elemeket. Az általánosság megszorítása nélkül feltehető, hogy az eredmény $a \leqslant b$ és $c \leqslant d$. A következő lépésben hasonlítsuk össze a b és d elemeket. Az általánosság megszorítása nélkül feltehető, hogy az eredmény $b \leqslant d$. Eddig összesen három összehasonlítást végeztünk. Ezután az e elemet az a, b, d elemek közé két összehasonlítás felhasználásával beszúrhatjuk: először a középső elemmel, majd az eredménytől függően a szélsők valamelyikével hasonlítjuk össze e-t. Mivel már tudjuk, hogy $c \leqslant d$, végül a c elemet $d \leqslant e$ esetén az a, b elemek közé, különben pedig vagy az e, a, b vagy az a, e, b vagy az a, b, e elemek közé kell beszúrni. Ehhez ismét elegendő két összehasonlítás.

Buborék rendezés

Feladat.
Feladatunk az $A[1:n]$ tömb rendezése. Egy lehetséges megoldás a következő. A tömb végéről indulva a rosszul álló szomszédos elemeket cserélgetve eljutunk a tömb elejéig. Ekkor a legkisebb elem(ek egyike) $A[1]$-ben van. Utána ismételjük ezt az $A[2:n]$ résztömbre, majd az $A[3:n]$ résztömbre, stb. Végül A rendezett lesz. A módszer neve buborék rendezés.

Írjuk meg a buborék rendezés algoritmust!

Megoldás.

```
BuborékRendezés(A,n)
for i=1 to n-1 do
  for j=n downto i+1 do
    if A[j]<A[j-1] then Csere(A[j],A[j-1])
```

A legrosszabb esetben $n(n-1)/2$ az összehasonlítások, és ugyanennyi a cserék száma (ez akkor fordul elő, ha a tömb kezdetben csökkenően rendezett).

Javított buborék rendezés

Feladat.
Vegyük észre, hogy ha a buborék rendezés külső ciklusának egy iterációja során egyetlen cserét sem hajtottunk végre, akkor felesleges minden további összehasonlítás, a tömb rendezett. Sőt ennél kicsit több is igaz: a külső ciklus egy iterációja után az iteráció során utolsónak végrehajtott csere helyétől balra eső rész kész, így ezután elég a jobbra eső résszel foglalkozni.

Írjuk meg a buborék rendezés algoritmusnak az előbbi észrevételt kihasználó javított változatát!

Megoldás.

```
JavítottBuborékRendezés(A,n)
i=1
while i<=n-1 do
  s=n
  for j=n downto i+1 do
    if A[j]<A[j-1] then
      Csere(A[j],A[j-1])
      s=j
  i=s
```

A legrosszabb esetben most is $n(n-1)/2$ az összehasonlítások, és ugyanennyi a cserék száma (ez akkor fordul elő, ha a tömb kezdetben csökkenően rendezett).

Nem rekurzív összefésüléses rendezés

Feladat.
Írjuk meg az összefésülés rendezés algoritmust iteratív formában!

```
Összefésülés(A,p,q,r)
k=q-p+1
l=r-q
for i=1 to k do
  B[i]=A[p+i-1]
for j=1 to l do
  C[j]=A[q+j]
B[k+1]=INFINITY
C[l+1]=INFINITY
i=1 : j=1
for m=p to r do
  if B[i]<=C[j]
    then
      A[m]=B[i]
      i=i+1
    else
      A[m]=C[j]
      j=j+1
```

```
RekurzívÖsszefésülésesRendezés(A,p,r)
if p<r then
  q=[(p+r)/2]
  RekurzívÖsszefésülésesRendezés(A,p,q)
  RekurzívÖsszefésülésesRendezés(A,q+1,r)
  Összefésülés(A,p,q,r)
```

```
RekurzívÖsszefésülésesRendezés(A,1,n)
```

Megoldás.
Első lépésben az $A[1:n]$ tömbben szomszédos párokat rendezzük. Utána a már rendezett szomszédos párokból rendezett négyeseket alakítunk ki összefésüléssel (ugyanazt az eljárást használva, mint a rekurzív változat), majd ezekből rendezett nyolcasokat, és így tovább. A költség itt is $O(n \log n)$.

```
IteratívÖsszefésülésesRendezés(A,n)
i=2
while i<2*n do
  r=0
  while r<n do
    p=r+1
    r=r+i
    q=[(p+r)/2]
    if q<n then
      r=min(r,n)
      Összefésülés(A,p,q,r)
  i=2*i
```

Összefésülés variáció

Feladat.
Írjuk meg az összefésülés algoritmust a tömbök végére beszúrt segédelemek használata nélkül!

Megoldás.
A $B[1:k]$ és $C[1:l]$ tömböket fésüljük össze az $A[1:k+l]$ tömbbe.

```
Összefésülés(B,k,C,l,A,k+l)
i=1
j=1
m=0
while i<=k AND j<=l do
  m=m+1
  if B[i]<=C[j]
    then
      A[m]=B[i]
      i=i+1
    else
      A[m]=C[j]
      j=j+1
while i<=k do
  m=m+1
  A[m]=B[i]
  i=i+1
```

```
while j<=l do
  m=m+1
  A[m]=C[j]
  j=j+1
```

Adott összeg keresése

Feladat.
Adott egész számoknak egy $A[1:n]$ tömbje és egy b egész szám. Adjunk $O(n \log n)$ költségű algoritmust, amely eldönti, hogy van-e két olyan eleme a tömbnek, amelyek összege éppen b.

Megoldás.
Rendezzük először az $A[1:n]$ tömböt mondjuk összefésüléses rendezéssel! Ezek után egymás után minden egyes $1 \leqslant i \leqslant n-1$ indexre bináris kereséssel ellenőrizzük, hogy a $b - A[i]$ szám benne van-e a (rendezett) $A[i+1:n]$ résztömbben. Ha valamikor találunk ilyen elemet, akkor megállunk, és a kérdésre "igen" a válasz. Egyébként nincs két olyan eleme a tömbnek, amelyek összege b.

Jegyezzük meg, hogy az i-edik iterációban, amikor a $b - A[i]$ értéket keressük, nem kell törődnünk az $A[1:i]$ résztömb elemeivel. Valóban, ha lenne olyan $j < i$ index, amelyre $A[j] + A[i] = b$, akkor az eljárás már a j-edik iterációban befejeződött volna, amikor is $A[j]$-hez kerestük a $b - A[j]$ értéket.

Az összefésüléses rendezés költsége $O(n \log n)$. Az egyes bináris keresések költsége $O(\log n)$, így a keresések összköltsége szintén $O(n \log n)$. Ennélfogva az algoritmus teljes költsége $O(n \log n)$.

Inverziók száma

Feladat.
Adott különböző számoknak egy $A[1:n]$ tömbje. Azt mondjuk, hogy valamely $1 \leqslant i < j \leqslant n$ esetén az (i, j) pár az A egy inverziója, ha $A[i] > A[j]$.

(A) Soroljuk fel a $\langle 2, 3, 8, 6, 1 \rangle$ tömb inverzióit!

(B) Az $A[1:n]$ tömb inverzióinak száma mikor maximális?

(C) Mi a kapcsolat a beillesztéses rendezés költsége és a rendezendő tömb inverzióinak száma között?

```
BeillesztésesRendezés(A,n)
for j=2 to n do
  s=A[j]
  i=j-1
  while i>0 AND A[i]>s do
    A[i+1]=A[i]
    i=i-1
  A[i+1]=s
```

(D) Adjunk $O(n \log n)$ költségű algoritmust, amely meghatározza az $A[1:n]$ tömb inverzióinak számát!

Megoldás.
(A) Öt inverzió van:

$$(1,5),(2,5),(3,4),(3,5),(4,5).$$

(B) Az inverziók száma legfeljebb annyi, mint a lehetséges indexpárok száma, vagyis

$$\binom{n}{2}.$$

Ha a tömb monoton csökkenően rendezett, akkor az inverziók száma éppen ennyi.

(C) A beillesztéses rendezés költsége $O(n+I)$, ahol I a rendezendő tömb inverzióinak száma. Ez a következőképpen látható be. A while ciklusok költségét nem számítva az algoritmus költsége $O(n)$. A while ciklusok minden iterációjában pontosan egy inverzió megszűnik. Mire az algoritmus befejeződik, egyetlen inverzió sem marad. Ez csak úgy lehet, ha a while ciklusokbeli iterációk száma összesen éppen I. Ezért az algoritmus teljes költsége $O(n+I)$.

(D) Módosítsuk a rekurzív összefésüléses rendezés algoritmust a következőképpen.

```
InverziókSzámaÖsszefésüléssel(A,p,q,r)
k=q-p+1
l=r-q
for i=1 to k do
  B[i]=A[p+i-1]
for j=1 to l do
  C[j]=A[q+j]
B[k+1]=INFINITY
C[l+1]=INFINITY
```

```
i=1
j=1
inv=0
for m=p to r do
  if B[i]<C[j]
    then
      A[m]=B[i]
      i=i+1
    else
      A[m]=C[j]
      inv=inv+k-i+1
      j=j+1
return inv

InverziókSzáma(A,p,r)
inv=0
if p<r then
  q=[(p+r)/2]
  inv=inv+InverziókSzáma(A,p,q)
  inv=inv+InverziókSzáma(A,q+1,r)
  inv=inv+InverziókSzámaÖsszefésüléssel(A,p,q,r)
return inv

InverziókSzáma(A,1,n)
```

Először az $A[1:n]$ tömböt az $A[1:q]$ és az $A[q+1:n]$ résztömbökre bontjuk, ahol $q = [(n+1)/2]$, és rekurzívan meghatározzuk az egyes résztömbök inverziószámát. Ezután következik azon (i', j') inverziók összeszámlálása, amelyekre $1 \leqslant i' \leqslant q < j' \leqslant n$. Ezt valósítja meg az `InverziókSzámaÖsszefésüléssel` eljárás. Jegyezzük meg, hogy mire az eljárás meghívásra kerül, az $A[1:q]$ és az $A[q+1:n]$ résztömbök már rendezettek.

Belátjuk az `InverziókSzámaÖsszefésüléssel` eljárás helyességét. Tegyük fel, hogy az eljárás során éppen az $A[p:q]$ résztömb $A[i']$ és az $A[q+1:r]$ résztömb $A[j']$ értékének összehasonlításánál tartunk, ahol $p \leqslant i' \leqslant q < j' \leqslant r$. Azt is tegyük fel, hogy már minden olyan inverziót felderítettünk, amelynek az első komponense p és $i'-1$ között van, vagy a második komponense $q+1$ és $j'-1$ között.

- Ha $A[i'] < A[j']$, akkor $A[j'] < A[j'+1] < \cdots < A[n]$ miatt nincs olyan eddig nem látott inverzió, amelynek első komponense i', így ennél a lépésnél a felderített inverziók száma nem nő.

- Ha $A[i'] > A[j']$, akkor $A[q] > \cdots > A[i'+1] > A[i']$ miatt (i', j'), $(i'+1, j'), \ldots, (q, j')$ eddig nem látott új inverziók, így ennél a lépésnél a felderített inverziók száma $(q - i' + 1)$-gyel nő.

Innen az `InverziókSzámaÖsszefésüléssel` eljárás helyessége adódik.

Az algoritmus teljes költsége az összefésüléses rendezéséhez hasonlóan $O(n \log n)$.

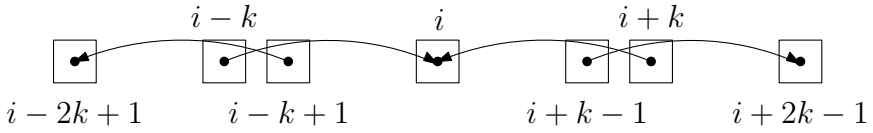
Majdnem rendezett tömb

Feladat.
Adott különböző számoknak egy növekvően rendezett $A[1 : n]$ tömbje. A tömb elemeit valaki megkeveri, de csak annyira, hogy minden egyes elem új helye az eredetitől legfeljebb k távolságra esik. Adjunk hatékony algoritmust az eredeti állapot helyreállítására!

Megoldás.
A megkevert tömb szerkezetére vonatkozó kulcsészrevétel a következő: bármely $A[i]$ elem indexe legfeljebb $4k - 2$ inverzióban szerepelhet; az inverzió másik tagja szükségképpen az

$$i - 2k + 1, i - 2k + 2, \ldots, i + 2k - 2, i + 2k - 1$$

indexek közül kerül ki.



Így a megkevert tömb inverzióinak száma legfeljebb $n(4k - 2)$.

Rendezzük most a megkevert tömböt beillesztéses rendezéssel. Ennek költsége $O(n + I)$, ahol I a tömb inverzióinak száma. Itt $I = O(nk)$, így a visszarendezés költsége $O(nk)$.

A feladat megoldható $O(n \log k)$ költséggel is. Az egyszerűség kedvéért tegyük fel, hogy n osztható $2k$-val. A megkevert tömb szerkezetére vonatkozó fenti kulcsészrevételből következik, hogy az alábbi $t = 2k$ darab (rész)tömb rendezett:

$$\begin{aligned}
A_1 &= \langle A[1], A[2k+1], A[4k+1], \ldots, A[n-(2k-1)] \rangle, \\
A_2 &= \langle A[2], A[2k+2], A[4k+2], \ldots, A[n-(2k-2)] \rangle, \\
&\vdots \\
A_t &= \langle A[2k], A[4k], A[6k], \ldots, A[n] \rangle.
\end{aligned}$$

A visszarendezéshez ezért elég az $A_1, A_2, \ldots, A_t$ tömböket összefésülni. Ennek egy hatékony megvalósítása a következő. Első menetben összefésüljük az A_1 és A_2 tömböket az A_{12} tömbbe, az A_3 és A_4 tömböket az A_{34} tömbbe, és így tovább. Második menetben összefésüljük A_{12} és A_{34} tömböket az A_{1234} tömbbe, az A_{56} és A_{78} tömböket az A_{5678} tömbbe, és így tovább. Folytassuk ezt amíg van legalább két tömb.

Világos, hogy az eljárás $\lceil \log k \rceil$ menet után befejeződik. Egy menetben az összefésülések költsége $O(n)$, így az eljárás teljes költsége $O(n \log k)$.

Módosítás rendezett tömbben

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

Megoldás.
Vegyünk fel két segédtömböt, jelölje ezeket R és S, és osszuk szét az A tömb elemeit R és S között a következőképpen:

```
Szétoszt(A,n)
j=0 : l=0
for i=1 to n do
  if j=0
    then
      j=j+1
      R[j]=A[i]
    else
      if R[j]<=A[i]
        then
          j=j+1
          R[j]=A[i]
        else
          l=l+2
          S[l]=A[i]
          S[l-1]=R[j]
          j=j-1
return R,j,S,l
```

Először is vegyük észre, hogy az R tömb monoton növekvően rendezett. Tekintsük ezután az S tömböt. Az S tömbbe párosával kerülnek az elemek,

mindig egy olyan $(A[h], A[i])$ pár, amelyre $h < i$ és $A[h] > A[i]$. Mivel az A tömb eredetileg monoton növekvően rendezett volt, egy ilyen pár csak akkor jelenhet meg, ha az $A[h]$ és $A[i]$ elemek közül legalább az egyik megváltozott. Feltételünk szerint k elem változott meg, ezért legfeljebb k ilyen pár lehet az S tömbben.

Az algoritmus ezek után a következő. Az A tömb szétosztása után rendezzük az S tömböt, majd fésüljük össze a rendezett R és S tömböket.

A szétosztás költsége $O(n)$, az S tömb rendezéséé $O(k \log k)$, az összefésülésé pedig ismét $O(n)$. Így az algoritmus teljes költsége $O(n + k \log k)$.

Gyorsrendezés

Feladat.
Adott különböző számoknak egy $A[1 : n]$ tömbje. A tömb rendezésére egy lehetséges módszer a következő. Vegyük a tömb egy véletlen elemét. Mozgassuk a tömb elejére az ennél kisebb, a végére pedig az ennél nagyobb elemeket, majd a kiválasztott elemet helyezzük el a nála kisebbek, illetve a nála nagyobbak közé. Ezzel a kiválasztott elem a rendezés szerinti helyére kerül. Rendezzük ezután a kiválasztott elem előtti és mögötti résztömböket különkülön rekurzív módon. A módszer neve gyorsrendezés.

(A) Írjuk meg a partícionáló eljárást és a rendező algoritmust!

(B) Mutassuk meg, hogy az algoritmus összehasonlítások számában mért költségének várható értéke $O(n \log n)$.

Megoldás.

(A) Lássuk először a partícionáló eljárást, amely az $A[p : r]$ résztömböt rendezi át helyben. Az eljárás a véletlen partícionáló elem (átrendezés utáni) indexével tér vissza. Az összehasonlítások száma $r - p$.

```
Partícionálás(A,p,r)
i=Random(p,r)
csere(A[i],A[r])
i=p-1
for j=p to r-1 do
  if A[j]<A[r] then
    i=i+1
    csere(A[i],A[j])
csere(A[i+1],A[r])
return i+1
```

Ezután jöjjön a rendező algoritmus. Az algoritmust az $(A, 1, n)$ paraméterekkel kell meghívni.

```
Gyorsrendezés(A,p,r)
if p<r then
  q=Partícionálás(A,p,r)
  Gyorsrendezés(A,p,q-1)
  Gyorsrendezés(A,q+1,r)
```

Vegyük észre, hogy a rekurzió mélysége a legrosszabb esetben itt a tömb méretében lineáris. A következő változatban a rekurzió mélysége a tömb méretében logaritmikus a legrosszabb esetben is. Az algoritmust az $(A, 1, n)$ paraméterekkel kell meghívni.

```
MódosítottGyorsrendezés(A,p,r)
while p<r do
  q=Partícionálás(A,p,r)
  if q-p<r-q
    then
      MódosítottGyorsrendezés(A,p,q-1)
      p=q+1
    else
      MódosítottGyorsrendezés(A,q+1,r)
      r=q-1
```

(B) Nem megy az általánosság rovására, ha azt gondoljuk, hogy a rendezendő elemek az $1, 2, \ldots, n$ egész számok. Jelölje $T(n)$ az összehasonlítások számának várható értékét. Nyilván $T(0) = T(1) = 0$.

Legyen $T(n, j)$ az összehasonlítások számának várható értéke abban az esetben, amikor a legelőször választott partícionáló elem éppen j. Világos, hogy

$$T(n, j) = n - 1 + T(j - 1) + T(n - j).$$

A partícionáló elem véletlen választásának köszönhetően

$$T(n) = \frac{1}{n} \left(T(n, 1) + T(n, 2) + \cdots + T(n, n - 1) + T(n, n) \right).$$

Innen az előző összefüggéseket felhasználva kiküszöbölhetjük a $T(n, j)$ mennyiségeket:

$$T(n) = n - 1 + \frac{2}{n} \left(T(0) + T(1) + T(2) + \cdots + T(n - 1) \right).$$

Szorozzuk be mindkét oldalt n-nel:

$$nT(n) = n(n-1) + 2(T(0) + T(1) + T(2) + \cdots + T(n-1)),$$

majd írjuk fel ugyanezt a formulát $n-1$ esetén is:

$$(n-1)T(n-1) = (n-1)(n-2) + 2(T(0) + T(1) + T(2) + \cdots + T(n-2)).$$

A két formulát egymásból kivonva kapjuk, hogy

$$nT(n) - (n-1)T(n-1) = 2(n-1) + 2T(n-1),$$

illetve átrendezve

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2(n-1)}{n(n+1)}.$$

Növeljük a jobb oldal második tagját:

$$\frac{T(n)}{n+1} < \frac{T(n-1)}{n} + \frac{2}{n}.$$

Ezt ismételten önmagába helyettesítve azt nyerjük, hogy

$$\begin{aligned}
\frac{T(n)}{n+1} &< \frac{2}{n} + \frac{T(n-1)}{n} \\
&< \frac{2}{n} + \frac{2}{n-1} + \frac{T(n-2)}{n-1} \\
&< \frac{2}{n} + \frac{2}{n-1} + \frac{2}{n-2} + \frac{T(n-3)}{n-2} < \cdots \\
&< \frac{2}{n} + \frac{2}{n-1} + \frac{2}{n-2} + \cdots + \frac{2}{2} + \frac{T(1)}{2} \\
&= \frac{2}{n} + \frac{2}{n-1} + \frac{2}{n-2} + \cdots + \frac{2}{2}.
\end{aligned}$$

Innen

$$\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n} < \int_1^n \frac{dx}{x} = \ln n$$

(Riemann alsó közelítő összeg) felhasználásával

$$T(n) = O(n \log n)$$

adódik.

Megjegyezzük, hogy összehasonlítások számának várható értékére $T(n) \leqslant 1.39\, n \log n$ is igazolható.

Csavarok és anyák

Feladat.
Adott egy dobozban n különböző méretű csavar, egy másik dobozban pedig a hozzájuk illő anyák. Sajnos sem a csavarokat nem tudjuk egymással összehasonlítani, sem az anyákat. Azt tudjuk csak kipróbálni, hogy egy csavar külső átmérője kisebb, nagyobb vagy egyenlő egy anya belső átmérőjénél (megpróbáljuk az anyát rácsavarni a csavarra).

Adjunk hatékony randomizált algoritmust az egymásnak megfelelő csavarok és anyák összepárosítására! Az algoritmus összehasonlításokban mért költségének várható értéke $O(n \log n)$ legyen!

Megoldás.
Válasszunk véletlenszerűen egy c csavart. Keressük meg a c csavarhoz illő a anyát, eközben válogassuk szét a többi anyát az a anyánál kisebbekre és nagyobbakra (az előző feladat partícionáló eljárásához hasonló módon). Ez n darab összehasonlítás. Az a anya segítségével válogassuk szét a csavarokat is a c csavarnál kisebbekre és nagyobbakra. Ez további $n-1$ darab összehasonlítás. Így az összehasonlítások száma összesen $2n-1$. Ezután rekurzívan párosítsuk a c-nél kisebb csavarokat az a-nál kisebb anyákkal, illetve a c-nél nagyobb csavarokat az a-nál nagyobb anyákkal.

Elemezzük az algoritmus (összehasonlításokban mért) költségét. Nem megy az általánosság rovására, ha azt gondoljuk, hogy a csavarok átmérői az $1, 2, \ldots, n$ egész számok. Jelölje $T(n)$ az összehasonlítások számának várható értékét. Nyilván $T(0) = T(1) = 0$.

Legyen $T(n, j)$ az összehasonlítások számának várható értéke abban az esetben, amikor a legelőször választott csavar éppen a j átmérőjű. Világos, hogy

$$T(n, j) = 2n - 1 + T(j-1) + T(n-j).$$

A csavar véletlen választásának köszönhetően

$$T(n) = \frac{1}{n}\left(T(n,1) + T(n,2) + \cdots + T(n,n-1) + T(n,n)\right).$$

Innen az előző összefüggéseket felhasználva kiküszöbölhetjük a $T(n, j)$ mennyiségeket:

$$T(n) = 2n - 1 + \frac{2}{n}\left(T(0) + T(1) + T(2) + \cdots + T(n-1)\right).$$

Szorozzuk be mindkét oldalt n-nel:

$$nT(n) = n(2n-1) + 2(T(0) + T(1) + T(2) + \cdots + T(n-1)),$$

majd írjuk fel ugyanezt a formulát $n-1$ esetén is:

$$(n-1)T(n-1) = (n-1)(2n-3) + 2(T(0) + T(1) + T(2) + \cdots + T(n-2)).$$

A két formulát egymásból kivonva kapjuk, hogy

$$nT(n) - (n-1)T(n-1) = 4n - 3 + 2T(n-1),$$

illetve átrendezve

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{4n-3}{n(n+1)}.$$

Növeljük a jobb oldal második tagját:

$$\frac{T(n)}{n+1} < \frac{T(n-1)}{n} + \frac{4}{n}.$$

Ezt ismételten önmagába helyettesítve azt nyerjük, hogy

$$\begin{aligned}
\frac{T(n)}{n+1} &< \frac{4}{n} + \frac{T(n-1)}{n} \\
&< \frac{4}{n} + \frac{4}{n-1} + \frac{T(n-2)}{n-1} \\
&< \frac{4}{n} + \frac{4}{n-1} + \frac{4}{n-2} + \frac{T(n-3)}{n-2} < \cdots \\
&< \frac{4}{n} + \frac{4}{n-1} + \frac{4}{n-2} + \cdots + \frac{4}{2} + \frac{T(1)}{2} \\
&= \frac{4}{n} + \frac{4}{n-1} + \frac{4}{n-2} + \cdots + \frac{4}{2}.
\end{aligned}$$

Innen

$$\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n} < \int_1^n \frac{dx}{x} = \ln n$$

(Riemann alsó közelítő összeg) felhasználásával

$$T(n) = O(n \log n)$$

adódik.

Bináris számok rendezése

Feladat.
Adott d jegyű bináris számok egy $A[1:n]$ tömbje. Tervezzünk $O(nd)$ költségű algoritmust a tömb rendezésére!

Megoldás.
Mozgassuk először a tömb elejére azokat a bináris számokat, amelyek 2^{d-1} helyiértékén 0 áll, a végére pedig azokat, amelyeknek 1. Az így kialakult két résztömbben külön-külön mozgassuk a résztömb elejére azokat a bináris számokat, amelyek 2^{d-2} helyiértékén 0 áll, a végére pedig azokat, amelyeknek 1, és így tovább.

Az algoritmus ezek után a következő. A `Bit(x,i)` függvény egy x bináris szám 2^{i-1} helyiértékén álló számjegyét adja vissza konstans időben. Az algoritmust az $(A, d, 1, n)$ paraméterekkel kell meghívni.

```
BinárisSzámokRendezése(A,i,l,h)
if i>0 AND h>l then
  j=l
  k=h
  while j<k do
    while j<k AND Bit(A[j],i)=0 do
      j=j+1
    while j<k AND Bit(A[k],i)=1 do
      k=k-1
    if j<k then
      Csere(A[j],A[k])
      j=j+1
      k=k-1
  if Bit(A[j],i)=1 then
      j=j-1
  BinárisSzámokRendezése(A,i-1,l,j)
  BinárisSzámokRendezése(A,i-1,j+1,h)
```

Maximális növekedés

Feladat.
Adott valós számoknak egy $A[1:n]$ tömbje. Adjunk hatékony algoritmust, amely meghatároz két olyan $1 \leqslant i \leqslant j \leqslant n$ indexet, amelyekre $A[j] - A[i]$ maximális!

Megoldás.
A megoldás alapötlete a következő. Bontsuk fel az $A[1:n]$ tömböt az $A[1:m]$ és $A[m+1:n]$ résztömbökre, ahol $m = [(n+1)/2]$.

(1) Rekurzívan keressünk olyan $1 \leqslant i \leqslant j \leqslant m$ indexeket, amelyekre $A[j] - A[i]$ maximális az $A[1:m]$ résztömbön.

(2) Rekurzívan keressünk olyan $m+1 \leqslant i \leqslant j \leqslant n$ indexeket, amelyekre $A[j]-A[i]$ maximális az $A[m+1:n]$ résztömbön.

(3) Vizsgáljuk meg, hogy mely $1 \leqslant i \leqslant m < j \leqslant n$ indexekre lesz $A[j]-A[i]$ maximális. Világos, hogy itt az optimumot azok az i és j indexek adják, amelyekre $A[i]$ az $A[1:m]$ résztömb egy minimális, $A[j]$ pedig az $A[m+1:n]$ résztömb egy maximális eleme.

A három részfeladat megoldása között nyilván szerepel az eredeti probléma megoldása is; válasszuk tehát azt a megoldást, amelyre $A[j]-A[i]$ maximális.

Lássuk az algoritmust ezek után. Az algoritmust az $(A, 1, n)$ paraméterekkel kell meghívni.

```
MaximálisNövekedés(A,l,r)
if l=r then return (l,l)
m=[(l+r)/2]
(x,x')=MaximálisNövekedés(A,l,m)
(y,y')=MaximálisNövekedés(A,m+1,r)
z=MinimumKiválasztás(A,l,m)
z'=MaximumKiválasztás(A,m+1,r)
if A[x']-A[x]>=A[y']-A[y] AND A[x']-A[x]>=A[z']-A[z]
  then return (x,x')
if A[y']-A[y]>=A[x']-A[x] AND A[y']-A[y]>=A[z']-A[z]
  then return (y,y')
if A[z']-A[z]>=A[x']-A[x] AND A[z']-A[z]>=A[y']-A[y]
  then return (z,z')
```

A `MaximálisNövekedés(A,l,r)` eljárás az (i, j) értékekkel tér vissza, ahol $l \leqslant i \leqslant j \leqslant r$ olyan indexek, amelyekre $A[j]-A[i]$ maximális az $A[l:r]$ résztömbön. A `MinimumKiválasztás(A,l,m)` eljárás az $A[l:m]$ résztömb egy minimális elemének indexével, a `MaximumKiválasztás(A,m+1,r)` eljárás pedig az $A[m+1:r]$ résztömb egy maximális elemének indexével tér vissza.

Az utóbbi két eljárás költsége a résztömbök elemszámában lineáris. Az algoritmus költségére így a

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) & \text{ha } n > 1, \\ O(1) & \text{ha } n = 1 \end{cases}$$

rekurzív képlet adódik. Az egyszerűség kedvéért legyen $n = 2^k$. Ekkor

alkalmas c_1 és c_2 pozitív konstansokkal

$$\begin{aligned}
T(n) &\leqslant c_1 n + 2T(n/2) \\
&\leqslant c_1 n + 2(c_1(n/2) + 2T(n/2^2)) \\
&= c_1 \cdot 2n + 2^2 T(n/2^2) \\
&\leqslant c_1 \cdot 2n + 2^2(c_1(n/2^2) + 2T(n/2^3)) \\
&= c_1 \cdot 3n + 2^3 T(n/2^3) \leqslant \cdots \\
&\leqslant c_1 \cdot kn + 2^k T(n/2^k) \\
&= c_1 n \log_2 n + nT(1) \\
&\leqslant c_1 n \log_2 n + c_2 n.
\end{aligned}$$

Innen

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

Az algoritmus gyenge pontja az $A[l : m]$ résztömb minimumának és az $A[m+1 : r]$ résztömb maximumának a meghatározása; ehhez végiglépkedünk a résztömbökön, aminek költsége arányos a résztömbök elemszámával. Vegyük észre, hogy ez igazából felesleges, az $A[l : m]$ résztömb minimumának és az $A[m + 1 : r]$ résztömb maximumának a meghatározása rekurzívan is történhet.

Lássuk az algoritmus ezen változatát. Az algoritmust az $(A, 1, n)$ paraméterekkel kell meghívni.

```
JavítottMaximálisNövekedés(A,l,r)
if l=r then return (l,l,l,l)
m=[(l+r)/2]
(x,x',l_min,l_max)=JavítottMaximálisNövekedés(A,l,m)
(y,y',r_min,r_max)=JavítottMaximálisNövekedés(A,m+1,r)
z=l_min
z'=r_max
if A[x']-A[x]>=A[y']-A[y] AND A[x']-A[x]>=A[z']-A[z]
  then return (x,x',min(l_min,r_min),max(l_max,r_max))
if A[y']-A[y]>=A[x']-A[x] AND A[y']-A[y]>=A[z']-A[z]
  then return (y,y',min(l_min,r_min),max(l_max,r_max))
if A[z']-A[z]>=A[x']-A[x] AND A[z']-A[z]>=A[y']-A[y]
  then return (z,z',min(l_min,r_min),max(l_max,r_max))
```

A `JavítottMaximálisNövekedés(A,l,r)` eljárás az $(i, j, \min, \max)$ értékekkel tér vissza, ahol $l \leqslant i \leqslant j \leqslant r$ olyan indexek, amelyekre $A[j] - A[i]$

maximális az $A[l:r]$ résztömbön, továbbá min és max az $A[l:r]$ résztömb egy minimális, illetve egy maximális elemének indexe.

Most az algoritmus költségére a

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(1) & \text{ha } n > 1, \\ O(1) & \text{ha } n = 1 \end{cases}$$

rekurzív képlet adódik. Az egyszerűség kedvéért legyen ismét $n = 2^k$. Ekkor alkalmas c_1 és c_2 pozitív konstansokkal

$$\begin{aligned} T(n) &\leqslant c_1 + 2T(n/2) \\ &\leqslant c_1 + 2(c_1 + 2T(n/2^2)) \\ &= c_1 + 2c_1 + 2^2 T(n/2^2) \\ &\leqslant c_1 + 2c_1 + 2^2(c_1 + 2T(n/2^3)) \\ &= c_1 + 2c_1 + 2^2 c_1 + 2^3 T(n/2^3) \leqslant \cdots \\ &\leqslant c_1 + 2c_1 + 2^2 c_1 + \cdots + 2^{k-1} c_1 + 2^k T(n/2^k) \\ &= c_1(1 + 2 + 2^2 + \cdots + 2^{k-1}) + 2^k T(n/2^k) \\ &= c_1(2^k - 1) + 2^k T(n/2^k) \\ &= c_1(n-1) + nT(1) \\ &\leqslant c_1(n-1) + c_2 n \\ &= (c_1 + c_2)n - c_1. \end{aligned}$$

Innen

$$T(n) = O(n).$$

Legközelebbi pontpár

Feladat.
Adott egy $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ ponthalmaz a síkon. Szokásos módon, a $p_i = (x_i, y_i)$ és $p_j = (x_j, y_j)$ pontok távolsága

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Adjunk hatékony algoritmust a két legközelebbi pont meghatározására (ha több ilyen pár is van, akkor megelégszünk az egyikkel)!

Megoldás.
A legegyszerűbb megközelítési mód a következő: nézzük végig az összes pontpárt, és válasszuk ki hol lesz a távolság (négyzete) minimális.

```
dist(p,q)
return (q.x-p.x)^2+(q.y-p.y)^2;

LegközelebbiPontpár(P,n)
min_dist=INFINITY
for i=1 to n-1 do
  for j=i+1 to n do
    d=dist(P[i],P[j])
    if d<min_dist then
      min_dist=d
      min_i=i
      min_j=j
return (P[min_i],P[min_j])
```

Az algoritmus költsége arányos a pontpárok számával, ami

$$\binom{n}{2} = \frac{n(n-1)}{2} = O(n^2).$$

A következőkben a legközelebbi pontpár meghatározására egy oszd meg és uralkodj algoritmust ismertetünk, amelynek költsége $O(n \log n)$. Ez jóval kedvezőbb, mint az előző algoritmus költsége.

Minden rekurzív hívásnál átadásra kerülnek a $\mathcal{P}$ ponthalmaz egy $\mathcal{Q}$ részhalmazának pontjai egy, az x koordináták szerint monoton növekvően rendezett X tömbben (azonos x koordinátájú pontok esetén a kisebb y koordinátájú jön előbb), valamint egy, az y koordináták szerint monoton növekvően rendezett Y tömbben (azonos y koordinátájú pontok esetén a kisebb x koordinátájú jön előbb). Hogy ne kelljen minden egyes rekurzív hívás előtt rendezésre pazarolni az időt, már az algoritmus elején előállítjuk $\mathcal{P}$ pontjainak az x, illetve az y koordináták szerint monoton növekvően rendezett tömbjeit, így a rekurzív hívások előtt csak kiválogatásokat kell végezni.

Egy adott rekurzív lépés a következő. Ha $|\mathcal{Q}| \leq 3$, akkor páronkénti vizsgálattal határozzuk meg a minimális távolságot. Természetesen az érdekes eset, amikor $|\mathcal{Q}| > 3$, ilyenkor az alábbiak szerint járunk el.

Először egy olyan ℓ függőleges egyenest keresünk, amely $\mathcal{Q}$-t két olyan $\mathcal{Q}_L$ és $\mathcal{Q}_R$ részre osztja, amelyekre

- $|\mathcal{Q}_L| = \lceil |\mathcal{Q}|/2 \rceil$ és $|\mathcal{Q}_R| = \lfloor |\mathcal{Q}|/2 \rfloor$,
- $\mathcal{Q}_L$ minden pontja az ℓ egyenes bal oldalán van vagy illeszkedik ℓ-re,
- $\mathcal{Q}_R$ minden pontja az ℓ egyenes jobb oldalán van vagy illeszkedik ℓ-re.

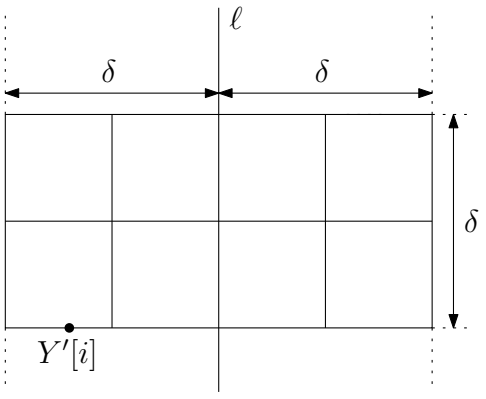
Válogassuk szét a pontokat az X tömbből az X_L és X_R tömbökbe: X_L-be kerüljenek a $\mathcal{Q}_L$-beli pontok, X_R-be pedig a $\mathcal{Q}_R$-beliek. Az X_L és X_R tömbök is rendezettek az x koordináták szerint. Válogassuk szét a pontokat az Y tömbből is az Y_L és Y_R tömbökbe: Y_L-be kerüljenek a $\mathcal{Q}_L$-beli pontok, Y_R-be pedig a $\mathcal{Q}_R$-beliek. Az Y_L és Y_R tömbök is rendezettek az y koordináták szerint.

Ezután keressük meg rekurzívan a $\mathcal{Q}_L$ és $\mathcal{Q}_R$ halmazokban a legközelebbi pontpárt. Az első hívás bemenete az X_L és Y_L tömbök, míg a második hívásé az X_R és Y_R tömbök. Legyen $\mathcal{Q}_L$-ben, illetve $\mathcal{Q}_R$-ben a minimális távolság δ_L, illetve δ_R, és legyen $\delta = \min(\delta_L, \delta_R)$. Most a legközelebbi pontpár

- vagy az egyik rekurzív hívás által megtalált δ távolságú páros,
- vagy egy olyan pár, amelynek egyik pontja $\mathcal{Q}_L$-ben, a másik $\mathcal{Q}_R$-ben van.

A következő lépés annak meghatározása, hogy ez utóbbi párok között van-e olyan, amelyben a pontok távolsága kisebb, mint δ. Vegyük észre, hogy ha van ilyen pár, akkor annak egyik pontja sem lehet δ-nál nagyobb távolságra ℓ-től. Válogassuk ki az Y tömbből azokat a pontokat, amelyek ℓ-től mért távolsága legfeljebb δ. Jelölje a kapott tömböt Y'. Az Y' tömb is rendezett az y koordináták szerint.

Most minden egyes $p \in Y'$ ponthoz keressük meg Y' azon pontjait, melyek p-től mért távolsága legfeljebb δ, és amelyek a p-n átmenő vízszintes egyenesen vagy az fölött vannak. Nem nehéz látni, hogy legfeljebb 7 ilyen pont jöhet számításba. Valóban, ha $i < j$ és $Y'[i]$ valamint $Y'[j]$ távolsága legfeljebb δ, akkor ez a két pont az ábrán látható $\delta \times 2\delta$ oldalú téglalapban van.



Ám a téglalapban lévő 8 kis négyzet egyikében sem lehet egynél több pont Y'-ből (az ℓ egyenesen lévő $\mathcal{Q}_L$-beli pontokat a bal oldali, a $\mathcal{Q}_R$-beli pontokat

pedig a jobb oldali kis négyzetekbe sorolva), hiszen egy ilyen kis négyzet átmérője $\delta\sqrt{2}/2 < \delta$. Innen az állítás adódik.

Így minden $p \in Y'$ pontra elég kiszámolni p távolságát az Y'-ben utána következő 7 ponttól. Az így kapott távolságok minimuma legyen δ'. Világos, hogy ha $\delta' < \delta$, akkor $\mathcal{Q}$-ban a minimális távolság δ', egyébként a minimális távolság δ.

Jelölje $T(n)$ a költséget a kezdeti előrendezések nélkül egy n elemű ponthalmaz esetén. A rekurzív hívások költsége $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, ehhez jön még az X_L, X_R, Y_L, Y_R és Y' rendezett tömbök meghatározásának, illetve δ' kiszámításának költsége. Ez utóbbi műveletek költsége $O(n)$, így a költségre az előrendezések nélkül a

$$T(n) = \begin{cases} T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) & \text{ha } n > 3, \\ O(1) & \text{ha } n \leq 3. \end{cases}$$

rekurzív képlet adódik. Lényegében ugyanezzel a rekurzív képletettel találkoztunk a maximális növekedésről szóló feladatnál, az ottani gondolatmenet szerint ennek megoldása

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

Mivel az előrendezések is megvalósíthatók $O(n \log n)$ költséggel (például összefésüléses rendezést használva), ezért az algoritmus teljes költsége szintén $O(n \log n)$.

Legkisebb és legnagyobb elem

Feladat.
Adjunk hatékony algoritmust, amely meghatározza egy n elemű tömb legkisebb és legnagyobb elemét!

Megoldás.
Ha külön keressük a minimumot és a maximumot, akkor összesen $2n - 2$ összehasonlítással célt érünk. Azonban ennél jóval kevesebb összehasonlítás is elég!

Elemek helyett elempárokkal fogunk dolgozni. A bemeneti elempárokat először egymással hasonlítjuk össze. Ezután a minimum a kisebbek, a maximum a nagyobbak közül kerül ki.

Ha n páros, a minimum és a maximum kezdeti értékét az első két elem összehasonlítása után állítjuk be. Ha n páratlan, a minimum és a maximum kezdeti értéke is az első elem.

Az összehasonlítások száma páros n esetén

$$n/2 + 2(n/2 - 1) = 3n/2 - 2,$$

páratlan n esetén

$$(n-1)/2 + 2(n-1)/2 = 3(n-1)/2,$$

ami mindkét esetben legfeljebb $3\lfloor n/2 \rfloor$.

Második legnagyobb elem

Feladat.
Mutassuk meg, hogy egy n elemű tömb második legnagyobb elemének meghatározásához $n + \lceil \log_2 n \rceil - 2$ összehasonlítás elegendő! Feltehetjük, hogy az elemek mind különbözők.

Megoldás.
Állítsuk az elemeket párokba, hasonlítsuk össze a párok tagjait, majd a nagyobbnak adódott elemekkel járjunk el ugyanígy. Végül $\lceil \log_2 n \rceil$ menetben kiderül, hogy melyik a legnagyobb elem. Összesen $n-1$ összehasonlítást végzünk, mivel a legnagyobbikon kívül minden elem pontosan egyszer bizonyul az összehasonlítások során kisebbik elemnek.

A második legnagyobb elem nyilván azon elemek között van, amelyek csak a legnagyobb elemmel történő összehasonlítások során bizonyultak kisebbik elemnek. Ilyen jelöltekből $\lceil \log_2 n \rceil$ van, közülük $\lceil \log_2 n \rceil - 1$ összehasonlítással eldönthető, hogy melyik a legnagyobb.

Öt szám közül a középső

Feladat.
Mutassuk meg, hogy különböző számok egy $A[1:5]$ tömbjének elemei közül a középső 6 összehasonlítással meghatározható!

Megoldás.

```
KözépsőElem(A)
if A[2]<A[1] then Csere(A[1],A[2])
if A[4]<A[3] then Csere(A[3],A[4])
if A[3]<A[1] then
  Csere(A[1],A[3])
  Csere(A[2],A[4])
if A[5]<A[2] then Csere(A[2],A[5])
```

```
if A[3]<A[2] then
  Csere(A[2],A[3])
  Csere(A[5],A[4])
if A[5]<A[3] then Csere(A[3],A[5])
return A[3]
```

Az első három összehasonlítás (és esetleges cserék) után $A[1] < A[2]$, $A[3] < A[4]$ és $A[1] < A[3]$. Ennélfogva $A[1]$ biztos kisebb, mint $A[2]$, $A[3]$ és $A[4]$, így középső elemként nem jöhet szóba. Marad tehát négy elem, amelyek közül a másodikat kell kiválasztani. Ráadásul itt azt is tudjuk, hogy $A[3] < A[4]$.

A következő két összehasonlítás (és esetleges cserék) után $A[2] < A[5]$, $A[3] < A[4]$ és $A[2] < A[3]$. Ennélfogva $A[2]$ biztos kisebb, mint $A[3]$, $A[4]$ és $A[5]$, így második elemként nem jöhet szóba, továbbá $A[4]$ biztos nagyobb, mint $A[2]$ és $A[3]$, így második elemként szintén nem jöhet szóba. Marad tehát $A[3]$ és $A[5]$, melyek közül a kisebb a keresett elem.

A hatodik összehasonlítás (és esetleges csere) után $A[3]$ a keresett elem.

A k-adik elem kiválasztása

Feladat.
Adott különböző valós számok egy $A[1 : n]$ tömbje és egy $1 \leq k \leq n$ egész. Határozzuk meg a tömb k-adik elemét! Az eljárás költsége $O(n)$ legyen!

Megoldás.
Először egy olyan módszert ismertetünk, amelynél az összehasonlítások számának várható értéke $O(n)$. Hívjuk meg az $A[1 : n]$ tömbre a `Partícionálás` eljárást, amelyet a gyorsrendezés is használ. Legyen j a véletlen partícionáló elem (átrendezés utáni) indexe, ezzel tér vissza a `Partícionálás` eljárás.

- Ha $j = k$, akkor kész vagyunk, a k-adik elem $A[j]$.
- Ha $j > k$, akkor a k-adik elemet az $A[1 : j - 1]$ résztömbben keressük tovább rekurzívan, szintén mint k-adik elemet.
- Ha $j < k$, akkor a k-adik elemet az $A[j + 1 : n]$ résztömbben keressük tovább rekurzívan, de most mint $(k - j)$-edik elemet, hiszen tudjuk, hogy az $A[1 : j]$ résztömb minden eleme megelőzi a k-adik elemet.

Lássuk az algoritmust ezek után. Az algoritmust az $(A, 1, n, k)$ paraméterekkel kell meghívni, és a tömb k-adik elemét adja vissza.

```
Kiválaszt(A,p,r,i)
if p=r then return A[p]
q=Partícionálás(A,p,r)
s=q-p+1
if i=s
  then
    return A[q]
  else
    if i<s
      then return Kiválaszt(A,p,q-1,i)
      else return Kiválaszt(A,q+1,r,i-s)
```

Jelölje $T(n,k)$ az algoritmus összehasonlítások számában mért költségének várható értékét. Nyilván $T(1,1)=0$. Ha pedig $n>1$, akkor a partícionáló elem véletlen választásának köszönhetően fennáll a következő rekurzív összefüggés:

$$T(n,k) = n-1+\frac{1}{n}\left(\sum_{j=1}^{k-1} T(n-j,k-j) + \sum_{j=k+1}^{n} T(j-1,k)\right).$$

Teljes indukcióval belátjuk, hogy $T(n,k)\leqslant 4n$. Ez nyilván igaz, ha $n=1$. Nagyobb n-ekre pedig az indukciós feltevést használva

$$\begin{aligned}
T(n,k) &\leqslant n-1+\frac{1}{n}\left(\sum_{j=1}^{k-1} 4(n-j) + \sum_{j=k+1}^{n} 4(j-1)\right)\\
&= n-1+\frac{4}{n}\left(\sum_{j=1}^{k-1} (n-j) + \sum_{j=k+1}^{n} (j-1)\right)\\
&= n-1+\frac{4}{n}\left(\sum_{j=1}^{k-1} (n-j) + \sum_{j=k}^{n-1} j\right)\\
&= n-1+\frac{4}{n}\left((k-1)n - \sum_{j=1}^{k-1} j + \sum_{j=1}^{n-1} j - \sum_{j=1}^{k-1} j\right)\\
&= n-1+\frac{4}{n}\left((k-1)n - 2\sum_{j=1}^{k-1} j + \sum_{j=1}^{n-1} j\right)\\
&= n-1+\frac{4}{n}\left((k-1)n - (k-1)k + \frac{(n-1)n}{2}\right)\\
&= n-1+\frac{4}{n}\left((k-1)(n-k) + \frac{(n-1)n}{2}\right).
\end{aligned}$$

A számtani és mértani közép közötti egyenlőtlenség szerint

$$(k-1)(n-k) \leqslant \left(\frac{(k-1)+(n-k)}{2}\right)^2 = \frac{(n-1)^2}{4},$$

így

$$\begin{aligned} T(n,k) &\leqslant n-1+\frac{4}{n}\left(\frac{(n-1)^2}{4}+\frac{(n-1)n}{2}\right) \\ &= n-1+\frac{(n-1)^2}{n}+2(n-1) \\ &\leqslant 4(n-1) < 4n. \end{aligned}$$

A következő eljárásnál az összehasonlítások száma legrosszabb esetben is $O(n)$.

(1) Ha $n < 25$, akkor a halmaz k-adik elemét az elemeket rendezve határozzuk meg. Ehhez $12n$ összehasonlítás bőven elég (pl. buborék rendezés).

(2) Tegyük fel, hogy $n \geq 25$. Az n elemet ötös csoportokba soroljuk, elhagyva az esetleg kimaradó $1-4$ elemet. A csoportok száma $[n/5]$.

(3) Minden ötös csoportnak meghatározzuk a rendezés szerinti középső elemét. Ez az előző feladat szerint csoportonként legfeljebb 6 összehasonlítást igényel, így itt a költség legfeljebb $6[n/5]$.

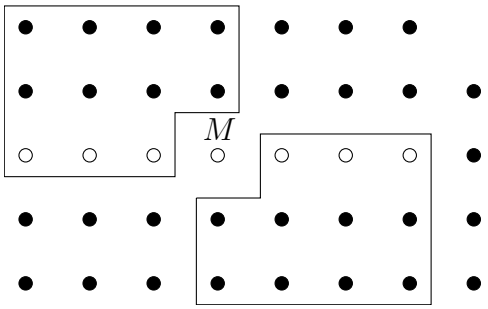
(4) Az imént meghatározott $[n/5]$ középső elem közül a kiválasztási eljárásunkkal rekurzívan meghatározzuk a középső elemet. Ez az M elem nem kisebb azon ötös csoportok három legkisebb eleménél, melyek középső elemei nem nagyobbak M-nél. Ezek száma nyilván legalább

$$3[([n/5]+1)/2] = 3[(n+5)/10].$$

Könnyű ellenőrizni, hogy

$$3[(n+5)/10] > [n/4],$$

ha $n \geq 25$. Így ha $n \geq 25$, akkor M nagyobb legalább $[n/4]$ elemnél. Hasonlóan adódik, hogy ha $n \geq 25$, akkor M kisebb legalább $[n/4]$ elemnél.



(5) A gyorsrendezésnél is használt módszerrel rendezzük át az A tömböt úgy, hogy elől legyenek az M-nél kisebb, hátul pedig az M-nél nagyobb elemek. Ez $n-1$ összehasonlítást igényel. Jelölje A' és A'' az A tömbnek az M elem előtti, illetve az M elem utáni részét. Az előzőek szerint az A' és A'' résztömbök elemszáma legalább $[n/4]$.

(6) Ha $k = |A'| + 1$, akkor a k-adik elem éppen M. Ha $k \leq |A'|$, akkor a kiválasztási eljárásunkkal rekurzívan keressük meg A'-ben a k-adik elemet. Végül ha $k > |A'|+1$, akkor ismét csak a kiválasztási eljárásunkkal rekurzívan keressük meg A''-ben a $(k - |A'| - 1)$-edik elemet.

Jelölje $T(n)$ az összehasonlítások számát n elem és tetszőleges k mellett. Világos, hogy az algoritmus költségére fennáll a

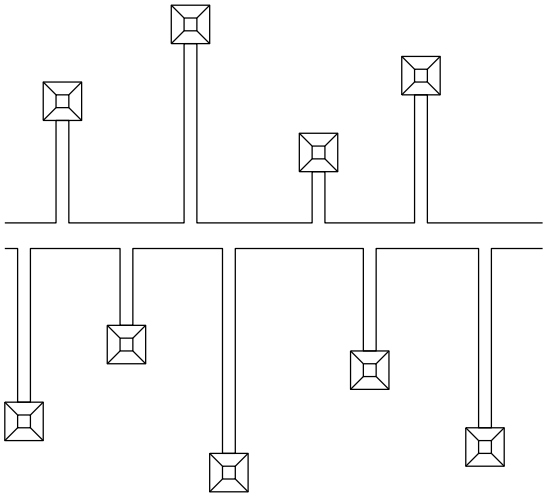
$$T(n) \leq \begin{cases} 12n & \text{ha } n \leq 24, \\ n + 6[n/5] + T([n/5]) + T([3n/4]) & \text{ha } n > 24 \end{cases}$$

rekurzív összefüggés. Innen teljes indukcióval adódik, hogy $T(n) \leq 44n$. Ez nyilván igaz, ha $n \leq 24$. Nagyobb n-ekre pedig az indukciós feltevést használva

$$\begin{aligned} T(n) &\leq n + 6[n/5] + T([n/5]) + T([3n/4]) \\ &\leq n + 6[n/5] + 44[n/5] + 44[3n/4] \\ &\leq n + 6 \cdot n/5 + 44 \cdot n/5 + 44 \cdot 3n/4 \\ &= 20n/20 + 6 \cdot 4n/20 + 44 \cdot 4n/20 + 44 \cdot 15n/20 \\ &= 44 \cdot 20n/20 = 44n. \end{aligned}$$

Fővezeték tervezése

Feladat.
Egy olajvállalat olyan fővezetéket tervez, amelyik egy n kutat működtető olajmezőt szel át keletről nyugat felé. Mindegyik kút északról vagy délről közvetlen csővel, a legrövidebb úton csatlakozik a fővezetékhez.



A kutak x és y koordinátáit ismerve, adjunk lineáris költségű algoritmust a fővezeték optimális helyének meghatározására, amikor is a bekötő csövek hosszának az összege minimális! Feltehetjük, hogy az x, illetve az y koordináták mind különbözők.

Megoldás.
Először azzal az esettel foglalkozunk, amikor $n = 2k + 1$ páratlan szám. Megmutatjuk, hogy ekkor az optimális fővezeték illeszkedik arra az O kútra, amelynek y koordinátája a $(k + 1)$-edik a kutak y koordinátái között.

Jelölje s a bekötő csövek hosszának összegét abban a helyzetben, amikor a fővezeték illeszkedik az O kútra. Tekintsünk most egy olyan helyzetet, amikor a fővezeték az előző helyzettől $d > 0$ távolságra van, mondjuk észak felé (a másik eset hasonlóan intézhető el). Jelölje s' a bekötő csövek hosszának összegét ebben a helyzetben. Az O kútnak, továbbá az O alatt levő k kútnak a bekötő csöve d-vel növekedett az előző helyzethez képest. Azok a kutak, amelyek bekötő csöve rövidebb lett az előző helyzethez képest, mind az O kút fölött vannak, így számuk legfeljebb k. Ezen kutak bekötő csövének hossza nyilván legfeljebb d-vel csökkenhetett. Következésképpen

$$s' \geqslant s + (k + 1)d - kd = s + d > s,$$

azaz a bekötő csövek hosszának összege most nagyobb az előző helyzethez képest. Ebből következik az állítás.

Ezután nézzük azt az esetet, amikor $n = 2k$ páros szám. Megmutatjuk, hogy ekkor az optimális fővezeték azon O' és O'' kutak között halad (beleértve azt a helyzetet is, amikor valamelyikre illeszkedik), amelyek y koordinátái a k-adik, illetve a $(k + 1)$-edik a kutak y koordinátái között.

Jelölje s a bekötő csövek hosszának összegét egy olyan a helyzetben, amikor a fővezeték az O' és O'' kutak között halad (beleértve azt a helyzetet is, amikor valamelyikre illeszkedik). Tekintsünk egy olyan helyzetet, amikor a fővezeték az előző helyzettől $d > 0$ távolságra van, de még mindig az O' és O'' kutak között. Világos, hogy ekkor k kút bekötő csövének hossza d-vel növekedett, és ugyanennyi kút bekötő csövének hossza d-vel csökkent, így a bekötő csövek hosszának összege s maradt. Ennélfogva bármely fővezeték, amely az O' és O'' kutak között halad (beleértve azt a helyzetet is, amikor valamelyikre illeszkedik) a bekötő csövek összhosszának szempontjából egyformán jó.

Mikor a fővezeték illeszkedik az O'' kútra, a bekötő csövek hosszának összege s. Tekintsünk most egy olyan helyzetet, amikor a fővezeték ettől helyzettől $d > 0$ távolságra van észak felé. Jelölje s' a bekötő csövek hosszának összegét ez utóbbi helyzetben. Az O'' kútnak, továbbá az O'' alatt levő k kútnak a bekötő csöve d-vel növekedett az előző helyzethez képest. Azok a kutak, amelyek bekötő csöve rövidebb lett az előző helyzethez képest, mind az O'' kút fölött vannak, így számuk legfeljebb $k - 1$. Ezen kutak bekötő csövének hossza nyilván legfeljebb d-vel csökkenhetett. Következésképpen

$$s' \geqslant s + (k + 1)d - (k - 1)d = s + 2d > s,$$

azaz a bekötő csövek hosszának összege most nagyobb az előző helyzethez képest.

Hasonlóan, mikor a fővezeték illeszkedik az O' kútra, a bekötő csövek hosszának összege szintén s. Tekintsünk most egy olyan helyzetet, amikor a fővezeték ettől helyzettől $d > 0$ távolságra van dél felé. Jelölje s' a bekötő csövek hosszának összegét ez utóbbi helyzetben. Az O' kútnak, továbbá az O' fölött levő k kútnak a bekötő csöve d-vel növekedett az előző helyzethez képest. Azok a kutak, amelyek bekötő csöve rövidebb lett az előző helyzethez képest, mind az O' kút alatt vannak, így számuk legfeljebb $k - 1$. Ezen kutak bekötő csövének hossza nyilván legfeljebb d-vel csökkenhetett. Következésképpen

$$s' \geqslant s + (k + 1)d - (k - 1)d = s + 2d > s,$$

azaz a bekötő csövek hosszának összege most is nagyobb az előző helyzethez képest. Ebből következik az állítás.

Mindezek alapján a fővezeték optimális helyének megtalálásához páratlan n esetén elég azt a kutat megtalálni, amelynek y koordinátája az összes kút y koordinátái közül a középső, illetve páros n esetén azt a két kutat, amelyek y koordinátái az összes kút y koordinátái közül a középsők. Ennek költsége a k-adik elem meghatározására szolgáló algoritmussal $O(n)$.

Többségi elem

Feladat.
Ha egy $A[1:n]$ tömbben van olyan elem, amely több, mint $n/2$ példányban fordul elő, akkor azt a tömb többségi elemének nevezzük (nyilvánvaló, hogy legfeljebb egy többségi elem lehet egy tömbben). A tömb elemei nem feltétlenül egy rendezett halmazból valók, ezért csak azt tudjuk ellenőrizni, hogy két elem megegyezik-e, más típusú összehasonlításokat nem tudunk végezni. Adjunk hatékony algoritmust annak eldöntésére, hogy a tömb tartalmaz-e többségi elemet!

Megoldás.
Vegyük észre, hogy ha $n \geqslant 3$, és az $A[1:n]$ tömbben van többségi elem, továbbá $A[1] \neq A[2]$, akkor az $A[3:n]$ résztömbben is van többségi elem, és az megegyezik az $A[1:n]$ tömb többségi elemével. Ez a következőképpen látható be. Tegyük fel, hogy egy x elem $k > n/2$ példányban fordul elő az $A[1:n]$ tömbben. Most két eset van.

- Ha az x elem nem egyezik meg az $A[1]$ és az $A[2]$ elemek egyikével sem, akkor az x elem $k > n/2 > (n-2)/2$ példányban fordul elő az $A[3:n]$ résztömbben, így az $A[3:n]$ résztömbben is többségi elem.

- Ha az x elem az $A[1]$ és az $A[2]$ elemek közül pontosan az egyikkel egyezik meg, akkor az x elem $k-1 > n/2-1 = (n-2)/2$ példányban fordul elő az $A[3:n]$ résztömbben, így az $A[3:n]$ résztömbben is többségi elem.

Az alábbi algoritmus $O(n)$ költséggel találja meg a tömb többségi elemét, ha létezik.

```
TöbbségiElem(A)
i=0
j=1
while j<=n do
  if A[j]<>A[i+1] then
    i=i+2
    Csere(A[j],A[i])
  j=j+1
if i=n
  then
    return nil
```

```
else
  s=0
  for k=1 to n do
    if A[k]=A[i+1] then s=s+1
  if s>n/2
    then return A[i+1]
    else return nil
```

Az első (while) ciklus leszűkíti a keresést legfeljebb egyetlen jelöltre. Jegyezzük meg, hogy a while ciklus minden iterációs lépésének végrehajtása után teljesülnek a következők:

- az $A[1:i]$ résztömb egymástól különböző elemekből álló párokból áll,
- az $A[i+1, j-1]$ résztömbben azonos elemek vannak.

Ezután ellenőrizzük, hogy $i < n$ esetén a talált jelölt, az $A[i+1]$ elem, valóban több, mint $n/2$ példányban fordul elő a tömbben.

Adatszerkezetek

Verem

Feladat.
A verem olyan adatszerkezet, amelyből a legutoljára beérkezett elemet törölhetjük, így a verem utolsóként be, elsőként ki stratégiát valósít meg. Hogyan valósítható meg egy verem egyetlen $V[1:n]$ tömbben? A műveletek költsége $O(1)$ legyen!

Megoldás.
A vermeken értelmezett `Beszúr` művelet neve `Verembe`, az argumentum nélküli `Töröl` művelet neve pedig `Veremből`.

Egy legfeljebb n elemet tartalmazó verem a következőképpen valósítható meg egy $V[1:n]$ tömbbel. A tömb $teteje[V]$ attribútuma a verembe legutoljára betett elem indexe. A verem a $V[1], V[2], \ldots, V[teteje[V]]$ elemekből áll, amelyek közül $V[1]$ a verem legalsó, $V[teteje[V]]$ pedig a legfelső eleme. Kezdetben $teteje[V] = 0$.

Ha $teteje[V] = 0$, akkor a verem nem tartalmaz elemet és üresnek nevezzük. Az `Üres?` lekérdező művelettel megállapítható, hogy egy verem üres-e. Ha üres veremből próbálunk elemet kivenni, akkor azt mondjuk, hogy a verem alulcsordul; ezt általában hibának tekintjük. Ha $teteje[V] = n$, akkor a verem tele van. A `Tele?` lekérdező művelettel megállapítható, hogy egy verem tele van-e. Ha tele lévő verembe próbálunk elemet beszúrni, akkor azt mondjuk, hogy a verem túlcsordul; általában ezt is hibának tekintjük.

```
Üres?(V)
if teteje[V]=0
  then return igaz
  else return hamis

Tele?(V)
if teteje[V]=n
  then return igaz
```

```
  else return hamis

Verembe(V,x)
if Tele?(V)
  then
    error "túlcsordulás"
  else
    teteje[V]=teteje[V]+1
    V[teteje[V]]=x
```

A művelet megvalósításának költsége nyilván $O(1)$.

```
Veremből(V)
if Üres?(V)
  then
    error "alulcsordulás"
  else
    x=V[teteje[V]]
    teteje[V]=teteje[V]-1
    return x
```

A művelet megvalósításának költsége itt is nyilván $O(1)$.

Sor

Feladat.
A sor olyan adatszerkezet, amelyből a legrégebben beérkezett elemet lehet törölni, így a sor elsőként be, elsőként ki stratégiát valósít meg. Hogyan valósítható meg egy sor egyetlen $S[1 : n]$ tömbben? A műveletek költsége $O(1)$ legyen!

Megoldás.
A sorokon értelmezett `Beszúr` művelet neve `Sorba`, az argumentum nélküli `Töröl` művelet neve pedig `Sorból`.

Egy legfeljebb n elemet tartalmazó sor a következőképpen valósítható meg egy $S[1 : n]$ tömbbel. A tömb *eleje*$[S]$ attribútuma a sor első elemét indexeli, a *vége*$[S]$ attribútum pedig annak a helynek az indexe, amelyre a legközelebb beérkező elemet el fogjuk helyezni. A sorban az elemek rendre az *eleje*$[S]$, *eleje*$[S] + 1, \ldots,$ *vége*$[S] - 1$ indexű helyeket foglalják el a ciklikus elrendezés szabálya szerint: a tömb végére érve az n-edik hely után az első következik. Kezdetben *eleje*$[S] =$ *vége*$[S] = 1$.

Az alul- illetve túlcsordulásból adódó hibák kezelésének megkönnyítésére bevezetjük a $hossz[S]$ attribútumot, amelyben a sorban lévő elemek számát tároljuk. Kezdetben $hossz[S] = 0$.

```
Sorba(S,x)
if hossz[S]=n
  then
    error "túlcsordulás"
  else
    S[vége[S]]=x
    if vége[S]=n
      then vége[S]=1
      else vége[S]=vége[S]+1
    hossz[S]=hossz[S]+1
```

A művelet megvalósításának költsége nyilván $O(1)$.

```
Sorból(S)
if hossz[S]=0
  then
    error "alulcsordulás"
  else
    x=S[eleje[S]]
    if eleje[S]=n
      then eleje[S]=1
      else eleje[S]=eleje[S]+1
    hossz[S]=hossz[S]-1
    return x
```

A művelet megvalósításának költsége itt is nyilván $O(1)$.

Két verem

Feladat.
Hogyan valósítható meg két verem egyetlen $V[1:n]$ tömbben úgy, hogy egyik verem se csorduljon túl addig, amíg együttes elemszámuk nem haladja meg n-et? A műveletek költsége továbbra is $O(1)$ legyen!

Megoldás.
A `Beszúr` műveletek neve `EgyikVerembe` és `MásikVerembe`, az argumentum nélküli `Töröl` műveletek neve pedig `EgyikVeremből` és `MásikVeremből` lesz.

A tömb $EgyikVeremTeteje[V]$, illetve $MásikVeremTeteje[V]$ attribútumai az egyik, illetve a másik verembe legutoljára betett elem indexe. Az egyik verem a $V[1], V[2], \ldots, V[EgyikVeremTeteje[V]]$ elemekből áll, amelyek közül $V[1]$ a verem legalsó eleme, $V[EgyikVeremTeteje[V]]$ a verem legfelső eleme. A másik verem a $V[n], V[n-1], \ldots, V[MásikVeremTeteje[V]]$ elemekből áll, amelyek közül $V[n]$ a verem legalsó eleme, $V[MásikVeremTeteje[V]]$ a verem legfelső eleme.

Ha $EgyikVeremTeteje[V] = 0$, akkor az egyik verem nem tartalmaz elemet és üresnek nevezzük. Az `EgyikVeremÜres?` lekérdező művelettel megállapítható, hogy az egyik verem üres-e. Ha $MásikVeremTeteje[V] = n+1$, akkor a másik verem nem tartalmaz elemet és üresnek nevezzük. A `MásikVeremÜres?` lekérdező művelettel megállapítható, hogy a másik verem üres-e.

Ha $EgyikVeremTeteje[V] = MásikVeremTeteje[V]-1$, akkor a vermek tele vannak. A `Tele?` lekérdező művelettel megállapítható, hogy a vermek tele vannak-e.

Kezdetben $EgyikVeremTeteje[V] = 0$ és $MásikVeremTeteje[V] = n+1$.

```
EgyikVeremÜres?(V)
if EgyikVeremTeteje[V]=0
  then return igaz
  else return hamis

MásikVeremÜres?(V)
if MásikVeremTeteje[V]=n+1
  then return igaz
  else return hamis

Tele?(V)
if EgyikVeremTeteje[V]=MásikVeremTeteje[V]-1
  then return igaz
  else return hamis

EgyikVerembe(V,x)
if Tele?(V)
  then
    error "túlcsordulás"
  else
    EgyikVeremTeteje[V]=EgyikVeremTeteje[V]+1
    V[EgyikVeremTeteje[V]]=x
```

```
MásikVerembe(V,x)
if Tele?(V)
  then
    error "túlcsordulás"
  else
    MásikVeremTeteje[V]=MásikVeremTeteje[V]-1
    V[MásikVeremTeteje[V]]=x

EgyikVeremből(V)
if EgyikVeremÜres?(V)
  then
    error "alulcsordulás"
  else
    x=V[EgyikVeremTeteje[V]]
    EgyikVeremTeteje[V]=EgyikVeremTeteje[V]-1
    return x

MásikVeremből(V)
if MásikVeremÜres?(V)
  then
    error "alulcsordulás"
  else
    x=V[MásikVeremTeteje[V]]
    MásikVeremTeteje[V]=MásikVeremTeteje[V]+1
    return x
```

A műveletek megvalósításának költsége nyilván $O(1)$.

Kétvégű sor

Feladat.
A kétvégű sor olyan adatszerkezet, amelynek mindkét végén lehet elemet beszúrni és elemet törölni is. Hogyan valósítható meg egy kétvégű sor egyetlen $S[1:n]$ tömbben? A műveletek költsége $O(1)$ legyen!

Megoldás.
A `Beszúr` műveletek neve `SorElejére`, illetve `SorVégére`, míg az argumentum nélküli `Töröl` műveletek neve `SorElejéről`, illetve `SorVégéről` lesz.

A tömb *eleje*$[S]$ attribútuma annak a helynek az indexe, amelyre a legközelebbi `SorElejére` művelettel beérkező elemet el fogjuk helyezni. A tömb *vége*$[S]$ attribútuma pedig annak a helynek az indexe, amelyre a legközelebbi

`SorVégére` művelettel beérkező elemet el fogjuk helyezni. A kétvégű sorban az elemek rendre az $eleje[S]+1, eleje[S]+2, \ldots,$ $vége[S]-1$ indexű helyeket foglalják el a ciklikus elrendezés szabálya szerint: a tömb végére érve az n-edik hely után az első következik. Kezdetben $eleje[S] = vége[S] = 1$.

Az alul- illetve túlcsordulásból adódó hibák kezelésének megkönnyítésére bevezetjük a $hossz[S]$ attribútumot, amelyben a sorban lévő elemek számát tároljuk.

```
SorElejére(S,x)
if hossz[S]=n
  then
    error "túlcsordulás"
  else
    S[eleje[S]]=x
    if eleje[S]=1
      then eleje[S]=n
      else eleje[S]=eleje[S]-1
    hossz[S]=hossz[S]+1

SorVégére(S,x)
if hossz[S]=n
  then
    error "túlcsordulás"
  else
    S[vége[S]]=x
    if vége[S]=n
      then vége[S]=1
      else vége[S]=vége[S]+1
    hossz[S]=hossz[S]+1

SorElejéről(S)
if hossz[S]=0
  then
    error "alulcsordulás"
  else
    if eleje[S]=n
      then eleje[S]=1
      else eleje[S]=eleje[S]+1
    hossz[S]=hossz[S]-1
    return S[eleje[S]]

SorVégéről(S)
```

```
if hossz[S]=0
  then
    error "alulcsordulás"
  else
    if vége[S]=1
      then vége[S]=n
      else vége[S]=vége[S]-1
    hossz[S]=hossz[S]-1
    return S[vége[S]]
```

A művelet megvalósításának költsége nyilván $O(1)$.

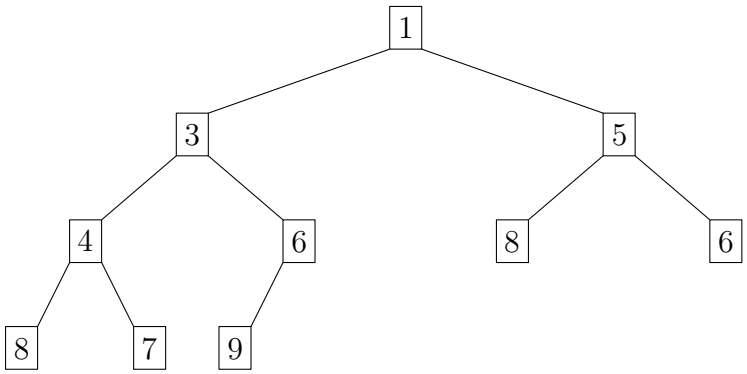
Kupac (elsőbbségi sor)

Feladat.
A kupac (elsőbbségi sor) adatszerkezettel egy rendezett halmaz egy A véges részhalmazát tároljuk. Két művelet tartozik az adatszerkezethez:

- a rendezett halmaz egy elemének hozzávétele az A halmazhoz,
- az A halmaz minimális elemének törlése.

Valósítsuk meg minél hatékonyabban az adatszerkezetet! Az A halmaz elemeit egyetlen tömbben tároljuk!

Megoldás.
A kupac adatszerkezetet legegyszerűbben úgy képzelhetjük el, mint egy olyan teljes bináris fa, melyben a gyökértől eltekintve tetszőleges csúcs eleme legalább akkora, mint a csúcs szülőjében lévő elem. A teljes bináris fák levelei legfeljebb két szinten, a legalsón és esetleg a felette lévőn helyezkednek el, és legfeljebb egy kivételével minden nem levél csúcsnak két gyereke van.



Egy *kupacméret*[A] csúcsú teljes bináris fa természetes módon ábrázolható egy $A[1 : \textit{kupacméret}[A]]$ tömbben: $A[1]$ a gyökérben lévő elem, $A[2]$ és $A[3]$ a gyökér bal, illetve jobb gyerekében lévő elemek, $A[4]$ és $A[5]$ a gyökér bal gyerekének bal, illetve jobb gyerekében lévő elemek, $A[6]$ és $A[7]$ a gyökér jobb gyerekének bal, illetve jobb gyerekében lévő elemek, és így tovább, általánosan, egy csúcsban, illetve annak bal és jobb gyerekében lévő elemek a tömb i, illetve $2i$ és $2i + 1$ indexű helyén találhatók. A teljes bináris fákra fent megfogalmazott kupac tulajdonság a tömbben azt jelenti, hogy minden $2 \leqslant i \leqslant \textit{kupacméret}[A]$ indexre $A[i] \geqslant A[[i/2]]$. Íme az előző példának megfelelő tömb:

1	2	3	4	5	6	7	8	9	10
1	3	5	4	6	8	6	8	7	9

Foglaljunk le egy $A[1 : \textit{maxkupacméret}[A]]$ tömböt az adatszerkezetnek, ahol *maxkupacméret*[A] egy észszerű felső korlát a kupac elemszámára. Az A tömbnek mindig csak egy $A[1 : \textit{kupacméret}[A]]$ résztömbjét használjuk a kupac tárolására, ahol *kupacméret*[A] a kupac aktuális elemszáma. Kezdetben *kupacméret*[A] $= 0$.

Mivel több alkalmazásnál is előjön, először azt vizsgáljuk, hogy miképpen tudunk az $A[1 : \textit{kupacméret}[A]]$ tömbben tetszőlegesen elhelyezett elemekből kupacot építeni. A tömböt az előbb ismertetett módon teljes bináris fának képzelve, legyen f a fa egy csúcsa, f_1 és f_2 az f bal, illetve jobb gyereke. Tegyük fel, hogy a kupac építésénél pillanatnyilag ott tartunk, amikor az f_1 és f_2 gyökerű részfák már kupacok. Ha az f gyökerű részfa nem kupac, akkor szükségképpen az f csúcs eleme nagyobb, mint az f_1 vagy az f_2 csúcs eleme. Ha f_1 eleme kisebb, mint f_2 eleme, akkor az f csúcs elemét cseréljük fel az f_1 csúcs elemével, és hívjuk meg rekurzívan ugyanezt az eljárást az f_1 gyökerű részfára. Ellenkező esetben az f csúcs elemét cseréljük fel az f_2 csúcs elemével, és hívjuk meg rekurzívan ugyanezt az eljárást az f_2 gyökerű részfára. Az eljárás végrehajtása után nyilván már az f gyökerű részfára is teljesülni fog a kupac tulajdonság.

```
Kupacol(A,i)
l=2i
r=2i+1
if l<=kupacméret[A] AND A[l]<A[i]
  then legkisebb=l
  else legkisebb=i
if r<=kupacméret[A] AND A[r]<A[legkisebb]
  then legkisebb=r
```

```
if legkisebb<>i then
  csere(A[i],A[legkisebb])
  Kupacol(A,legkisebb)
```

Ezek után a kupacépítés algoritmusa már egyszerű: a fa csúcsaira lentről felfelé, szintenként jobbról balra haladva hívjuk meg a `Kupacol` eljárást.

```
KupacotÉpít(A)
for i=kupacméret[A] downto 1 do
  Kupacol(A,i)
```

Megjegyezzük, hogy a $[\mathit{kupacméret}[A]/2]$-nél nagyobb indexű csúcsok levelek, ezért az i ciklusváltozó gond nélkül indulhat ettől az indextől is.

Próbáljuk megbecsülni egy n elemű kupac építésének (összehasonlításokban mért) költségét! Legyen a teljes bináris fa szintjeinek száma l (az első szint az, ahol a gyökér van). Mivel a fa levelei az utolsó két szinten helyezkednek el, ezért $2^{l-1} \leqslant n$. Ebből speciálisan azonnal következik, hogy $l \leqslant 1 + \log_2 n$. Minden $1 \leqslant i \leqslant l$ esetén az i-edik szinten legfeljebb 2^{i-1} csúcs lehet, és az ezen a szinten elhelyezkedő csúcsokra a `Kupacol` eljárás (a rekurzív hívásokkal együtt) legfeljebb $2(l-i)$ összehasonlítást végez. Így az egész algoritmus során az összehasonlítások száma legfeljebb

$$\sum_{i=1}^{l} 2(l-i)2^{i-1} = \sum_{i=1}^{l} (l-i)2^i.$$

A $j = l - i$ (azaz $i = l - j$) helyettesítés után az összehasonlítások száma így becsülhető:

$$\sum_{j=0}^{l-1} j2^{l-j} = 2^l \sum_{j=0}^{l-1} \frac{j}{2^j}.$$

Teljes indukcióval megmutatjuk, hogy

$$\sum_{j=0}^{l-1} \frac{j}{2^j} = 2 - \frac{l+1}{2^{l-1}}$$

minden l pozitív egész számra. Ha $l = 1$, akkor ez triviálisan teljesül. Legyen ezután $l > 1$ és tegyük fel, hogy minden l-nél kisebb pozitív egészre igaz az állítás. Most használva az indukciós feltevést

$$\sum_{j=0}^{l-1} \frac{j}{2^j} = 2 - \frac{l}{2^{l-2}} + \frac{l-1}{2^{l-1}} = 2 - \frac{2l-l+1}{2^{l-1}} = 2 - \frac{l+1}{2^{l-1}}.$$

Innen az összehasonlítások számára a $2^{l+1} \leqslant 4n$ felső korlát adódik.

Következzenek az adatszerkezethez tartozó műveletek! Lássuk először a minimális elemet törlő eljárást! Világos, hogy a minimális elem a fa gyökerében van. Töröljük a kupacból ezt az elemet, tegyük a gyökérbe a fa legalsó szintjének legjobboldalibb csúcsában tárolt elemet, töröljük magát a csúcsot, majd a kupac tulajdonság helyreállításához hívjuk meg a gyökérre a `Kupacol` eljárást. A költség nyilván $O(\log n)$ egy n elemű kupac esetén.

```
Mintöröl(A)
if kupacméret[A]=0 then
  error "alulcsordulás"
min=A[1]
A[1]=A[kupacméret[A]]
kupacméret[A]=kupacméret[A]-1
Kupacol(A,1)
return min
```

Lássuk ezután az új elemet beszúró eljárás! Adjunk egy új levelet a fához ügyelve a teljesség megtartására. Tegyük ebbe a levélbe az új elemet, majd az új elem felfelé mozgatásával állítsuk helyre a kupac tulajdonságot. A költség itt is nyilván $O(\log n)$ egy n elemű kupac esetén.

```
Beszúr(A,újelem)
if kupacméret[A]=maxkupacméret[A] then
  error "túlcsordulás"
kupacméret[A]=kupacméret[A]+1
i=kupacméret[A]
A[i]=újelem
while i>1 AND A[[i/2]]>A[i] do
  csere(A[i],A[[i/2]])
  i=[i/2]
```

Megjegyezzük, hogy a kupac adatszerkezet lehetőséget ad arra is, hogy egy n elemű tömböt konstans méretű segédmemóriát használva $O(n \log n)$ költséggel rendezzünk.

```
Kupacrendezés(A)
KupacotÉpít(A)
n=kupacméret[A]
for i=n downto 2 do
  csere(A[1],A[i])
  kupacméret[A]=kupacméret[A]-1
  Kupacol(A,1)
```

Láncolt lista megfordítása

Feladat.
Adjunk olyan nem rekurzív eljárást, amely csupán konstans méretű segédmemóriát használva $O(n)$ költséggel megfordít egy n elemű egyszeresen láncolt listát!

Megoldás.
Az egyszeresen láncolt listákban az elemek a *kulcs* mező mellett egyetlen mutató mezőt tartalmaznak, amelyet *köv* azonosít. A lista egy x eleme esetén $köv[x]$ az x rákövetkezőjére mutat a listában. Ha $köv[x] = nil$, akkor az x elemnek nincs rákövetkezője, vagyis x a lista utolsó eleme.

A listának egyetlen attribútuma van, az első elemre mutató $fej[L]$ pointer. Ha $fej[L] = nil$, akkor a lista üres.

A következő eljárás egymás után törli az elemeket az L lista elejéről, és beszúrja az S lista elejére (az S lista kezdetben üres). Végül a $fej[L]$ attribútum értékét $fej[S]$-re állítja.

```
LáncoltListátFordít(L)
fej[S]=nil
while fej[L]<>nil do
  x=fej[L]
  fej[L]=köv[x]
  köv[x]=fej[S]
  fej[S]=x
fej[L]=fej[S]
```

Az eljárás költsége nyilván $O(n)$.

Láncolt lista rendezése

Feladat.
Adjunk olyan eljárást, amely csupán konstans méretű segédmemóriát használva $O(n^2)$ költséggel rendez egy n elemű egyszeresen láncolt listát!

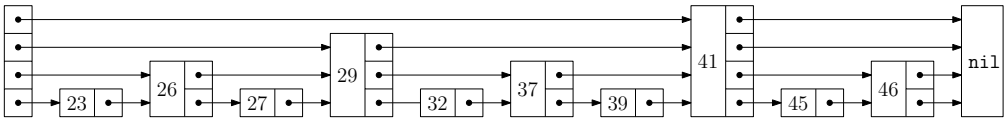
Megoldás.
A következő eljárás egymás után törli az elemeket az L lista elejéről, és beszúrja az S listába a beillesztéses rendezésnél látottakhoz hasonló módon (az S lista kezdetben üres). Végül a $fej[L]$ attribútum értékét $fej[S]$-re állítja.

```
LáncoltListátRendez(L)
fej[S]=nil
while fej[L]<>nil do
  x=fej[L]
  fej[L]=köv[x]
  y=fej[S]
  if y=nil
    then
      köv[x]=nil
      fej[S]=x
    else
      if kulcs[x]<=kulcs[y]
        then
          köv[x]=y
          fej[S]=x
        else
          while köv[y]<>nil AND kulcs[köv[y]]<kulcs[x] do
            y=köv[y]
          köv[x]=köv[y]
          köv[y]=x
fej[L]=fej[S]
```

Az eljárás költsége nyilván $O(n^2)$.

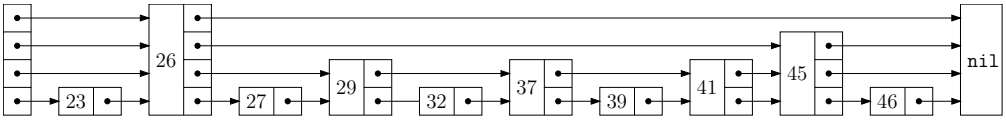
Ugráló lista

Feladat.
Egy n elemű rendezett láncolt listában egy elem megkereséséhez a lista összes elemének vizsgálatára szükség lehet. Ha a láncolást megvalósító pointereken felül minden második elemből egy pointer két elemmel előbbre mutat, akkor a keresésnél elég legfeljebb $\lceil (n+1)/2 \rceil$ elemet megvizsgálni. Ha ezen kívül még minden negyedik elemből egy pointer négy elemmel előbbre mutat, akkor a keresésnél elég legfeljebb $\lceil (n+1)/4 \rceil + 1$ elemet megvizsgálni. Általában, ha a láncolást megvalósító pointereken felül minden 2^i-edik elemben elhelyezünk egy 2^i elemmel előbbre mutató pointert ($i = 1, 2, \ldots, [\log_2 n]$), akkor a keresésnél elég legfeljebb $\lceil \log_2(n+1) \rceil$ elemet megvizsgálni. Jegyezzük meg, hogy mindehhez a pointerek számát csupán kétszeresére növeltük.



Ebben az adatszerkezetben a keresés művelet gyorsan megvalósítható, azonban a beszúrással és a törléssel gondok vannak.

Egy olyan elemet, amelynek pontosan k előremutató pointere van k magasságú elemnek fogunk nevezni. Világos, hogy ha minden 2^i-edik elem tartalmaz egy 2^i elemmel előbbre mutató pointert ($i = 0, 1, \ldots, [\log_2 n]$), akkor az elemek 50%-ának egy a magassága, 25%-ának kettő, $12,5$%-ának három, és így tovább. Mi lenne, ha az elemek magasságát véletlenszerűen választanánk meg, csupán arra ügyelnénk, hogy a fenti arányok megmaradjanak? Persze ekkor egy legalább i magasságú elemnek az (alulról számított) i-edik előremutató pointere általában nem 2^{i-1} elemmel mutatna előre, hanem a listában utána következő, legalább i magasságú elemre. Az adatszerkezet neve ugráló lista.



Tegyük fel, hogy az elemek kulcsai különbözők. Implementáljuk az ugráló lista keresés, beszúrás és törlés műveleteit! A műveletek költségének várható értéke $O(\log n)$ legyen n elemű ugráló lista esetén!

Megoldás.
Egy elem magasságát véletlenszerűen választjuk meg mikor beszúrjuk, az adatszerkezet aktuális elemszámára való tekintet nélkül. Egy k magasságú elemnek k előremutató pointere van, ezeket 1-től k-ig indexeljük, alulról felfelé. Látni fogjuk, hogy egy elem magasságát nem szükséges az elemben tárolni. A lista legnagyobb magasságú elemének magasságát a lista magasságának nevezzük. Az üres lista magassága megállapodás szerint 1. Programozástechnikai okból a lista magasságát felülről korlátozzuk egy alkalmasan megválasztott `MaxMagasság` konstanssal. Ennek értéke általában $\log_2 N$, ahol N egy észszerű kettőhatvány felső korlát a lista elemszámára.

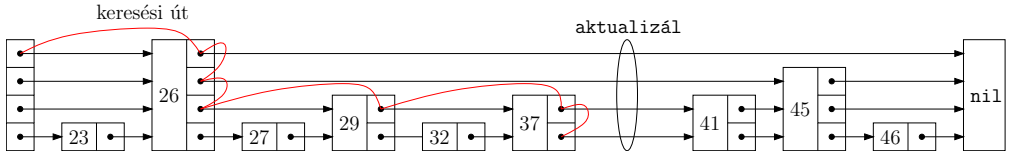
A lista listafej attribútumának `MaxMagasság` számú pointere van, ezek 1-től `MaxMagasság`-ig vannak indexelve, szintén alulról felfelé. Azok a pointerek, amelyek a lista magasságánál nagyobb indexűek `nil`-re mutatnak.

Allokálunk még egy `nil` elemet, amelynek kulcsa nagyobb minden legális kulcsnál. Az ugráló lista minden szintje `nil`-ben végződik. Kezdetben az ugráló lista magassága 1 és a listafej minden pointere `nil`-re mutat.

A keresés megvalósítása egyszerű: egy adott szinten a pointerek mentén addig lépkedünk, hogy éppen ne ugorjuk át a keresett elemet, amikor ez már nem lehetséges, az adott ponttól ugyanezt egy szinttel lejjebb folytatjuk.

```
Keresés(lista,kulcs)
x=lista.listafej
for i=lista.magasság downto 1 do
  while x.következő[i].kulcs<kulcs do
    x=x.következő[i]
{ha a kulcs a listában van, épp előtte állunk legalul}
x=x.következő[1]
if x.kulcs=kulcs
  then return x
  else error "a kulcs nincs a listában"
```

Beszúrás és törlés esetén először keresünk, majd a mutatókat manipuláljuk. Mindkét eljárás felhasznál egy `aktualizál[1:MaxMagasság]` vektort (lokális változó), ennek i-edik koordinátája egy pointer, amely a beszúrás/törlés helyétől balra, azon belül a legjobboldalibb, legalább i magasságú elemre mutat.



Itt épp (mondjuk) a 39 kulcsú elem beszúrása vagy a 41 kulcsú elem törlése zajlik.

Ha a beszúrás olyan elemet generál, amelynek magassága nagyobb, mint az aktuális listamagasság, a listamagasságot növelni kell, valamint inicializálni kell a `aktualizál` vektor megfelelő részét. A törlés után ellenőrizni kell, hogy nem a lista maximális magasságú elemét töröltük-e; ha igen, a listamagasságot csökkenteni kell.

Lássuk először a beszúrás eljárást.

```
Beszúr(lista,kulcs,érték)
x=lista.listafej
for i=lista.magasság downto 1 do
  while x.következő[i].kulcs<kulcs do
    x=x.következő[i]
  aktualizál[i]=x
x=x.következő[1]
if x.kulcs=kulcs
  then
    x.érték=érték
  else
```

```
    újmagasság=VéletlenMagasság()
    if újmagasság>lista.magasság then
      for i=lista.magasság+1 to újmagasság do
        aktualizál[i]=lista.listafej
      lista.magasság=újmagasság
    x=ElemetLétrehoz(újmagasság,kulcs,érték)
    for i=1 to újmagasság do
      x.következő[i]=aktualizál[i].következő[i]
      aktualizál[i].következő[i]=x
```

Egy adósságunk van:

```
VéletlenMagasság()
újmagasság=1
while újmagasság<MaxMagasság AND Random()<1/2 do
  újmagasság=újmagasság+1
return újmagasság
```

Világos, hogy ily módon, elvárásainknak megfelelően, a legalább i magasságú csúcsok fele legalább $i+1$ magasságú lesz.

Jöjjön ezek után a törlés eljárás.

```
Töröl(lista,kulcs)
x=lista.listafej
for i=lista.magasság downto 1 do
  while x.következő[i].kulcs<kulcs do
    x=x.következő[i]
  aktualizál[i]=x
x=x.következő[1]
if x.kulcs=kulcs then
  for i=1 to lista.magasság do
    if aktualizál[i].következő[i]<>x then break
    aktualizál[i].következő[i]=x.következő[i]
  ElemetFelszabadít(x)
  while lista.magasság>1 AND
      lista.listafej.következő[lista.magasság]=nil do
    lista.magasság=lista.magasság-1
```

A beszúrás és törlés eljárások költsége nyilván arányos egy adott elem megkeresésének költségével, ezért elég a keresés eljárás költségével foglalkozni. Egy adott elem megkeresésének költsége arányos a bejárt keresési út hosszával (az összehasonlítások száma ennél eggyel nagyobb).

Egy ugráló lista szerkezete csak az elemek számától és egy véletlenszám generátor által meghatározott értékektől függ, az ugráló listát előállító műveletek sorozata nem számít. Feltesszük, hogy a felhasználónak nincs hozzáférése az elemek magasság értékeihez (pl. egy rosszakaró nem tudja az egynél nagyobb magasságú elemeket csak úgy törölni). A bejárt keresési út hosszának várható értékére adunk felső becslést.

Röviden átismételjük a szükséges valószínűségszámítási fogalmakat. Az X és Y nem negatív, független valószínűségi változókra használni fogjuk az alábbi jelöléseket:

$$X =_{\text{prob}} Y \iff \forall u,\ \text{Prob}\{X > u\} = \text{Prob}\{Y > u\},$$
$$X \leqslant_{\text{prob}} Y \iff \forall u,\ \text{Prob}\{X > u\} \leqslant \text{Prob}\{Y > u\}.$$

Ismert, hogy ha $X \leqslant_{\text{prob}} Y$, akkor várható értékeikre is $E(X) \leqslant E(Y)$.

Legyen t pozitív egész szám, p pedig egy valószínűség. Jelölje $B(t,p)$ azt a valószínűségi változót, melynek értéke t független kísérlet sorozatában a sikerek száma, ahol a siker valószínűsége minden kísérletnél p. Ismert, hogy $E(B(t,p)) = tp$.

Legyen s pozitív egész szám, p pedig egy valószínűség. Jelölje $NB(s,p)$ azt a valószínűségi változót, melynek értéke független kísérletek sorozatában az s-edik siker bekövetkezte előtti kudarcok száma, ahol a siker valószínűsége minden kísérletnél p. Ismert, hogy $E(NB(s,p)) = s(1-p)/p$.

A keresési út hosszának megállapításához a keresési út mentén visszafelé fogunk lépkedni, mindig egyet fel vagy egyet balra. Bár a lista elemeinek a magasságai a keresés végrehajtásakor már rögzítettek, úgy fogunk csinálni, mintha egy elem magassága csak akkor határozódna meg, mikor a keresési úton visszafelé haladva éppen áthaladunk rajta.

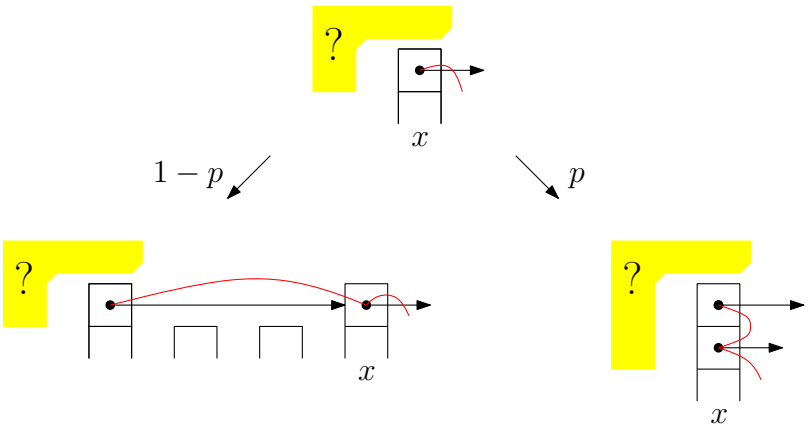
A mágikus konstansok elkerülése érdekében az elemzés során élni fogunk azzal az általánosítással, hogy a `VéletlenMagasság()` eljárás ciklusmagjában a magasság növelésének valószínűsége $1/2$ helyett p.

A keresési úton visszafelé haladva egy adott pillanatban a helyzet a következő: egy x elem i-edik előre mutató pointerénél vagyunk, semmit nem tudunk az x-től balra lévő elemek magasságairól és x magasságáról is csak annyit tudunk, hogy az legalább i.

Tegyük fel, hogy x nem a listafej. Ez biztosítható, ha a listát ideiglenesen kiterjesztjük bal oldali irányban a végtelenbe.

- Ha az x elem magassága i, akkor a keresési úton visszafelé haladva balra fogunk lépni.
- Ha az x elem magassága nagyobb, mint i, akkor a keresési úton visszafelé haladva felfelé fogunk lépni.

Annak a valószínűsége, hogy a második eset áll fenn nyilván p.



Egy végtelen listában a balra menő lépések száma, amelyeket az előtt teszünk, hogy egy szinttel feljebb lépnénk, a "kudarcok" száma (bal oldali szituáció) az első "siker" (jobb oldali szituáció) előtt, független kísérletek olyan sorozatában, ahol a siker valószínűsége p. Ennélfogva a lépések száma, amíg egy végtelen listában egy szinttel feljebb kerülünk, a bevezetett valószínűségi jelölést használva,

$$=_{\text{prob}} 1 + NB(1, p).$$

Analóg módon a lépések száma, amíg egy végtelen listában a legalsó szintről feljutunk az $L(n)$-edik szintre

$$=_{\text{prob}} (L(n) - 1) + NB(L(n) - 1, p).$$

Mivel a lista nem végtelen, ezért a lépések száma, amíg a legalsó szintről feljutunk az $L(n)$-edik szintre

$$\leqslant_{\text{prob}} (L(n) - 1) + NB(L(n) - 1, p).$$

Indoklásul itt elég arra hivatkozni, hogy ha elérjük a listafejet, akkor egyszerűen elkezdünk felfelé lépkedni, anélkül, hogy egyetlen további balra lépést végrehajtanánk.

Ha elértünk az $L(n)$-edik szintre, a hátra lévő balra lépések száma nyilván legfeljebb annyi, mint a lista legalább $L(n)$ magasságú elemeinek száma. Legyen $L(n) = \log_{1/p} n$. Ekkor a listában lévő legalább $L(n)$ magasságú elemek száma

$$=_{\text{prob}} B(n, p^{L(n)-1}) = B(n, 1/np).$$

A hátra lévő felfelé lépések számának meghatározásához legyen M egy n elemű ugráló lista magasságához tartozó valószínűségi változó. Annak a valószínűsége, hogy egy elem magassága nagyobb, mint k nyilván p^k, így

$$\text{Prob}\{M > k\} = 1 - (1 - p^k)^n.$$

A Bernoulli egyenlőtlenség szerint

$$(1 - p^k)^n > 1 - np^k,$$

ahonnan

$$\text{Prob}\{M > k\} < np^k = p^{k - L(n)}.$$

Hajtsuk végre az $i = k - L(n)$ helyettesítést:

$$\text{Prob}\{(M - L(n)) > i\} < p^i.$$

Most vegyük észre, hogy

$$\text{Prob}\{(NB(1, 1-p) + 1) > i\} = p^i.$$

Innen következik, hogy $M - L(n)$, vagyis az $L(n)$-edik szint elérése utáni felfelé lépések száma

$$\leqslant_{\text{prob}} NB(1, 1-p) + 1.$$

Ennélfogva az $L(n)$-edik szint elérése utáni balra és felfelé lépések száma összesen

$$\leqslant_{\text{prob}} B(n, 1/np) + NB(1, 1-p) + 1.$$

Mindent összevetve a teljes keresési út hossza

$$\leqslant_{\text{prob}} (L(n) - 1) + NB(L(n) - 1, p) + B(n, 1/np) + NB(1, 1-p) + 1.$$

Ennek várható értéke

$$L(n) - 1 + \frac{(L(n) - 1)(1-p)}{p} + \frac{1}{p} + \frac{p}{1-p} + 1 = \frac{L(n)}{p} + \frac{1}{1-p},$$

ami $p = 1/2$ esetén $2 \log_2 n + 2$.

Bináris fák levelei

Feladat.
Mutassuk meg, hogy egy bináris fa levél csúcsainak száma eggyel nagyobb, mint azoké, amelyeknek két gyerekük van!

Megoldás.
Az állítást a bináris fa csúcsszámára vonatkozó teljes indukcióval bizonyítjuk. Tekintsünk egy n csúcsú T bináris fát. Jelölje $l(T)$ a levél csúcsok számát, $b(t)$ pedig azokét, amelyeknek két gyerekük van.

Ha $n = 1$, akkor az állítás triviálisan igaz. Legyen ezután $n > 1$, és legyen v a fa egy levél csúcsa. Mivel a fának egynél több csúcsa van, ezért v nem a gyökér, következésképpen van szülője. Jelölje a v csúcs szülőjét u.

Tekintsük most azt a T' fát, amelyet T-ből a v levél (és a rá illeszkedő él) törlésével kapunk.

- Ha u-nak v volt az egyetlen gyereke T-ben, akkor u levél csúcs lesz a T' fában, így $l(T') = l(T)$ és $b(T') = b(T)$. Alkalmazva az indukciós feltevést T'-re, az állítás adódik.

- Ha u-nak két gyereke volt T-ben, akkor u-ból nem lesz levél csúcs T'-ben, ugyanakkor nem marad olyan csúcs sem, amelynek két gyereke van. Így ekkor $l(T') = l(T) - 1$ és $b(T') = b(T) - 1$. Alkalmazva az indukciós feltevést T'-re, az állítás ismét adódik.

Keresési sorozatok

Feladat.
Egy bináris keresőfa kulcsai az 1 és 1000 közötti egész számok közül valók. A fában a 363 kulcsot akarjuk megkeresni. Az alábbi sorozatok közül melyek nem lehetnek keresési sorozatok?

(A) 2, 252, 401, 398, 330, 344, 397, 363.

(B) 924, 220, 911, 244, 898, 258, 362, 363.

(C) 925, 202, 911, 240, 912, 245, 363.

(D) 2, 399, 387, 219, 266, 382, 381, 278, 363.

(E) 935, 278, 347, 621, 299, 392, 358, 363.

Megoldás.
(A) Igen.

(B) Igen.

(C) Nem. Miután a 911 kulcsnál balra mentünk, nem találkozhattunk a nála nagyobb 912 kulccsal.

(D) Igen.

(E) Nem. Miután a 347 kulcsnál jobbra mentünk, nem találkozhattunk a nála kisebb 299 kulccsal.

Előző és következő elem bináris keresőfákban

Feladat.
(A) Mutassuk meg, hogy ha egy bináris keresőfa valamely csúcsának van bal oldali gyereke, akkor a megelőzőjének nincs jobb oldali gyereke!

(B) Mutassuk meg, hogy ha egy bináris keresőfa valamely csúcsának van jobb oldali gyereke, akkor a rákövetkezőjének nincs bal oldali gyereke!

(C) Mutassuk meg, hogy ha egy bináris keresőfa valamely csúcsának van bal oldali gyereke, akkor a megelőzője a csúcs bal oldali részfájának a maximális eleme!

(D) Mutassuk meg, hogy ha egy bináris keresőfa valamely csúcsának van jobb oldali gyereke, akkor a rákövetkezője a csúcs jobb oldali részfájának a minimális eleme!

(E) Mutassuk meg, hogy ha egy bináris keresőfa valamely csúcsának nincs bal oldali gyereke, akkor a megelőzője a csúcs legalacsonyabban fekvő olyan őse, amelynek a csúcs jobb oldali leszármazottja!

(F) Mutassuk meg, hogy ha egy bináris keresőfa valamely csúcsának nincs jobb oldali gyereke, akkor a rákövetkezője a csúcs legalacsonyabban fekvő olyan őse, amelynek a csúcs bal oldali leszármazottja!

Megoldás.
(A) Legyen x a fa egy olyan csúcsa, amelynek van bal oldali gyereke. A fa inorder bejárásában az x csúcs bal oldali részfájának csúcsai közvetlenül x előtt jönnek. Így x megelőzője a bal oldali részfában van. Legyen s az x csúcs megelőzője. Most ha s-nek lenne jobb oldali gyereke, az az inorder bejárásban s után, de még x előtt jönne, ellentmondva annak, hogy az x csúcs megelőzője s.

(B) Legyen x a fa egy olyan csúcsa, amelynek van jobb oldali gyereke. A fa inorder bejárásában az x csúcs jobb oldali részfájának csúcsai közvetlenül x után jönnek. Így x rákövetkezője a jobb oldali részfában van. Legyen s az x csúcs rákövetkezője. Most ha s-nek lenne bal oldali gyereke, az az inorder bejárásban x után, de még s előtt jönne, ellentmondva annak, hogy az x csúcs rákövetkezője s.

(C) Legyen x a fa egy olyan csúcsa, amelynek van bal oldali gyereke. A fa inorder bejárásában az x csúcs bal oldali részfájának csúcsai közvetlenül x

előtt jönnek. Így x megelőzője a bal oldali részfában van, nevezetesen annak az inorder bejárás szerinti utolsó eleme, azaz a maximuma.

(D) Legyen x a fa egy olyan csúcsa, amelynek van jobb oldali gyereke. A fa inorder bejárásában az x csúcs jobb oldali részfájának csúcsai közvetlenül x után jönnek. Így x rákövetkezője a jobb oldali részfában van, nevezetesen annak az inorder bejárás szerinti első eleme, azaz a minimuma.

(E) Legyen x a fa egy olyan csúcsa, amelynek nincs bal oldali gyereke, és legyen y az x csúcs legalacsonyabban fekvő olyan őse, amelynek x jobb oldali leszármazottja. Most az y csúcsnak van jobb oldali gyereke, így (D) szerint y rákövetkezője y jobb oldali részfájának a minimális eleme. Legyen x' az y csúcs rákövetkezője.

Indirekt tegyük fel, hogy $x' \neq x$. Legyen x és x' legalacsonyabban fekvő közös őse z. Ha $z = x$, akkor x' az x csúcs jobb oldali részfájában van, hiszen a bal oldali üres, így a fa inorder bejárásában x előbb jön, mint x', ami ellentmondás. Ha $z \neq x$, akkor a feltétel szerint x a z csúcs bal oldali részfájában van, így x' szükségképpen a jobb oldaliban vagy $z = x'$. Ám ekkor a fa inorder bejárásában x megint csak előbb jön, mint x', ami ellentmondás.

Ebből következik, hogy az y csúcs rákövetkezője az x csúcs, és így az x csúcs megelőzője az y csúcs.

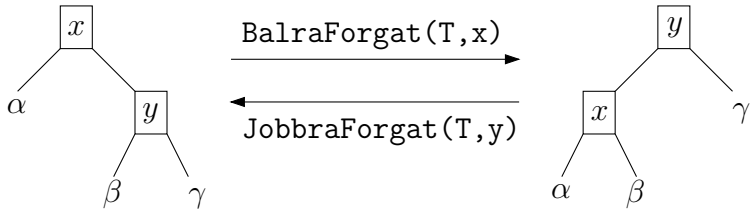
(F) Legyen x a fa egy olyan csúcsa, amelynek nincs jobb oldali gyereke, és legyen y az x csúcs legalacsonyabban fekvő olyan őse, amelynek x bal oldali leszármazottja. Most az y csúcsnak van bal oldali gyereke, így (C) szerint y megelőzője y bal oldali részfájának a maximális eleme. Legyen x' az y csúcs megelőzője.

Indirekt tegyük fel, hogy $x' \neq x$. Legyen x és x' legalacsonyabban fekvő közös őse z. Ha $z = x$, akkor x' az x csúcs bal oldali részfájában van, hiszen a jobb oldali üres, így a fa inorder bejárásában x később jön, mint x', ami ellentmondás. Ha $z \neq x$, akkor a feltétel szerint x a z csúcs jobb oldali részfájában van, így x' szükségképpen a bal oldaliban vagy $z = x'$. Ám ekkor a fa inorder bejárásában x megint csak később jön, mint x', ami ellentmondás.

Ebből következik, hogy az y csúcs megelőzője az x csúcs, és így az x csúcs rákövetkezője az y csúcs.

Forgatás

Feladat.
Mutassuk meg, hogy bármely, az $1, 2, \ldots, n$ kulcsokat tartalmazó n csúcsú bináris keresőfa forgatásokkal átalakítható bármely, szintén az $1, 2, \ldots, n$ kulcsokat tartalmazó n csúcsú bináris keresőfává!



Azt is mutassuk meg, hogy $O(n)$ forgatás mindig elég!

Megoldás.
Először megmutatjuk, hogy bármely n csúcsú bináris keresőfa legfeljebb $n-1$ forgatással átalakítható egy jobbra tartó lánccá.

Kezdetben a fa jobb szélén haladó úton van legalább egy csúcs, nevezetesen a fa gyökere. Ha a fa még nem egy jobbra tartó lánc, akkor a fának van olyan x csúcsa, amelynek y szülője a fa jobb szélén haladó úton van. Hívjuk meg az y csúcsra a jobbra forgató eljárást; ezután már x is rajta lesz a jobb szélen haladó úton, miközben onnan egyetlen csúcs sem tűnik el. Így a forgatás eredményeként a fa jobb szélén haladó út hossza eggyel nőtt. Ennélfogva legfeljebb $n-1$ jobbra forgatás után mind az n csúcs a jobb szélen haladó úton lesz, vagyis a fa egy jobbra tartó lánccá alakul.

Ezek után egy T_1 fát a következőképpen alakíthatunk át egy T_2 fává. A T_1 fa jobbra forgatások egy legfeljebb $n-1$ elemű

$$(j_1^{(1)}, j_2^{(1)}, \ldots, j_k^{(1)})$$

sorozatával egy jobbra tartó lánccá alakítható. Hasonlóan, a T_2 fa is jobbra forgatások egy legfeljebb $n-1$ elemű

$$(j_1^{(2)}, j_2^{(2)}, \ldots, j_l^{(2)})$$

sorozatával egy jobbra tartó lánccá alakítható. Jelölje a $j_1^{(2)}, j_2^{(2)}, \ldots, j_l^{(2)}$ jobbra forgatásoknak megfelelő inverz balra forgatásokat rendre $b_1^{(2)}, b_2^{(2)}, \ldots, b_l^{(2)}$. A balra forgatások

$$(b_l^{(2)}, b_{l-1}^{(2)}, \ldots, b_1^{(2)})$$

sorozata a (T_1-ből kapott) jobbra tartó láncot világos módon a T_2 fává alakítja. Így a legfeljebb $2n-2$ elemű

$$(j_1^{(1)}, j_2^{(1)}, \ldots, j_k^{(1)}, b_l^{(2)}, b_{l-1}^{(2)}, \ldots, b_1^{(2)})$$

forgatássorozat a T_1 fát a T_2 fává alakítja.

Piros-fekete fák egyesítése

Feladat.
Legyenek F_1 és F_2 piros-fekete fák. Tegyük fel, hogy az F_2-ben szereplő kulcsok mind nagyobbak az F_1-ben szereplő kulcsoknál. Mutassuk meg, hogy a két piros-fekete fa $O(\log n)$ költséggel összefűzhető egyetlen piros-fekete fává, ahol n a két fában szereplő kulcsok együttes száma!

Megoldás.
Ha F_2 egyetlen kulcsot tartalmaz, akkor szúrjuk be ezt a kulcsot F_1-be. Ennek költsége nyilván $O(\log n)$.

Tegyük fel ezután, hogy F_2 egynél több kulcsot tartalmaz. Keressük meg és töröljük F_2 legkisebb kulcsú csúcsát. Jelölje a törölt csúcsot x, a törlés után kapott piros-fekete fát pedig F_3.

Határozzuk meg az F_1 fa gyökerének az $\mathrm{fm}(\mathit{gyökér}[F_1])$ és az F_3 fa gyökerének az $\mathrm{fm}(\mathit{gyökér}[F_3])$ fekete-magasságát. Ehhez elég a gyökértől egy tetszőleges levélig vezető úton a fekete csúcsokat összeszámolni. Ha $\mathrm{fm}(\mathit{gyökér}[F_1]) = \mathrm{fm}(\mathit{gyökér}[F_3])$, akkor tekintsük a következő piros-fekete fát. A fa gyökere legyen az x csúcs, ennek színe legyen fekete; az x csúcs bal oldali részfája legyen F_1, a jobb oldali részfája pedig F_3. Ez a fa nyilván megfelel a kívánalmaknak.

Tegyük fel ezután, hogy $\mathrm{fm}(\mathit{gyökér}[F_1]) \neq \mathrm{fm}(\mathit{gyökér}[F_3])$. Legyen mondjuk $\mathrm{fm}(\mathit{gyökér}[F_1]) > \mathrm{fm}(\mathit{gyökér}[F_3])$; a fordított eset hasonlóan kezelhető. Az F_1 fában a gyökértől indulva, és rendületlenül jobb felé haladva keressük meg azt a fekete csúcsot, jelölje ezt a csúcsot y, amelynek fekete-magassága $\mathrm{fm}(\mathit{gyökér}[F_3])$. Jegyezzük meg, hogy a fenti úton az egymás után következő fekeke csúcsok fekete-magassága egyesével csökken. Most az F_1 fa y gyökerű részfáját helyettesítsük azzal a fával, amelynek gyökere az x csúcs, ennek bal oldali részfája az F_1 fa y gyökerű részfája, jobb oldali részfája pedig F_3. Legyen az x csúcs színe piros, és hívjuk meg az x csúcsra a beszúrás eljárás piros-fekete tulajdonságot helyreállító részét (úgy, mintha csupán az x csúcsot szúrtuk volna be a fába). Az így kapott fa nyilván megfelel a kívánalmaknak.

Az x csúcs törlésének költsége $O(\log n)$. Ugyanennyi $\mathrm{fm}(\mathit{gyökér}[F_1])$ és $\mathrm{fm}(\mathit{gyökér}[F_3])$ meghatározásának a költsége is.

- Ha $\mathrm{fm}(\mathit{gyökér}[F_1]) = \mathrm{fm}(\mathit{gyökér}[F_3])$, akkor az utolsó fázis konstans költségű.

- Ha $\mathrm{fm}(\mathit{gyökér}[F_1]) \neq \mathrm{fm}(\mathit{gyökér}[F_3])$, akkor meg kell még találnunk az y csúcsot; ennek költsége $O(\log n)$. Ezután jön az y gyökerű részfa lecserélése, ami konstans költségű, majd a piros-fekete tulajdonság helyreállítása, ami ismét $O(\log n)$ költségű.

Mindkét esetben az algoritmus teljes költsége $O(\log n)$.

Még egyszer a k-adik elem kiválasztásáról

Feladat.
Valós számok egy véges halmazát szeretnénk tárolni olyan adatszerkezettel, amely a szokásos keres, beszúr, töröl műveleteken kívül támogatja a halmaz k-adik elemének kiválasztását. Valósítsuk meg minél hatékonyabban az adatszerkezetet!

Megoldás.
Az adatszerkezet egy olyan T piros-fekete fa, amelynek minden csúcsában egy kiegészítő információt tárolunk; a szokásos mezőkön kívül minden x csúcsban jelen van egy $méret[x]$ mező is. Ez a mező az x gyökerű részfa $nil[T]$-től különböző csúcsainak számát tartalmazza, az x csúcsot is beleértve. A $méret[nil[T]] = 0$ konvenciót alkalmazva a következő azonosság azonnal adódik:

$$méret[x] = méret[bal[x]] + méret[jobb[x]] + 1$$

a fa tetszőleges $nil[T]$-től különböző x csúcsára.

Keressük tehát a k-adik elemet. Ez a T fa inorder bejárás szerinti k-adik eleme. Vegyük észre, hogy a T gyökerében lévő kulcsot megelőző kulcsok száma $méret[bal[gyökér[T]]]$, így $r = méret[bal[gyökér[T]]] + 1$ nem más, mint a gyökérben lévő kulcs sorszáma. Három eset lehetséges.

- Ha $r = k$, akkor kész vagyunk, a k-adik elem a gyökérben van.
- Ha $r > k$, akkor a k-adik elemet a bal oldali részfában keressük tovább rekurzívan, szintén mint k-adik elemet.
- Ha $r < k$, akkor a k-adik elemet a jobb oldali részfában keressük tovább rekurzívan, de most mint $(k - r)$-edik elemet, hiszen tudjuk, hogy a bal oldali részfában, illetve a gyökérben levő r kulcs megelőzi a k-adik elemet.

Lássuk az algoritmust ezek után. Az algoritmust a $(gyökér[T], k)$ paraméterekkel kell meghívni.

```
PF-Kiválaszt(x,i)
r=méret[bal[x]]+1
if i=r
  then
    return x
```

```
else
  if i<r
    then return PF-Kiválaszt(bal[x],i)
    else return PF-Kiválaszt(jobb[x],i-r)
```

Az algoritmus a k-adik elemre mutató pointerrel tér vissza.

Mivel minden rekurzív híváskor egy szinttel alacsonyabbra kerülünk a fában, ezért a `PF-Kiválaszt` algoritmus költsége arányos a fa magasságával. Ennélfogva a `PF-Kiválaszt` algoritmus költsége $O(\log n)$, ha a fa n kulcsot tartalmaz.

Hátra van még annak igazolása, hogy a *méret* mezők mind beszúrás, mind törlés során a műveletek aszimptotikus költségének változása nélkül aktualizálhatók. Tegyük fel, hogy a kulcsok száma n.

Piros-fekete fák esetén a beszúrás két menetben történik. Az első menetben a gyökértől haladunk addig a csúcsig, amelynek az új csúcs a gyereke lesz. A második menetben felfelé haladunk a fában, csúcsok színét változtatgatjuk, illetve forgatásokat végzünk a piros-fekete tulajdonság fenntartása érdekében. Az első menetben az aktualizálás csupán annyi, hogy minden olyan x csúcsra, amelyen áthaladunk, a *méret*$[x]$ értéket eggyel megnöveljük, az új csúcs *méret* értékét pedig egyre állítjuk. Mivel a keresőúton $O(\log n)$ csúcs van, így itt a pótlólagos költség $O(\log n)$. A második menetben a forgatások okoznak csak változást a piros-fekete fa szerkezetén. Azonban a forgatás lokális művelet, csak annál a két csúcsnál változik a *méret* érték, amely kapcsolat mentén történik a forgatás. Mivel legfeljebb két forgatást kell végezni piros-fekete fába történő beszúrásnál, ezért a második menet pótlólagos költsége $O(1)$. Így beszúrásnál a *méret* értékek a művelet aszimptotikus költségének változása nélkül aktualizálhatók, következésképpen a beszúrás költsége marad $O(\log n)$.

A törlés algoritmus szintén két menetből áll: az első menetben a tényleges törlést hajtjuk végre, míg a másodikban csúcsok színét változtatgatjuk, és legfeljebb három forgatást végzünk. Az első menetben a *méret* értékek aktualizálása úgy történik, hogy az eltávolított y csúcstól a gyökérig felfelé haladva az úton található összes x csúcsra a *méret*$[x]$ értéket eggyel csökkentjük. Mivel ezen az úton $O(\log n)$ csúcs van, így itt a pótlólagos költség $O(\log n)$. A második menet pótlólagos költsége $O(1)$, ahogy azt már a beszúrásnál láttuk. Így itt is igaz, hogy a *méret* értékek a művelet aszimptotikus költségének változása nélkül aktualizálhatók, következésképpen a törlés költsége marad $O(\log n)$.

Minimális távolság

Feladat.
Valós számok egy véges halmazát szeretnénk tárolni olyan adatszerkezettel, amely a szokásos keres, beszúr, töröl műveleteken kívül támogatja a halmaz két legközelebbi elemének kiválasztását, illetve azok távolságának meghatározását. Valósítsuk meg minél hatékonyabban az adatszerkezetet!

Megoldás.
Az adatszerkezet egy olyan T piros-fekete fa, amelynek minden $nil[T]$-től különböző csúcsában három kiegészítő információt tárolunk; a szokásos mezőkön kívül minden x csúcsban jelen van egy $min[x]$, egy $max[x]$ és egy $mintáv[x]$ mező is. Ezek a mezők rendre az x gyökerű részfa legkisebb kulcsát, legnagyobb kulcsát, illetve két legközelebbi elemének távolságát tartalmazzák. Ha x mindkét gyereke $nil[T]$, akkor $mintáv[x] = \infty$ definíció szerint. Vegyük észre, hogy a fa bármely $nil[T]$-től különböző x csúcsának három új mezőjére a következők teljesülnek:

$$min[x] = \begin{cases} min[bal[x]] & \text{ha } bal[x] \neq nil[T], \\ kulcs[x] & \text{különben,} \end{cases}$$

$$max[x] = \begin{cases} max[jobb[x]] & \text{ha } jobb[x] \neq nil[T], \\ kulcs[x] & \text{különben,} \end{cases}$$

$$mintav[x] = \min \begin{cases} mintav[bal[x]] & (\infty \text{ ha } bal[x] = nil[T]), \\ mintav[jobb[x]] & (\infty \text{ ha } jobb[x] = nil[T]), \\ kulcs[x] - max[bal[x]] & (\infty \text{ ha } bal[x] = nil[T]), \\ min[jobb[x]] - kulcs[x] & (\infty \text{ ha } jobb[x] = nil[T]). \end{cases}$$

Ezek után a két legközelebbi elem távolsága egyszerűen megkapható: ez nem más mint a $mintav[gyökér[T]]$ mező értéke. A művelet költsége $O(1)$.

Az adatszerkezet támogatja két konkrét legközelebbi elem meghatározását is. A gyökértől indulva vizsgáljuk, hogy a *mintav* mező értékét hogyan kaptuk.

- Ha ez az érték megegyezik valamelyik részfa *mintav* mezőjének az értékével, akkor folytassuk a vizsgálatot rekurzívan ebben a részfában.
- Ha viszont ez az érték a *kulcs* mező és a bal oldali részfa legnagyobb vagy a jobb oldali részfa legkisebb elemének a különbsége, akkor egy legközelebbi elempár a *kulcs* mező értéke, és ez a maximum, illetve minimum érték.

Az eljárás költsége $O(\log n)$ ha a fa n kulcsot tartalmaz.

Hátra van még annak igazolása, hogy a *min*, *max* és *mintav* mezők mind beszúrás, mind törlés során a műveletek aszimptotikus költségének változása nélkül aktualizálhatók. Tegyük fel, hogy a kulcsok száma n.

Piros-fekete fák esetén a beszúrás két menetben történik: az első menetben beillesztjük az új x csúcsot a fába, míg a másodikban csúcsok színét változtatgatjuk, és forgatásokat végzünk a piros-fekete tulajdonság fenntartása érdekében. Az x csúcsnál a három új mező értékének a beállítása nyilván konstans költségű. Ezután az x csúcstól a gyökérig felfelé haladva, az út minden csúcsánál aktualizáljuk a három új mező értékét. Mivel ezen az úton $O(\log n)$ csúcs van, ezért az első menet pótlólagos költsége $O(\log n)$. A második menetben csak a forgatások okoznak változást a piros-fekete fa szerkezetén. Azonban a forgatás lokális művelet, csak annál a két csúcsnál változnak a *min*, *max* és *mintav* értékek, amely kapcsolat mentén történik a forgatás. Mivel legfeljebb két forgatást kell végezni piros-fekete fába történő beszúrásnál, így a második menet pótlólagos költsége $O(1)$. Következésképpen a beszúrás költsége marad $O(\log n)$.

A törlés algoritmus szintén két menetből áll: az első menetben a tényleges törlést hajtjuk végre, míg a másodikban csúcsok színét változtatgatjuk, és legfeljebb három forgatást végzünk. Az első menetben az eltávolított y csúcstól a gyökérig felfelé haladva, az út minden csúcsánál aktualizáljuk az új mezők értékét. Mivel ezen az úton $O(\log n)$ csúcs van, így itt a pótlólagos költség $O(\log n)$. A második menet pótlólagos költsége szintén $O(1)$, ahogy azt már a beszúrásnál láttuk. Következésképpen a törlés költsége marad $O(\log n)$.

UNIÓ-HOLVAN

Feladat.
Az UNIÓ-HOLVAN adatszerkezettel egy n elemű S halmaz olyan $U_1, U_2, \ldots, U_m$ részhalmazait tároljuk, amelyekre

$$U_i \cap U_j = \emptyset, \text{ ha } i \neq j,$$

valamint

$$U_1 \cup U_2 \cup \cdots \cup U_m = S,$$

teljesül, más szóval amelyek S egy partícióját adják. Két művelet tartozik az adatszerkezethez:

- UNIÓ$(U_i, U_j) = (\{U_1, U_2, \ldots, U_m\} \cup \{U_i \cup U_j\}) \setminus \{U_i, U_j\}$,
- HOLVAN(v) eredménye az S halmaz egy v elemére annak az U_i halmaznak a neve, amely tartalmazza v-t.

Valósítsuk meg minél hatékonyabban az adatszerkezetet! Az adatszerkezet tipikus alkalmazásainál több UNIÓ és HOLVAN műveletből álló lépéssort hajtunk végre abból a kezdőhelyzetből indulva, amelyben minden U_j halmaz egyelemű. A műveletek implementálásánál mi is élhetünk ezzel a feltevéssel.

Megoldás.
Az adatszerkezetet legegyszerűbben úgy képzelhetjük el, hogy egy U_j részhalmaznak egy gyökeres, felfelé irányított fa felel meg. U_j elemeit a fa csúcsaiban tároljuk, egy szülőmutatóval együtt (ami *nil* a gyökérnél). Egy részhalmaz neve legyen az őt ábrázoló fa gyökere. A gyökérben nyilvántartjuk még a fa méretét is. Ezek a fák természetes módon tárolhatók egy S elemeivel indexelt tömbben; ennek a v indexű eleme egy mutató a v fabeli szülőjére.

Az UNIÓ(U_i, U_j) művelet megvalósítása ezek után a következő. Tegyük fel, hogy $|U_i| \leqslant |U_j|$; csatoljuk hozzá gyerekként az U_i fa gyökerét az U_j fa gyökeréhez.

A HOLVAN(v) művelet megvalósítása sem sokkal bonyolultabb. A $v \in S$ elemet tartalmazó részhalmaz nevét, azaz a megfelelő fa gyökerét, a szülőkhöz menő mutatók végigkövetésével találhatjuk meg.

Tegyük fel, hogy UNIÓ(U_i, U_j) hívásakor az U_i és U_j halmazok a gyökereikkel adottak. Az UNIÓ költsége ekkor nyilván $O(1)$, hiszen a gyökereknél adminisztráljuk a fák méretét, és csak a kisebbik fa gyökerének szülőmutatóját kell beállítani.

A HOLVAN művelet költségének meghatározásához vegyük észre, hogy amikor egy v csúcs új gyökér alá kerül, akkor egy szinttel távolodik a gyökértől, fájának mérete pedig legalább a duplájára nő. Ezért v legfeljebb $\log_2 n$ alkalommal kerülhet új gyökér alá, így v-től a gyökérig vezető úton az élek száma legfeljebb $\log_2 n$. Innen azonnal adódik, hogy a HOLVAN költsége $O(\log n)$.

Dinamikus programozás és mohó módszer

Sakktábla

Feladat.
Tegyük fel, hogy adott egy $n \times n$-es sakktábla, és egy figura, amelyet a tábla alsó sorából kell eljuttatni a felső sorba úgy, hogy minden mezőről csak a következő három mezőre léphetünk:

- a pillanatnyi mező feletti mezőre,
- a pillanatnyi mező felett lévő mezőtől balra levő mezőre, feltéve, hogy nem a bal szélső oszlopban állunk,
- a pillanatnyi mező felett lévő mezőtől jobbra levő mezőre, feltéve, hogy nem a jobb szélső oszlopban állunk.

Ha az x mezőről (megengedett módon) az y mezőre lépünk, akkor $f(x, y)$ forintot kapunk.

Adjunk hatékony algoritmust egy olyan út megtalálására, amely mentén a lehető legtöbb pénzt gyűjthetjük be! Az alsó sor akármelyik mezőjéről indulhatunk, és a felső sor akármelyik mezőjére érkezhetünk.

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Minden $1 \leqslant i, j \leqslant n$ esetén jelölje (i, j) a sakktábla i-edik sorának és j-edik oszlopának a kereszteződésében lévő mezőt. Tekintsünk most az (i, j) mezőn végződő utak közül egy olyat, amely a lehető legtöbb pénzt hozza. Jelölje ezt az utat P. Ha $i > 1$, akkor P-n szükségképpen az $(i-1, j')$ mezőről érkezünk meg az (i, j) mezőre, ahol $j' \in \{j-1, j, j+1\}$. Vizsgáljuk meg P-nek az $(i-1, j')$ mezőig terjedő P' szakaszát. Állítjuk, hogy P' az $(i-1, j')$ mezőn végződő utak közül egy legtöbb pénzt hozó. Tegyük fel, hogy ez nincs így, azaz az $(i-1, j')$ mezőn végződő utak között van olyan P'' út, amely több pénzt hoz, mint P'.

Ám ekkor a P út P' szakaszát P''-re cserélve egy olyan (i,j)-n végződő utat kapunk, amely több pénzt hoz, mint P, ellentmondva P optimalitásának.

Minden $1 \leqslant i,j \leqslant n$ esetén jelölje $d[i,j]$ az (i,j) mezőn végződő utak mentén begyűjthető pénz maximumát. Most $d[1,j] = 0$ minden $1 \leqslant j \leqslant n$ esetén. Ha pedig $i > 1$, akkor a fentiekkel összhangban

$$d[i,j] = \max \begin{cases} d[i-1,j-1] + f((i-1,j-1),(i,j)) & \text{ha } j > 1, \\ d[i-1,j] + f((i-1,j),(i,j)) & \\ d[i-1,j+1] + f((i-1,j+1),(i,j)) & \text{ha } j < n. \end{cases}$$

Az egész sakktáblán begyűjthető pénz maximuma

$$\max\{d[n,j] \mid 1 \leqslant j \leqslant n\}.$$

Annak érdekében, hogy a legtöbb pénzt hozó utat is meg tudjuk adni még egy jelölést vezetünk be. Minden $2 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant n$ esetén legyen $w[i,j]$ az (i,j) mezőn végződő, legtöbb pénzt hozó úton az utolsó előtti mező oszlopa.

```
Sakktábla(n,f)
for j=1 to n do
  d[1,j]=0
for i=2 to n do
  for j=1 to n do
    d[i,j]=-INFINITY
    if j>1 then
      d[i,j]=d[i-1,j-1]+f((i-1,j-1),(i,j))
      w[i,j]=j-1
    if d[i-1,j]+f((i-1,j),(i,j))>d[i,j] then
      d[i,j]=d[i-1,j]+f((i-1,j),(i,j))
      w[i,j]=j
    if j<n AND d[i-1,j+1]+f((i-1,j+1),(i,j))>d[i,j] then
      d[i,j]=d[i-1,j+1]+f((i-1,j+1),(i,j))
      w[i,j]=j+1
opt=d[n,1]
t=1
for j=2 to n do
  if opt<d[n,j] then
    opt=d[n,j]
    t=j
return d,w,opt,t
```

A legtöbb pénzt hozó utat a következő rekurzív eljárással határozhatjuk meg. Az eljárást a (w, n, t) paraméterekkel kell meghívni, ahol t az az $1 \leqslant j \leqslant n$ index, amelyre $d[n, j]$ maximális.

```
LépésekNyomtatása(w,i,j)
if i>1 then LépésekNyomtatása(w,i-1,w[i,j])
print "(" i "," j ")"
```

A `Sakktábla` eljárás költsége $O(n^2)$, a `LépésekNyomtatása` eljárásé pedig $O(n)$.

Leghosszabb közös részsorozat

Feladat.
Egy $Z = (z_1, z_2, \ldots, z_k)$ sorozat részsorozata egy $X = (x_1, x_2, \ldots, x_n)$ sorozatnak, ha léteznek olyan $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$ indexek, hogy minden $1 \leqslant j \leqslant k$ esetén $z_j = x_{i_j}$. Azt mondjuk, hogy egy Z sorozat közös részsorozata az X és Y sorozatoknak, ha Z részsorozata X-nek és Y-nak is. Adjunk hatékony algoritmust az $X = (x_1, x_2, \ldots, x_n)$ és $Y = (y_1, y_2, \ldots, y_m)$ sorozatok egy leghosszabb közös részsorozatának meghatározására!

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Szükségünk lesz a prefix fogalmára. Az $X = (x_1, x_2, \ldots, x_n)$ sorozat $X_i = (x_1, x_2, \ldots, x_i)$ részsorozatát az X sorozat i-edik prefixének nevezzük tetszőleges $0 \leqslant i \leqslant n$ esetén.

Állítás. Legyen $X = (x_1, x_2, \ldots, x_n)$ és $Y = (y_1, y_2, \ldots, y_m)$ két sorozat és $Z = (z_1, z_2, \ldots, z_k)$ ezek egy leghosszabb közös részsorozata. Ekkor

(1) ha $x_n = y_m$, akkor $z_k = x_n = y_m$ és Z_{k-1} egy leghosszabb közös részsorozata X_{n-1}-nek és Y_{m-1}-nek,

(2) ha $x_n \neq y_m$ és $z_k \neq x_n$, akkor Z egy leghosszabb közös részsorozata X_{n-1}-nek és Y-nak,

(3) ha $x_n \neq y_m$ és $z_k \neq y_m$, akkor Z egy leghosszabb közös részsorozata X-nek és Y_{m-1}-nek.

Bizonyítás.
(1) Ha $z_k \neq x_n$, akkor az $x_n = y_m$ elemet hozzávehetnénk Z-hez, ami által X és Y egy $k + 1$ hosszú közös részsorozatát kapnánk, ellentmondás. Így

$z_k = x_n = y_m$. Világos, hogy Z_{k-1} közös részsorozata X_{n-1}-nek és Y_{m-1}-nek. Ha lenne ennél hosszabb közös részsorozata X_{n-1}-nek és Y_{m-1}-nek, akkor ehhez z_k-t hozzáfűzve X-nek és Y-nak egy Z-nél hosszabb közös részsorozatát kapnánk, ami nem lehetséges. Így Z_{k-1} szükségképpen egy leghosszabb közös részsorozata X_{n-1}-nek és Y_{m-1}-nek.

(2) Ha $z_k \neq x_n$, akkor Z közös részsorozata X_{n-1}-nek és Y-nak. Ha X_{n-1}-nek és Y-nak volna Z-nél hosszabb közös részsorozata, az Z-nél hosszabb közös részsorozata lenne X-nek és Y-nak is, ellentmondás.

(3) Ugyanúgy, mint az előbb.

Az előző állítás szerint csak egy vagy két részfeladat van, amit meg kell vizsgálni az $X = (x_1, x_2, \ldots, x_n)$ és $Y = (y_1, y_2, \ldots, y_m)$ sorozatok egy leghosszabb közös részsorozatának megkeresésekor.

- Ha $x_n = y_m$, akkor elég X_{n-1} és Y_{m-1} egy leghosszabb közös részsorozatát megtalálni, amelyhez az $x_n = y_m$ elemet csatolva X és Y egy leghosszabb közös részsorozatához jutunk.

- Ha $x_n \neq y_m$, akkor két részfeladatot kell megoldani: X_{n-1} és Y, illetve X és Y_{m-1} egy-egy leghosszabb közös részsorozatát kell megkeresni. Ezek közül a hosszabb X-nek és Y-nak is egy leghosszabb közös részsorozata lesz.

Ezek után az X_i és Y_j sorozatok leghosszabb közös részsorozatának hosszát jelölje $c[i,j]$ minden $0 \leqslant i \leqslant n$ és $0 \leqslant j \leqslant m$ esetén. Ha $i = 0$ vagy $j = 0$, azaz a sorozatok legalább egyikének a hossza nulla, akkor természetesen a leghosszabb közös részsorozat hossza is nulla. A többi esetben a fenti észrevétel igazít el. Mindent összevetve a $c[i,j]$-re vonatkozó rekurzív képlet a következő:

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

Egy leghosszabb közös részsorozat megtalálása érdekében még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant m$ esetén jelölje $b[i,j]$ a c tömb azon elemének indexét, ahonnan $c[i,j]$ értékének meghatározásakor a legjobb értéket kaptuk.

```
LeghosszabbKözösRészsorozat(X,n,Y,m)
for i=1 to n do
   c[i,0]=0
for j=0 to m do
```

```
   c[0,j]=0
for i=1 to n do
   for j=1 to m do
      if x_i=y_j
         then
            c[i,j]=c[i-1,j-1]+1
            b[i,j]=(i-1,j-1)
         else
            if c[i-1,j]>=c[i,j-1]
               then
                  c[i,j]=c[i-1,j]
                  b[i,j]=(i-1,j)
               else
                  c[i,j]=c[i,j-1]
                  b[i,j]=(i,j-1)
return c,b
```

A $c[0:n, 0:m]$ tömb elemeit sorfolytonosan töltjük ki, először az első sort balról jobbra, aztán a másodikat, és így tovább. A leghosszabb közös részsorozat hossza $c[n, m]$. Az eljárás költsége nyilván $O(nm)$.

Az b tömb felhasználásával könnyű egy leghosszabb közös részsorozatot megtalálni: a $b[n, m]$ elemből indulva egyszerűen csak végig kell haladni a "mutatók" mentén a tömbön. Amikor egy $b[i, j]$ elemről a $b[i-1, j-1]$ elemre lépünk, az azt jelenti, hogy az $x_i = y_j$ elem benne van a leghosszabb közös részsorozatban. Ezzel a módszerrel a leghosszabb közös részsorozat elemeit fordított sorrendben kapjuk meg.

A következő rekurzív eljárás a leghosszabb közös részsorozat elemeit a helyes, eredeti sorrendben nyomtatja ki. Az eljárást a (b, X, n, m) paraméterekkel kell meghívni.

```
Nyomtat(b,X,i,j)
if i=0 OR j=0 then return
if b[i,j]=(i-1,j-1)
   then
      Nyomtat(b,X,i-1,j-1)
      print x_i
   else
      if b[i,j]=(i-1,j)
         then Nyomtat(b,X,i-1,j)
         else Nyomtat(b,X,i,j-1)
```

Az eljárás költsége $O(n+m)$, hiszen minden rekurzív hívásnál i és j legalább egyikének csökken az értéke.

Leghosszabb monoton növekvő részsorozat

Feladat.
Adjunk hatékony algoritmust egy n elemű számsorozat leghosszabb monoton növekvő részsorozatának meghatározására!

Megoldás.
A feladat visszavezethető a leghosszabb közös részsorozat feladatra. Legyen a bemeneti tömb $T[1:n]$. Másoljuk át a T tömböt a T' tömbbe, rendezzük a T' tömb elemeit növekvő sorrendbe, végül határozzuk meg T és T' egy leghosszabb közös részsorozatát. Az eredmény éppen T egy leghosszabb monoton növekvő részsorozata.

A másolás költsége $O(n)$, a rendezésé $O(n \log n)$, a leghosszabb közös részsorozat meghatározásáé pedig $O(n^2)$. Így az algoritmus teljes költsége $O(n^2)$.

Mutatunk egy (szintén dinamikus programozás) direkt megoldást is. Legyen megint $T[1:n]$ a bemeneti tömb. Először is vegyük észre, hogy ha egy $T[j]$-re végződő maximális hosszúságú monoton növekvő részsorozat utolsó előtti tagja $T[i]$, akkor a sorozat $T[i]$-ig terjedő része maximális hosszúságú a $T[i]$-re végződő monoton növekvő részsorozat között. Valóban, ha lenne ennél hosszabb $T[i]$-re végződő monoton növekvő részsorozat, akkor az eredeti sorozat $T[i]$-ig terjedő részét erre cserélve egy az eredetinél hosszabb $T[j]$-re végződő monoton növekvő részsorozatot kapnánk, ami ellentmondás.

Minden $0 \leqslant j \leqslant n$ esetén jelölje $l[j]$ a $T[j]$-re végződő monoton növekvő részsorozatok elemszámának a maximumát.

- Ha $j = 0$, akkor $l[j] = 0$ definíció szerint.
- Ha $1 \leqslant j \leqslant n$ és egy $T[j]$-re végződő maximális elemszámú monoton növekvő részsorozat utolsó előtti eleme $T[i]$, akkor az előző megjegyzéssel összhangban $l[j] = 1 + l[i]$. Ha a $T[j]$-re végződő maximális elemszámú monoton növekvő részsorozatnak nincs utolsó előtti eleme, akkor természetesen $l[j] = 1$. Ez akkor fordul elő, ha $T[j]$ kisebb az összes előtte lévő elemnél.

A fenti rekurzív egyenlet feltételezi, hogy ismerjük i értékét, bár ez nem igaz. Vizsgáljuk meg ezért az összes lehetőséget, és válasszuk ki a legjobbat. Így

$$l[j] = \begin{cases} 0 & \text{ha } j = 0, \\ 1 + \max\{l[i] \mid 0 \leqslant i \leqslant j-1 \text{ és } T[i] \leqslant T[j]\} & \text{ha } j > 0. \end{cases}$$

(A rekurzív egyenlet kompakt felírása érdekében legyen $T[0] = -\infty$ definíció szerint.) A leghosszabb monoton növekvő részsorozat hossza

$$L = \max\{l[j] \mid 1 \leqslant j \leqslant n\}.$$

A leghosszabb monoton növekvő részsorozat megtalálásához még egy jelölést vezetünk be. Minden $1 \leqslant j \leqslant n$ esetén legyen $b[j]$ egy $T[j]$-re végződő maximális hosszúságú monoton növekvő részsorozat utolsó előtti elemének az indexe. Legyen továbbá B egy leghosszabb monoton növekvő részsorozat utolsó elemének az indexe.

```
LeghosszabbMonotonNövekvőRészsorozat(T,n)
l[0]=0
for i=1 to n do
  j=0
  for h=1 to i-1 do
    if T[h]<=T[i] AND l[h]>l[j] then j=h
  l[i]=l[j]+1
  b[i]=j
L=l[1]
for i=2 to n do
  if l[i]>L then
    L=l[i]
    B=i
return l,b,L,B
```

Egy leghosszabb monoton növekvő részsorozatot a következő rekurzív eljárással határozhatunk meg. Az eljárást a (T, b, B) paraméterekkel kell meghívni.

```
Nyomtat(T,b,i)
if i>0 then
  Nyomtat(T,b,b[i])
  print T[i]
```

A `LeghosszabbMonotonNövekvőRészsorozat` eljárás költsége $O(n^2)$, a `Nyomtat` eljárásé pedig $O(n)$.

Erdős-Szekeres tétel

Feladat.
Mutassuk meg, hogy valós számok tetszőleges $n^2 + 1$ elemű sorozatának van $n + 1$ elemű monoton csökkenő vagy monoton növekvő részsorozata!

Megoldás.
Indirekt tegyük fel, hogy létezik olyan n^2+1 elemű sorozat, amelynek nincs $n+1$ elemű monoton növekvő, illetve monoton csökkenő részsorozata sem. Legyen $(x_1, x_2, \ldots, x_{n^2+1})$ egy ilyen sorozat.

Minden $1 \leqslant i \leqslant n^2+1$ esetén rendeljük hozzá az x_i elemhez az (a_i, b_i) rendezett párt, ahol a_i a leghosszabb x_i-vel kezdődő monoton növő, b_i pedig a leghosszabb x_i-vel kezdődő monoton csökkenő részsorozat hossza. Feltevésünk szerint $1 \leqslant a_i \leqslant n$ és $1 \leqslant b_i \leqslant n$, így a lehetséges (a_i, b_i) rendezett párok száma n^2. Mivel a sorozat elemszáma ennél eggyel nagyobb, ezért a skatulya elvvel összhangban létezik a sorozatnak két olyan tagja, amelyekhez ugyanazt a rendezett párt rendeltük. Legyen x_j és x_k két ilyen tag, ahol $j < k$.

Most vegyük észre, hogy ha $x_j \leqslant x_k$, akkor egy x_k-val kezdődő leghosszabb monoton növekvő részsorozat első eleme elé x_j-t beszúrva egy eggyel hosszabb, x_j-vel kezdődő monoton növekvő részsorozatot kapunk, ellentmondva annak, hogy az x_j-vel, illetve x_k-val kezdődő leghosszabb monoton növekvő részsorozatok ugyanolyan hosszúak.

Hasonlóan, ha $x_j \geqslant x_k$, akkor egy x_k-val kezdődő leghosszabb monoton csökkenő részsorozat első eleme elé x_j-t beszúrva egy eggyel hosszabb, x_j-vel kezdődő monoton csökkenő részsorozatot kapunk, ellentmondva annak, hogy az x_j-vel, illetve x_k-val kezdődő leghosszabb monoton csökkenő részsorozatok ugyanolyan hosszúak.

Szekvenciaillesztés

Feladat.
Tegyük fel, hogy adottak az $X = (x_1, x_2, \ldots, x_m)$ és $Y = (y_1, y_2, \ldots, y_n)$ karaktersorozatok. Tekintsük az $\{1, 2, \ldots, m\}$ és $\{1, 2, \ldots, n\}$ halmazokat, amelyekkel az X és Y karaktersorozatok pozícióit reprezentáljuk, és tekintsük a két halmaz egy M párosítását. Egy ilyen M párosítást szekvenciaillesztésnek nevezünk, ha tetszőleges $(i, j), (i', j') \in M$ és $i < i'$ esetén $j < j'$.

A szekvenciaillesztéseknek létezik egy elég természetes ábrázolási módja. Tekintsük például a `gatcggcat` és `caatgtgaatc` szekvenciákat és az

$$\{(1,1), (2,3), (3,4), (5,5), (6,7), (7,8), (8,9), (9,10)\}$$

szekvenciaillesztést. Ezt a következőképpen szemléltethetjük:

```
g - a t c g - g c a t -
c a a t - g t g a a t c
```

Úgy is interpretálhatjuk ezt a képet, hogy a `gatcggcat` szekvenciától eljuthatunk a `caatgtgaatc` szekvenciához, ha az első pozíción levő `g` karaktert felcseréljük a `c` karakterre, majd az első pozíció után beszúrjuk az `a` karaktert, ezután töröljük a negyedik pozíción levő `c` karaktert, majd az ötödik pozíció után beszúrjuk a `t` karaktert, aztán a hetedik pozíción levő `c` karaktert felcseréljük az `a` karakterre, végül a kilencedik pozíció után beszúrjuk a `c` karaktert.

Legyen M egy szekvenciaillesztés az X és Y karaktersorozatok között. Az M szekvenciaillesztés költségét a következőképpen definiáljuk.

- Létezik egy $g > 0$ konstans, amely a hézag költségét reprezentálja. Az X és Y minden olyan pozíciójára, amely nem szerepel egyik M-beli párban sem, felszámolunk g költséget.
- Az ábécé minden (`"p"`,`"q"`) karakterpárjához létezik egy c[`"p"`, `"q"`] $\geqslant 0$ konstans, amely a `"p"` karakternek a `"q"` karakterhez párosításának költségét reprezentálja (általában feltesszük, hogy c[`"p"`, `"p"`] $= 0$ az ábécé tetszőleges `"p"` karakterére, bár ez nem szükségszerű). Minden $(i, j) \in M$ párra felszámolunk $c[x_i, y_j]$ költséget.

Az M szekvenciaillesztés költsége legyen ezek után a fent definiált költségek összege.

Adjunk hatékony algoritmust az X és Y karaktersorozatok minimális költségű szekvenciaillesztésének meghatározására! Minél kisebb ez a költség, annál hasonlóbbnak fogjuk tekinteni az X és Y karaktersorozatokat.

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Szükségünk lesz a prefix fogalmára. Az $X = (x_1, x_2, \ldots, x_m)$ karaktersorozat $X_i = (x_1, x_2, \ldots, x_i)$ részsorozatát az X sorozat i-edik prefixének nevezzük tetszőleges $0 \leqslant i \leqslant m$ esetén.

Állítás. Legyen M optimális szekvenciaillesztés az X és Y karaktersorozatok között. Ekkor a következők legalább egyike fennáll:

(1) $(m, n) \in M$,

(2) az X karaktersorozat m-edik pozíciója nem szerepel az M-beli párok egyikében sem,

(3) az Y karaktersorozat n-edik pozíciója nem szerepel az M-beli párok egyikében sem.

Bizonyítás. Indirekt tegyük fel, hogy $(m, n) \notin M$ és léteznek olyan $i < m$ és $j < n$ pozitív egészek, amelyekre $(m, j) \in M$ és $(i, n) \in M$. Azonban ez ellentmond a szekvenciaillesztés definíciójának: itt $(i, n), (m, j) \in M$ és $i < m$, miközben $n > j$. Innen az állítás adódik.

Állítás. Legyen M optimális szekvenciaillesztés az X és Y karaktersorozatok között.

(1) Ha $(m, n) \in M$, akkor $M \setminus \{(m, n)\}$ optimális szekvenciaillesztés az X_{m-1} és Y_{n-1} prefixek között.

(2) Ha az X karaktersorozat m-edik pozíciója nem szerepel az M-beli párok egyikében sem, akkor M optimális szekvenciaillesztés az X_{m-1} prefix és Y között.

(3) Ha az Y karaktersorozat n-edik pozíciója nem szerepel az M-beli párok egyikében sem, akkor M optimális szekvenciaillesztés X és az Y_{n-1} prefix között.

Bizonyítás.
(1) Ha lenne $M \setminus \{(m, n)\}$-nél kisebb költségű szekvenciaillesztés az X_{m-1} és Y_{n-1} prefixek között, akkor ehhez hozzávéve az (m, n) párt egy M-nél kisebb költségű szekvenciaillesztést kapnánk X és Y között, ami ellentmond M optimalitásának.

(2) Ha lenne M-nél kisebb költségű szekvenciaillesztés az X_{m-1} prefix és Y között, akkor ugyanez M-nél kisebb költségű szekvenciaillesztés lenne X és Y között is, ami ellentmond M optimalitásának.

(3) Ha lenne M-nél kisebb költségű szekvenciaillesztés X és az Y_{n-1} prefix között, akkor ugyanez M-nél kisebb költségű szekvenciaillesztés lenne X és Y között is, ami ellentmond M optimalitásának.

Mit jelent ez?

- Ha az X és Y karaktersorozatok közötti optimális szekvenciaillesztés tartalmazza az (m, n) párt, akkor az optimális szekvenciaillesztés költsége nyilván $c[x_m, y_n]$ plusz az X_{m-1} és Y_{n-1} prefixek közötti optimális szekvenciaillesztés költsége.

- Ha az X és Y karaktersorozatok közötti optimális szekvenciaillesztés egyik párjában sem szerepel az X karaktersorozat m-edik pozíciója, akkor az optimális szekvenciaillesztés költsége nyilván g plusz az X_{m-1} prefix és Y közötti optimális szekvenciaillesztés költsége.

- Ha az X és Y karaktersorozatok közötti optimális szekvenciaillesztés egyik párjában sem szerepel az Y karaktersorozat n-edik pozíciója, akkor az optimális szekvenciaillesztés költsége nyilván g plusz az X és az Y_{n-1} prefix közötti optimális szekvenciaillesztés költsége.

Az optimális szekvenciaillesztés költségének meghatározásánál mindhárom lehetőséget számba kell venni és a legkisebb költségűt választani nyilvánvaló módon.

Jelölje ezek után $a[i,j]$ az X_i és Y_j prefixek közötti optimális szekvenciaillesztés költségét minden $0 \leqslant i \leqslant m$ és $0 \leqslant j \leqslant n$ esetén. Nyilván $a[i,0] = ig$ és $a[0,j] = jg$ minden $0 \leqslant i \leqslant m$ és $0 \leqslant j \leqslant n$ esetén. A többi esetben pedig az előző észrevétellel összhangban

$$a[i,j] = \min(c[x_i, y_j] + a[i-1, j-1], g + a[i-1, j], g + a[i, j-1]).$$

Egy minimális költségű szekvenciaillesztés megtalálása érdekében még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant m$ esetén jelölje $b[i,j]$ az a tömb azon elemének indexét, ahonnan $a[i,j]$ értékének meghatározásakor a legkisebb értéket kaptuk.

```
Szekvenciaillessztés(X,m,Y,n)
for i=0 to m do
  a[i,0]=i*g
for j=0 to n do
  a[0,j]=j*g
for i=1 to m do
  for j=1 to n do
    a[i,j]=c[x_i,y_j]+a[i-1,j-1]
    b[i,j]=(i-1,j-1)
    if a[i,j]>g+a[i-1,j] then
      a[i,j]=g+a[i-1,j]
      b[i,j]=(i-1,j)
    if a[i,j]>g+a[i,j-1] then
      a[i,j]=g+a[i,j-1]
      b[i,j]=(i,j-1)
return a,b
```

Az $a[0:m, 0:n]$ tömb elemeit sorfolytonosan töltjük ki, először az első sort balról jobbra, aztán a másodikat, és így tovább. Az optimális szekvenciaillesztés költsége $a[m,n]$. Az eljárás költsége nyilván $O(mn)$.

Az b tömb felhasználásával könnyű egy optimális szekvenciaillesztést megtalálni: a $b[m,n]$ elemből indulva egyszerűen csak végig kell haladni a "mutatók" mentén a tömbön. Amikor egy $b[i,j]$ elemről a $b[i-1, j-1]$ elemre

lépünk, az azt jelenti, hogy az (i, j) pár hozzátartozik az optimális szekvenciaillesztéshez. A következő rekurzív eljárás az optimális szekvenciaillesztés elemeit nyomtatja ki. Az eljárást a (b, m, n) paraméterekkel kell meghívni.

```
Nyomtat(b,i,j)
if i=0 OR j=0 then return
if b[i,j]=(i-1,j-1)
  then
    Nyomtat(b,i-1,j-1)
    print '(' i ',' j ')'
  else
    if b[i,j]=(i-1,j)
      then Nyomtat(b,i-1,j)
      else Nyomtat(b,i,j-1)
```

Az eljárás költsége $O(m+n)$, hiszen minden rekurzív hívásnál i és j legalább egyikének csökken az értéke.

Leghosszabb palindrom részsorozat

Feladat.
Egy $P[1 : k]$ karakterláncot palindromnak nevezünk, ha balról jobbra és jobbról balra olvasva is ugyanaz, vagyis $P[i] = P[k-i+1]$ minden $1 \leqslant i \leqslant k$ esetén.

Adjunk hatékony algoritmust egy $S[1 : n]$ karakterlánc leghosszabb palindrom részsorozatának megtalálására!

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Először is vegyük észre a következőt.

Állítás. Legyen $S[1 : n]$ egy karaktersorozat és $P[1 : k]$ ennek egy leghosszabb palindrom részsorozata. Ekkor

(1) ha $S[1] = S[n]$, akkor $P[1] = S[1] = S[n] = P[k]$ és $P[2 : k-1]$ egy leghosszabb palindrom részsorozata $S[2 : n-1]$-nek,

(2) ha $S[1] \neq S[n]$ és $P[k] \neq S[n]$, akkor $P[1 : k]$ egy leghosszabb palindrom részsorozata $S[1 : n-1]$-nek,

(3) ha $S[1] \neq S[n]$ és $P[1] \neq S[1]$, akkor $P[1 : k]$ egy leghosszabb palindrom részsorozata $S[2 : n]$-nek.

Bizonyítás.
(1) Ha $P[k] \neq S[n]$, akkor az $S[n]$ karaktert hozzáfűzhetnénk $P[1:k]$ elejéhez és végéhez, ami által $S[1:n]$ egy $k+2$ hosszú palindrom részsorozatát kapnánk, ellentmondás. Így $P[1] = S[1] = S[n] = P[k]$. Világos, hogy $P[2:k-1]$ palindrom részsorozata $S[2:n-1]$-nek. Ha lenne ennél hosszabb palindrom részsorozata is $S[2:n-1]$-nek, akkor ennek elejéhez és végéhez az $S[n]$ karaktert hozzáfűzve $S[1:n]$-nek egy $P[1:k]$-nál hosszabb palindrom részsorozatát kapnánk, ami nem lehetséges. Így $P[2:k-1]$ szükségképpen egy leghosszabb palindrom részsorozata $S[2:n]$-nek.

(2) Ha $S[1] \neq S[n]$ és $P[k] \neq S[n]$, akkor $P[1:k]$ palindrom részsorozata $S[1:n-1]$-nek. Ha $S[1:n-1]$-nek volna $P[1:k]$-nál hosszabb palindrom részsorozata, az $P[1:k]$-nál hosszabb palindrom részsorozata lenne $S[1:n]$-nek is, ellentmondás.

(3) Ugyanúgy, mint az előbb.

Az előző állítás szerint csak egy vagy két részfeladat van, amit meg kell vizsgálni az $S[1:n]$ karakterlánc egy leghosszabb palindrom részsorozatának megkeresésekor.

- Ha $S[1] = S[n]$, akkor elég $S[2:n-1]$ egy leghosszabb palindrom részsorozatát megtalálni, amelynek elejéhez és végéhez az $S[1] = S[n]$ karaktert fűzve $S[1:n]$ egy leghosszabb palindrom részsorozatához jutunk.

- Ha $S[1] \neq S[n]$, akkor két részfeladatot kell megoldani: $S[1:n-1]$, illetve $S[2:n]$ egy-egy leghosszabb palindrom részsorozatát kell megkeresni. Ezek közül a hosszabb $S[1:n]$-nek is egy leghosszabb palindrom részsorozata lesz.

Ezek után minden $1 \leqslant i \leqslant j \leqslant n$ esetén jelölje $l[i,j]$ az $S[i:j]$ karakterlánc leghosszabb palindrom részsorozatának a hosszát. Ha $i = j$, akkor a leghosszabb palindrom részsorozat hossza természetesen 1. Ha $j = i+1$ akkor a leghosszabb palindrom részsorozat hossza $S[i] = S[j]$ esetén 2, különben pedig 1. A többi esetben a fenti észrevétel igazít el:

$$l[i,j] = \begin{cases} l[i+1,j-1]+2 & \text{ha } S[i] = S[j], \\ \max(l[i+1,j], l[i,j-1]) & \text{ha } S[i] \neq S[j]. \end{cases}$$

Az optimális megoldás értékét alulról felfelé történő módon határozzuk meg.

```
LeghosszabbPalindromRészsorozat(S,n)
for i=1 to n do
  l[i,i]=1
for i=1 to n-1 do
  if S[i]=S[i+1]
    then l[i,i+1]=2
    else l[i,i+1]=1
for d=2 to n-1 do
  for i=1 to n-d do
    j=i+d
    if S[i]=S[j]
      then l[i,j]=l[i+1,j-1]+2
      else l[i,j]=max(l[i+1,j],l[i,j-1])
return l[1,n]
```

Az algoritmus költsége $O(n^2)$.

Magát egy leghosszabb palindrom részsorozatot az l tömb ismeretében a következő eljárással határozhatunk meg. Az eljárást az $(S, 1, n)$ paraméterekkel kell meghívni. Az eljárás költsége $O(n)$.

```
Nyomtat(S,i,j)
if j=i then
  print S[i]
  return
if j=i+1 then
  if S[i]=S[j]
    then
      print S[i]
      print S[j]
      return
    else
      print S[i]
      return
if S[i]=S[j] then
  print S[i]
  Nyomtat(S,i+1,j-1)
  print S[j]
  return
if l[i+1,j]>l[i,j-1]
  then Nyomtat(S,i+1,j)
  else Nyomtat(S,i,j-1)
```

Leghosszabb ismétlődés

Feladat.
Tekintsünk egy $S[1:n]$ karakterláncot. Ha léteznek olyan $1 \leqslant i < j \leqslant n$ indexek, hogy $S[i:i+k] = S[j:j+k]$ valamilyen $k \geqslant 0$ egészre, akkor azt mondjuk, hogy az S karakterláncban előfordul egy $k+1$ hosszúságú ismétlődés.

Adjunk hatékony algoritmust az $S[1:n]$ karakterláncban előforduló leghosszabb ismétlődés megtalálására!

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Szükségünk lesz a prefix fogalmára. Az x és y karakterláncok xy konkatenációja az a karakterlánc, melyben x karaktereit y karakterei követik. A w karakterlánc az x karakterlánc prefixe, ha van olyan nem üres y karakterlánc, hogy $x = wy$. Minden $1 \leqslant i < j \leqslant n$ esetén jelölje $p[i,j]$ az $S[i:n]$ és $S[j:n]$ karakterláncok leghosszabb közös prefixének a hosszát. Vegyük észre, hogy tetszőleges $1 \leqslant i < j \leqslant n$ esetén

$$p[i,j] = \begin{cases} 0 & \text{ha } S[i] \neq S[j], \\ 1 + p[i+1, j+1] & \text{ha } S[i] = S[j]. \end{cases}$$

(A rekurzív egyenlet kompakt felírása érdekében legyen $p[i, n+1] = 0$ definíció szerint minden $1 \leqslant i \leqslant n$ esetén.)

Ezek után egy leghosszabb ismétlődő karakterlánc hossza világos módon

$$L = \max\{p[i,j] \mid 1 \leqslant i < j \leqslant n\}.$$

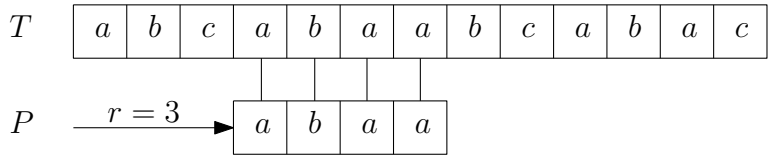
```
LeghosszabbIsmétlődés(S,n)
for i=1 to n do
  p[i,n+1]=0
for i=n-1 downto 1 do
  for j=i+1 to n do
    if S[i]=S[j]
      then p[i,j]=1+p[i+1,j+1]
      else p[i,j]=0
L=0
for i=1 to n-1 do
  for j=i+1 to n do
    if p[i,j]>L then L=p[i,j]
return L
```

Az algoritmus költsége $O(n^2)$.

Ha egy leghosszabb ismétlődő karakterláncra is kíváncsiak vagyunk, akkor valahányszor frissítjük L értékét az utolsó egymásba ágyazott ciklus belsejében, jegyezzük fel azt is, hogy melyik (i, j) párnál történt ez meg.

Mintaillesztés

Feladat.
Szövegszerkesztő programokban gyakori feladat megkeresni egy szövegben egy minta összes előfordulását. Tegyük fel, hogy a szöveget egy n elemű $T[1:n]$ tömb, a mintát pedig egy m elemű $P[1:m]$ tömb tartalmazza, ahol $m \leqslant n$. Azt mondjuk, hogy a P minta r eltolással előfordul a T szövegben, vagy másképpen fogalmazva a T szöveg $(r+1)$-edik pozíciójára illeszkedik, ha $0 \leqslant r \leqslant n-m$ és $T[r+1:r+m] = P[1:m]$, azaz minden $j = 1, 2, \ldots, m$ esetén $T[r+j] = P[j]$. Ha a P minta r eltolással előfordul T-ben, akkor r érvényes eltolás, ellenkező esetben r érvénytelen eltolás.



Adjunk hatékony algoritmust egy adott P minta összes érvényes eltolásának megtalálására egy rögzített T szövegben!

Megoldás.
Kezdjük néhány elnevezéssel. Az x és y karakterláncok xy konkatenációja az a karakterlánc, melyben x karaktereit y karakterei követik. A w karakterlánc az x karakterlánc valódi prefixe, ha van olyan nem üres y karakterlánc, hogy $x = wy$. Hasonlóan, a w karakterlánc az x karakterlánc valódi szuffixe, ha van olyan nem üres y karakterlánc, hogy $x = yw$. A w karakterlánc az x karakterlánc szegélye, ha w az x leghosszabb olyan valódi prefixe, amely egyben valódi szuffixe is x-nek. Tehát w a leghosszabb olyan karakterlánc, amelyre alkalmas nem üres u és v karakterláncokkal $x = wu$ és $x = vw$.

Először a P mintához előállítunk egy $S[1:m]$ tömböt, ahol $S[j]$ definíció szerint a $P[1:j]$ karakterlánc szegélyének hossza minden $1 \leqslant j \leqslant m$ esetén. Ehhez dinamikus programozást használunk.

Nyilvánvaló módon $S[1] = 0$. Tegyük fel ezután, hogy $j > 1$ és az $S[1:j-1]$ résztömböt már kitöltöttük. Ekkor $S[j]$ a következőképpen számítható.

Jelölje w a $P[1:j]$ karakterlánc szegélyét és l a w hosszát. Ekkor $S[j]=l$. Először is jegyezzük meg, hogy $S[j]$ értéke legfeljebb $S[j-1]+1$ lehet, hiszen ha $P[1:l]=P[j-l+1:j]$, akkor $P[1:l-1]=P[j-l+1:j-1]$ is nyilván igaz, vagyis a $P[1:j-1]$ karakterláncnak van olyan $l-1$ hosszú valódi prefixe, amely egyben valódi szuffixe is.

Ha $P[j]=P[S[j-1]+1]$, akkor $S[j]=S[j-1]+1$, és viszont. Ezután tegyük fel, hogy $P[j]\neq P[S[j-1]+1]$. Ekkor persze $S[j]\leqslant S[j-1]$. Most a $w[1:l-1]$ karakterlánc valódi prefixe a $P[1:S[j-1]]$ karakterláncnak és valódi szuffixe a $P[j-S[j-1]:j-1]$ karakterláncnak. Ám ez utóbbi két karakterlánc az S tömb definíciója szerint azonos, így a w karakterlánc valódi prefixe és egyben valódi szuffixe is annak a karakterláncnak, amit úgy kapunk, hogy a $P[1:S[j-1]]$ karakterlánc végére fűzzük a $P[j]$ karaktert. Vegyük észre, hogy ha lenne ennek a karakterláncnak l-nél hosszabb olyan valódi prefixe, amely egyben valódi szuffixe is, akkor ez a $P[1:j]$ karakterláncnak szintén egy l-nél hosszabb olyan valódi prefixe lenne, amely egyben valódi szuffixe is, ellentmondás. Következésképpen w szegélye annak a karakterláncnak, amit úgy kapunk, hogy a $P[1:S[j-1]]$ karakterlánc végére fűzzük a $P[j]$ karaktert.

Lássuk ezek után az S tömb elemeit számító eljárást.

```
SzegélySzámítás(P,m)
S[1]=0
k=0
for j=2 to m do
  while k>0 AND P[k+1]<>P[j] do
    k=S[k]
  if P[k+1]=P[j] then k=k+1
  S[j]=k
return S
```

Meghatározzuk az S tömb kitöltésének a karakterösszehasonlítások számában mért költségét. Jelölje b_j az $S[j]$ elem számításánál felhasznált karakterösszehasonlítások számát (ebbe most $S[1:j-1]$ számításának költségét nem értjük bele). A k változó értéke kezdetben $S[j-1]$, ez a while ciklus magjának minden lefutása után legalább eggyel csökken. A ciklusmag legalább b_j-2 alkalommal végrehajtásra kerül, ezért a while ciklusból kilépve k értéke legfeljebb $S[j-1]-b_j+2$. Ezután k értéke eggyel nőhet még, így végső értéke, ami $S[j]$, legfeljebb $S[j-1]-b_j+3$. Ennélfogva

$$b_j\leqslant S[j-1]-S[j]+3.$$

Az összehasonlítások száma ezek után

$$\sum_{j=2}^{m} b_j = \sum_{j=2}^{m} (S[j-1] - S[j] + 3) = S[1] - S[m] + 3(m-1) \leqslant 3(m-1).$$

Így az S tömb kitöltésének költége $O(m)$.

Ezután lássuk magát az algoritmust. Tegyük fel, hogy éppen a T szöveg i-edik karakterét a P minta $(k+1)$-edik karakterével hasonlítjuk össze, és már tudjuk, hogy $T[i-k] = P[1], \ldots, T[i-1] = P[k]$, azaz a minta első k karakterénél illeszkedést találtunk.

Először tegyük fel, hogy $P[k+1] = T[i]$. Ha $k+1 < m$, akkor a szövegben és a mintában is egy hellyel jobbra lépünk (k és i értéke eggyel nő). Ha $k+1 = m$, akkor találtunk egy P-vel azonos részkarakterláncot T-ben. Ilyenkor a szövegben egy hellyel jobbra lépünk, a mintában pedig $m - S[m]$ hellyel vissza (k értéke $S[m]$ lesz, i értéke eggyel nő). Most egyrészt P első $S[m]$ darab karaktere illeszkedni fog a T szöveg i-edik karaktere előtti $S[m]$ hosszú részhez, másrészt ha kevesebb hellyel lépünk vissza, akkor S definíciója miatt a T szöveg i-edik karaktere előtt nem illeszkedhet minden karakter.

Tegyük fel ezután, hogy $P[k+1] \neq T[i]$. Ha $k = 0$, azaz éppen P első karakterénél tartunk, akkor a T szövegben egy hellyel jobbra lépünk (k értéke nem változik, i értéke eggyel nő). Ha $k > 0$, akkor a mintában $k - S[k]$ hellyel visszalépünk (k értéke $S[k]$ lesz, i nem változik). Ez utóbbi esetben egyrészt P első $S[k]$ darab karaktere illeszkedni fog a T szöveg i-edik karaktere előtt lévő $S[k]$ hosszú részhez, másrészt ha kevesebb hellyel lépünk vissza, akkor S definíciója miatt a T szöveg i-edik karaktere előtt nem illeszkedhet minden karakter.

```
Mintaillesztés(T,n,P,m)
S=SzegélySzámítás(P,m)
k=0
for i=1 to n do
  while k>0 AND P[k+1]<>T[i] do
    k=S[k]
  if P[k+1]=T[i] then k=k+1
  if k=m then
    print 'A minta illeszkedik a(z) ' i-m+1 '. pozícióra'
    k=S[k]
```

A költséget itt is a karakterösszehasonlítások számában mérjük. Vizsgáljuk az i és az $i-k$ mennyiségeket az egymás utáni karakterösszehasonlítások pillanataiban. Az első karakterösszehasonlításkor $i = 1$ és $k = 0$, tehát

$i - k = 1$. Vegyük észre, hogy az i és $i - k$ mennyiségek semelyik karakterösszehasonlításkor sem kisebbek, mint az azt megelőző karakterösszehasonlításkor. Vegyük észre azt is, hogy a while ciklusok utáni n darab karakterösszehasonlítást leszámítva, az i és $i-k$ mennyiségek legalább egyike minden karakterösszehasonlításkor nagyobb, mint az azt megelőző karakterösszehasonlításkor. Innen $i \leqslant n$ és $i - k \leqslant n$ felhasználásával következik, hogy a karakterösszehasonlítások száma összesen legfeljebb $2n - 1 + n = 3n - 1$.

Így az algoritmus teljes költsége az S tömb számításával együtt $O(m+n)$.

Nyomtatás

Feladat.
Feladatunk az $s_1, s_2, \ldots, s_n$ szavakból álló bekezdés kinyomtatása. A szavak rendre $l_1, l_2, \ldots, l_n$ karakterből állnak. A nyomtató egy sorba összesen M karaktert tud nyomtatni. Tegyük fel, hogy $l_i \leqslant M$ minden $1 \leqslant i \leqslant n$ esetén. Ha egy sor az s_i-től s_j-ig terjedő szavakat tartalmazza, akkor a szavak között mindig egy szóköz van, míg a sor végén további

$$M - j + i - \sum_{m=i}^{j} l_m$$

extra szóköz. Ez utóbbi mennyiségnek természetesen nem negatívnak kell lenni.

Adjunk hatékony algoritmust, amely a lehető "legszebben" nyomtatja ki a bekezdést, vagyis amely minimalizálja az utolsó sor kivételével a sorok végén található extra szóközök számának köbeinek összegét!

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Minden $1 \leqslant i \leqslant j \leqslant n$ esetén legyen

$$e[i, j] = M - j + i - \sum_{m=i}^{j} l_m.$$

Ez a mennyiség az s_i-től s_j-ig terjedő szavakat tartalmazó sor végén lévő extra szóközök száma. Jegyezzük meg, hogy $e[i, j]$ negatív is lehet; ez azt jelenti, hogy az s_i-től s_j-ig terjedő szavak nem férnek el egy sorban.

Szintén minden $1 \leqslant i \leqslant j \leqslant n$ esetén legyen

$$k[i, j] = \begin{cases} \infty & \text{ha } e[i, j] < 0, \\ 0 & \text{ha } j = n \text{ és } e[i, j] \geqslant 0, \\ e[i, j]^3 & \text{különben.} \end{cases}$$

Ezzel a mennyiséggel járul hozzá az s_i-től s_j-ig terjedő szavakat tartalmazó sor az optimalizálandó összeghez, amennyiben szerepel a kinyomtatott bekezdésben. Jegyezzük meg, hogy a $k[i,j]$ mennyiséget végtelennek definiáljuk abban az esetben, amikor az s_i-től s_j-ig terjedő szavak nem férnek el egy sorban; ezzel biztosítjuk, hogy az optimális megoldásban a sorok nem fognak sehol túlcsordulni (ne feledjük, önmagában minden szó elfér egy sorban). Azt is jegyezzük meg, hogy a $k[i,n]$ mennyiségeket nullának definiáljuk (már ha az s_i-től s_n-ig terjedő szavak elférnek egy sorban); ezzel biztosítjuk, hogy az utolsó sor végén lévő extra szóközök nem számítanak bele az optimalizálandó összegbe.

Tekintsük most az $s_1, s_2, \ldots, s_j$ szavak egy optimális elrendezését. Tegyük fel hogy ebben az utolsó sor az s_i szóval kezdődik. Vegyük észre, hogy ekkor az utolsó előtti sorig terjedő rész az $s_1, s_2, \ldots, s_{i-1}$ szavak egy optimális elrendezése. Valóban, ha nem így lenne, akkor ezt a részt lecserélve az $s_1, s_2, \ldots, s_{i-1}$ szavak egy optimális elrendezésére az $s_1, s_2, \ldots, s_j$ szavak egy "szebb" elrendezéséhez jutnánk, ami ellentmondás.

Minden $0 \leqslant j \leqslant n$ esetén jelölje $c[j]$ az $s_1, s_2, \ldots, s_j$ szavak optimális elrendezésének soraihoz rendelt $k[\,]$ értékek összegét.

- Ha $j = 0$, akkor $c[j] = 0$ triviális módon.
- Ha $1 \leqslant j \leqslant n$ és az optimális elrendezés utolsó sora az s_i szóval kezdődik, akkor az előző megjegyzéssel összhangban

$$c[j] = c[i-1] + k[i,j].$$

Ez a rekurzív egyenlet feltételezi, hogy ismerjük i értékét, bár ez nem igaz. Azonban i legfeljebb j különböző értéket vehet fel; vizsgáljuk meg az összes lehetőséget, és válasszuk ki a legjobbat. Így

$$c[j] = \begin{cases} 0 & \text{ha } j = 0, \\ \min\{c[i-1] + k[i,j] \mid 1 \leqslant i \leqslant j\} & \text{ha } j > 0. \end{cases}$$

A "legszebb" nyomtatás megtalálása érdekében még egy jelölést vezetünk be. Minden $1 \leqslant j \leqslant n$ esetén legyen $p[j]$ annak az s_i szónak az indexe, amellyel az utolsó sor kezdődik az $s_1, s_2, \ldots, s_j$ szavak optimális elrendezésénél.

```
Nyomtatás(l,n,M)
for i=1 to n do
  e[i,i]=M-l[i]
  for j=i+1 to n do
```

```
    e[i,j]=e[i,j-1]-l[j]-1
for i=1 to n do
  for j=i to n do
    if e[i,j]<0
      then k[i,j]=INFINITY
      else
        if j=n AND e[i,j]>=0
          then k[i,j]=0
          else k[i,j]=e[i,j]^3
c[0]=0
for j=1 to n do
  c[j]=INFINITY
  for i=1 to j do
    if c[i-1]+k[i,j]<c[j] then
      c[j]=c[i]+k[i,j]
      p[j]=i
return c,p
```

Az optimális tördelést a következő rekurzív eljárással határozhatjuk meg. Az eljárást a (p, n) paraméterekkel kell meghívni.

```
Tördelés(p,j)
i=p[j]
if i=1
  then h=1
  else h=Tördelés(p,i-1)+1
print (h,i,j)
return h
```

Az eljárás az optimálisan tördelt bekezdés sorainak számával tér vissza. A kinyomtatott (h, i, j) hármasok azt mutatják, hogy a h-adik sor az s_i-től s_j-ig terjedő szavakat tartalmazza ($h = 1, 2, \ldots$).

A `Nyomtatás` eljárás költsége $O(n^2)$, a `Tördelés` eljárásé pedig $O(n)$.

Csillagászati megfigyelések

Feladat.
Tegyük fel, hogy adott csillagászati megfigyelések egy $M = \{m_1, m_2, \ldots, m_n\}$ halmaza, amelyeket különböző kutatócsoportok a Hubble űrtávcsővel szeretnének elvégezni. A megfigyelések nem szakíthatók meg, és az űrtávcső egy

időben legfeljebb egy megfigyelést tud végrehajtani. Minden m_i megfigyelésnek adott az s_i kezdési és az f_i befejezési időpontja, ahol $0 \leqslant s_i < f_i < \infty$. Ezen kívül minden m_i megfigyelésnek adott a w_i tudományos súlya, ahol $w_i > 0$. Az m_i és m_j megfigyeléseket egymást kizáróknak nevezzük, ha az $[s_i, f_i]$ és $[s_j, f_j]$ intervallumoknak van közös belső pontja. Ellenkező esetben azt mondjuk, hogy az m_i és m_j megfigyelések kompatibilisek. Feladatunk páronként kompatibilis M-beli megfigyelések egy olyan $\{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ részhalmazának a meghatározása, amelyre $w_{i_1} + w_{i_2} + \cdots + w_{i_k}$ maximális.

(A) Tervezzünk hatékony mohó algoritmust, amely megoldja a problémát abban az esetben, amikor minden megfigyelés tudományos súlya ugyanakkora!

(B) Tervezzünk hatékony dinamikus programozás algoritmust, amely megoldja a problémát abban az esetben is, amikor a megfigyelések tudományos súlya nem feltétlenül ugyanakkora!

Megoldás.
(A) Ha a megfigyelések tudományos súlya ugyanakkora, akkor a feladat egyszerűen páronként kompatibilis M-beli megfigyelések egy maximális elemszámú $\{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ részhalmazának a meghatározása. Tegyük fel, hogy a megfigyelések a befejezési idejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$f_1 \leqslant f_2 \leqslant \cdots \leqslant f_n.$$

A kiválasztott megfigyeléseket egy H halmazban fogjuk tárolni. Kezdetben legyen $H = \emptyset$. Használni fogunk még egy G listát, amely mindig a még nem vizsgált megfigyeléseket tartalmazza a befejezési idejük szerint monoton növekvően rendezetten. Kezdetben G az összes M-beli megfigyelésből áll. Először rakjuk át G-ből H-ba a legkisebb befejezési időpontú megfigyelést, majd töröljük G-ből az ezzel egymást kizáró megfigyeléseket, amíg egy kompatibilis megfigyelést nem találunk. Ismételjük ezek után a következőt, amíg G-ből nem töröltük az összes megfigyelést. Rakjuk át G-ből H-ba a legkisebb befejezési időpontú megfigyelést, majd töröljük G-ből az ezzel egymást kizáró megfigyeléseket, amíg egy kompatibilis megfigyelést nem találunk.

Legyen $\mathcal{H} = \{m_{j_1}, m_{j_2}, \ldots, m_{j_l}\}$ az algoritmus által kiválasztott megfigyelések halmaza, ahol $j_1 < j_2 < \cdots < j_l$. Világos módon a $\mathcal{H}$-beli megfigyelések páronként kompatibilisek. Belátjuk, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak. Tekintsük a feladat egy $\mathcal{O} = \{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ optimális megoldását, ahol $i_1 < i_2 < \cdots < i_k$. Nyilván $l \leqslant k$.

Először teljes indukcióval megmutatjuk, hogy $f_{j_r} \leqslant f_{i_r}$ minden $1 \leqslant r \leqslant l$ esetén. Ha $r = 1$, akkor ez nyilvánvaló, hiszen az algoritmus elsőként a legkisebb befejezési idejű m_1 megfigyelést választja ki. Legyen most $r > 1$,

és tegyük fel, hogy $f_{j_{r-1}} \leqslant f_{i_{r-1}}$. Mivel az $\mathcal{O}$-beli megfigyelések páronként kompatibilisek, ezért $f_{i_{r-1}} \leqslant s_{i_r}$, így $f_{j_{r-1}} \leqslant s_{i_r}$. Ez azt jelenti, hogy az m_{i_r} megfigyelés benne volt abban a halmazban, amelyből az algoritmus a legkisebb befejezési időpontú m_{j_r} megfigyelést választotta, következésképpen $f_{j_r} \leqslant f_{i_r}$.

Indirekt tegyük fel ezután, hogy $\mathcal{H}$ nem egy optimális megoldása a feladatnak, vagyis $l < k$. Az előzőek szerint $f_{j_l} \leqslant f_{i_l}$. Mivel $l < k$, ezért $\mathcal{O}$ tartalmaz egy $f_{i_{l+1}}$ megfigyelést is. Mivel az $\mathcal{O}$-beli megfigyelések páronként kompatibilisek, ezért $f_{i_l} \leqslant s_{i_{l+1}}$, következésképpen $f_{j_l} \leqslant s_{i_{l+1}}$. Ez viszont azt jelenti, hogy miután töröltük G-ből az m_{j_l} megfigyeléssel egymást kizáró megfigyeléseket, az $m_{i_{l+1}}$ megfigyelés még G-ben maradt, így az algoritmus nem fejeződhetett be, ellentmondás. Ebből következik, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak.

Az algoritmus lépésszáma a kezdeti rendezés nélkül $O(n)$, azzal együtt $O(n \log n) + O(n) = O(n \log n)$.

(B) Tegyük fel ismét, hogy a megfigyelések a befejezési idejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$f_1 \leqslant f_2 \leqslant \cdots \leqslant f_n.$$

Minden $2 \leqslant i \leqslant n$ esetén legyen $\gamma(i)$ az a legnagyobb $j < i$ index, amelyre az m_j és m_i megfigyelések kompatibilisek. Ha az m_j és m_i megfigyelések egymást kizáróak minden $1 \leqslant j < i$ esetén, akkor legyen $\gamma(i) = 0$, továbbá legyen $\gamma(1) = 0$. A $\gamma(i)$ értékek a megfigyelések befejezési idejeinek monoton növekvően rendezett sorozatában történő bináris kereséssel egyszerűen meghatározhatók.

Minden $1 \leqslant i \leqslant n$ esetén jelölje $v[i]$ azon feladat egy optimális megoldásában a megfigyelések összsúlyát, amikor az $m_1, m_2, \ldots, m_i$ megfigyelések közül választhatunk. A formulák kompakt felírhatóságának érdekében legyen $v[0] = 0$.

Legyen most $1 \leqslant i \leqslant n$, és legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, mikor az $m_1, m_2, \ldots, m_i$ megfigyelések közül választhatunk.

- Ha m_i szerepel az $\mathcal{O}$-beli megfigyelések között, akkor $\mathcal{O}' = \mathcal{O} \setminus \{m_i\}$ egy optimális megoldása annak a feladatnak, amikor az $m_1, m_2, \ldots, m_{\gamma(i)}$ események közül választhatunk (ne feledjük, hogy az $m_{\gamma(i)+1}, \ldots, m_{i-2}$, m_{i-1} megfigyelések egymást kizáróak az m_i megfigyeléssel). Valóban, ha lenne $\mathcal{O}'$-nél nagyobb összsúlyú megoldása annak a feladatnak, amikor az $m_1, m_2, \ldots, m_{\gamma(i)}$ megfigyelések közül választhatunk, akkor ehhez hozzávéve az m_i megfigyelést egy $\mathcal{O}$-nál nagyobb összsúlyú megoldását kapnánk annak a feladatnak, mikor az $m_1, m_2, \ldots, m_i$ megfigyelések

közül választhatunk, ellentmondás. Az $\mathcal{O}$-beli megfigyelések összsúlya ebben az esetben $w_i + v[\gamma(i)]$.

- Ha m_i nem szerepel az $\mathcal{O}$-beli megfigyelések között, akkor $\mathcal{O}$ nyilván egy optimális megoldása annak a feladatnak, mikor az $m_1, m_2, \ldots, m_{i-1}$ megfigyelések közül választhatunk. Ebben az esetben az $\mathcal{O}$-beli megfigyelések összsúlya $v[i-1]$.

Mivel nem tudjuk, hogy m_i szerepel-e az $\mathcal{O}$-beli megfigyelések között, ezért mindkét lehetőséget megvizsgáljuk, és a kedvezőbbet választjuk. Így

$$v[i] = \max(w_i + v[\gamma(i)], v[i-1]).$$

A $v[i]$ értékek kiszámítását úgy tekinthetjük, mintha egy $n+1$ elemű tömböt töltenénk ki balról jobbra. A feladat optimális megoldásában a megfigyelések összsúlya $v[n]$.

Minden $v[i]$ érték $O(1)$ lépésben számítható, így a $v[0:n]$ tömb kitöltésének lépésszáma $O(n)$. Ehhez jön még a kezdeti rendezés és a $\gamma(i)$ értékek meghatározásának lépésszáma, ami $O(n \log n)$. Így az algoritmus teljes lépésszáma $O(n \log n) + O(n) = O(n \log n)$.

Ha minden $1 \leqslant i \leqslant n$ esetén feljegyezzük, hogy $v[i]$ számításánál kedvezőbb volt-e az m_i megfigyelést kiválasztani vagy nem, akkor magát egy optimális megoldást is könnyen rekonstruálhatunk.

Fesztiválbérlet

Feladat.
Egy zenei fesztivál szervezői a fesztiválbérlet árát a következők alapján szeretnék meghatározni. Minden programhoz megállapítanak egy elvi részvételi díjat, amit teljes egészében ki kellene fizetni egy látogatónak, ha a programon akárcsak részben részt vesz. Legalább mennyit kellene így fizetni egy látogatónak, ha a fesztivál ideje alatt végig részt venne valamilyen programon? Feltehetjük, hogy a fesztiválon soha nincs üresjárat, mindig zajlik legalább egy program.

Formálisan, adott a fesztiválnak az s kezdési és az f befejezési időpontja. Adott továbbá a programok $P = \{p_1, p_2, \ldots, p_n\}$ halmaza. Minden p_i programnak adott az s_i kezdési és az f_i befejezési időpontja, ahol $s \leqslant s_i < f_i \leqslant f$. Feltételünk szerint

$$[s_1, f_1] \cup [s_2, f_2] \cup \cdots \cup [s_n, f_n] = [s, f].$$

Ezen kívül minden p_i programnak adott a w_i részvételi díja, ahol $w_i > 0$. Feladatunk P-beli programok egy olyan $\{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ részhalmazának a meghatározása, amelyre

$$[s_{i_1}, f_{i_1}] \cup [s_{i_2}, f_{i_2}] \cup \cdots \cup [s_{i_k}, f_{i_k}] = [s, f]$$

és amelyre $w_{i_1} + w_{i_2} + \cdots + w_{i_k}$ minimális.

(A) Tervezzünk hatékony mohó algoritmust, amely megoldja a problémát abban az esetben, amikor minden program elvi részvételi díja ugyanakkora!

(B) Tervezzünk hatékony dinamikus programozás algoritmust, amely megoldja a problémát abban az esetben is, amikor a programok elvi részvételi díja nem feltétlenül ugyanakkora!

Megoldás.

(A) Ha a programok elvi részvételi díja ugyanakkora, akkor a feladat egyszerűen P-beli programok egy olyan minimális elemszámú $\{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ részhalmazának a meghatározása, amelyre

$$[s_{i_1}, f_{i_1}] \cup [s_{i_2}, f_{i_2}] \cup \cdots \cup [s_{i_k}, f_{i_k}] = [s, f].$$

Tegyük fel, hogy a programok a kezdési idejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n.$$

A kiválasztott programokat egy H halmazban fogjuk tárolni. Kezdetben legyen $H = \emptyset$. Használni fogunk még egy G listát, amely mindig a még nem vizsgált programokat tartalmazza a kezdési idejük szerint monoton növekvően rendezetten. Kezdetben G az összes P-beli programból áll. Először töröljük G-ből azokat a programokat, amelyek kezdési időpontja s, majd ezek közül vegyünk hozzá H-hoz egy legkésőbbi befejezési időpontút. Ismételjük ezek után a következőt, amíg G-ből nem töröltük az összes programot. Tegyük fel, hogy H-hoz legutóbb az m_j programot vettük hozzá. Ha $f_j = f$, akkor töröljük G-ből az összes programot (ezzel befejeződik az algoritmus). Ha viszont $f_j < f$, akkor töröljük G-ből azokat a programokat, amelyek az f_j időpont előtt vagy éppen az f_j időpontban kezdődnek, majd ezek közül vegyünk hozzá H-hoz egy legkésőbbi befejezési időpontút (jegyezzük meg, hogy ez a program szükségképpen az f_j időpont után fejeződik be).

Legyen $\mathcal{H} = \{p_{j_1}, p_{j_2}, \ldots, p_{j_l}\}$ az algoritmus által kiválasztott programok halmaza, ahol $j_1 < j_2 < \cdots < j_l$. Világos módon

$$[s_{j_1}, f_{j_1}] \cup [s_{j_2}, f_{j_2}] \cup \cdots \cup [s_{j_l}, f_{j_l}] = [s, f].$$

Belátjuk, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak. Tekintsük a feladat egy $\mathcal{O} = \{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ optimális megoldását, ahol $i_1 < i_2 < \cdots < i_k$. Nyilván $l \geqslant k$.

Először teljes indukcióval megmutatjuk, hogy $f_{j_r} \geqslant f_{i_r}$ minden $1 \leqslant r \leqslant k$ esetén. Ha $r = 1$, akkor ez nyilvánvaló, hiszen az algoritmus elsőként az s időpontban kezdődő programok közül egy legnagyobb befejezési időpontút választ ki. Legyen most $r > 1$, és tegyük fel, hogy $f_{j_{r-1}} \geqslant f_{i_{r-1}}$. Mivel $\mathcal{O}$ egy optimális megoldása a feladatnak, ezért $s_{i_r} \leqslant f_{i_{r-1}}$. Másrészt az $f_{j_{r-1}}$ időpont előtt vagy éppen az $f_{j_{r-1}}$ időpontban kezdődő programok közül m_{j_r} egy legkésőbbi befejezési időpontú. Ennélfogva $f_{j_r} \geqslant f_{i_r}$.

Indirekt tegyük fel ezután, hogy $\mathcal{H}$ nem egy optimális megoldása a feladatnak, vagyis $l > k$. Az előzőek szerint $f_{j_k} \geqslant f_{i_k}$. Mivel $\mathcal{O}$ egy optimális megoldása a feladatnak, ezért $f_{i_k} = f$, következésképpen $f_{j_k} = f$ szintén. Ez viszont azt jelenti, hogy az algoritmus az m_{j_k} program kiválasztása után befejeződött, ami ellentmond annak, hogy $l > k$. Ebből következik, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak.

Az algoritmus lépésszáma a kezdeti rendezés nélkül $O(n)$, azzal együtt $O(n \log n) + O(n) = O(n \log n)$.

(B) Tegyük fel, hogy a programok a kezdési idejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n.$$

Feltételünk szerint $s_1 = s$.

A formulák kompakt felírhatóságának érdekében legyen $s_{n+1} = f$, és legyen m az a legkisebb $2 \leqslant j \leqslant n+1$ index, amelyre $s_j > s$. Legyen továbbá minden $m \leqslant i \leqslant n+1$ esetén $\gamma(i)$ az a legnagyobb $j < i$ index, amelyre $s_j < s_i$.

Ezek után minden $m \leqslant i \leqslant n+1$ esetén jelölje $v[i]$ azon feladat egy optimális megoldásában a programokért fizetendő részvételi díjak összegét, amikor a $p_1, p_2, \ldots, p_{\gamma(i)}$ programok közül választhatunk, és a fesztivál elejétől az s_i időpontig kell kitölteni az időt.

Legyen most $m \leqslant i \leqslant n+1$, és legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor a $p_1, p_2, \ldots, p_{\gamma(i)}$ programok közül választhatunk, és a fesztivál elejétől az s_i időpontig kell kitölteni az időt. Világos, hogy ekkor van olyan $\mathcal{O}$-beli p_j program, amelyre $f_j \geqslant s_i$. Ez a program $\mathcal{O}$ optimalitása miatt egyértelmű, és az összes többi $\mathcal{O}$-beli program, ha van ilyen, az s_j időpont előtt kezdődik. Ha $s_j = s$, akkor nyilván $\mathcal{O} = \{p_j\}$, a költség pedig w_j. Ha viszont $s_j > s$, akkor $\mathcal{O}' = \mathcal{O} \setminus \{p_j\}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor a $p_1, p_2, \ldots, p_{\gamma(j)}$ programok közül választhatunk (ezek kezdődnek az s_j időpont előtt), és a fesztivál elejétől az

s_j időpontig kell kitölteni az időt. Valóban, ha lenne az utóbbi feladatnak $\mathcal{O}'$-nél kedvezőbb megoldása, akkor ezt a megoldást a p_j programmal kiegészítve egy $\mathcal{O}$-nál kedvezőbb megoldását kapnánk annak a feladatnak, amikor a $p_1, p_2, \ldots, p_{\gamma(i)}$ programok közül választhatunk, és a fesztivál elejétől az s_i időpontig kell kitölteni az időt, ellentmondás. A költség ekkor $w_j + v[j]$. Egy probléma van: nem ismerjük j értékét. Azonban j legfeljebb $\gamma(i) \leqslant n$ különböző értéket vehet fel; vizsgáljuk meg az összes lehetőséget, és válasszuk ki a legkedvezőbbet. Így

$$v[i] = \min(\{w_j \mid 1 \leqslant j \leqslant \gamma(i),\ f_j \geqslant s_i \text{ és } s_j = s\} \cup \\ \cup \{w_j + v[j] \mid 1 \leqslant j \leqslant \gamma(i),\ f_j \geqslant s_i \text{ és } s_j > s\}).$$

A $v[i]$ értékek kiszámítását úgy tekinthetjük, mintha egy $n + 2 - m$ elemű tömböt töltenénk ki balról jobbra. A feladat optimális megoldásában a programokért fizetendő részvételi díjak összege $v[n + 1]$.

Minden $v[i]$ érték $O(n)$ lépésben számítható, így a $v[m : n + 1]$ tömb kitöltésének lépésszáma $O(n^2)$. Ehhez jön még a kezdeti rendezés lépésszáma, ami $O(n \log n)$, valamint a $\gamma(i)$ értékek meghatározásának lépésszáma, ami $O(n)$. Így az algoritmus lépésszáma $O(n \log n) + O(n) + O(n^2) = O(n^2)$.

Ha minden $m \leqslant i \leqslant n + 1$ esetén feljegyezzük, hogy $v[i]$ számításánál mely p_j program választása bizonyult a legkedvezőbbnek, akkor magát egy optimális megoldást is könnyen rekonstruálhatunk.

Átlagos várakozási idő

Feladat.
Egy étterembe a déli nyitáskor n vendég érkezik. A vendégek jól ismerik a választékot, így azonnal rendelnek. A V_i vendég által rendelt étel elkészítése t_i ideig tart. A szakács egyszerre csak egy étellel tud foglalkozni, és ha valamelyiket elkezdte, akkor azt addig nem hagyja abba, amíg teljesen el nem készült vele. Ha elkészült egy étel, azt rögtön felszolgálják a megfelelő vendégnek. Adjunk hatékony algoritmust annak eldöntésére, hogy a szakács milyen sorrendben készítse el az ételeket, ha azt szeretnénk, hogy a w_i várakozási idők átlaga a lehető legkisebb legyen!

Például ha három vendég van, akik olyan ételeket rendeltek, amelyek elkészítési ideje $t_1 = 2$, $t_2 = 4$, $t_3 = 3$, és a rendeléseket a $(3, 1, 2)$ sorrendben teljesíti az étterem, akkor a várakozási idők $w_3 = 3$, $w_1 = 5$, $w_2 = 9$, az átlagos várakozási idő pedig $17/3$.

Megoldás.
Az algoritmus egyszerű: a szakács az elkészítési idő szerint monoton növekvő sorrendben készítse el az ételeket. Az algoritmus helyessége a következő két állításból adódik.

Állítás. Az ételek elkészítésének van olyan optimális ütemezése, amely szerint a legrövidebb elkészítési idejű ételt készítik el elsőnek.

Bizonyítás. Jelölje $i_{\min}$ valamelyik legrövidebb elkészítési idő indexét, és legyen $\sigma\colon \{1,2,\dots,n\} \to \{1,2,\dots,n\}$ az ételek elkészítésének egy optimális ütemezése. Ha $\sigma(1) = i_{\min}$, akkor kész vagyunk. Ezért tegyük fel, hogy $\sigma(1) \neq i_{\min}$.

Megmutatjuk, hogy az ételek elkészítésének van olyan σ' optimális ütemezése, amelyben $\sigma'(1) = i_{\min}$. Legyen

$$\sigma'\colon \{1,2,\dots,n\} \to \{1,2,\dots,n\},\ i \mapsto \begin{cases} i_{\min} & \text{ha } i = 1, \\ \sigma(1) & \text{ha } i = i^*, \\ \sigma(i) & \text{különben,} \end{cases}$$

ahol $i_{\min} = \sigma(i^*)$. A σ ütemezés mellett a várakozási idők

$$w_{\sigma(i)} = \sum_{j=1}^{i} t_{\sigma(j)}$$

minden $1 \leqslant i \leqslant n$ esetén, az átlagos várakozási idő pedig

$$W(\sigma) = \frac{1}{n}\sum_{i=1}^{n} w_{\sigma(i)} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} t_{\sigma(j)} = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)t_{\sigma(i)}.$$

Hasonlóan, a σ' ütemezés mellett a várakozási idők

$$w_{\sigma'(i)} = \sum_{j=1}^{i} t_{\sigma'(j)}$$

minden $1 \leqslant i \leqslant n$ esetén, az átlagos várakozási idő pedig

$$W(\sigma') = \frac{1}{n}\sum_{i=1}^{n} w_{\sigma'(i)} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} t_{\sigma'(j)} = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)t_{\sigma'(i)}.$$

Vessük össze a σ és σ' ütemezések melletti átlagos várakozási időket:

$$W(\sigma') - W(\sigma) = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)\left(t_{\sigma'(i)} - t_{\sigma(i)}\right) =$$
$$\frac{1}{n}\left(n\left(t_{\sigma'(1)} - t_{\sigma(1)}\right) + (n-i^*+1)\left(t_{\sigma'(i^*)} - t_{\sigma(i^*)}\right)\right) =$$
$$\frac{1}{n}\left(n\left(t_{i_{\min}} - t_{\sigma(1)}\right) + (n-i^*+1)\left(t_{\sigma(1)} - t_{i_{\min}}\right)\right) =$$
$$\frac{1}{n}(i^*-1)\left(t_{i_{\min}} - t_{\sigma(1)}\right) \leqslant 0,$$

hiszen $i^* > 1$ és $t_{i_{\min}} \leqslant t_{\sigma(1)}$. Mivel σ egy optimális ütemezés, így $W(\sigma') < W(\sigma)$ nem lehetséges. Ennélfogva $W(\sigma') = W(\sigma)$, ami azt jelenti, hogy σ' is egy optimális ütemezés.

Állítás. Tegyük fel, hogy az ételek elkészítésének van olyan optimális ütemezése, amely szerint a V_k vendég által rendelt ételt készítik el elsőnek. Tegyük fel azt is, hogy

$$\sigma' \colon \{1, 2, \ldots, n-1\} \to \{1, \ldots, k-1, k+1, \ldots, n\}$$

a többi étel elkészítésének egy optimális ütemezése. Ekkor

$$\sigma \colon \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\},\ i \mapsto \begin{cases} k & \text{ha } i = 1, \\ \sigma'(i-1) & \text{ha } 2 \leqslant i \leqslant n \end{cases}$$

az összes étel elkészítésének egy optimális ütemezése.

Bizonyítás. Indirekt tegyük fel, hogy σ nem optimális ütemezése az összes étel elkészítésének. Legyen $\pi \colon \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ az ételek elkészítésének egy olyan optimális ütemezése, amely szerint a V_k vendég által rendelt ételt készítik el elsőnek, azaz amelyre $\pi(1) = k$. Ekkor a π ütemezéshez tartozó $W(\pi)$ átlagos várakozási idő kisebb, mint a σ ütemezéshez tartozó $W(\sigma)$ átlagos várakozási idő.

Tekintsük most a V_k vendég által rendelt ételen kívüli ételek elkészítésének a

$$\pi' \colon \{1, 2, \ldots, n-1\} \to \{1, \ldots, k-1, k+1, \ldots, n\},\ i \mapsto \pi(i+1)$$

ütemezését. Vessük össze a π' és σ' ütemezések melletti átlagos várakozási

időket:

$$W(\pi') - W(\sigma') = \frac{1}{n-1}\sum_{i=1}^{n-1} w_{\pi'(i)} - \frac{1}{n-1}\sum_{i=1}^{n-1} w_{\sigma'(i)} =$$
$$\frac{n}{n-1}\left(\frac{1}{n}\left(t_k + \sum_{i=1}^{n-1}\left(t_k + w_{\pi'(i)}\right)\right) - \frac{1}{n}\left(t_k + \sum_{i=1}^{n-1}\left(t_k + w_{\sigma'(i)}\right)\right)\right) =$$
$$\frac{n}{n-1}\left(\frac{1}{n}\sum_{i=1}^{n} w_{\pi(i)} - \frac{1}{n}\sum_{i=1}^{n} w_{\sigma(i)}\right) = \frac{n}{n-1}\left(W(\pi) - W(\sigma)\right) < 0,$$

ami ellentmondás, hiszen σ' egy optimális ütemezése a V_k vendég által rendelt ételen kívüli ételek elkészítésének. Következésképpen σ az összes étel elkészítésének egy optimális ütemezése.

Határidős feladatok ütemezése

Feladat.
Tegyük fel, hogy adott feladatok egy $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ halmaza, amelyek végrehajtása egy közös erőforrás használatával lehetséges csak. Az erőforrás egyszerre egy feladattal tud foglalkozni, és ha valamelyiket elkezdte, akkor azt addig nem hagyja abba, amíg teljesen el nem készült vele. Minden a_i feladathoz adott a t_i végrehajtási ideje, és egy d_i határidő, amelyre a feladat végrehajtásával el kellene készülni.

Adjunk hatékony algoritmust $\mathcal{A}$-beli feladatok egy olyan $\{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$ maximális elemszámú részhalmazának a meghatározására, amelyre a következők teljesülnek. Minden $1 \leqslant j \leqslant k$ esetén az a_{i_j} feladathoz hozzárendelhető egy $s_{i_j} \geqslant 0$ kezdési és egy f_{i_j} befejezési időpont úgy, hogy $f_{i_j} - s_{i_j} = t_{i_j}$ és $f_{i_j} \leqslant d_{i_j}$, továbbá bármely $1 \leqslant j < h \leqslant k$ esetén az $[s_{i_j}, f_{i_j}]$ és $[s_{i_h}, f_{i_h}]$ intervallumok a végpontjaiktól eltekintve páronként diszjunktak. Az első feltétel azt jelenti, hogy minden a_{i_j} feladat határidőre elkészül, a második pedig azt, hogy az erőforrás egyesével hajtja végre a feladatokat, megszakítás nélkül.

Megoldás.
Tegyük fel, hogy a feladatok a határidejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n.$$

A kiválasztott feladatokat egy H halmazban fogjuk tárolni. Kezdetben legyen $H = \emptyset$. Most egymás után, minden $1 \leqslant i \leqslant n$ esetén vegyük hozzá az

a_i feladatot H-hoz. Ha így

$$\sum_{a_j \in H} t_j > d_i$$

adódik, akkor töröljünk egy leghosszabb elvégzési idejű feladatot H-ból. Végül a H-beli feladatokat az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendbe osszuk be. Jelölje $\mathcal{H}$ a feladatok ilyen módon történő ütemezését. Jegyezzük meg, hogy a $\mathcal{H}$ ütemezésben minden feladat legkésőbb a határidejére elkészül.

A feladatok n száma szerinti teljes indukcióval bizonyítjuk, hogy $\mathcal{H}$ egy optimális megoldása a problémának. Ha $n = 1$, akkor ez nyilvánvaló. Legyen ezután $n > 1$, és tegyük fel, hogy $n - 1$ feladat esetén igaz az állítás.

Ha $\mathcal{H}$ az összes feladatot tartalmazza, akkor optimális megoldása a problémának világos módon. Tegyük fel ezért, hogy van olyan feladat, amely nem szerepel $\mathcal{H}$-ban. Legyen ezek közül a_k az a feladat, amelyet először törölt az algoritmus H-ból, mondjuk az a_i feladat hozzávételekor.

Megmutatjuk, hogy a problémának van olyan $\mathcal{O}$ optimális megoldása, amelyben a feladatok az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendben követik egymást, és amelyben szintén nem szerepel az a_k feladat. Az világos, hogy a problémának létezik olyan optimális megoldása, amelyben a feladatok az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendben követik egymást. Tegyük fel, hogy egy ilyen $\mathcal{O}'$ optimális megoldásban szerepel az a_k feladat. Mivel

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

ezért az $a_1, a_2, \ldots, a_i$ feladatok között szükségképpen van olyan a_l feladat, amely nem szerepel $\mathcal{O}'$-ben. Legyen $\mathcal{O}$ az az ütemezés, amelyet úgy kapunk $\mathcal{O}'$-ből, hogy az a_k feladatot lecseréljük az a_l feladatra, majd a feladatokat az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendbe osszuk be. Most az $a_1, a_2, \ldots, a_i$ közül való $\mathcal{O}$-beli feladatok legkésőbb a határidejükre mind elkészülnek, hiszen i megválasztása miatt ugyanez akkor is teljesül, ha az összes $a_1, \ldots, a_{k-1}, a_{k+1}, \ldots, a_i$ feladatot osztjuk be az első naptól kezdve, szünet nélkül. Másrészt $t_k \geqslant t_l$ miatt az $a_{i+1}, a_{i+2}, \ldots, a_n$ közül való $\mathcal{O}$-beli feladatok szintén legkésőbb a határidejükre mind elkészülnek, hiszen semelyik nincs későbbre ütemezve $\mathcal{O}$-ban, mint amikorra $\mathcal{O}'$-ben volt. Ebből következik, hogy $\mathcal{O}$ is egy optimális megoldása a problémának.

Nyilván a $\mathcal{H}$-beli feladatok száma nem haladhatja meg az $\mathcal{O}$-beli feladatok számát, hiszen az utóbbi ütemezés optimális megoldása a problémának. Másrészt a mohó algoritmust a feladatok $\mathcal{A} \setminus \{a_k\}$ halmazára futtatva szintén a $\mathcal{H}$ ütemezést kapjuk, ami az indukciós feltevés szerint optimális a feladatok

$\mathcal{A} \setminus \{a_k\}$ halmazára. Mivel az $\mathcal{O}$ ütemezésben is csak $\mathcal{A} \setminus \{a_k\}$ halmazbeli feladatok vannak, így az $\mathcal{O}$-beli feladatok száma sem haladhatja meg a $\mathcal{H}$-beli feladatok számát. Ebből következik, hogy a $\mathcal{H}$-beli feladatok száma megegyezik az $\mathcal{O}$-beli feladatok számával, következésképpen $\mathcal{H}$ is optimális megoldása az eredeti problémának.

A H-beli feladatokat az elvégzési idők szerint rendezett maximum kupacban tárolva a soron következő feladat beszúrásának, valamint egy maximális elvégzési idejű feladat esetleges törlésének lépésszáma $O(\log n)$. A törlés szükségessége $O(1)$ lépésben eldönthető, ha egy segédváltozóban nyilvántartjuk a kupacban levő feladatok elvégzési idejének összegét. Ezt minden $1 \leqslant i \leqslant n$ esetén végrehajtva az összlépésszám $O(n \log n)$. Ehhez jön még a kezdeti rendezés $O(n \log n)$ költsége, így az algoritmus teljes költsége $O(n \log n)$.

Ütemezés minimális késéssel

Feladat.
Tegyük fel, hogy adott feladatok egy $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ halmaza, amelyek végrehajtása egy közös erőforrás használatával lehetséges csak. Az erőforrás egyszerre egy feladattal tud foglalkozni, és ha valamelyiket elkezdte, akkor azt addig nem hagyja abba, amíg teljesen el nem készült vele. Minden a_i feladathoz adott a t_i végrehajtási ideje, és egy d_i határidő, amelyre a feladat végrehajtásával el kellene készülni.

Tegyük fel, hogy minden feladatot végre kell hajtanunk, de az megengedett, hogy bizonyosakat a határidejük után fejezzünk csak be. Minden a_i feladathoz rendeljük hozzá azt a t_i hosszú intervallumot, amikor az erőforrást a feladat végrehajtására akarjuk használni. Jelölje ezt az intervallumot $[s_i, f_i]$, ahol $s_i \geqslant 0$ és $f_i = s_i + t_i$ természetes módon. Mivel az erőforrás egyszerre egy feladattal tud csak foglalkozni, ezeknek az intervallumoknak a végpontjaiktól eltekintve páronként diszjunktaknak kell lenni.

Azt fogjuk mondani, hogy egy a_i feladatot késve hajtunk végre, ha $f_i > d_i$. Ebben az esetben jelölje $l_i = d_i - f_i$ a késés nagyságát. Ha egy a_i feladat a határidején belül végrehajtásra kerül, akkor legyen $l_i = 0$ definíció szerint.

Adjunk hatékony algoritmust a feladatok egy olyan ütemezésének megkeresésére, amelynél az $L = \max\{l_i \mid 1 \leqslant i \leqslant n\}$ maximális késés a lehető legkevesebb!

Megoldás.
Tegyük fel, hogy a feladatok a határidejük szerint monoton növekvően ren-

dezettek (ha ez nem teljesül, akkor először rendezzük őket):

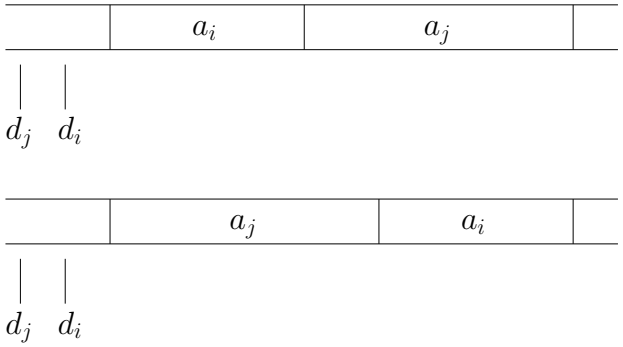
$$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n.$$

Ütemezzük a feladatokat ebben a sorrendben: legyen a_1 kezdési ideje $s_1 = 0$, befejezési ideje $f_1 = s_1 + t_1$, legyen a_2 kezdési ideje $s_2 = f_1$, befejezési ideje $f_2 = s_2 + t_2$, és így tovább. A költség nyilván $O(n \log n)$.

Belátjuk, hogy ez az ütemezés optimális. Jelölje $\mathcal{P}$ az algoritmus által előállított ütemezést. Jegyezzük meg, hogy ebben nincs üresjárat, amikor az erőforrás nem foglalkozik egyik feladattal sem, miközben vannak még elvégzésre várók.

Nyilvánvaló, hogy az optimális ütemezések között is kell lenni üresjárat nélkülinek. Jelöljön $\mathcal{O}$ egy ilyen optimális ütemezést. Ha $\mathcal{O} = \mathcal{P}$, akkor készen vagyunk. Tegyük fel ezért, hogy $\mathcal{O} \neq \mathcal{P}$. Ekkor az $\mathcal{O}$ ütemezésben szükségképpen vannak olyan a_i és a_j közvetlenül egymás után következő feladatok, amelyekre $i > j$.

Tekintsük most azt az $\mathcal{O}'$ ütemezést, amelyet $\mathcal{O}$-ból az a_i és a_j feladatok felcserélésével kapunk.



Vizsgáljuk meg, hogy a cserével növekedhetett-e a maximális késés! Csak az a_i és a_j feladatok ütemezése változott meg, így a többi feladatnál a késés a csere után ugyanannyi, mint a csere előtt volt. A cserével az a_j feladat előbbre került, így a késése nyilván nem növekedett.

Nézzük ezután az a_i feladatot. Jelölje a_j befejezési idejét az $\mathcal{O}$ ütemezésnél f_j, késését pedig l_j. Ekkor a_i befejezési ideje az $\mathcal{O}'$ ütemezésnél $f'_i = f_j$. Jelölje a_i késését az $\mathcal{O}'$ ütemezésnél l'_i.

Ha $l'_i = 0$, vagyis a_i nem késik az $\mathcal{O}'$ ütemezésnél, akkor nyilván nem késhetett az $\mathcal{O}$ ütemezésnél sem, hiszen ott korábbra volt ütemezve. Ha pedig $l'_i > 0$, akkor $d_i \geqslant d_j$ figyelembe vételével

$$l'_i = f'_i - d_i = f_j - d_i \leqslant f_j - d_j = l_j$$

adódik, vagyis a_i késése az $\mathcal{O}'$ ütemezésnél nem lehet nagyobb, mint a_j késése az $\mathcal{O}$ ütemezésnél. Mindebből következik, hogy a maximális késés az $\mathcal{O}'$ ütemezésnél nem lehet nagyobb, mint az $\mathcal{O}$ ütemezésnél.

Algoritmusunk helyessége innen már egyszerűen adódik. Az optimális $\mathcal{O}$ ütemezésből legfeljebb $\binom{n}{2}$ szomszédos feladat cseréjével eljuthatunk a $\mathcal{P}$ ütemezéshez (vö. buborék rendezés). Mivel a maximális késés egyik lépésben sem növekszik, így $\mathcal{P}$ szükségképpen szintén egy optimális ütemezés.

Egynapos munkák ütemezése

Feladat.
Egy megbízható vállalkozót sokan keresnek meg különböző megrendelésekkel. Tegyük fel, hogy a vállalkozó minden szóban forgó munkát egy nap alatt el tud végezni. Tegyük fel azt is, hogy a vállalkozó egy nap csak egy munkán dolgozik. Minden munkához tartozik egy határidő és egy munkadíj. Egy munkáért a munkadíjat akkor kapja meg a vállalkozó, ha azt legkésőbb a határidőre elvégzi.

Adjunk hatékony algoritmust annak eldöntésére, hogy a vállalkozó mely munkákat vállalja el, és azokat mikor végezze el, ha a bevételét maximalizálni akarja!

Megoldás.
Először is vegyük észre, hogy a feladatnak mindig van olyan optimális megoldása, amelyben az összes elvállalt munkát legkésőbb az n-edik nap elvégezzük, hiszen az üres napokat meg tudjuk szüntetni a feltételek megsértése nélkül a munkák alkalmas előbbre ütemezésével. Ebből következik, hogy ha minden olyan munkának a határidejét az n-edik napra hozzuk előre, amelyiknek a határideje az n-edik nap után van, akkor az így módosított feladat egy optimális megoldásában az összbevétel nyilván ugyanannyi lesz, mint az eredeti feladat esetén.

Rendezzük a munkákat a munkadíj szerint csökkenő sorrendbe. Jelölje $M_1, M_2, \ldots, M_n$ a munkákat az így kialakult sorrendben. Minden $1 \leqslant i \leqslant n$ esetén az M_i munka határidejét jelölje d_i, az érte kapható munkadíjat pedig p_i. A fentiekkel összhangban feltehető, hogy $d_i \leqslant n$ minden $1 \leqslant i \leqslant n$ esetén.

Most egymás után minden $1 \leqslant i \leqslant n$ esetén ütemezzük be az M_i munkát a d_i-edik, illetve az azt megelőző napok közül a legkésőbbi olyanra, amely még szabad. Ha az összes ilyen nap foglalt már, akkor a munkát ne vállaljuk el. Jelölje $\mathcal{H}$ a munkák ilyen módon történő ütemezését.

Először belátjuk, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak. Tekintsük a munkák egy optimális $\mathcal{O}$ ütemezését. Ha $\mathcal{O} = \mathcal{H}$, akkor készen vagyunk. Tegyük fel ezért, hogy $\mathcal{O} \neq \mathcal{H}$. Legyen k a legkisebb olyan index, amelyre M_k ütemezése különbözik $\mathcal{O}$-ban és $\mathcal{H}$-ban. Ekkor minden $j < k$ esetén az M_j munka

- vagy nem szerepel sem $\mathcal{O}$-ban sem $\mathcal{H}$-ban,
- vagy szerepel $\mathcal{O}$-ban és $\mathcal{H}$-ban is, mindkétszer ugyanarra a napra ütemezve.

Négy esetet különböztethetünk meg.

(1) Az M_k munka szerepel $\mathcal{O}$-ban, de nem szerepel $\mathcal{H}$-ban. Mivel M_k szerepel $\mathcal{O}$-ban, ezért a $\mathcal{H}$-t előállító algoritmusunk k-adik lépésében a d_k-adik, illetve az azt megelőző napok között szükségképpen van még szabad nap. Ám ez ellentmond annak, hogy $\mathcal{H}$-ból kihagytuk M_k-t.

(2) Az M_k munka szerepel $\mathcal{H}$-ban, de nem szerepel $\mathcal{O}$-ban. Ha arra a napra, amelyre $\mathcal{H}$ ütemezi M_k-t, $\mathcal{O}$ nem ütemez egyetlen munkát sem, akkor M_k-t vegyük hozzá $\mathcal{O}$-hoz erre a napra ütemezve. Ezzel az összbevétel nyilván nem csökken. Ha viszont arra a napra, amelyre $\mathcal{H}$ ütemezi M_k-t, $\mathcal{O}$ is ütemez egy M_j munkát, akkor $\mathcal{O}$-ban cseréljük le M_j-t M_k-ra. Mivel $j > k$, ezért $p_j \leqslant p_k$, következésképpen az összbevétel most sem csökken.

(3) Az M_k munka szerepel $\mathcal{O}$-ban és $\mathcal{H}$-ban is, de az előbbiben korábbi napra van ütemezve, mint az utóbbiban. Ha arra a napra, amelyre $\mathcal{H}$ ütemezi M_k-t, $\mathcal{O}$ nem ütemez egyetlen munkát sem, akkor M_k-t ütemezzük át $\mathcal{O}$-ban erre a napra. Ezzel az összbevétel nem változik. Ha viszont arra a napra, amelyre $\mathcal{H}$ ütemezi M_k-t, $\mathcal{O}$ is ütemez egy M_j munkát, akkor $\mathcal{O}$-ban cseréljük fel M_j-t és M_k-t. Mivel M_j így előbbre kerül, a csere megengedett. Az összbevétel nyilván most sem változik.

(4) Az M_k munka szerepel $\mathcal{O}$-ban és $\mathcal{H}$-ban is, de az utóbbiban korábbi napra van ütemezve, mint az előbbiben. Ez ellentmond annak, hogy a $\mathcal{H}$-t előállító algoritmusunk a k-adik lépésben az M_k munkát a d_k, illetve az azt megelőző napok között a legkésőbbi, még szabad napra ütemezte.

Ezzel beláttuk, hogy az optimális $\mathcal{O}$ ütemezés áttranszformálható egy olyan optimális $\mathcal{O}'$ ütemezéssé, hogy $\mathcal{O}'$-ben és $\mathcal{H}$-ban már M_k ütemezése is megegyezik. Az eljárást folytatva $\mathcal{O}$ lépésről-lépésre áttranszformálható $\mathcal{H}$-ba az összbevétel csökkenése nélkül. Következésképpen $\mathcal{H}$ is egy optimális ütemezés.

Az algoritmus költsége nagyban függ attól, hogy a k-adik lépésben hogyan találjuk meg d_k-adik, illetve az azt megelőző napok közül a legkésőbbi olyat, amely még szabad. Kézenfekvő ötlet, hogy d_k-adik naptól egyesével

haladjunk visszafelé. Ennek költsége nyilván $O(n)$. Ezt minden $1 \leqslant k \leqslant n$ esetén végrehajtva az összköltség $O(n^2)$.

Ehhez jön még a kezdeti rendezés $O(n \log n)$ költsége, így az algoritmus teljes költsége $O(n^2)$.

Az UNIÓ-HOLVAN adatszerkezet használatával lényegesen hatékonyabban is végezhető a keresés. Tartsuk nyilván az egymás utáni foglalt napok maximális halmazait (amelyeket a közvetlenül előttük lévő szabad nappal azonosítunk), valamint a szabad napokat, mint egyelemű halmazokat. Az algoritmus k-adik lépésében ellenőrizzük, hogy a d_k-adik nap szabad-e, illetve ha nem, akkor a foglalt napok mely blokkjához tartozik. Az első esetben M_k-t a d_k-adik napra ütemezzük, a másodikban a blokkot közvetlenül megelőző szabad napra (illetve eldobjuk, ha nincs a blokk előtt szabad nap). Ha az a nap, amelyre M_k-t ütemeztük, foglalt napok blokkjaival szomszédos, akkor képezzük a blokkok unióját. A költség most csupán $O(\log n)$.

Az algoritmus ezen implementációjának teljes költsége így $O(n \log n)$.

Hosszabb munkák ütemezése

Feladat.
Egy vállalkozónak n megrendelése van különböző munkákra. Minden munkához tartozik egy elvégzési idő, egy határidő és egy munkadíj. A vállalkozó egyszerre csak egy munkán tud dolgozni, és ha egy munkát elkezdett, azt annak befejezéséig nem szakítja meg. Egy munkáért a munkadíjat akkor kapja meg a vállalkozó, ha azt legkésőbb a határidőre elvégzi. Tegyük fel, hogy a munkák elvégzési ideje és a határidők pozitív egész számok (mondjuk napok). Mely munkákat vállalja el a vállalkozó, és azokat mikor végezze el, ha a bevételét maximalizálni akarja?

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Jelölje a munkákat M_1, $M_2, \ldots, M_n$. Minden $1 \leqslant i \leqslant n$ esetén az M_i munka végrehajtási idejét jelölje t_i, a határidejét d_i, az érte kapható munkadíjat pedig p_i. Tegyük fel, hogy a munkák a határidejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n.$$

Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant d_n$ esetén jelölje $c[i, j]$ azon feladat egy optimális megoldásában az összbevételt, amikor az $M_i, M_{i+1}, \ldots, M_n$ munkák közül választhatunk, és az első munkát legkorábban a j-edik nap kezdhetjük

el. A formulák kompakt felírhatóságának érdekében legyen $c[i,j] = 0$, ha $i = n+1$ vagy $j = d_n + 1$.

Legyen most $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant d_n$ továbbá legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor az $M_i, M_{i+1}, \ldots, M_n$ munkák közül választhatunk, és az első munkát legkorábban a j-edik nap kezdhetjük el. Ha $j + t_i - 1 > d_i$, akkor M_i nyilván nem szerepelhet $\mathcal{O}$-ban, így $\mathcal{O}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor az $M_{i+1}, \ldots, M_n$ munkák közül választhatunk, és az első munkát legkorábban a j-edik nap kezdhetjük el. Ebben az esetben a bevétel $c[i+1,j]$. Tegyük fel ezután, hogy $j + t_i - 1 \leqslant d_i$.

- Ha M_i szerepel az $\mathcal{O}$-beli munkák között, akkor az az $\mathcal{O}'$ ütemezés, amelyet úgy kapunk $\mathcal{O}$-ból, hogy elhagyjuk az M_i munkát, az M_i elé ütemezett munkákat pedig mind t_i nappal későbbre ütemezzük (a munkáknak a határidők szerinti rendezettsége miatt ezek továbbra is határidőn belül vannak), egy optimális megoldása annak a feladatnak, amikor az $M_{i+1}, \ldots, M_n$ munkák közül választhatunk, és az első munkát legkorábban a $(j+t_i)$-edik nap kezdhetjük el. Valóban, ha lenne $\mathcal{O}'$-nél nagyobb bevételt hozó ütemezés arra a feladatra, amikor az $M_{i+1}, \ldots, M_n$ munkák közül választhatunk, és az első munkát legkorábban a $(j+t_i)$-edik nap kezdhetjük el, akkor ehhez hozzávéve az M_i munkát j-edik napi kezdéssel, egy $\mathcal{O}$-nál nagyobb bevételt hozó ütemezést kapnánk arra a feladatra, amikor az $M_i, M_{i+1}, \ldots, M_n$ munkák közül választhatunk, és az első munkát legkorábban a j-edik nap kezdhetjük el, ellentmondás. A bevétel ekkor $p_i + c[i+1, j+t_i]$.

- Ha M_i nem szerepel az $\mathcal{O}$-beli munkák között, akkor $\mathcal{O}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor az $M_{i+1}, \ldots, M_n$ munkák közül választhatunk, és az első munkát legkorábban a j-edik nap kezdhetjük el. A bevétel ekkor $c[i+1,j]$.

Mivel nem tudjuk, hogy M_i szerepel-e az $\mathcal{O}$-beli munkák között, ezért mindkét lehetőséget megvizsgáljuk, és a kedvezőbbet választjuk. Így

$$c[i,j] = \begin{cases} c[i+1,j] & \text{ha } j + t_i - 1 > d_i, \\ \max(p_i + c[i+1, j+t_i], c[i+1,j]) & \text{ha } j + t_i - 1 \leqslant d_i. \end{cases}$$

Annak érdekében, hogy munkák egy optimális megoldást adó részhalmazát is meg tudjuk adni, még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant d_n$ esetén legyen $b[i,j]$ a `TRUE` logikai érték, ha $c[i,j]$ meghatározásánál azt találtuk, hogy az M_i munka beválasztásával nagyobb bevételt hozó megoldást kapunk, mint anélkül, egyébként legyen $b[i,j]$ a `FALSE` logikai érték.

```
Ütemezés(n,t,d,p)
for j=1 to d[n]+1 do
  c[n+1,j]=0
for i=n downto 1 do
  c[i,d[n]+1]=0
  b[i,d[n]+1]=FALSE
  for j=d[n] downto 1 do
    if j+t[i]-1>d[i]
      then
        c[i,j]=c[i+1,j]
        b[i,j]=FALSE
      else
        if p[i]+c[i+1,j+t[i]]>c[i+1,j]
          then
            c[i,j]=p[i]+c[i+1,j+t[i]]
            b[i,j]=TRUE
          else
            c[i,j]=c[i+1,j]
            b[i,j]=FALSE
return c,b
```

A $c[i,j]$ értékek kiszámítását úgy tekinthetjük, mintha egy $(n+1)\times(d_n+1)$ méretű táblázatot töltenénk ki alulról felfelé, soronként jobbról balra. A feladat optimális megoldásában az összbevétel $c[1,1]$.

Minden $c[i,j]$ érték konstans költséggel számítható, így az algoritmus teljes költsége $O(nd_n)$. Jegyezzük meg, hogy az algoritmus nem polinomiális!

A b tömb felhasználásával munkák egy optimális megoldást adó részhalmazát könnyen meghatározhatjuk.

```
MunkákNyomtatása
j=1
for i=1 to n do
  if b[i,j]=TRUE then
    print i '. munka'
    j=j+t[i]
```

Egy optimális ütemezéshez jutunk ezek után, ha az így kapott sorozatban szereplő munkákat az első naptól kezdődően, egymás után, szünet nélkül osztjuk be.

Hátizsák feladat

Feladat.
Adottak a $t_1, t_2, \ldots, t_n$ tárgyak. A t_i tárgy súlya w_i, értéke v_i, ahol w_i és v_i pozitív egész számok. Kiválasztandó a tárgyaknak egy olyan részhalmaza, amelyben a tárgyak értékének összege a lehető legnagyobb, súlyuk összege viszont nem halad meg egy adott W pozitív egész számot (a hátizsák kapacitását)!

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant W$ esetén jelölje $c[i, j]$ azon feladat egy optimális megoldásában a tárgyak értékeinek az összegét, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk. A formulák kompakt felírhatóságának érdekében legyen $c[i, j] = 0$, ha $i = 0$ vagy $j = 0$.

Legyen most $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant W$, továbbá legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk. Ha $w_i > j$, akkor t_i nyilván nem szerepelhet $\mathcal{O}$-ban, így $\mathcal{O}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Ebben az esetben az összérték $c[i-1, j]$. Tegyük fel ezután, hogy $w_i \leqslant j$.

- Ha t_i szerepel az $\mathcal{O}$-beli tárgyak között, akkor $\mathcal{O}' = \mathcal{O} \setminus \{t_i\}$ egy optimális megoldása annak a feladatnak, amikor a hátizsák kapacitása $j - w_i$ és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Valóban, ha lenne $\mathcal{O}'$-nél nagyobb összértékű megoldása annak a feladatnak, amikor a hátizsák kapacitása $j - w_i$ és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk, akkor ehhez hozzávéve a t_i tárgyat egy $\mathcal{O}$-nál nagyobb összértékű megoldását kapnánk annak a feladatnak, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk, ellentmondás. Az összérték ekkor $v_i + c[i-1, j-w_i]$.

- Ha t_i nem szerepel az $\mathcal{O}$-beli tárgyak között, akkor $\mathcal{O}$ nyilván egy optimális megoldása annak a feladatnak, mikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Az összérték ekkor $c[i-1, j]$.

Mivel nem tudjuk, hogy t_i szerepel-e az $\mathcal{O}$-beli tárgyak között, ezért mindkét lehetőséget megvizsgáljuk, és a kedvezőbbet választjuk. Így

$$c[i,j] = \begin{cases} c[i-1,j] & \text{ha } w_i > j, \\ \max(v_i + c[i-1, j-w_i], c[i-1,j]) & \text{ha } j \geqslant w_i. \end{cases}$$

Annak érdekében, hogy tárgyak egy optimális megoldást adó részhalmazát is meg tudjuk adni, még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant W$ esetén legyen $d[i,j]$ a `TRUE` logikai érték, ha $c[i,j]$ meghatározásánál azt találtuk, hogy a t_i tárgy beválasztásával nagyobb értékű megoldást kapunk, mint anélkül, egyébként legyen $d[i,j]$ a `FALSE` logikai érték.

```
Hátizsák(n,v,w,W)
for j=0 to W do
  c[0,j]=0
for i=1 to n do
  c[i,0]=0
  d[i,0]=FALSE
  for j=1 to W do
    if w[i]>j
      then
        c[i,j]=c[i-1,j]
        d[i,j]=FALSE
      else
        if v[i]+c[i-1,j-w[i]]>c[i-1,j]
          then
            c[i,j]=v[i]+c[i-1,j-w[i]]
            d[i,j]=TRUE
          else
            c[i,j]=c[i-1,j]
            d[i,j]=FALSE
return c,d
```

A $c[i,j]$ értékek kiszámítását úgy tekinthetjük, mintha egy $(n+1)\times(W+1)$ méretű táblázatot töltenénk ki felülről lefelé, soronként balról jobbra. A feladat optimális megoldásának értéke $c[n,W]$.

Minden $c[i,j]$ érték konstans költséggel számítható, így az algoritmus teljes költsége $O(nW)$. Jegyezzük meg, hogy az algoritmus nem polinomiális!

A d tömb felhasználásával tárgyak egy optimális megoldást adó részhalmazát a következőképpen határozhatjuk meg:

```
TárgyakNyomtatása
j=W
for i=n downto 1 do
  if d[i,j]=TRUE then
    print i '. tárgy'
    j=j-w[i]
```

Házidolgozat

Feladat.
Egy házidolgozat n kérdésből áll, jelölje ezeket $t_1, t_2, \ldots, t_n$. Minden $1 \leqslant i \leqslant n$ esetén tudjuk, hogy a t_i kérdés megválaszolása m_i perc, és a helyes válaszért v_i pont jár (m_i és v_i pozitív egész számok). A számunkra kívánt érdemjegy eléréséhez legalább V pontot kell összegyűjteni (V szintén pozitív egész szám). Kiválasztandó a kérdéseknek egy olyan részhalmaza, amelyek megválaszolásával a lehető legrövidebb idő alatt érhető el legalább V pont!

Megoldás.
A feladatot dinamikus programozással oldjuk meg. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant V$ esetén jelölje $c[i, j]$ a legrövidebb időt, amely alatt csak a $t_1, t_2, \ldots, t_i$ kérdések közül választva legalább j pont elérhető. A formulák kompakt felírhatóságának érdekében legyen $c[0, j] = \infty$ minden $1 \leqslant j \leqslant V$ esetén.

Legyen most $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant V$, továbbá legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor legalább j pontot kell összegyűjtenünk, és csak a $t_1, t_2, \ldots, t_i$ kérdések közül választhatunk.

- Ha t_i szerepel az $\mathcal{O}$-beli kérdések között, és $v_i \geqslant j$, akkor $\mathcal{O} = \{t_i\}$ nyilvánvaló módon. A felhasznált idő ekkor m_i. Másrészt, ha t_i szerepel az $\mathcal{O}$-beli kérdések között, és $v_i < j$, akkor $\mathcal{O}' = \mathcal{O} \setminus \{t_i\}$ egy optimális megoldása annak a feladatnak, amikor legalább $j - v_i$ pontot kell összegyűjtenünk, és csak a $t_1, t_2, \ldots, t_{i-1}$ kérdések közül választhatunk. Valóban, ha lenne $\mathcal{O}'$-nél rövidebb összidejű megoldása annak a feladatnak, amikor legalább $j - v_i$ pontot kell összegyűjtenünk, és csak a $t_1, t_2, \ldots, t_{i-1}$ kérdések közül választhatunk, akkor ehhez hozzávéve a t_i kérdést egy $\mathcal{O}$-nál rövidebb összidejű megoldását kapnánk annak a feladatnak, mikor legalább j pontot kell összegyűjtenünk, és csak a $t_1, t_2, \ldots, t_i$ kérdések közül választhatunk, ellentmondás. A felhasznált idő ekkor $m_i + c[i-1, j-v_i]$.

- Ha t_i nem szerepel az $\mathcal{O}$-beli kérdések között, akkor $\mathcal{O}$ nyilván egy optimális megoldása annak a feladatnak, mikor legalább j pontot kell összegyűjtenünk, és csak a $t_1, t_2, \ldots, t_{i-1}$ kérdések közül választhatunk. A felhasznált idő ekkor $c[i-1, j]$.

Mivel nem tudjuk, hogy t_i szerepel-e az $\mathcal{O}$-beli kérdések között, ezért mindkét lehetőséget megvizsgáljuk, és a kedvezőbbet választjuk. Így

$$c[i,j] = \begin{cases} \min(m_i, c[i-1,j]) & \text{ha } v_i \geqslant j, \\ \min(m_i + c[i-1, j-v_i], c[i-1,j]) & \text{ha } v_i < j. \end{cases}$$

Annak érdekében, hogy a kérdések egy optimális megoldást adó részhalmazát is meg tudjuk adni, még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant V$ esetén legyen $d[i,j]$ a `TRUE` logikai érték, ha $c[i,j]$ meghatározásánál azt találtuk, hogy a t_i kérdés beválasztásával rövidebb összidejű megoldást kapunk, mint anélkül, egyébként legyen $d[i,j]$ a `FALSE` logikai érték.

```
Házidolgozat(n,m,v,V)
for j=1 to V do
  c[0,j]=INFINITY
for i=1 to n do
  for j=1 to W do
    if v[i]>=j
      then
        if m[i]<c[i-1,j]
          then
            c[i,j]=m[i]
            d[i,j]=TRUE
          else
            c[i,j]=c[i-1,j]
            d[i,j]=FALSE
      else
        if m[i]+c[i-1,j-v[i]]<c[i-1,j]
          then
            c[i,j]=m[i]+c[i-1,j-v[i]]
            d[i,j]=TRUE
          else
            c[i,j]=c[i-1,j]
            d[i,j]=FALSE
return c,d
```

A $c[i,j]$ értékek kiszámítását úgy tekinthetjük, mintha egy $(n+1) \times V$ méretű táblázatot töltenénk ki felülről lefelé, soronként balról jobbra. A feladat optimális megoldásának értéke $c[n,V]$.

Minden $c[i,j]$ érték konstans költséggel számítható, így az algoritmus teljes költsége $O(nV)$. Jegyezzük meg, hogy az algoritmus nem polinomiális!

A d tömb felhasználásával a kérdések egy optimális megoldást adó részhalmazát a következőképpen határozhatjuk meg:

```
KérdésekNyomtatása
j=V
i=n
while j>0 AND i>0 do
  if d[i,j]=TRUE then
    print i '. feladat'
    j=j-v[i]
  i=i-1
```

Visszajáró pénz

Feladat.
A visszajáró pénz problémánál adott pénzmennyiséget kell kifizetni a lehető legkevesebb érme felhasználásával (adott címletű érméből tetszőlegesen sok áll a rendelkezésünkre).

(A) Tervezzünk mohó algoritmust, amely megoldja a visszajáró pénz problémát abban az esetben, amikor a felhasználható érmék $1, 5, 10, 25$ címletűek!

(B) Adjunk meg olyan címlethalmazt, amelyre a mohó algoritmus nem feltétlenül ad optimális megoldást! A címletek között szerepeljen az 1 érték, hogy minden pénzmennyiség kifizethető legyen!

(C) Tervezzünk dinamikus programozás algoritmust, amely minden olyan esetben megoldja a visszajáró pénz problémát, amikor a címletek között szerepel az 1 érték!

(D) Tervezzünk hatékony algoritmust annak eldöntésére, hogy adott címlethalmaz esetén használható-e a mohó algoritmus a visszajáró pénz probléma megoldására!

Megoldás.
Egy észrevétellel kezdjük, amelyet a mohó és a dinamikus programozás algoritmusnál is felhasználunk majd. Tekintsük egy legkevesebb érmét felhasználó megoldását annak a feladatnak, amikor a kifizetendő pénzmennyiség n. Legyen a megoldásban felhasznált érmék száma s. Ha most elhagyjuk ezek közül valamelyik érmét, legyen ez mondjuk egy c címletű, akkor a megmaradt $s-1$ érme egy legkevesebb érmét felhasználó megoldását adja annak a feladatnak, amikor a kifizetendő pénzmennyiség $n-c$. Valóban, ha ez utóbbi pénzmennyiség kifizethető lenne $(s-1)$-nél kevesebb érmével is, akkor ezeket az érméket az elhagyott c címletű érmével kiegészítve az eredeti feladat egy s-nél kevesebb érmét felhasználó megoldásához jutnánk, ami ellentmondás.

(A) A visszajáró pénz problémáját megoldja a következő algoritmus. Legyen a kifizetendő pénzmennyiség n.

- A 25-ös címletű érmék száma legyen $[n/25]$. Ezek után a kifizetendő pénzmennyiség $n' = n - 25[n/25]$.
- A 10-es címletű érmék száma legyen $[n'/10]$. Ezek után a kifizetendő pénzmennyiség $n'' = n' - 10[n'/10]$.
- Az 5-ös címletű érmék száma legyen $[n''/5]$. Ezek után a kifizetendő pénzmennyiség $n''' = n'' - 5[n''/5]$.
- Az 1-es címletű érmék száma legyen n'''.

Az algoritmust kicsit másképpen is megfogalmazhatjuk. Legyen a kifizetendő pénzmennyiség n. Ha $n = 0$, akkor az optimális megoldás 0 érméből áll. Ha $n > 0$, akkor keressük meg azt a legnagyobb c címletet, amely kisebb vagy egyenlő, mint n. Adjunk ki egy c címletű érmét, majd oldjuk meg rekurzívan azt a problémát, amikor a kifizetendő pénzmennyiség $n - c$.

Megmutatjuk, hogy ha a kifizetendő pénzmennyiség n, és a címletek közül c a legnagyobb olyan, amely kisebb vagy egyenlő, mint n, akkor az adott pénzmennyiség kifizethető a lehető legkevesebb számú érmével úgy is, hogy az érmék között van c címletű. Innen a megoldás elején tett megjegyzés felhasználásával az algoritmus helyessége következik.

Tekintsük a feladat egy optimális megoldását. Ha ebben a megoldásban szerepel c címletű érme, akkor készen vagyunk. Tegyük fel ezért, hogy nem szerepel. Most négy esetet különböztethetünk meg.

(1) Ha $1 \leqslant n < 5$, akkor $c = 1$. Feltételünk szerint a vizsgált optimális megoldás nem tartalmaz 1-es címletű érmét, ami nyilván lehetetlen.

(2) Ha $5 \leqslant n < 10$, akkor $c = 5$. Feltételünk szerint a vizsgált optimális megoldás nem tartalmaz 5-ös címletű érmét, tehát csak 1-es címletű érmékből áll. Ezek közül ötöt egy 5-ös címletű érmére cserélve egy kevesebb érméből álló megoldáshoz jutunk, ami ellentmondás.

(3) Ha $10 \leqslant n < 25$, akkor $c = 10$. Feltételünk szerint a vizsgált optimális megoldás nem tartalmaz 10-es címletű érmét. Ekkor az 1-es és 5-ös címletű érmék között vannak olyanok, amelyek összege 10, ezeket egy 10-es címletű érmére cserélve egy kevesebb érméből álló megoldáshoz jutunk, ami ellentmondás.

(4) Ha $n \geqslant 25$, akkor $c = 25$. Feltételünk szerint a vizsgált optimális megoldás nem tartalmaz 25-ös címletű érmét. Ha tartalmaz három 10-es címletű érmét, akkor ezeket egy 25-ös és egy 5-ös címletű érmére cserélve egy kevesebb érméből álló megoldáshoz jutunk, ami ellentmondás. Ha legfeljebb két 10-es címletű érmét tartalmaz, akkor az 1-es, 5-ös és 10-es címletű érmék között vannak olyanok, amelyek összege 25, ezeket egy 25-ös címletű érmére

cserélve ismét egy kevesebb érméből álló megoldáshoz jutunk, ami szintén ellentmondás.

Az algoritmus második változatának, amikor egyszerre mindig csak egy érmét választunk, költsége $O(k)$, ahol k a kifizetéshez szükséges érmék minimális száma. Mivel $k \leqslant n$, ezért itt a költség $O(n)$.

Az algoritmus első változatának költsége $O(1)$.

(B) Legyenek a címletek $1, 10, 25$, és legyen a kifizetendő pénzmenynyiség 30. Most a mohó módszerrel ezt egy 25-ös címletű és öt 1-es címletű érmével fizetjük ki. Ennél jobb megoldás a három 10-es címletű érmével történő kifizetés.

(C) Legyenek a felhasználható címletek $c_1 > c_2 > \cdots > c_m = 1$, a kifizetendő pénzmennyiség pedig legyen n.

Minden $0 \leqslant j \leqslant n$ esetén jelölje $v[j]$ a kifizetésben szereplő érmék minimális számát, amikor a kifizetendő pénzmennyiség j. Ha $j = 0$, akkor $v[j] = 0$ nyilvánvaló módon. Ha $1 \leqslant j \leqslant n$ és egy legkevesebb érméből álló kifizetésben szerepel c_i címletű érme, akkor $v[j] = 1 + v[j - c_i]$ a megoldás elején tett megjegyzéssel összhangban. Ez a rekurzív egyenlet feltételezi, hogy ismerjük i értékét, miközben ez nem igaz. Azonban i legfeljebb m különböző értéket vehet fel; vizsgáljuk meg az összes lehetőséget, és válasszuk ki a legjobbat. Így most

$$v[j] = 1 + \min\{v[j - c_i] \mid 1 \leqslant i \leqslant m,\ c_i \leqslant j\}.$$

Egy legkevesebb érméből álló kifizetés megtalálása érdekében még egy jelölést vezetünk be. Minden $1 \leqslant j \leqslant n$ esetén legyen $d[j]$ az a c_i címlet, amelyre $v[j] = 1 + v[j - c_i]$.

```
VisszajáróPénz(n,c)
v[0]=0
for j=1 to n do
  v[j]=INFINITY
  for i=1 to m do
    if j>=c[i] AND 1+v[j-c[i]]<v[j] then
      v[j]=1+v[j-c[i]]
      d[j]=c[i]
return v,d
```

Egy minimális számú érmével történő kifizetést a következő rekurzív eljárással határozhatunk meg. Az eljárást az (n, d) paraméterekkel kell meghívni.

```
Kifizet(j,d)
if j>0 then
  print d[j]
  Kifizet(j-d[j],d)
```

A `VisszajáróPénz` algoritmus költsége $O(nm)$, a `Kifizet` eljárásé pedig $O(n)$. Jegyezzük meg, hogy a `VisszajáróPénz` algoritmus nem polinomiális!
(D) Először is vezessük be a $\boldsymbol{c} = (c_1, c_2, \ldots, c_m)$ jelölést. Adott $n \in \mathbb{N}$ esetén egy $\boldsymbol{r} = (r_1, r_2, \ldots, r_m) \in \mathbb{N}^m$ sorozatot az n egy $\boldsymbol{c}$-beli reprezentációjának fogunk nevezni, ha

$$\boldsymbol{rc} = r_1c_1 + r_2c_2 + \cdots + r_mc_m = n.$$

Vezessük még be az $|\boldsymbol{r}| = r_1 + r_2 + \cdots + r_m$ jelölést, erre a mennyiségre az $\boldsymbol{r}$ reprezentáció méreteként fogunk hivatkozni. A visszajáró pénz feladat ebben a megfogalmazásban adott $n \in \mathbb{N}$ egy minimális méretű $\boldsymbol{c}$-beli reprezentációjának a megtalálása.

Definiáljuk az $\mathbb{N}^m$-beli sorozatok halmazán a $\preccurlyeq_1$, $\preccurlyeq_2$, $\preccurlyeq_3$ rendezési relációkat a következőképpen. Legyenek $\boldsymbol{x} = (x_1, x_2, \ldots, x_m) \in \mathbb{N}^m$ és $\boldsymbol{y} = (y_1, y_2, \ldots, y_m) \in \mathbb{N}^m$. Azt fogjuk mondani, hogy

- $\boldsymbol{x} \preccurlyeq_1 \boldsymbol{y}$, ha vagy $\boldsymbol{x} = \boldsymbol{y}$ vagy pedig $x_i < y_i$ a legkisebb olyan $1 \leqslant i \leqslant m$ indexre, amelyre $x_i \neq y_i$,
- $\boldsymbol{x} \preccurlyeq_2 \boldsymbol{y}$, ha vagy $|\boldsymbol{x}| > |\boldsymbol{y}|$ vagy pedig $|\boldsymbol{x}| = |\boldsymbol{y}|$ és $\boldsymbol{x} \preccurlyeq_1 \boldsymbol{y}$,
- $\boldsymbol{x} \preccurlyeq_3 \boldsymbol{y}$, ha $x_i \leqslant y_i$ minden $1 \leqslant i \leqslant m$ esetén.

Megjegyezzük, hogy tetszőleges $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{N}^m$ sorozatokra

$$\boldsymbol{x} \preccurlyeq_k \boldsymbol{y} \iff \boldsymbol{x} + \boldsymbol{z} \preccurlyeq_k \boldsymbol{y} + \boldsymbol{z}$$

mindhárom rendezési reláció esetén.

A továbbiakban, ha ez nem okoz félreértést, a $\boldsymbol{c}$-beli reprezentáció kifejezés elejéről el fogjuk hagyni a $\boldsymbol{c}$-beli jelzőt. Adott n esetén jelölje $\boldsymbol{g}(n)$ a $\preccurlyeq_1$ rendezés szerint maximális reprezentációját n-nek. A $\boldsymbol{g}(n)$ sorozatra az n mohó reprezentációjaként fogunk hivatkozni. Vegyük észre, hogy ha $\boldsymbol{g}(n) = (g_1, g_2, \ldots, g_m)$, akkor a visszajáró pénz problémára adott mohó algoritmus éppen g_i darab c_i címletű érmét használ n kifizetéséhez minden $1 \leqslant i \leqslant m$ esetén. Jelölje továbbá adott n esetén $\boldsymbol{h}(n)$ a $\preccurlyeq_2$ rendezés szerint maximális reprezentációját n-nek. A $\boldsymbol{h}(n)$ sorozatra az n minimális reprezentációjaként fogunk hivatkozni. Jegyezzük meg, hogy $\boldsymbol{h}(n)$ az n minimális méretű reprezentációi közül a $\preccurlyeq_1$ rendezés szerint maximális.

Azt fogjuk mondani, hogy egy $\boldsymbol{c}$ pénzrendszer kanonikus, ha $\boldsymbol{g}(n) = \boldsymbol{h}(n)$ minden $n \in \mathbb{N}$ esetén. Ez azt jelenti, hogy a visszajáró pénz problémára adott mohó algoritmus az adott $\boldsymbol{c}$ pénzrendszert használva mindig optimális megoldást ad. Célunk ebben a megfogalmazásban a kanonikus pénzrendszerek azonosítása.

1. Állítás. Legyenek $n', n'' \in \mathbb{N}$. Ha $n' < n''$, akkor $\boldsymbol{g}(n') \prec_1 \boldsymbol{g}(n'')$.

Bizonyítás. Ha $n' < n''$, akkor $\boldsymbol{r} = \boldsymbol{g}(n') + (0, 0, \ldots, 0, n'' - n')$ egy reprezentációja n''-nek. Most egyrészt $\boldsymbol{g}(n') \prec_1 \boldsymbol{r}$ nyilvánvaló módon, másrészt $\boldsymbol{r} \preccurlyeq_1 \boldsymbol{g}(n'')$ a mohó reprezentáció definíciójával összhangban. Így $\boldsymbol{g}(n') \prec_1 \boldsymbol{g}(n'')$.

2. Állítás. Legyenek $n', n'' \in \mathbb{N}$, és legyen $\boldsymbol{r'}$ egy reprezentációja n'-nek, valamint $\boldsymbol{r''}$ egy reprezentációja n''-nek. Tegyük fel, hogy $\boldsymbol{r'} \preccurlyeq_3 \boldsymbol{r''}$.

(1) Ha $\boldsymbol{r''}$ mohó reprezentációja n''-nek, akkor $\boldsymbol{r'}$ is mohó reprezentációja n'-nek.

(2) Ha $\boldsymbol{r''}$ minimális reprezentációja n''-nek, akkor $\boldsymbol{r'}$ is minimális reprezentációja n'-nek.

Bizonyítás. Egy általános észrevétellel kezdjük a bizonyítást. Legyen $\boldsymbol{r}$ egy tetszőleges reprezentációja n'-nek. Ekkor $\boldsymbol{rc} = \boldsymbol{r'c}$. Innen $\boldsymbol{rc} + \boldsymbol{r''c} = \boldsymbol{r'c} + \boldsymbol{r''c}$, illetve átrendezve $\boldsymbol{r''c} - \boldsymbol{r'c} + \boldsymbol{rc} = \boldsymbol{r''c}$, majd kiemelve $(\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r})\boldsymbol{c} = \boldsymbol{r''c}$. Feltételünk szerint $\boldsymbol{r'} \preccurlyeq_3 \boldsymbol{r''}$, így $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r} \in \mathbb{N}^m$, következésképpen $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r}$ is egy reprezentációja n''-nek. Lássuk ezután a két állítás nagyon hasonló bizonyítását.

(1) Mivel $\boldsymbol{r''}$ mohó reprezentációja n''-nek, ezért $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r} \preccurlyeq_1 \boldsymbol{r''}$. Ebből következik, hogy $\boldsymbol{r''} + \boldsymbol{r} \preccurlyeq_1 \boldsymbol{r''} + \boldsymbol{r'}$, ahonnan $\boldsymbol{r} \preccurlyeq_1 \boldsymbol{r'}$ adódik. Ez viszont csak akkor lehetséges n' teteszőleges $\boldsymbol{r}$ reprezentációjára, ha $\boldsymbol{r'}$ mohó reprezentációja n'-nek.

(2) Mivel $\boldsymbol{r''}$ minimális reprezentációja n''-nek, ezért $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r} \preccurlyeq_2 \boldsymbol{r''}$. Ebből következik, hogy $\boldsymbol{r''} + \boldsymbol{r} \preccurlyeq_2 \boldsymbol{r''} + \boldsymbol{r'}$, ahonnan $\boldsymbol{r} \preccurlyeq_2 \boldsymbol{r'}$ adódik. Ez viszont csak akkor lehetséges n' teteszőleges $\boldsymbol{r}$ reprezentációjára, ha $\boldsymbol{r'}$ minimális reprezentációja n'-nek.

Tegyük fel ezek után, hogy egy $\boldsymbol{c}$ pénzrendszer nem kanonikus. Legyen n a legkisebb olyan természetes szám, amelyre $\boldsymbol{g}(n) \neq \boldsymbol{h}(n)$. Legyen $\boldsymbol{g}(n) = (g_1, g_2, \ldots, g_m)$ és $\boldsymbol{h}(n) = (h_1, h_2, \ldots, h_m)$. Megmutatjuk, hogy minden $1 \leqslant$

$i \leqslant m$ esetén g_i és h_i közül legalább az egyik nulla. Indirekt tegyük fel, hogy ez nem teljesül, azaz valamely $1 \leqslant i \leqslant m$ esetén g_i és h_i is pozitív. Ekkor

$$\boldsymbol{r'} = (g_1, \ldots, g_{i-1}, g_i - 1, g_{i+1}, \ldots, g_m) \in \mathbb{N}^m$$

és

$$\boldsymbol{r''} = (h_1, \ldots, h_{i-1}, h_i - 1, h_{i+1}, \ldots, h_m) \in \mathbb{N}^m$$

az n-nél kisebb $n - c_i \in \mathbb{N}$ szám olyan reprezentációi, amelyekre $\boldsymbol{r'} \prec_3 \boldsymbol{g}(n)$ és $\boldsymbol{r''} \prec_3 \boldsymbol{h}(n)$, így a 2. Állítás szerint $\boldsymbol{r'} = \boldsymbol{g}(n - c_i)$ és $\boldsymbol{r''} = \boldsymbol{h}(n - c_i)$. Ez viszont azt jelenti, hogy $\boldsymbol{g}(n - c_i) \neq \boldsymbol{h}(n - c_i)$, ellentmondva n minimalitásának.

Az n szám minimális reprezentációjában legyen h_α az első és h_β az utolsó pozitív tag. Ekkor az előzőekkel összhangban az n szám mohó reprezentációjában $g_\alpha = 0$. Ám $\boldsymbol{h}(n) \prec_1 \boldsymbol{g}(n)$ miatt ekkor g_i szükségképpen pozitív valamely $i < \alpha$ esetén. Ebből rögtön adódik, hogy $\alpha > 1$.

Meglepő kapcsolat áll fenn n minimális és $c_{\alpha-1} - 1$ mohó reprezentációja között. Legyen $\boldsymbol{g}(c_{\alpha-1} - 1) = (f_1, f_2, \ldots, f_m)$. Ekkor

$$h_i = \begin{cases} f_i & \text{ha } i < \beta, \\ f_i + 1 & \text{ha } i = \beta, \\ 0 & \text{ha } i > \beta. \end{cases}$$

Ez a következőképpen látható be. Mivel g_i pozitív valamely $i < \alpha$ esetén, ezért $n \geqslant c_{\alpha-1}$. Másrészt a 2. Állítással összhangban

$$(h_1, \ldots, h_{\beta-1}, h_\beta - 1, h_{\beta+1}, \ldots, h_m) = \boldsymbol{h}(n - c_\beta).$$

Ugyanakkor $\boldsymbol{h}(n - c_\beta) = \boldsymbol{g}(n - c_\beta)$, hiszen $n - c_\beta < n$ és feltételünk szerint n a legkisebb olyan természetes szám, amelynek minimális és mohó reprezentációja különböző. Felhasználva, hogy $h_i = 0$ minden $i < \alpha$ esetén, ebből $n - c_\beta < c_{\alpha-1}$ és így $n - c_\beta \leqslant c_{\alpha-1} - 1$ azonnal adódik. Ennélfogva az 1. Állítás szerint

$$\boldsymbol{g}(n - c_\beta) \preccurlyeq_1 \boldsymbol{g}(c_{\alpha-1} - 1).$$

Tekintsük most a $c_{\alpha-1} - 1$ számot. Mivel $c_{\alpha-1} - 1 \geqslant c_\alpha$, ezért $c_{\alpha-1} - 1$ mohó reprezentációjában $f_\alpha \geqslant 1$. Ismét a 2. Állítással összhangban

$$(f_1, \ldots, f_{\alpha-1}, f_\alpha - 1, f_{\alpha+1}, \ldots, f_m) = \boldsymbol{g}(c_{\alpha-1} - 1 - c_\alpha).$$

Másrészt

$$(h_1, \ldots, h_{\alpha-1}, h_\alpha - 1, h_{\alpha+1}, \ldots, h_m) = \boldsymbol{h}(n - c_\alpha)$$

szintén a 2. Állítás szerint. Ennélfogva

$$(h_1, \ldots, h_{\alpha-1}, h_\alpha - 1, h_{\alpha+1}, \ldots, h_m) = \boldsymbol{g}(n - c_\alpha),$$

hiszen $n - c_\alpha < n$ és feltételünk szerint n a legkisebb olyan természetes szám, amelynek minimális és mohó reprezentációja különböző. Idézzük fel, hogy $c_{\alpha-1} \leqslant n$, következésképpen $c_{\alpha-1} - 1 - c_\alpha < n - c_\alpha$, így az 1. Állítás szerint $\boldsymbol{g}(c_{\alpha-1} - 1 - c_\alpha) \prec_1 \boldsymbol{g}(n - c_\alpha)$. Mindkét sorozat α-adik tagját eggyel megnövelve, az előbbiek figyelembe vételével,

$$\boldsymbol{g}(c_{\alpha-1} - 1) \prec_1 \boldsymbol{h}(n)$$

adódik.

Végül tekintsük a

$$\boldsymbol{g}(n - c_\beta) \preccurlyeq_1 \boldsymbol{g}(c_{\alpha-1} - 1) \prec_1 \boldsymbol{h}(n)$$

láncot. Mivel a $\boldsymbol{g}(n - c_\beta)$ és a $\boldsymbol{h}(n)$ sorozatok csak a β-adik tagban különböznek, ezért $f_i = h_i$ minden $i < \beta$ esetén. Ugyanakkor $\boldsymbol{g}(c_{\alpha-1} - 1) \prec_1 \boldsymbol{h}(n)$ miatt $f_j < h_j$ valamely $j \geqslant \beta$ indexre. Ám $h_i = 0$ minden $i > \beta$ esetén, ezért $j = \beta$ lehetséges csak, ennélfogva $f_\beta < h_\beta$. Másrészt $\boldsymbol{g}(n - c_\beta) \preccurlyeq_1 \boldsymbol{g}(c_{\alpha-1} - 1)$ miatt $h_\beta - 1 \leqslant f_\beta$, így $h_\beta - 1 \leqslant f_\beta < h_\beta$, ahonnan $h_\beta = f_\beta + 1$ következik.

Ezek után egy $\boldsymbol{c}$ pénzrendszerről már könnyen eldönthető, hogy kanonikus vagy nem. Minden $2 \leqslant \alpha \leqslant \beta \leqslant m$ esetén állítsuk elő a $\boldsymbol{g}(c_{\alpha-1} - 1) = (f_1, f_2, \ldots, f_m)$ sorozatot a mohó algoritmussal, illetve az $\boldsymbol{r} = (r_1, r_2, \ldots, r_m)$ sorozatot, ahol

$$r_i = \begin{cases} f_i & \text{ha } i < \beta, \\ f_i + 1 & \text{ha } i = \beta, \\ 0 & \text{ha } i > \beta. \end{cases}$$

Ha $\boldsymbol{c}$ nem kanonikus, akkor a fentiek szerint az így kapott $\boldsymbol{r}$ sorozatok között találni fogunk olyat, amelyre $|\boldsymbol{g}(\boldsymbol{rc})| > |\boldsymbol{r}|$. Mivel a szóba jövő $\boldsymbol{r}$ sorozatok száma $O(m^2)$, és minden sorozatról $O(m)$ lépésben eldönthető ennek a tulajdonság teljesülése, ezért az algoritmus teljes lépésszáma $O(m^3)$.

Tánciskola

Feladat.
Egy tánciskolába n fiú és n lány jár.

(A) Adjunk hatékony algoritmust egy olyan párosítás megtalálására, amelyben a párok közötti magasságkülönbségek maximuma a lehető legkisebb!

(B) Adjunk hatékony algoritmust egy olyan párosítás megtalálására, amelyben a párok közötti magasságkülönbségek összege a lehető legkisebb!

Megoldás.
(A) Rendezzük a fiúkat és a lányokat külön-külön nagyság szerint csökkenő sorrendbe. Jelölje az így kialakult sorrendet $f_1, f_2, \ldots, f_n$ a fiúk esetén, $l_1, l_2, \ldots, l_n$ pedig a lányok esetén. Ezek után a legmagasabb fiú párja legyen a legmagasabb lány, a második legmagasabb fiú párja a második legmagasabb lány, és így tovább, végül a legalacsonyabb fiú párja a legalacsonyabb lány.

Az algoritmus költsége nyilván $O(n \log n)$. Jegyezzük meg, hogy ha azzal a természetes feltételezéssel élünk, hogy a magasságok (centiméterben mérve) mondjuk 140 és 210 közötti egész számok, akkor az algoritmus költsége például számlálva szétosztó rendezést alkalmazva $O(n)$.

Megmutatjuk, hogy az algoritmus által előállított $\mathcal{G}$ párosítás optimális. Tekintsük a tánciskolába járó fiúk és lányok egy optimális $\mathcal{O}$ párosítását. Ha $\mathcal{O} = \mathcal{G}$, akkor készen vagyunk. Tegyük fel ezért, hogy $\mathcal{O} \neq \mathcal{G}$.

Legyen i a legkisebb olyan index, amelyre az f_i fiú partnere különbözik $\mathcal{O}$-ban és $\mathcal{G}$-ben. Jelölje az f_i fiú $\mathcal{O}$-beli partnerét l_j, továbbá az l_i lány $\mathcal{O}$-beli partnerét f_k. Itt szükségképpen $j > i$ és $k > i$.

Tekintsük most azt az $\mathcal{O}'$ párosítást, amelyet $\mathcal{O}$-ból úgy kapunk, hogy az (f_i, l_j) és (f_k, l_i) párokat lecseréljük az (f_i, l_i) és (f_k, l_j) párokra. Megmutatjuk, hogy ezzel a cserével a maximális magasságkülönbség nem növekedett. Ehhez nyilván elég belátni, hogy

$$\max(|m(f_i) - m(l_i)|, |m(f_k) - m(l_j)|) \leqslant \\ \leqslant \max(|m(f_i) - m(l_j)|, |m(f_k) - m(l_i)|).$$

(Itt $m()$ természetesen a tánciskolások magasságát jelenti.) Hat esetet különböztethetünk meg.
(1) $m(f_i) \geqslant m(f_k) \geqslant m(l_i) \geqslant m(l_j)$. Ekkor

$$|m(f_i) - m(l_i)| \leqslant |m(f_i) - m(l_j)|,$$
$$|m(f_k) - m(l_j)| \leqslant |m(f_i) - m(l_j)|,$$

így ebben az esetben igaz az állítás.
(2) $m(f_i) \geqslant m(l_i) \geqslant m(f_k) \geqslant m(l_j)$. Ekkor

$$|m(f_i) - m(l_i)| \leqslant |m(f_i) - m(l_j)|,$$
$$|m(f_k) - m(l_j)| \leqslant |m(f_i) - m(l_j)|,$$

így ebben az esetben igaz az állítás.
(3) $m(l_i) \geqslant m(f_i) \geqslant m(f_k) \geqslant m(l_j)$. Ekkor

$$|m(f_i) - m(l_i)| \leqslant |m(f_k) - m(l_i)|,$$
$$|m(f_k) - m(l_j)| \leqslant |m(f_i) - m(l_j)|,$$

így ebben az esetben igaz az állítás.
(4) $m(f_i) \geqslant m(l_i) \geqslant m(l_j) \geqslant m(f_k)$. Ekkor

$$|m(f_i) - m(l_i)| \leqslant |m(f_i) - m(l_j)|,$$
$$|m(f_k) - m(l_j)| \leqslant |m(f_k) - m(l_i)|,$$

így ebben az esetben igaz az állítás.
(5) $m(l_i) \geqslant m(f_i) \geqslant m(l_j) \geqslant m(f_k)$. Ekkor

$$|m(f_i) - m(l_i)| \leqslant |m(f_k) - m(l_i)|,$$
$$|m(f_k) - m(l_j)| \leqslant |m(f_k) - m(l_i)|,$$

így ebben az esetben igaz az állítás.
(6) $m(l_i) \geqslant m(l_j) \geqslant m(f_i) \geqslant m(f_k)$. Ekkor

$$|m(f_i) - m(l_i)| \leqslant |m(f_k) - m(l_i)|,$$
$$|m(f_k) - m(l_j)| \leqslant |m(f_k) - m(l_i)|,$$

így ebben az esetben igaz az állítás.

Ezzel beláttuk, hogy az optimális $\mathcal{O}$ párosítás áttranszformálható egy olyan optimális $\mathcal{O}'$ párosítássá, hogy $\mathcal{O}'$-ben és $\mathcal{G}$-ben már az f_i fiú partnere is ugyanaz a lány. Az eljárást folytatva $\mathcal{O}$ lépésről-lépésre áttranszformálható $\mathcal{G}$-be a maximális magasságkülönbség növelése nélkül. Következésképpen $\mathcal{G}$ is egy optimális párosítás.

(B) Megmutatjuk, hogy a fenti algoritmus a magasságkülönbségek összegét is minimalizálja. A bizonyítás hasonló az előzőhöz. Tekintsük a tánciskolába járó fiúk és lányok egy optimális $\mathcal{O}$ párosítását. Ha $\mathcal{O} = \mathcal{G}$, akkor készen vagyunk. Tegyük fel ezért, hogy $\mathcal{O} \neq \mathcal{G}$.

Legyen i a legkisebb olyan index, amelyre az f_i fiú partnere különbözik $\mathcal{O}$-ban és $\mathcal{G}$-ben. Jelölje az f_i fiú $\mathcal{O}$-beli partnerét l_j, továbbá az l_i lány $\mathcal{O}$-beli partnerét f_k. Itt szükségképpen $j > i$ és $k > i$.

Tekintsük most azt az $\mathcal{O}'$ párosítást, amelyet $\mathcal{O}$-ból úgy kapunk, hogy az (f_i, l_j) és (f_k, l_i) párokat lecseréljük az (f_i, l_i) és (f_k, l_j) párokra.

Megmutatjuk, hogy ezzel a cserével a magasságkülönbségek összege nem növekedett. Ehhez nyilván elég belátni, hogy

$$|m(f_i) - m(l_i)| + |m(f_k) - m(l_j)| \leqslant |m(f_i) - m(l_j)| + |m(f_k) - m(l_i)|,$$

illetve, ekvivalens módon,

$$\Delta = (|m(f_i) - m(l_i)| + |m(f_k) - m(l_j)|) - \\ - (|m(f_i) - m(l_j)| + |m(f_k) - m(l_i)|) \leqslant 0.$$

Most is hat esetet különböztethetünk meg.
(1) $m(f_i) \geqslant m(f_k) \geqslant m(l_i) \geqslant m(l_j)$. Ekkor

$$\Delta = ((m(f_i) - m(l_i)) + (m(f_k) - m(l_j))) - \\ - ((m(f_i) - m(l_j)) + (m(f_k) - m(l_i))) = 0,$$

így ebben az esetben igaz az állítás.
(2) $m(f_i) \geqslant m(l_i) \geqslant m(f_k) \geqslant m(l_j)$. Ekkor

$$\begin{aligned}\Delta &= ((m(f_i) - m(l_i)) + (m(f_k) - m(l_j))) - \\ &\quad - ((m(f_i) - m(l_j)) + (m(l_i) - m(f_k))) \\ &= 2(m(f_k) - m(l_i)) \leqslant 0,\end{aligned}$$

így ebben az esetben igaz az állítás.
(3) $m(l_i) \geqslant m(f_i) \geqslant m(f_k) \geqslant m(l_j)$. Ekkor

$$\begin{aligned}\Delta &= ((m(l_i) - m(f_i)) + (m(f_k) - m(l_j))) - \\ &\quad - ((m(f_i) - m(l_j)) + (m(l_i) - m(f_k))) \\ &= 2(m(f_k) - m(f_i)) \leqslant 0,\end{aligned}$$

így ebben az esetben igaz az állítás.
(4) $m(f_i) \geqslant m(l_i) \geqslant m(l_j) \geqslant m(f_k)$. Ekkor

$$\begin{aligned}\Delta &= ((m(f_i) - m(l_i)) + (m(l_j) - m(f_k))) - \\ &\quad - ((m(f_i) - m(l_j)) + (m(l_i) - m(f_k))) \\ &= 2(m(l_j) - m(l_i)) \leqslant 0,\end{aligned}$$

így ebben az esetben igaz az állítás.
(5) $m(l_i) \geqslant m(f_i) \geqslant m(l_j) \geqslant m(f_k)$. Ekkor

$$\begin{aligned}\Delta &= ((m(l_i) - m(f_i)) + (m(l_j) - m(f_k))) - \\ &\quad - ((m(f_i) - m(l_j)) + (m(l_i) - m(f_k))) \\ &= 2(m(l_j) - m(f_i)) \leqslant 0,\end{aligned}$$

így ebben az esetben igaz az állítás.
(6) $m(l_i) \geqslant m(l_j) \geqslant m(f_i) \geqslant m(f_k)$. Ekkor

$$\Delta = ((m(l_i) - m(f_i)) + (m(l_j) - m(f_k))) - \\ - ((m(l_j) - m(f_i)) + (m(l_i) - m(f_k))) = 0,$$

így ebben az esetben igaz az állítás.

Ezzel beláttuk, hogy az optimális $\mathcal{O}$ párosítás áttranszformálható egy olyan optimális $\mathcal{O}'$ párosítássá, hogy $\mathcal{O}'$-ben és $\mathcal{G}$-ben már az f_i fiú partnere is ugyanaz a lány. Az eljárást folytatva $\mathcal{O}$ lépésről-lépésre áttranszformálható $\mathcal{G}$-be a magasságkülönbségek összegének növelése nélkül. Következésképpen $\mathcal{G}$ is egy optimális párosítás.

Információtömörítés

Feladat.
Tegyük fel, hogy egy n különböző karakterből álló adatállományt akarunk tömörítetten, bitsorozatként tárolni. Ha fix hosszú kódszavakat használunk, akkor $\lceil \log_2 n \rceil$ bitre van szükség az n-féle karakter tárolására. Így egy m hosszú állomány tárolása összesen $m\lceil \log_2 n \rceil$ bitet igényel. Van ennél hatékonyabb módszer?

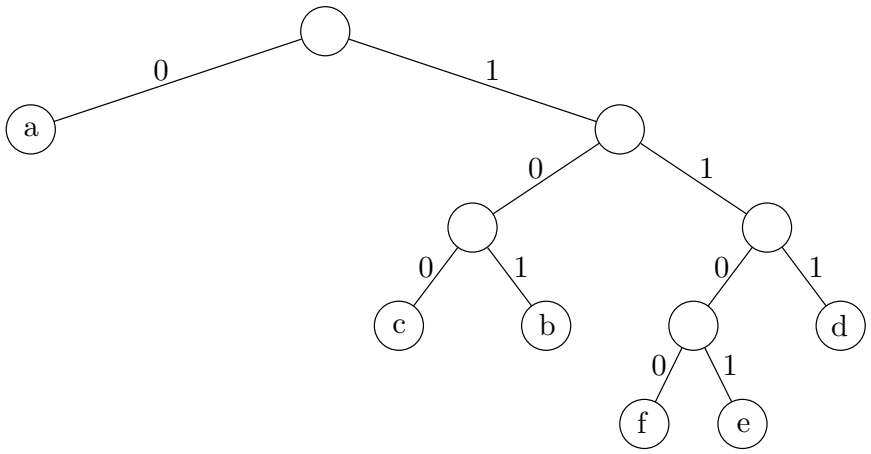
Ha megengedünk eltérő hosszú kódszavakat, akkor probléma lehet a dekódolással, vagyis az eredeti állománynak a kódolt bitsorozatból való visszanyerésével. Egy lehetséges megoldást kínál a következő. Egy bitsorozatokból álló kód prefix kód, ha egyik karakter kódja sem prefixe más karakter kódjának. Formálisan, ha x és y két különböző karakter kódja, akkor nincs olyan z bitsorozat, amelyre $xz = y$. Itt xz azt a bitsorozatot jelöli, melyben x bitjeit z bitjei követik. Egy prefix kóddal leírt állomány egyértelműen dekódolható. Az állomány elejétől addig megyünk a bitek mentén, amíg egy karakter kódszavát kapjuk. A prefix feltétel miatt csak ez lehet az első karakter. Ezt az eljárást addig folytatjuk, amíg a kódolt állomány végére nem érünk.

Egy prefix kódhoz természetes módon hozzárendelhetünk egy bináris fát. A fa levelei a kódolandó karakterek. Egy adott karakter kódját a gyökértől a karakterig vezető út ábrázolja: a 0 azt jelenti, hogy balra megyünk, az 1 pedig azt, hogy jobbra megyünk az úton a fában.

Ha adott egy prefix kód T fája, akkor egyszerűen kiszámítható az adatállomány kódolásához szükséges bitek száma. A C ábécé minden c karakterére jelölje $f[c]$ a c karakter előfordulási gyakoriságát az állományban, $d_T(c)$ pedig a c-t tartalmazó levél mélységét a T fában (ami éppen a c karakter kódjának hossza). A kódoláshoz szükséges bitek száma ekkor

$$B_T(C) = \sum_{c \in C} f[c] d_T(c).$$

Ezt az értéket a fa költségének fogjuk nevezni.



A feladat ezek után egy minimális költségű prefix kód előállítása a karakterek előfordulási gyakoriságának ismeretében.

Megoldás.
Először is vegyük észre, hogy egy adatállomány optimális prefix kódját mindig teljes bináris fa ábrázolja, azaz olyan fa, amelyben minden nem levél csúcsnak két gyermeke van. Valóban, ha lenne olyan belső csúcs, amelynek csak egy gyereke van, akkor ezt a csúcsot törölve (egyetlen gyerekét a helyére téve) a fa költségét csökkenteni tudnánk. Innen következik, hogy az optimális prefix kód fájának $|C|$ levele és $|C|-1$ belső csúcsa van (ld. a bináris fák leveleiről szóló feladatot).

A következő algoritmus egy optimális prefix kód fáját konstruálja meg. Az algoritmus alulról felfelé haladva építi meg ezt a fát. Kezdetben $|C|$ számú csúcs van, amelyek mindegyike levél, majd $|C|-1$ számú "összevonás" végrehajtása után alakul ki a végső fa. Két csúcs összevonásának eredménye egy új csúcs, melynek gyakorisága a két összevont csúcs gyakoriságának az összege. Mindig a két legkisebb gyakoriságú csúcsot vonjuk össze; ezek azonosítására egy f szerint kulcsolt Q kupacot használunk. Az algoritmus a megkonstruált fa gyökerével tér vissza.

```
OptimálisPrefixKód(C,f,n)
Q=KupacotÉpít(C,f,n)
for i=1 to n-1 do
  x=MinTöröl(Q)
  y=MinTöröl(Q)
  z=CsúcsotLétrehoz
  bal[z]=x
  jobb[z]=y
  f[z]=f[x]+f[y]
```

```
  Beszúr(Q,z)
return MinTöröl(Q)
```

A kupacépítés költsége $O(n)$. A for ciklus iterációinak száma $n-1$, így mivel az egyes kupacműveletek költsége $O(\log n)$, a ciklus összköltsége $O(n \log n)$. Ennélfogva az algoritmus teljes költsége $O(n \log n)$. Az algoritmus helyessége a következő két állításból adódik.

Állítás. Legyen x és y a két legkisebb gyakoriságú karakter C-ben. Ekkor van olyan optimális prefix kód, amelyben x és y kódszava ugyanolyan hosszú, és a két kódszó csak az utolsó bitben különbözik.

Bizonyítás. Tekintsünk egy optimális prefix kódot ábrázoló T fát. Legyen a és b ebben a fában a legmélyebben fekvő "testvér" csúcspár. Az általánosság megszorítása nélkül feltehetjük, hogy $f[a] \leqslant f[b]$ és $f[x] \leqslant f[y]$. Mivel $f[x]$ és $f[y]$ a két legkisebb gyakoriság, ezért $f[x] \leqslant f[a]$ és $f[y] \leqslant f[b]$.

Cseréljük fel T-ben az x és a csúcsokat. Jelölje az új fát T'. Ezután cseréljük fel T'-ben az y és b csúcsokat. Jelölje az új fát T''. A T és T' fák költségének különbsége:

$$\begin{aligned} B_T(C) - B_{T'}(C) &= \sum_{c \in C} f[c] d_T(c) - \sum_{c \in C} f[c] d_{T'}(c) \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\ &= (f[a] - f[x])(d_T(a) - d_T(x)) \geqslant 0, \end{aligned}$$

hiszen egyrészt az a karakter gyakorisága legalább akkora, mint az x karakteré, másrészt a legalább olyan mélyen van T-ben, mint x. Hasonlóan, a T' és T'' fák költségének különbsége:

$$\begin{aligned} B_{T'}(C) - B_{T''}(C) &= \sum_{c \in C} f[c] d_{T'}(c) - \sum_{c \in C} f[c] d_{T''}(c) \\ &= f[y] d_{T'}(y) + f[b] d_{T'}(b) - f[y] d_{T''}(y) - f[b] d_{T''}(b) \\ &= f[y] d_{T'}(y) + f[b] d_{T'}(b) - f[y] d_{T'}(b) - f[b] d_{T'}(y) \\ &= (f[b] - f[y])(d_{T'}(b) - d_{T'}(y)) \geqslant 0, \end{aligned}$$

hiszen egyrészt a b karakter gyakorisága legalább akkora, mint az y karakteré, másrészt b legalább olyan mélyen van T'-ben, mint y.

Kaptuk tehát, hogy

$$B_T(C) \geqslant B_{T'}(C) \quad \text{és} \quad B_{T'}(C) \geqslant B_{T''}(C),$$

így

$$B_T(C) \geqslant B_{T''}(C).$$

Figyelembe véve, hogy T optimális prefix kódot ábrázoló fa, ez csak úgy lehetséges, ha

$$B_T(C) = B_{T''}(C).$$

Ebből következik, hogy T'' is egy optimális prefix kódot ábrázoló fa, amelyben x és y maximális mélységű testvércsúcsok. Innen az állítás adódik.

Állítás. Legyen x és y a két legkisebb gyakoriságú karakter C-ben. Most tekintsük azt a C' ábécét, amelyet úgy kapunk C-ből, hogy eltávolítjuk x-et és y-t, majd hozzáadunk egy új z karaktert, melyre $f[z] = f[x] + f[y]$. A többi karakter gyakorisága nem változik. Legyen T' egy optimális prefix kódot ábrázoló fa a C' ábécéhez. Ekkor az a T fa, amelyet úgy kapunk T'-ből, hogy a z csúcshoz gyerekként hozzákapcsoljuk x-et és y-t, egy optimális prefix kódot ábrázoló fa a C ábécéhez.

Bizonyítás. Először is jegyezzük meg, hogy

$$\begin{aligned} B_T(C) - B_{T'}(C') &= f[x]d_T(x) + f[y]d_T(y) - f[z]d_{T'}(z) \\ &= (f[x] + f[y])(d_{T'}(z) + 1) - (f[x] + f[y])d_{T'}(z) \\ &= f[x] + f[y]. \end{aligned}$$

Ezek után indirekt tegyük fel, hogy T nem egy optimális prefix kódot ábrázoló fa a C ábécéhez. Legyen T'' egy optimális prefix kódot ábrázoló fa a C ábécéhez. Az előző állítás alapján feltehetjük, hogy x és y testvérek T''-ben. Most

$$B_T(C) > B_{T''}(C).$$

Legyen T''' az a fa, amelyet úgy kapunk T''-ből, hogy eltávolítjuk az x és y csúcsokat, ezek közös szülőjét z-vel jelöljük, és ehhez a z csúcshoz hozzárendeljük az $f[z] = f[x] + f[y]$ gyakoriságot. Ekkor

$$\begin{aligned} B_{T''}(C) - B_{T'''}(C') &= f[x]d_{T''}(x) + f[y]d_{T''}(y) - f[z]d_{T'''}(z) \\ &= (f[x] + f[y])(d_{T'''}(z) + 1) - (f[x] + f[y])d_{T'''}(z) \\ &= f[x] + f[y]. \end{aligned}$$

Így a T''' fára

$$B_{T'''}(C') = B_{T''}(C) - (f[x] + f[y]) < B_T(C) - (f[x] + f[y]) = B_{T'}(C'),$$

ami ellentmond annak, hogy T' egy optimális prefix kódot ábrázoló fa a C' ábécéhez. Ebből következik, hogy T egy optimális prefix kódot ábrázoló fa a C ábécéhez.

Gráfalgoritmusok

Szuperforrás

Feladat.
Egy $G = (V, E)$ irányított gráf $s \in V$ csúcsát szuperforrásnak nevezzük, ha minden $v \in V \setminus \{s\}$ csúcsra teljesül, hogy $(s, v) \in E$ és $(v, s) \notin E$. Nyilvánvaló, hogy egy irányított gráfban legfeljebb egy szuperforrás lehet.

Adjunk hatékony algoritmust annak eldöntésére, hogy egy C szomszédsági mátrixával adott $G = (V, E)$ irányított gráfban van-e szuperforrás! Ha van, akkor az algoritmus adja is meg azt!

Megoldás.
A következő módszer azon az egyszerű észrevételen alapul, hogy tetszőleges $v_i, v_j \in V$ különböző csúcsokra, ha $(v_i, v_j) \in E$, akkor v_j nem lehet szuperforrás, ha pedig $(v_i, v_j) \notin E$, akkor v_i nem lehet szuperforrás.

```
Szuperforrás(G)
n=|V|
i=1
j=n
while i<>j do
  if C[i,j]=1
    then j=j-1
    else i=i+1
for k=1 to n do
  if k<>i AND (C[i,k]<>1 OR C[k,i]<>0) then return nil
return i
```

Az első ciklus leszűkíti a keresést egyetlen jelöltre, a második ciklus pedig ezt a jelöltet ellenőrzi. Az eljárás költsége $O(|V|)$.

Összefüggő gráfok

Feladat.
Mutassuk meg, hogy bármely olyan algoritmusnak, amely egy szomszédsági mátrixával adott irányítatlan gráf összefüggőségét dönti el, a legrosszabb esetben az összes csúcspár összekötöttségét meg kell vizsgálnia! (Vagyis létezik olyan gráf, amelyre az algoritmusnak az összes csúcspár összekötöttségét meg kell vizsgálnia.)

Megoldás.
Indirekt tegyük fel, hogy van olyan algoritmus, amely bármely n csúcsú gráfról legfeljebb $\binom{n}{2} - 1$ csúcspár összekötöttségét megvizsgálva meg tudja állapítani, hogy a gráf összefüggő vagy nem.

Indítsuk el az algoritmust. Az "ellenpélda" gráfot az algoritmus futásával párhuzamosan hozzuk létre a következőképpen. Ha az algoritmus rákérdez egy csúcspár összekötöttségére, nemmel válaszolunk, ha az eddig igenlően megválaszolt és a még meg nem kérdezett "élek" együtt összefüggő gráfot alkotnak. Különben igennel válaszolunk.

Először is jegyezzük meg, hogy az algoritmus soha nem juthat arra a következtetésre, hogy a vizsgált gráf nem összefüggő. Feltételünk szerint az algoritmus legfeljebb $\binom{n}{2} - 1$ csúcspár összekötöttségét megvizsgálva már biztosan tudja, hogy a gráf összefüggő.

Legyenek u és v olyan csúcsok, amelyek összekötöttségére az algoritmus nem kérdezett rá. Mivel az igenlően megválaszolt élek összefüggő gráfot alkotnak, ebben a gráfban nyilván létezik egy u-t és v-t összekötő P út. Tekintsük ennek az útnak egy tetszőleges (w, z) élét. Amikor az algoritmus rákérdezett a w és z csúcsok összekötöttségére, azért válaszoltunk igennel, mert az addig igenlően megválaszolt és a még meg nem kérdezett "élek" együtt már nem alkottak összefüggő gráfot. Ám ez lehetetlen, hisz ebben a gráfban szerepelt az (u, v) él és P-nek a (w, z)-n kívüli élei, amelyek együtt egy w-t és z-t összekötő utat alkotnak.

Fokszámsorozatok

Feladat.
Adjunk hatékony algoritmust annak eldöntésére, hogy adott $d_1, d_2, \ldots, d_n$ nem negatív egész számokhoz létezik-e olyan n csúcsú (egyszerű) gráf, amelyben pontosan ezek a fokszámok!

Megoldás.
Először bebizonyítjuk a következő állítást.

Állítás. Akkor és csak akkor létezik n csúcsú (egyszerű) gráf a $0 \leqslant d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$ fokszámokkal, ha létezik $n-1$ csúcsú (egyszerű) gráf a $d'_1, d'_2, \ldots, d'_{n-1}$ fokszámokkal, ahol

$$d'_j = \begin{cases} d_j & \text{ha } 1 \leqslant j \leqslant n - d_n - 1, \\ d_j - 1 & \text{ha } n - d_n \leqslant j \leqslant n-1. \end{cases}$$

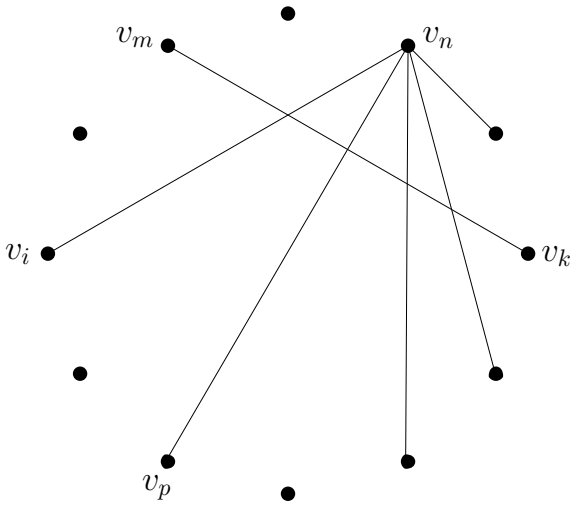
Szemléletesen úgy fogalmazhatjuk az állítást, hogy ha van az adott fokszámokkal (egyszerű) gráf, akkor van olyan (egyszerű) gráf is az adott fokszámokkal, amelyben a legnagyobb fokú csúcs az utána következő legnagyobb fokúakkal van összekötve.

Bizonyítás. Ha a $d'_1, d'_2, \ldots, d'_{n-1}$ számok egy G' gráf fokszámai, akkor egy új v csúcsot felvéve és összekötve G'-nek a $d'_j = d_j - 1$ fokú csúcsaival ($n - d_n \leqslant j \leqslant n-1$), a kapott n csúcsú gráf fokszámai éppen $d_1, d_2, \ldots, d_n$ lesznek.

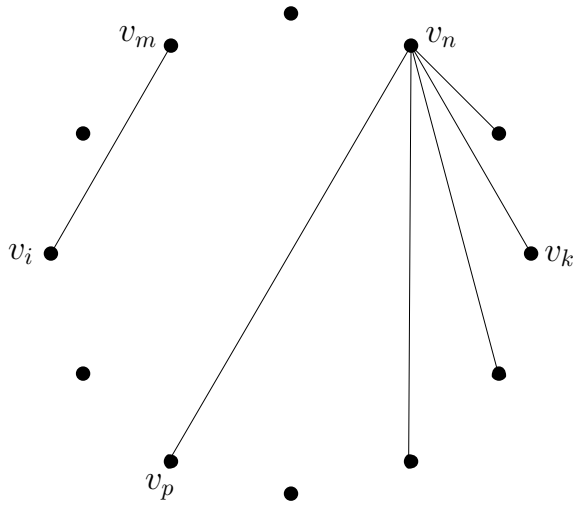
Megfordítva, tegyük fel, hogy létezik olyan G gráf, amelyben a csúcsok fokszámai $d_1, d_2, \ldots, d_n$. Minden $1 \leqslant j \leqslant n$ esetén jelölje a d_j fokú csúcsot v_j, és legyen $p = n - d_n$. Állítjuk, hogy G választható úgy, hogy v_n szomszédos a $v_p, \ldots, v_{n-1}$ csúcsok mindegyikével.

Azon n csúcsú gráfok közül, amelyekben minden $1 \leqslant j \leqslant n$ esetén a v_j csúcs foka d_j, válasszuk G-t úgy, hogy a v_n csúcs a $v_p, \ldots, v_{n-1}$ csúcsok közül a lehető legtöbbel legyen szomszédos. Megmutatjuk, hogy ekkor G-ben a v_n csúcs a $v_p, \ldots, v_{n-1}$ csúcsok mindegyikével szomszédos.

Indirekt tegyük fel, hogy G-ben a v_n csúcs nem szomszédos a v_k csúccsal valamely $p \leqslant k \leqslant n-1$ esetén. Ekkor a v_n csúcsnak szomszédosnak kell lenni egy v_i csúccsal valamely $1 \leqslant i \leqslant p-1$ esetén, hiszen v_n foka $n-p$. Mivel $i \leqslant p-1$ és $k \geqslant p$, így $d_i \leqslant d_k$, következésképpen van olyan v_i-től és v_k-tól különböző v_m csúcs, amely v_k-val szomszédos, de v_i-vel nem.



Itt a G gráfban $\{v_n, v_i\}$ és $\{v_k, v_m\}$ él, míg $\{v_n, v_k\}$ és $\{v_i, v_m\}$ nem azok. Ha most G-ből elhagyjuk $\{v_n, v_i\}$ és $\{v_k, v_m\}$ éleket, és helyettük behúzzuk a $\{v_n, v_k\}$ és $\{v_i, v_m\}$ éleket, akkor olyan G^* gráfot kapunk, amelyekben a fokszámok ugyanazok, mint G-ben.



Ez a G^* viszont G-nél több $\{v_n, v_k\}$ típusú élt tartalmaz ($p \leqslant k \leqslant n-1$), ami ellentmondás.

Így van olyan $d_1, d_2, \ldots, d_n$ fokszámokkal rendelkező gráf, amelyben a v_n csúcs a $v_p, \ldots, v_{n-1}$ csúcsok mindegyikével szomszédos. Ebből a gráfból elhagyva a v_n csúcsot a rá illeszkedő élekkel együtt, olyan gráfot kapunk, amelyben a fokszámok $d'_1, d'_2, \ldots, d'_{n-1}$.

Az állítás egyben eljárást is ad a $d_1, d_2, \ldots, d_n$ fokszámsorozat realizálására. A legnagyobb fokú csúcsot kössük össze az utána következő legnagyobb fokú csúcsokkal. Ezután töröljük a legnagyobb fokú csúcsot, és csökkentsük eggyel az azzal összekötött csúcsok fokszámát. Ezt az eljárást ismételve felrajzolhatunk egy $d_1, d_2, \ldots, d_n$ fokszámokkal rendelkező gráfot. Megjegyezzük, hogy az eljárás közben a csúcsok sorrendje változhat. Akkor akadunk el, ha valamelyik csúcs foka nagyobb, mint ahány tőle különböző csúcs van éppen. Ilyenkor az eredeti fokszámsorozat nem realizálható.

Legrövidebb utak száma

Feladat.
Legyen $G = (V, E)$ egy szomszédsági listákkal adott irányított vagy irányítatlan gráf, és $s, t \in V$ két tetszőleges csúcs. Adjunk hatékony algoritmust

az s-ből t-be vezető legrövidebb, azaz legkevesebb élből álló, utak számának meghatározására!

Megoldás.
Nem csak a t-be, hanem az összes többi csúcsba vezető legrövidebb utak számát meghatározhatjuk a szélességi keresés algoritmus alábbi módosításával.

```
LegrövidebbUtakSzáma(G,s)
for V\{s} minden u csúcsára do
  d[u]=INFINITY
  n[u]=0
d[s]=0
n[s]=1
SorInicializálás(Q)
Sorba(Q,s)
while Q<>EMPTYSET do
  u=Sorból(Q)
  for A[u] minden v csúcsára do
    if d[v]=d[u]+1 then n[v]=n[v]+n[u]
    if d[v]=INFINITY then
      d[v]=d[u]+1
      n[v]=n[u]
      Sorba(Q,v)
```

Minden $v \in V$ csúcsra az s-ből v-be vezető legrövidebb utak hosszát a $d[v]$, számát pedig az $n[v]$ változó tartalmazza.

Az algoritmus helyességének belátásához elég megjegyezni, hogy az s csúcsból egy v csúcsba vezető legrövidebb utak száma megegyezik az s csúcsból azokba a $v_1, v_2, \ldots, v_k$ csúcsokba vezető legrövidebb utak számának összegével, amelyekre $(v_1, v), (v_2, v), \ldots, (v_k, v) \in E$ és $d[v_1] = d[v_2] = \cdots = d[v_k] = d[v] - 1$.

Az algoritmus költsége világos módon $O(|V| + |E|)$.

Vizeskancsók

Feladat.
Van három vizeskancsónk, egyenként 4, 7 és 10 liter űrtartalmúak. Kezdetben a 4 és 7 literes kancsók tele vannak vízzel, a 10 literes kancsó pedig üres. Egy megengedett lépés a következő: egy kancsóból vizet töltünk át egy másik kancsóba, amíg vagy az egyik kancsó ki nem ürül, vagy a másik tele nem lesz.

Adjunk hatékony algoritmust annak eldöntésére, hogy megengedett lépések sorozatával elérhető-e az az állapot, amikor a két kisebb kancsó valamelyikében 2 liter víz van! Ha ez az állapot elérhető, az algoritmus adja meg az ehhez szükséges lépések minimális számát is!

Megoldás.
Definiáljunk egy $G = (V, E)$ irányított gráfot a következőképpen. A gráf csúcsai legyenek a lehetséges állapotok (melyik kancsóban mennyi víz van). Ezek egész számok olyan (a_1, a_2, a_3) rendezett hármasai, amelyekre

$$0 \leqslant a_1 \leqslant 4,$$
$$0 \leqslant a_2 \leqslant 7,$$
$$0 \leqslant a_3 \leqslant 10,$$

továbbá

$$a_1 + a_2 + a_3 = 11.$$

Az (a_1, a_2, a_3) csúcsból akkor vezessen egy irányított él az (a'_1, a'_2, a'_3) csúcsba, ha az első állapotból a második egy megengedett lépéssel elérhető. Ehhez két dolognak kell teljesülni:

- a két rendezett hármas pontosan két koordinátában különbözik (a harmadikban megegyeznek),
- ha i és j a két koordináta, amelyekben a rendezett hármasok különböznek, akkor vagy $a'_i = 0$, vagy $a'_j = 0$, vagy $a'_i = S_i$, vagy $a'_j = S_j$, ahol $S_1 = 4$, $S_2 = 7$, $S_3 = 10$ a kancsók űrtartalma.

A feladat annak eldöntése, hogy a $(4, 7, 0)$ csúcsból vezet-e út a $(2, *, *)$ vagy a $(*, 2, *)$ csúcsok valamelyikébe, ahol $*$ tetszőleges értéket jelöl.

Futtassuk le G-re a szélességi keresés algoritmust a $(4, 7, 0)$ kezdőcsúcsból. Ha a keresett csúcsok közül valamelyik elérhető, azt az algoritmus meg fogja találni, az ahhoz vezető legrövidebb úttal együtt. (Jegyezzük meg, hogy nem szükséges az egész gráfot előre megszerkeszteni, elég mindig annak a csúcsnak a szomszédait előállítani, amelynél éppen tartunk.) A $(2, 7, 2)$ állapot hat lépéssel elérhető: $(4, 7, 0) \to (0, 7, 4) \to (0, 1, 10) \to (4, 1, 6) \to (0, 5, 6) \to (4, 5, 2) \to (2, 7, 2)$.

Kannibálok és misszionáriusok

Feladat.
Egy folyó egyik partján 3 misszionárius és 3 kannibál áll, és át szeretnének jutni a másik partra. Ehhez rendelkezésükre áll egy kétszemélyes csónak,

egy pár evezővel (így egy ember is át tud kelni vele). A kannibáloknak sajnos van egy vad szokásuk: ha valamelyik parton többségbe kerülnek, akkor ott megeszik a misszionáriusokat (az érkező csónakban ülő emberek már az adott parton lévőkhöz számítanak). Adjunk hatékony algoritmust annak eldöntésére, hogy át tud-e jutni mindenki! Igenlő válasz esetén egy legkevesebb fordulóból álló megoldást is javasoljon az algoritmus!

Megoldás.
Az általánosság megszorítása nélkül feltehető, hogy a jobb partról a bal partra akar átjutni a társaság. Definiáljunk egy $G = (V, E)$ irányított gráfot a következőképpen. A gráf csúcsai legyenek a lehetséges állapotok (melyik parton hány kannibál és hány misszionárius van, illetve hol van a csónak). Ezek olyan $(k, m, p) \in \{0, 1, 2, 3\} \times \{0, 1, 2, 3\} \times \{B, J\}$ rendezett hármasok, ahol k és m a jobb parton lévő kannibálok, illetve misszionáriusok száma, p a csónak helye, továbbá $k = m$ ha $m \notin \{0, 3\}$. Ez utóbbi azt fejezi ki, hogy ha a misszionáriusok nincsenek mindhárman ugyanazon a parton, akkor egyik parton sem lehet több kannibál, mint misszionárius.

A (k, m, p) csúcsból akkor vezessen egy irányított él a (k', m', p') csúcsba, ha az első állapotból a második egy megengedett lépéssel elérhető. Ehhez a következőnek kell teljesülni:

- ha $p = J$, akkor $p' = B$ és

$$\begin{aligned} &k' = k - 1, m' = m \text{ vagy} \\ &k' = k, m' = m - 1 \text{ vagy} \\ &k' = k - 1, m' = m - 1 \text{ vagy} \\ &k' = k - 2, m' = m \text{ vagy} \\ &k' = k, m' = m - 2, \end{aligned}$$

- ha $p = B$, akkor $p' = J$ és

$$\begin{aligned} &k' = k + 1, m' = m \text{ vagy} \\ &k' = k, m' = m + 1 \text{ vagy} \\ &k' = k + 1, m' = m + 1 \text{ vagy} \\ &k' = k + 2, m' = m \text{ vagy} \\ &k' = k, m' = m + 2. \end{aligned}$$

A feladat annak eldöntése, hogy a $(3, 3, J)$ csúcsból vezet-e út a $(0, 0, B)$ csúcsba.

Futtassuk le G-re a szélességi keresés algoritmust a $(3, 3, J)$ kezdőcsúcsból. Ha a $(0, 0, B)$ csúcs elérhető, azt az algoritmus meg fogja találni, az ahhoz vezető legrövidebb úttal együtt. (Jegyezzük meg, hogy nem szükséges az egész gráfot előre megszerkeszteni, elég mindig annak a csúcsnak a szomszédait előállítani, amelynél éppen tartunk.) A $(0, 0, B)$ állapot tizenegy lépéssel elérhető: $(3, 3, J) \to (1, 3, B) \to (2, 3, J) \to (0, 3, B) \to (1, 3, J) \to (1, 1, B) \to (2, 2, J) \to (2, 0, B) \to (3, 0, J) \to (1, 0, B) \to (2, 0, J) \to (0, 0, B)$.

Páros gráfok

Feladat.
Adjunk hatékony algoritmust annak eldöntésére, hogy egy irányítatlan összefüggő gráf páros vagy nem!

Megoldás.
Legyen $G = (V, E)$ egy irányítatlan gráf és $s \in V$. Futtassuk le G-re a szélességi keresés algoritmust az s kezdőcsúcsból, majd ellenőrizzük minden $(u, v) \in E$ élre, hogy $d[u]$ és $d[v]$ paritása különböző-e. Ha igen, akkor a gráf páros;

$$V_1 = \{v \in V \mid d[v] \text{ páros}\},$$
$$V_2 = \{v \in V \mid d[v] \text{ páratlan}\}$$

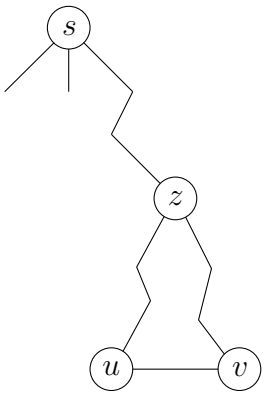
a csúcsok egy megfelelő partíciója.

Mit mondhatunk, ha valamely $(u, v) \in E$ élre $d[u]$ és $d[v]$ paritása megegyezik? Jegyezzük meg, hogy tetszőleges $(u, v) \in E$ élre $|d[u] - d[v]| \leqslant 1$, így ebben az esetben $d[u] = d[v]$. Jelölje a közös értéket i.

Legyen z az u és v csúcsok legközelebbi közös őse a szélességi fában, és legyen $d[z] = j$. Világos, hogy $j < i$. Tekintsük a G gráfnak azt a C körét, amely a szélességi fa z-ből u-ba vezető útjából, az (u, v) élből és a szélességi fa v-ből z-be vezető útjából áll. A C kör hossza

$$(i - j) + 1 + (i - j) = 2(i - j) + 1,$$

ami páratlan szám. Ám egy páratlan hosszúságú kör csúcsai semmiképp nem oszthatók két részre úgy, hogy élek csak különböző részekbe osztott csúcsok között menjenek, ezért ebben az esetben a gráf nem lehet páros.



Az algoritmus költsége $O(|V| + |E|)$.

Páros részgráfok

Feladat.
Mutassuk meg, hogy bármely gráf csúcsai két (diszjunkt) osztályba sorolhatók úgy, hogy a gráf éleinek legalább a fele különböző osztálybeli csúcsokat kössön össze!

Megoldás.
Az állítást a csúcsok száma szerinti teljes indukcióval bizonyítjuk. Ha a gráfnak csak egy vagy két csúcsa van, akkor az állítás nyilván igaz. Most tegyük fel, hogy minden legfeljebb $n-1$ csúcsú gráfra teljesül az állítás, és tekintsünk egy n csúcsú G gráfot. Hagyjuk el G egy tetszőleges x csúcsát a belőle kiinduló élekkel együtt. A kapott gráf legyen G'. Az indukciós feltevés szerint a G' gráf csúcsai két (diszjunkt) osztályba sorolhatók úgy, hogy G' éleinek legalább a fele különböző osztálybeli csúcsokat kössön össze. Az x csúcsot a két osztály közül ahhoz sorolva, amelyikben kevesebb szomszédja van G-ben (ha mindkét osztályban ugyanannyi a szomszédok száma, akkor akármelyik osztályt választhatjuk), a G gráf csúcsinak egy megfelelő felosztását kapjuk.

A bizonyítás egyben eljárást is ad a két osztály meghatározására. Kezdetben legyen mindkét osztály üres. Ezután vegyük sorra a gráf csúcsait, és az éppen következő csúcsot mindig ahhoz az osztályhoz soroljuk, amelyikben az adott pillanatban kevesebb szomszédja van (ha mindkét osztályban ugyanannyi a szomszédok száma, akkor akármelyik osztályt választhatjuk).

Pillangók

Feladat.
Néhány amatőr lepkegyűjtő egy gyűjtőútról n pillangóval tér vissza. Meg vannak győződve róla, hogy a pillangók két fajhoz tartoznak, de mivel a két faj nagyon hasonló, elég nehéz feladatnak tűnik direkt módon besorolni a pillangókat. Ezért a következő módszerhez folyamodnak. Sorba veszik az összes pillangópárt, tüzetesen megvizsgálják őket, és eldöntik, hogy azonos vagy különböző fajhoz tartoznak-e. Előfordul, hogy nagyon bizonytalanok a döntésben, ekkor nyitva hagyhatják az azonos fajhoz tartozás kérdését. Tegyük fel, hogy m esetben sikerült eldönteni, hogy a pillangók azonos vagy különböző fajhoz tartoznak-e.

Adjunk $O(m+n)$ költségű algoritmust, amely ellenőrzi, hogy a döntések konzisztensek-e!

Megoldás.
Definiáljunk egy $G = (V, E)$ élszínezett, irányítatlan gráfot a következőképpen. A gráf csúcsai feleljenek meg a befogott pillangóknak. Két csúcsot akkor kössön össze él, ha a csúcsoknak megfelelő pillangókról sikerült eldönteni, hogy azonos vagy különböző fajhoz tartoznak-e. Egy él színe legyen zöld, ha a végpontjainak megfelelő pillangók azonos fajhoz lettek sorolva, piros pedig ha különbözőkhöz. A döntések akkor konzisztensek, ha a csúcsokat be tudjuk osztani két A és B halmazba úgy, hogy minden zöld él azonos halmazba beosztott csúcsokat, míg minden piros él különböző halmazba beosztott csúcsokat köt össze.

A G gráf nem feltétlenül összefüggő. Jelölje G összefüggő komponenseit $G_1, G_2, \ldots, G_k$, és legyenek $s_1, s_2, \ldots, s_k$ rendre ezeknek az összefüggő komponenseknek egy-egy csúcsa. Az $s_1, s_2, \ldots, s_k$ csúcsokat soroljuk az A halmazhoz.

Minden $1 \leqslant i \leqslant k$ esetén futtassuk le a G_i összefüggő komponensre a szélességi keresés algoritmust az s_i kezdőpontból. Amikor egy még felderítetlen v csúcsot elérünk egy már elért u csúcs szomszédsági listájának vizsgálata során, osszuk be v-t ugyanabba a halmazba, mint ahol u van, ha az (u, v) él zöld, illetve a másik halmazba, mint ahol u van, ha az (u, v) él piros. Ezután vegyük sorba G éleit, és ellenőrizzük, hogy minden zöld él azonos halmazba beosztott csúcsokat, míg minden piros él különböző halmazba beosztott csúcsokat köt össze. Ha igen, akkor a döntések nyilván konzisztensek.

Ha nem, akkor viszont a döntések nem lehetnek konzisztensek. Ez azon múlik, hogy ha G csúcsai beoszthatók két A és B halmazba úgy, hogy minden zöld él azonos halmazba beosztott csúcsokat, míg minden piros él különböző halmazba beosztott csúcsokat köt össze, akkor egy ilyen beosztás lényegében

megegyezik a fenti eljárás által szolgáltatott beosztással (amelyet a szélességi feszítőfák éleinek vizsgálatára alapozva hoztunk). Annyi eltérés lehet csupán a két beosztás között, hogy egy-egy összefüggő komponensben A és B szerepet cserél, vagyis az adott komponononens egyik beosztásban A-hoz tartozó csúcsai a másikban B-hez tartoznak, és fordítva. Azonban ez az eltérés a döntések konzisztens voltát nem befolyásolja.

Az eljárás költsége $O(|E| + |V|) = O(m + n)$.

Legrövidebb kör

Feladat.
Legyen $G = (V, E)$ egy szomszédsági listákkal adott irányítatlan gráf.

(A) Adjunk hatékony algoritmust a gráf egy legrövidebb, azaz legkevesebb élből álló, egy adott $s \in V$ csúcson átmenő körének megkeresésére!

(B) Adjunk hatékony algoritmust a gráf egy legrövidebb, azaz legkevesebb élből álló körének megkeresésére!

Megoldás.
(A) Futtassuk le G-re a szélességi keresés algoritmust az s kezdőpontból. Legyenek $s_1, s_2, \ldots, s_k$ az s szomszédai. Ezután keressük meg a szélességi feszítőfa különböző s_i és s_j gyökerű részfái között futó G-beli élek közül azt az (u, v) élt, amelyre $d[u] + d[v]$ minimális. Ekkor (u, v) a végpontjait összekötő szélességi feszítőfa élekkel együtt egy megfelelő kört ad. Az (u, v) él megtalálásában segít, ha a bejárás során a csúcsok mellé feljegyezzük, hogy melyik s_i gyökerű részfába tartoznak. Ezt a jellemzőt a csúcsok a szülő csúcsaiktól öröklik. Az eljárás költsége $O(|E| + |V|)$.

Az eljárás helyessége a következőképpen látható be. Tetszőleges s-en átmenő kör (ha van ilyen), szükségképpen tartalmaz olyan (w, z) élt, amely a szélességi feszítőfa különböző s_i és s_j gyökerű részfái között fut. A kör s és w közötti szakaszának hossza legalább $d[w]$, míg z és s közötti szakaszának hossza legalább $d[z]$. Így a kör hossza legalább $d[w] + d[z] + 1$. Ebből következik, hogy az eljárás által megadott kör minimális hosszúságú.

(B) Futtassuk le a fenti algoritmust a gráf összes csúcsára, és a kapott legrövidebb körök közül válasszuk ki a legkevesebb élből állót. Az eljárás költsége $O(|V|(|E| + |V|)) = O(|V||E| + |V|^2)$.

Utak távoli csúcsok között

Feladat.
Legyen $G = (V, E)$ irányítatlan, összefüggő gráf és $s, t \in V$. Mutassuk meg, hogy ha s és t távolsága nagyobb, mint $|V|/2$, akkor van a gráfban olyan s-től és t-től különböző v csúcs, amelyen minden s-et t-vel összekötő út átmegy!

Megoldás.
Futtassuk le G-re a szélességi keresés algoritmust az s kezdőpontból. Jelölje $L_0, L_1, L_2, \ldots$ a gráf azon csúcsainak halmazát, amelyek s-től rendre $0, 1, 2, \ldots$ távolságra vannak. Nyilván $L_0 = \{s\}$. Tegyük fel, hogy $t \in L_k$. Tudjuk, hogy $k > |V|/2$.

Állítjuk, hogy az $L_1, L_2, \ldots, L_{k-1}$ halmazok között van olyan, amelyik egyetlen csúcsból áll. Valóban, ha az $L_1, L_2, \ldots, L_{k-1}$ halmazok mindegyike legalább két elemű lenne, akkor azt kapnánk, hogy

$$|V| \geqslant |L_0| + |L_1| + \cdots + |L_{k-1}| + |L_k| \geqslant 1 + 2(k-1) + 1 = 2k > |V|,$$

ami ellentmondás. Így van olyan L_i halmaz, amely egyetlen elemből áll, legyen ez a v csúcs.

Megmutatjuk, hogy v-n az összes s-et t-vel összekötő út átmegy. Tekintsük az $L = L_0 \cup L_1 \cup \cdots \cup L_{i-1}$ halmazt. Most $s \in L$ és $t \notin L$, így bármely s-et t-vel összekötő útnak ki kell lépni valahol L-ből. Ám vegyük észre, hogy bármely olyan él, amelynek egyik végpontja L-ben, a másik végpontja pedig L-en kívül van, szükségképpen L_{i-1} és L_i egy-egy csúcsát köti össze, hiszen tetszőleges $(u, w) \in E$ élre $|d[u] - d[w]| \leqslant 1$. Így bármely s-et t-vel összekötő útnak át kell menni egy L_i-beli csúcson, és mivel L_i egyetlen elemből áll, ezért szükségképpen a v-n.

Mélységi feszítő erdő

Feladat. Legyen $G = (V, E)$ egy irányított gráf, és futtassuk a mélységi bejárás algoritmust ezen a gráfon:

```
MélységiBejárás(G)
for V(G) minden u csúcsára do
  szín[u]=fehér
  szülő[u]=nil
  s[u]=0
  f[u]=0
idő=0
```

```
for V(G) minden u csúcsára do
  if szín[u]=fehér then MB(u)

MB(u)
idő=idő+1
szín[u]=szürke
s[u]=idő
for A[u] minden v csúcsára do
    if szín[v]=fehér then
      szülő[v]=u
      MB(v)
idő=idő+1
szín[u]=fekete
f[u]=idő
```

Tekintsük az algoritmus által létrehozott mélységi feszítő erdőt, amely G csúcsaiból valamint a $(\texttt{szülő}[v], v)$ élekből áll. Tetszőleges $x \in V$ csúcsra jelölje T_x a feszítő erdő x gyökerű részfájának csúcshalmazát, U_x azon $y \in V$ csúcsok halmazát, amelyekre $s[x] \leqslant s[y]$ és $f[y] \leqslant s[x]$, végül S_x azon $y \in V$ csúcsok halmazát, amelyek elérhetők x-ből olyan $(u, v) \in E$ élek mentén, melyek végpontjaira $s[u] \geqslant s[x]$ és $s[v] \geqslant s[x]$. Mutassuk meg, hogy tetszőleges $x \in V$ csúcsra

(A) $T_x = U_x$,

(B) $T_x = S_x$.

Megoldás.
(A) Elég belátni a következőt.

Állítás. Legyen u és v a gráf két tetszőleges csúcsa.

(1) Ha a v csúcs leszármazottja az u csúcsnak a mélységi feszítő erdőben, akkor $s[u] \leqslant s[v] < f[v] \leqslant f[u]$.

(2) Ha az u csúcs leszármazottja a v csúcsnak a mélységi feszítő erdőben, akkor $s[v] \leqslant s[u] < f[u] \leqslant f[v]$.

(3) Ha sem a v csúcs nem leszármazottja az u csúcsnak, sem az u csúcs nem leszármazottja a v csúcsnak a mélyégi feszítő erdőben, akkor $s[u] < f[u] < s[v] < f[v]$ vagy $s[v] < f[v] < s[u] < f[u]$.

Bizonyítás.
(1) Az állítást a mélységi feszítő erdő u-ból v-be vezető útjának hosszára vonatkozó teljes indukcióval bizonyítjuk. Ha $u = v$, akkor $s[u] \leqslant s[v] < f[v] \leqslant f[u]$ nyilvánvalóan teljesül.

Az indukciós lépésben legyen w a mélységi feszítő erdő u-ból v-be vezető útjának utolsó előtti csúcsa. Egyrészt az indukciós feltevés szerint $s[u] \leqslant s[w] < f[w] \leqslant f[u]$. Másrészt $s[w] \leqslant s[v] < f[v] \leqslant f[w]$, hiszen az $\texttt{MB}(v)$ eljárást az $\texttt{MB}(w)$ eljárás végrehajtása során hívjuk meg. Az utóbbi egyenlőtlenségláncot az előbbibe illesztve $s[u] \leqslant s[v] < f[v] \leqslant f[u]$ adódik.

(2) Ugyanúgy, mint az előbb.

(3) Először azt az esetet tekintjük, amikor u és v a mélységi feszítő erdő különböző komponenseiben vannak. Legyen u', illetve v' az u, illetve a v csúcsot tartalmazó komponens gyökere. Most egyrészt $s[u'] < f[u'] < s[v'] < f[v']$ vagy $s[v'] < f[v'] < s[u'] < f[u']$, hiszen a $\texttt{MélységiBejárás}(G)$ eljárás végrehajtása során vagy az $\texttt{MB}(u')$ eljárás végrehajtásának befejezése után hívjuk meg az $\texttt{MB}(v')$ eljárást vagy az $\texttt{MB}(v')$ eljárás végrehajtásának befejezése után hívjuk meg az $\texttt{MB}(u')$ eljárást. Másrészt $s[u'] \leqslant s[u] < f[u] \leqslant f[u']$ és $s[v'] \leqslant s[v] < f[v] \leqslant f[v']$, hiszen u leszármazottja u'-nek és v leszármazottja v'-nek a mélységi feszítő erdőben. Az utóbbi egyenlőtlenségláncokat az előbbiekbe illesztve

$$s[u'] \leqslant s[u] < f[u] \leqslant f[u'] < s[v'] \leqslant s[v] < f[v] \leqslant f[v'],$$

illetve

$$s[v'] \leqslant s[v] < f[v] \leqslant f[v'] < s[u'] \leqslant s[u] < f[u] \leqslant f[u'],$$

ahonnan $s[u] < f[u] < s[v] < f[v]$, illetve $s[v] < f[v] < s[u] < f[u]$ adódik.

Tekintsük ezek után azt az esetet, amikor u és v a mélységi feszítő erdő egy komponensében vannak. Feltételünk szerint a közös komponens gyökeréből az u, illetve a v csúcsba vezető út nem tartalmazza a v, illetve az u csúcsot. Legyen w a mélységi feszítő erdőben a közös komponens gyökeréből az u, illetve v csúcsokba vezető utak utolsó (a gyökértől legtávolabbi) közös csúcsa, u', illetve v' pedig a fenti utak w után következő első csúcsa. Most egyrészt $s[u'] < f[u'] < s[v'] < f[v']$ vagy $s[v'] < f[v'] < s[u'] < f[u']$, hiszen az $\texttt{MB}(w)$ eljárás végrehajtása során vagy az $\texttt{MB}(u')$ eljárás végrehajtásának befejezése után hívjuk meg az $\texttt{MB}(v')$ eljárást, vagy az $\texttt{MB}(v')$ eljárás végrehajtásának befejezése után hívjuk meg az $\texttt{MB}(u')$ eljárást. Másrészt $s[u'] \leqslant s[u] < f[u] \leqslant f[u']$ és $s[v'] \leqslant s[v] < f[v] \leqslant f[v']$, hiszen u leszármazottja u'-nek és v leszármazottja v'-nek a mélységi feszítő erdőben. Az utóbbi egyenlőtlenségláncokat az előbbiekbe illesztve

$$s[u'] \leqslant s[u] < f[u] \leqslant f[u'] < s[v'] \leqslant s[v] < f[v] \leqslant f[v']$$

illetve

$$s[v'] \leqslant s[v] < f[v] \leqslant f[v'] < s[u'] \leqslant s[u] < f[u] \leqslant f[u'],$$

ahonnan $s[u] < f[u] < s[v] < f[v]$, illetve $s[v] < f[v] < s[u] < f[u]$ adódik.

(B) A T_x halmaz azokból a csúcsokból áll, amelyek x-ből a mélységi feszítő erdő élei mentén elérhetők. Az x gyökerű részfa csúcsait x után érjük el a mélységi bejárás során, ezért a részfa bármely (u, v) élére $s[u] \geqslant s[x]$ és $s[v] \geqslant s[x]$. Így $T_x \subseteq S_x$.

A fordított irányú tartalmazás igazolásához indirekt tegyük fel, hogy létezik olyan $y \in S_x$ csúcs, amely nincs T_x-ben, vagyis nem leszármazottja x-nek a mélységi feszítő erdőben. Tekintsünk egy, az S_x meghatározásában szereplő x-ből y-ba menő utat. Az általánosság megszorítása nélkül feltehetjük, hogy ennek az útnak az y-tól különböző csúcsai mind T_x-ben vannak; ha ez nem teljesül, válasszuk y-nak az út x-hez legközelebbi, nem T_x-beli csúcsát. Jelölje ennek az útnak az utolsó előtti csúcsát v. Nyilván $v \in T_x$. Az $y \in S_x$ feltétel szerint $s[x] < s[y]$, ami az indirekt feltevéssel és az előző ponttal összhangban csak úgy lehetséges, hogy $s[x] < f[x] < s[y] < f[y]$. Ennélfogva y-t valamikor T_x csúcsai után látogatjuk meg, speciálisan $s[v] < s[y]$. A (v, y) élt biztos megvizsgáljuk mielőtt az $\texttt{MB}(v)$ eljárás befejeződne, ezért $s[y] < f[v]$. Ez viszont ismét az előző pont szerint csak úgy lehetséges, hogy $s[v] < s[y] < f[y] < f[v]$, következésképpen y leszármazottja v-nek a mélységi feszítő erdőben. Mindkét esetben $y \in T_x$ adódik, ellentmondva feltevésünknek. Így $S_x \subseteq T_x$ is igaz.

Erősen összefüggő komponensek

Feladat.
Azt mondjuk, hogy egy $G = (V, E)$ gráf erősen összefüggő, ha bármely $u, v \in V$ csúcsokra a gráfban vezet v-ből u-ba is és u-ból v-be is irányított út. Egy irányított gráf egy maximális erősen összefüggő részgráfját a gráf egy erősen összefüggő komponensének nevezzük. Más szavakkal, a G gráf egy H részgráfja a G egy erősen összefüggő komponense, ha H erősen összefüggő, továbbá G-nek nem létezik olyan H' erősen összefüggő részgráfja, amelynek H valódi részgráfja lenne. Adjunk lineáris költségű algoritmust a mélységi bejárás segítségével egy szomszédsági listákkal adott $G = (V, E)$ irányított gráf erősen összefüggő komponenseinek feltérképezésére!

Megoldás.
Végezzük el a G gráf egy mélységi bejárását, közben minden csúcsnak sorszámul adva az elhagyási idejét. Készítsük el ezután azt a $G' = (V, E')$ irányított gráfot, amelyet úgy kapunk G-ből, hogy minden él irányítását megfordítjuk,

azaz $(u, v) \in E'$ pontosan akkor áll fenn, ha $(v, u) \in E$. Végül végezzük el a G' gráf egy mélységi bejárását az első mélységi bejárás során legnagyobb elhagyási idejű csúcsból kiindulva; új kezdőcsúcsnak mindig az első mélységi bejárás során legnagyobb elhagyási idejű csúcsot válasszuk a maradékból. A G gráf erősen összfüggő komponenseinek csúcshalmazai az így kapott mélységi feszítőerdő fáinak csúcshalmazai lesznek.

Az első mélységi bejárás költsége $O(|V| + |E|)$. A G' gráf szomszédsági listáit a következőképpen készíthetjük el. Kezdetben legyen G' összes szomszédsági listája üres. Ezután végigmenve a G-beli szomszédsági listákon, ha egy v csúcsnak szomszédja az u csúcs, akkor szúrjuk be a v csúcsot az u csúcs G'-beli szomszédsági listájának elejére. A költség itt is nyilván $O(|V| + |E|)$. Végül a második mélységi bejárás költsége $O(|V|+|E|)$; ne feledjük a csúcsok az első mélységi bejárás után az elhagyási idejük szerint rendezetten állnak rendelkezésre. Így a teljes költség $O(|V| + |E|)$.

Hátra van még az algoritmus helyességének belátása.

Állítás. A $G = (V, E)$ irányított gráf tetszőleges $u, v \in V$ csúcsaira teljesül, hogy u és v pontosan akkor tartozik G egy erősen összefüggő komponenséhez, ha a második mélységi bejárás után u és v a mélységi feszítőerdő egy fájában vannak.

Bizonyítás. Először azt mutatjuk meg, hogy G egy erősen összefüggő komponensének csúcsai a második mélységi bejárás után a mélységi feszítő erdőnek ugyanabban a fájában vannak. Legyenek u és v a G gráf egy erősen összefüggő komponensének csúcsai. Mivel G és G' erősen összefüggő komponenseinek csúcshalmazai ugyanazok, ezért u és v a G' gráfban is ugyanabban az erősen összefüggő komponensben vannak. Jelölje K ennek az erősen összefüggő komponensnek a csúcshalmazát, és legyen x az a K-beli csúcs, amelyet a második mélységi bejárás során a legkorábban érünk el. Ekkor az előző feladatban bevezetett jelöléssel $K \subseteq S_x$, amiből a mélységi feszítő erdő részfáinak jellemzését adó előző feladat felhasználásával következik, hogy $u, v \in T_x$, ennélfogva u és v a mélységi feszítőerdőnek ugyabban a fájában vannak.

Ezek után megmutatjuk, hogy a második mélységi bejárás után kapott mélységi feszítőerdő egy fájának csúcsai a G gráfnak ugyanabban az erősen összefüggő komponensében vannak. Legyenek u és v a második mélységi bejárás után kapott mélységi feszítőerdő egy fájának csúcsai, és legyen z ennek a fának a gyökere. Jelölje L' a fában a z csúcsból az u csúcsba vezető utat. Az L'-beli élek irányítását megfordítva világos módon egy, az u csúcsból a z csúcsba vezető G-beli L utat kapunk.

Belátjuk, hogy G-ben a z csúcsból az u csúcsba is vezet út. Legyen y az

L útnak az a csúcsa, amelyet az első mélységi bejárás során a legkorábban érünk el. A mélységi feszítő erdő részfáinak jellemzését adó előző feladat felhasználásával következik, hogy L-nek az y és z közötti csúcsai az első mélységi bejárásnál y leszármazottai lesznek, így közülük y elhagyási ideje a legnagyobb. Ugyanakkor idézzük fel, hogy a második mélységi bejárás során új kezdőcsúcsnak mindig az első mélységi bejárás során legnagyobb elhagyási idejű csúcsot választjuk a maradékból, így szükségképpen z elhagyási ideje a legnagyobb az L'-beli (és így az L-beli) csúcsok közül. Ez viszont csak úgy lehetséges, ha $y = z$.

Összefoglalva az eddigieket, az L út csúcsai között az első mélységi bejárásnál z a legkisebbb elérési és legnagyobb elhagyási idejű. Speciálisan $s[z] \leqslant s[u]$ és $f[u] \leqslant f[z]$, ami az előző feladat szerint azt jelenti, hogy u leszármazottja z-nek az első mélységi bejárásnál. Ennélfogva G-ben vezet út z-ből u-ba. Hasonlóan igazolható, hogy G-ben szintén vezet út z-ből v-be és v-ből z-be, így u-ból v-be és v-ből u-ba is.

Topológikus rendezés

Feladat.
Legyen $G = (V, E)$ tetszőleges irányított gráf, ahol $|V| = n$. A G gráf egy topológikus rendezése a csúcsainak egy olyan $v_1, v_2, \ldots, v_n$ sorrendje, melyben $(v_i, v_j) \in E$ esetén $i < j$. Ismert, hogy G-nek akkor és csak akkor van topológikus rendezése, ha nem tartalmaz irányított kört. Ez például a következőképpen látható be.

Ha G tartalmaz irányított kört, akkor nyilván nem lehet topológikus rendezése, mert egy irányított kör csúcsainak magától értetődő módon nincs a kívánalmaknak megfelelő sorrendje.

Ezek után tegyük fel, hogy G nem tartalmaz irányított kört. Először is megjegyezzük, hogy ekkor G-ben van olyan csúcs, amibe nem fut be él. Ha ugyanis minden csúcsba menne él, akkor egy tetszőleges csúcsból kiindulva lépkedni tudnánk visszafelé az irányított éleken anélkül, hogy valahol elakadnánk. E séta során egyszer szükségképpen körbeérnénk, ami ellentmond a körmentességnek.

Az állítást a csúcsok száma szerinti teljes indukcióval bizonyítjuk. Ha G-nek csak egyetlen csúcsa van, akkor az állítás nyilván igaz. Most tegyük fel, hogy minden legfeljebb $n-1$ csúcsú körmentes irányított gráf topológikusan rendezhető, és tekintsünk egy n csúcsú G körmentes irányított gráfot. G-ből hagyjunk el egy olyan x csúcsot a belőle kiinduló élekkel együtt, amibe nem fut be él. A kapott gráf legyen G'. Nyilván G' sem tartalmaz irányított kört, így az indukciós feltevés szerint G'-nek létezik egy $w_1, w_2, \ldots, w_{n-1}$ topoló-

gikus rendezése. Mármost az $x, w_1, w_2, \ldots, w_{n-1}$ sorrend világos módon a G gráf egy topológikus rendezése.

A bizonyításban ott lapul egy topológikus rendezésre szolgáló algoritmus: válasszunk elsőnek egy olyan csúcsot, amelybe nem fut él, majd a maradékból szintén egy ilyen csúcsot másodiknak, és így tovább. Írjuk meg az algoritmust!

Megoldás.
A `TopológikusRendezés(G)` algoritmusban feltesszük, hogy a gráfot szomszédsági listákkal ábrázoljuk (egy u csúcs szomszédsági listáját a szokásos módon $A[u]$ jelöli). Az algoritmus egy Q sor segítségével kezeli azokat a csúcsokat, amelyekbe már nem fut be él.

```
TopológikusRendezés(G)
for V(G) minden u csúcsára do
  be_fok[u]=0
for V(G) minden u csúcsára do
  for A[u] minden v csúcsára do
    be_fok[v]=be_fok[v]+1
SorInicializálás(Q)
for V(G) minden u csúcsára do
  if be_fok[u]=0 then Sorba(Q,u)
while Q<>EMPTYSET do
  u=Sorból(Q)
  output u
  for A[u] minden v csúcsára do
    be_fok[v]=be_fok[v]-1
    if be_fok[v]=0 then Sorba(Q,v)
for V(G) minden u csúcsára do
  if be_fok[u]>0 then
    print "nincs topológikus rendezés"
```

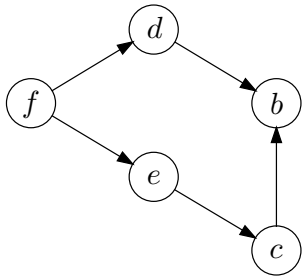
A `TopológikusRendezés(G)` algoritmus lineáris költségű, mert

- a kezdeti értékadás költsége $O(|V| + |E|)$,
- a sor inicializálása utáni for ciklus költsége $O(|V|)$,
- minden csúcsot legfeljebb egyszer teszünk be a sorba, így a while ciklusban a csúcsokhoz tartozó szomszédsági listákat legfeljebb egyszer olvassuk végig, amikor is a csúcsot kivesszük a sorból; ezért a while ciklus költsége szintén $O(|V| + |E|)$,

- az utolsó for ciklus költsége $O(|V|)$, igazából itt csak azt ellenőrizzük, hogy G tényleg körmentes volt-e.

Különböző topológikus rendezések

Feladat.
Hány topológikus rendezése van a következő gráfnak?



Megoldás.
A gráf topológikus rendezésében az első csúcs csak olyan lehet, amelybe nem érkezik be él. Ezért az első csúcs mindenképpen az f. A gráf topológikus rendezésében az utolsó csúcs csak olyan lehet, amelyből nem indul ki él. Ezért az utolsó csúcs mindenképpen a b. Hátra van még annak meghatározása, hogy d, e és c milyen sorrendben követhetik egymást a topológikus rendezés középső részén. Az (e, c) irányított él jelenléte miatt e-nek meg kell előzni c-t. Azonban d akárhol lehet e-hez és c-hez képest: mindkettőt megelőzheti, lehet a kettő között, jöhet mindkettő után.

Mindezek alapján három topológikus rendezése van a gráfnak:

$$f, d, e, c, b$$
$$f, e, d, c, b$$
$$f, e, c, d, b$$

Helytörténeti kutatás

Feladat.
Egy falusi iskola történelemtanára azzal a feladattal bízza meg a gyerekeket, hogy a falu hajdan élt lakóiról gyűjtsenek információt az öregektől. Az öregekkel folytatott beszélgetések során a gyerekeknek n emberről sikerül megtudni valamit. Jelölje ezeket az embereket $P_1, P_2, \ldots, P_n$.

A történelemtanár számára különösen érdekesek a következő típusú információk:

- valamilyen i és j indexekre a P_i ember előbb meghalt, mint P_j megszületett volna,
- valamilyen i és j indexekre a P_i és P_j emberek életének volt közös szakasza.

Mivel az emlékek idővel elhalványodnak, nem biztos, hogy ezek az információk mind helyesek. A történelemtanár ezért szeretné tudni, hogy az információk legalább konzisztensek-e. Adjunk hatékony algoritmust ennek eldöntésére!

Megoldás.
Definiáljunk egy G irányított gráfot a következőképpen. Minden $1 \leqslant i \leqslant n$ esetén a P_i embernek feleljen meg két csúcs, jelölje ezeket b_i és d_i, amelyek a P_i ember születésének és halálának (ismeretlen) időpontját reprezentálják. Minden $1 \leqslant i \leqslant n$ esetén vezessen irányított él a b_i csúcsból a d_i csúcsba. Ha tudjuk, hogy a P_i ember előbb meghalt, mint P_j megszületett volna, akkor vezessen irányított él a d_i csúcsból a b_j csúcsba. Másrészt ha tudjuk, hogy a P_i és P_j emberek életének volt közös szakasza, akkor vezessen egy-egy irányított él a b_i csúcsból a d_j csúcsba, illetve a b_j csúcsból a d_i csúcsba.

Ha G tartalmaz irányított kört, akkor az információk nyilván nem konzisztensek; a körön levő események közül nem tudunk kijelölni egy legkorábbit.

Ha G nem tartalmaz irányított kört, akkor viszont az információk konzisztensek; G topológikus rendezése megadja a születések és halálozások egy lehetséges sorrendjét.

Euler séta

Feladat.
A $G = (V, E)$ összefüggő, irányított gráf éleinek egy $(e_1, e_2, \ldots, e_m)$ permutációját Euler sétának nevezzük, ha minden $1 \leqslant i \leqslant m-1$ esetén az e_{i+1} él kezdőpontja megegyezik az e_i él végpontjával. Ha ezen kívül még az is teljesül, hogy az e_1 él kezdőpontja megegyezik az e_m él végpontjával, akkor zárt Euler sétáról beszélünk.

(A) Mutassuk meg, hogy G-ben akkor és csak akkor létezik zárt Euler séta ha be-fok(v) = ki-fok(v) minden $v \in V$ csúcsra!

(B) Tervezzünk $O(m)$ költségű algoritmust, amely G-ben megad egy zárt Euler sétát, ha létezik!

Megoldás.
(A) Először megmutatjuk, hogy ha a gráfban van zárt Euler séta, akkor be-fok(v) = ki-fok(v) minden $v \in V$ csúcsra. Induljunk el a gráf egy tetszőleges csúcsából, és járjuk be az éleket a zárt Euler séta mentén. Ekkor nyilvánvaló módon minden csúcsba annyiszor "megyünk be", ahányszor "kijövünk" onnan, így minden $v \in V$ csúcsra be-fok(v) = ki-fok(v).

Ezután belátjuk, hogy ha be-fok(v) = ki-fok(v) minden $v \in V$ csúcsra, akkor a gráfban van zárt Euler séta. Induljunk el a gráf egy tetszőleges csúcsából, és haladjunk az élek mentén ügyelve arra, hogy semelyik élen ne menjünk át kétszer. Ha olyan csúcsba érünk, amelyből már nem vezet ki eddig be nem járt él, akkor az csak a kiinduló csúcs lehet, mivel minden csúcs be-foka megegyezik a ki-fokával. Ilyen módon egy H zárt sétát kapunk (élek egy olyan ismétlődés nélküli $(e_{i_1}, e_{i_2}, \ldots, e_{i_k})$ sorozatát nevezzük zárt sétának, ahol minden $1 \leqslant j \leqslant k-1$ esetén az e_{i_j+1} él kezdőpontja megegyezik az e_{i_j} él végpontjával, továbbá az e_{i_1} él kezdőpontja megegyezik az e_{i_k} él végpontjával).

Ha H egy Euler séta, akkor kész vagyunk. Ellenkező esetben G összefüggősége miatt kell lenni olyan v csúcsnak H-n, amelyre illeszkedik G-nek még be nem járt éle. Mivel a G gráfból ha elhagyjuk a H-beli éleket, továbbra is teljesül, hogy bármely csúcs be-foka ugyanannyi mint a ki-foka, ezért v-ből szükségképpen kiindul legalább egy be nem járt él. Induljunk el a v csúcsból, és haladjunk az eddig be nem járt élek mentén, ügyelve arra, hogy továbbra se menjünk át kétszer egyik élen sem. Ha olyan csúcsba érünk, amelyből már nem vezet ki eddig be nem járt él, akkor az nyilván csak a v csúcs lehet. Jelölje az így kapott új zárt sétát H'. Ha most elindulunk a v csúcsból és először H-t, majd H'-t járjuk be, egy H-nál hosszabb zárt sétát kapunk. Folytatva ezt az eljárást, mivel fogynak a még be nem járt élek, előbb-utóbb G egy zárt Euler sétájához jutunk.

(B) A `ZártEulerSéta(G)` algoritmusban feltesszük, hogy a G gráfot szomszédsági listákkal ábrázoljuk (egy u csúcs szomszédsági listáját a szokásos módon $A[u]$ jelöli).

Az algoritmus a gráf éleinek egy mindkét irányban láncolt T listáját konstruálja meg, amelyen végighaladva épp egy zárt Euler sétát kapunk. Az algoritmus minden iterációban egy adott csúcsból induló zárt sétát épít, amelynek éleit egy mindkét irányban láncolt C listában tárolja. Az iteráció befejeztével az algoritmus a C listát befűzi T-be egy olyan él elé, amelynek kezdőpontja a kiindulási csúcs.

Használunk még egy L sort, amelynek elemei (csúcs, mutató) párok, ahol a mutató egy olyan élre mutat a T listában (illetve a C listában), amelynek kezdőpontja az adott csúcs.

```
ZártEulerSéta(G)
LáncoltListaInicializálás(T)
SorInicializálás(L)
legyen z tetszőleges csúcs G-ben
Sorba(L,(z,nil))
while L<>EMPTYSET do
  (v,hely(T,v))=Sorból(L)
  C=AdottCsúcsbólZártSéta(v)
  if hely(T,v)=nil
    then T=C
    else ListábaBefűz(T,hely(T,v),C)
return T

AdottCsúcsbólIndulóZártSéta(v)
LáncoltListaInicializálás(C)
u=v
while ki_fok(u)>0 do
  w=ListaElejérőlTöröl(A[u])
  ki_fok(u)=ki_fok(u)-1
  ListaVégéreBeszúr(C,(u,w))
  if ki_fok(u)>0 then Sorba(L,(u,hely(C,u)))
  u=w
return C
```

Az `AdottCsúcsbólIndulóZártSéta(v)` eljárás while ciklusának minden iterációjában törlünk egy csúcsot valamelyik szomszédsági listából, ezért az `AdottCsúcsbólIndulóZártSéta(v)` eljáráshívások összköltsége $O(|E|)$. Vegyük észre, hogy ez felső korlát a Sorba(L) műveletek számára is.

A `ZártEulerSéta(G)` algoritmusban a while ciklus iterációinak száma megegyezik a Sorból(L) műveletek számával, ami nyiván ugyanannyi, mint a Sorba(L) műveletek száma, vagyis $O(|E|)$. Mivel a T és a C listák mindkét irányban láncoltak, ezért C befűzése T-be konstans költséggel megvalósítható. Ez azt jelenti, hogy a `ZártEulerSéta(G)` algoritmus költsége az `AdottCsúcsbólIndulóZártSéta(v)` eljáráshívások költségét nem számolva $O(|E|)$.

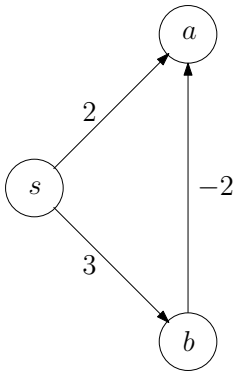
Mindent összevetve az algoritmus összköltsége $O(|E|)$.

Dijkstra algoritmus negatív élsúlyokkal

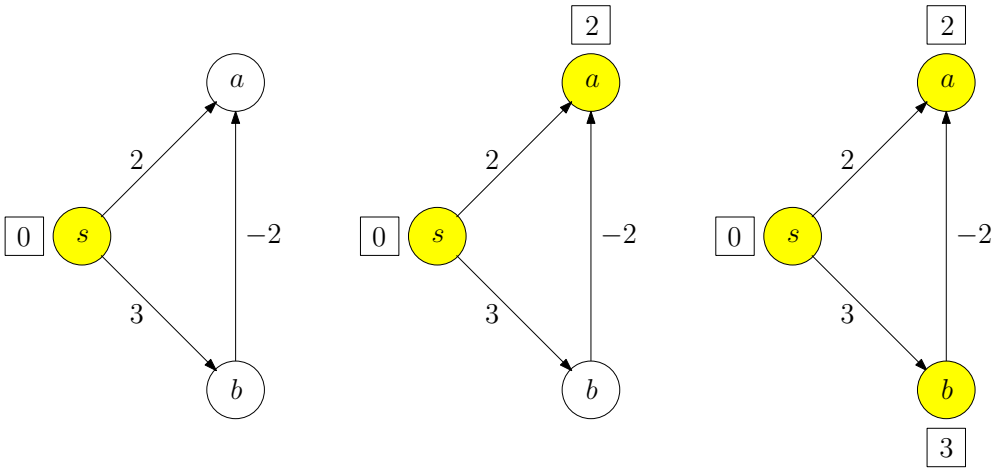
Feladat.
(A) Mutassuk meg, hogy a Dijkstra algoritmus nem feltétlenül találja meg egy súlyozott élű, irányított gráfban egy kezdőcsúcsból a legrövidebb utakat, ha az élsúlyok között negatív számok is szerepelnek!

(B) Legyen $G = (V, E)$ súlyozott élű, irányított gráf és $s \in V$. Tegyük fel, hogy a gráf élei közül egy negatív súlyú, a többi él súlya nem negatív (azt persze továbbra is feltesszük, hogy a gráf nem tartalmaz negatív összsúlyú irányított kört). Adjunk hatékony algoritmust az s csúcsból a többi csúcsba menő legrövidebb utak meghatározására!

Megoldás.
(A) Tekintsük a következő gráfot.



Futtassuk a Dijkstra algoritmust a gráfon az s kezdőcsúcsból.



Látható, hogy az algoritmus az s-ből a-ba vezető legrövidebb út hosszára a 2 értéket adja meg, miközben az (s, b, a) út hossza 1.

(B) Legyen $(u, v) \in E$ az egyetlen negatív súlyú él. Hagyjuk el a G gráfból (u, v)-t; jelölje az így kapott gráfot G'. Jegyezzük meg, hogy a G' gráfban az élsúlyok mind nem negatívak. Most tetszőleges $t \in V \setminus \{s\}$ csúcsra egy legrövidebb s-ből t-be vezető G-beli út hossza megegyezik

- egy legrövidebb s-ből t-be vezető G'-beli út hosszával, ha van olyan legrövidebb s-ből t-be vezető G-beli út, amely nem tartalmazza az (u, v) élt,
- egy legrövidebb s-ből u-ba vezető G'-beli út hosszával plusz az (u, v) él súlyával plusz egy legrövidebb v-ből t-be vezető G'-beli út hosszával, ha nincs olyan legrövidebb s-ből t-be vezető G-beli út, amely nem tartalmazza az (u, v) élt.

Ezek után a legrövidebb utak a következőképpen határozhatók meg. Futtassuk le a Dijkstra algoritmust a G' gráfon egyszer az s, egyszer pedig a v kezdőcsúcsból. Tetszőleges $t \in V \setminus \{s\}$ csúcsra egy legrövidebb s-ből t-be vezető G-beli út hossza legyen egy legrövidebb s-ből t-be vezető G'-beli út hossza, illetve egy legrövidebb s-ből u-ba vezető G'-beli út hossza plusz az (u, v) él súlya plusz egy legrövidebb v-ből t-be vezető G'-beli út hossza közül a kisebbik. A költség nyilván $O(|V|^2)$.

Legszélesebb utak

Feladat.
Legyen $G = (V, E)$ egy súlyozott élű irányított gráf a w nem negatív súlyfüggvénnyel. Ebben a feladatban az élsúlyok áteresztő képességeket reprezentálnak. A gráf egy $u \rightsquigarrow v$ irányított útjának szélessége legyen az út legkisebb súlyú élének súlya. Tetszőleges u és v csúcsokra legyen $b(u, v)$ egy maximális szélességű $u \rightsquigarrow v$ irányított út szélessége (ha u-ból nem vezet út v-be, akkor legyen $b(u, v) = 0$ definíció szerint). Adjunk hatékony algoritmust a gráf egy s csúcsából a többi csúcsba menő legszélesebb irányított utak megtalálására!

Megoldás.
Tegyük fel, hogy a G gráf az alábbi alakú C szomszédsági mátrixával adott:

$$C[u, v] = \begin{cases} \infty & \text{ha } u = v, \\ w(u, v) & \text{ha } u \neq v \text{ és } (u, v) \in E, \\ 0 & \text{különben.} \end{cases}$$

Használni fogunk egy a G csúcsaival indexelt B tömböt. A $B[u]$ értékekre úgy gondolhatunk, hogy azok minden időpillanatban az eljárás során addig megismert legszélesebb $s \rightsquigarrow u$ utak szélessége. A $B[u]$ mennyiségek mindenkor alsó közelítései lesznek a keresett $b(s,u)$ szélességeknek. Ezen kívül használunk még egy H halmazt, amelyben azokat a csúcsokat tároljuk, amelyekhez vezető legszélesebb utak szélességét már tudjuk.

```
LegszélesebbUtak(G,s)
H={s}
for minden u∈V csúcsra do
   B[u]=C[s,u]
for i=1 to |V|-1 do
   válasszunk egy olyan x∈V\H csúcsot, amelyre B[x] maximális
   H=H∪{x}
   for minden v∈V\H csúcsra do
      B[v]=max{B[v],min{B[x],C[x,v]}}
```

Az algoritmus helyességének belátásához hasznos lesz a különleges út fogalma: egy $s \rightsquigarrow z$ irányított út különleges, ha a z csúcsot kivéve minden csúcsa a H halmazban van. A különleges úttal elérhető csúcsok tehát éppen a H-ból egyetlen éllel elérhető csúcsok.

Állítás. A főciklus minden iterációs lépése után érvényesek a következők.

(1) Ha $u \in H$, akkor $B[u]$ egy legszélesebb $s \rightsquigarrow u$ út szélessége.

(2) Ha $u \in H$, akkor van olyan legszélesebb $s \rightsquigarrow u$ út, amelynek minden csúcsa H-ban van.

(3) Ha $v \in V \setminus H$, akkor $B[v]$ egy legszélesebb $s \rightsquigarrow v$ különleges út szélessége.

Bizonyítás. Teljes indukcióval bizonyítunk. A kezdőértékek beállítása után, amikor a főciklus még egyszer sem futott le, mindhárom állítás nyilvánvalóan igaz. Tegyük most fel, hogy mindhárom állítás igaz a j-edik iterációs lépés után, és tekintsük a $(j+1)$-edik iterációs lépést, amikor az eljárás mondjuk az x csúcsot választja be a H halmazba.

(1) Ezt elég csupán az x csúcsra igazolni, H többi csúcsára az indukciós feltevés adja az állítást. Indirekt tegyük fel, hogy $B[x] \neq b(s,x)$, azaz van olyan $s \rightsquigarrow x$ út, amelynek szélessége nagyobb, mint $B[x]$. Ez az út nem lehet különleges, hiszen a (3)-ra vonatkozó indukciós feltevés szerint $B[x]$ egy az

x-be vezető legszélesebb különleges út szélessége. Azonban a fenti út "eleje" különleges, mivel H-ból indul és egy H-n kívüli csúcsban végződik. Legyen y ezen út első olyan csúcsa, amely nincs H-ban. Az $s \rightsquigarrow y$ útdarab szélessége nyilván szintén nagyobb, mint $B[x]$. Másrészt az útdarab különleges, így a (3)-ra vonatkozó indukciós feltevés szerint szélessége nem nagyobb, mint $B[y]$. Ebből következik, hogy a j-edik iterációs lépés után $B[y] > B[x]$, ami lehetetlen, hiszen az algoritmusnak ekkor nem x-et kellett volna választania.

(2) Ezt is elég csupán az x csúcsra igazolni, hiszen H többi csúcsára az indukciós feltevés adja az állítást. Már beláttuk, hogy $b(s, x) = B[x]$. Ez utóbbi érték a (3)-ra vonatkozó indukciós feltevés szerint egy különleges $s \rightsquigarrow x$ út szélessége volt a $(j + 1)$-edik iterációs lépés előtt. Annak végeztével minden csúcs H-ban van.

(3) Egy legszélesebb olyan $s \rightsquigarrow v$ különleges út szélessége, amelynek utolsó előtti csúcsa u, az előző pont szerint $\min\{b(s, u), C[u, v]\}$. Emiatt az állítás egyenértékű a

$$B[v] = \max\{\min\{b(s, u), C[u, v]\} \mid u \in H\}$$

egyenlőséggel. A (3)-ra vonatkozó indukciós feltevés szerint a j-edik iteráció után

$$B[v] = \max\{\min\{b(s, u), C[u, v]\} \mid u \in H \setminus \{x\}\}.$$

Ezt a $(j+1)$-edik iterációban a $\min\{B[x], C[x, v]\}$ mennyiséggel vetjük össze. Azonban az első pont szerint itt $B[x] = b(s, x)$, így a $(j + 1)$-edik iterációban $B[v]$-be már az új H halmazra vett maximum kerül.

Következmény. Amikor $H = V$, akkor $B[u] = b(s, u)$ teljesül minden $u \in V$ csúcsra.

Vizsgáljuk meg az algoritmus költségét! A kezdőértékek beállításának költsége $O(|V|)$. A főciklus belsejében a maximumkeresés és a B tömb frissítésének költsége $O(|V|)$. Mivel a főciklus $(|V| - 1)$-szer fut le, a főciklus összköltsége $O(|V|^2)$. Mindezeket összevetve az algoritmus teljes költsége $O(|V|^2)$.

Általában nemcsak a legszélesebb utak szélességére, hanem magukra a legszélesebb utakra is kíváncsiak vagyunk. A költség nagyságrendjének megtartása mellett az algoritmust kibővíthetjük egy P tömb karbantartásával, amely minden csúcshoz megadja egy az eddig ismert hozzá vezető legszélesebb út utolsó előtti csúcsát.

Kezdetben legyen $P[u] = s$ minden $u \in V \setminus \{s\}$ csúcsra. Ezután csak annyit kell hozzátenni az algoritmushoz, hogy ha a főciklus belsejében valamelyik $v \in V \setminus H$ csúcsra $B[v]$-t növeljük, akkor legyen $P[v] = x$, hiszen

ekkor találtunk egy szélesebb különleges utat v-be, amelynek utolsó előtti csúcsa x. Végül minden $u \in V \setminus \{s\}$ csúcsra $P[u]$ egy legszélesebb $s \rightsquigarrow u$ út utolsó előtti csúcsa lesz.

A csúcsokba vezető legszélesebb utak ezek után a P tömb segítségével már egyszerűen felgombolyíthatók (visszafelé haladva).

Negatív összsúlyú irányított kör

Feladat.
Adjunk hatékony algoritmust annak eldöntésére, hogy egy súlyozott élű irányított gráf tartalmaz-e negatív összsúlyú irányított kört!

Megoldás.
Legyen $G = (V, E)$ egy súlyozott élű irányított gráf a $w\colon E \to \mathbb{R}$ súlyfüggvénnyel. Legyenek a G gráf csúcsai $v_1, v_2, \ldots, v_n$. Vegyünk fel egy új s csúcsot, és vezessünk s-ből nulla súlyú irányított éleket a $v_1, v_2, \ldots, v_n$ csúcsokba. Az így kialakult gráfot jelölje G'. Világos, hogy G-ben akkor és csak akkor van negatív összsúlyú irányított kör, ha ugyanez fennáll G'-re is.

Futtassuk le G'-re a Bellman-Ford algoritmust az s kezdőcsúccsal. Az algoritmus által szolgáltatott távolságértékeket jelölje $d[v_1], d[v_2], \ldots, d[v_n]$. Vegyük észre, hogy most $d[v_i] \leqslant 0$ minden $1 \leqslant i \leqslant n$ esetén. Vizsgáljuk meg ezután minden $(v_i, v_j) \in E$ élre, hogy teljesül-e a

$$d[v_j] \leqslant d[v_i] + w(v_i, v_j)$$

összefüggés.

Ha G'-ben nincs negatív összsúlyú irányított kör, akkor a $d[v_1], d[v_2], \ldots, d[v_n]$ értékek rendre az s-ből a $v_1, v_2, \ldots, v_n$ csúcsokba vezető legrövidebb utak hosszai, így a fenti összefüggéseknek nyilvánvalóan teljesülniük kell.

Megmutatjuk, hogy ha G'-ben van negatív összsúlyú irányított kör, akkor van olyan $(v_i, v_j) \in E$ él, amelyre

$$d[v_j] \leqslant d[v_i] + w(v_i, v_j)$$

nem teljesül. Legyen $C = (v_{i_0}, v_{i_1}, \ldots, v_{i_k})$ a gráf egy negatív összsúlyú irányított köre, ahol $v_{i_0} = v_{i_k}$. Ekkor

$$\sum_{j=1}^{k} w(v_{i_{j-1}}, v_{i_j}) < 0.$$

Indirekt tegyük fel, hogy minden $(v_i, v_j) \in E$ élre

$$d[v_j] \leqslant d[v_i] + w(v_i, v_j),$$

speciálisan minden $1 \leqslant j \leqslant k$ esetén

$$d[v_{i_j}] \leqslant d[v_{i_{j-1}}] + w(v_{i_{j-1}}, v_{i_j}).$$

Összegezzük ezeket az egyenlőtlenségeket:

$$\sum_{j=1}^{k} d[v_{i_j}] \leqslant \sum_{j=1}^{k} d[v_{i_{j-1}}] + \sum_{j=1}^{k} w(v_{i_{j-1}}, v_{i_j}).$$

Vegyük észre, hogy

$$\sum_{j=1}^{k} d[v_{i_j}] = \sum_{j=1}^{k} d[v_{i_{j-1}}].$$

Valóban, $v_{i_0} = v_{i_k}$ miatt a C kör minden csúcsa pontosan egyszer szerepel a két összegben. Azt is jegyezzük meg, hogy ezek az összegek végesek, hiszen $d[v_i] \leqslant 0$ minden $1 \leqslant i \leqslant n$ esetén. Innen

$$\sum_{j=1}^{k} w(v_{i_{j-1}}, v_{i_j}) \geqslant 0$$

adódik, ami ellentmondás.

Az algoritmus költsége $O(|V|^3)$.

Valuták átváltása

Feladat.
Tegyük fel, hogy adott n különböző valutanem, legyenek ezek $v_1, v_2, \ldots, v_n$, és a valutaárfolyamok egy $n \times n$-es T táblázata, amelynek $T[i, j]$ eleme azt mutatja meg, hogy a v_i valuta egy egységéért hány egységnyi vásárolható a v_j valutából ($1 \leqslant i, j \leqslant n$). Kezelési költség nincs.

(A) Az arbitrázs valutaváltások során az árfolyamok egyenetlenségeinek olyan hasznosítása, amikor egy valuta egy egységét ugyanazon valuta egy egységnél nagyobb értékére váltjuk át. Például tegyük fel, hogy 1 amerikai dollárért 46.4 indiai rúpiát, 1 indiai rúpiáért 2.5 japán jent és 1 japán jenért 0.0091 amerikai dollárt vehetünk. Ekkor 1 amerikai dollárt a valutaváltások ezen sorozatával

$$46.4 \times 2.5 \times 0.0091 = 1.0556$$

amerikai dollárra válthatunk át, ami 5.56% haszon. Adjunk hatékony algoritmust annak eldöntésére, hogy a megadott valutaárfolyamok mellett van-e lehetőség arbitrázsra!

(B) Tegyük fel, hogy a megadott valutaárfolyamok mellett nincs lehetőség arbitrázsra. Adjunk hatékony algoritmust átváltások egy olyan sorozatának megtalálására, amelynek során 50000 forintért a lehető legtöbb eurót vásárolhatjuk!

Megoldás.
(A) Tegyük fel, hogy a valutaárfolyamokat egy T táblázatban tároljuk. Arbitrázsra akkor van lehetőség, ha létezik valutaárfolyamoknak olyan $T[i_1, i_2]$, $T[i_2, i_3], \ldots, T[i_k, i_1]$ sorozata, hogy

$$T[i_1, i_2] \times T[i_2, i_3] \times \cdots \times T[i_k, i_1] > 1.$$

Ez a feltétel ekvivalens az

$$\frac{1}{T[i_1, i_2]} \times \frac{1}{T[i_2, i_3]} \times \cdots \times \frac{1}{T[i_k, i_1]} < 1$$

feltétellel, illetve mindkét oldal tízes alapú logaritmusát véve a

$$\log \frac{1}{T[i_1, i_2]} + \log \frac{1}{T[i_2, i_3]} + \cdots + \log \frac{1}{T[i_k, i_1]} < 0$$

feltétellel.

Tekintsük most azt a $G = (V, E)$ súlyozott élű irányított gráfot, amelynek csúcsai a $v_1, v_2, \ldots, v_n$ valuták, továbbá a v_i csúcsból $-\log T[i, j]$ súlyú él vezet a v_j csúcsba minden $1 \leqslant i, j \leqslant n$ esetén. Mit jelent G-re nézve a

$$\log \frac{1}{T[i_1, i_2]} + \log \frac{1}{T[i_2, i_3]} + \cdots + \log \frac{1}{T[i_k, i_1]} < 0$$

feltétel? Azt, hogy $(v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1})$ a G gráf egy negatív összhosszúságú irányított köre. Így arbitrázsra akkor van lehetőség, ha G-ben található negatív összhosszúságú irányított kör. Ilyen G-beli kör létezése $O(|V|^3)$ költséggel ellenőrizhető.
(B) Legyen v_1 a forint és v_n az euró. A feladat most valutaárfolyamok egy olyan $T[1, i_1], T[i_1, i_2], \ldots, T[i_k, n]$ sorozatának megtalálása, amelyre

$$T[1, i_1] \times T[i_1, i_2] \times \cdots \times T[i_k, n]$$

maximális. Ez ekvivalens azzal, hogy

$$\frac{1}{T[1, i_1]} \times \frac{1}{T[i_1, i_2]} \times \cdots \times \frac{1}{T[i_k, n]}$$

minimális, illetve tízes alapú logaritmust véve azzal, hogy

$$\log \frac{1}{T[1,i_1]} + \log \frac{1}{T[i_1,i_2]} + \cdots + \log \frac{1}{T[i_k,n]}$$

minimális.

Tekintsük ismét az előbb definiált $G = (V, E)$ gráfot. Mit jelent G-re nézve a

$$\log \frac{1}{T[1,i_1]} + \log \frac{1}{T[i_1,i_2]} + \cdots + \log \frac{1}{T[i_k,n]}$$

minimalitására vonatkozó feltétel? Azt, hogy $(v_1, v_{i_1}, \ldots, v_{i_k}, v_n)$ a G gráf egy v_1-ből v_n-be vezető legrövidebb útja.

Ilyen G-beli utat a Bellmann-Ford algoritmussal $O(|V|^3)$ költséggel találhatunk.

Gráfok centruma

Feladat.
Egy súlyozott élű, irányítatlan $G = (V, E)$ gráf v csúcsára legyen

$$e(v) = \max\{d(w, v) \mid w \in V\},$$

ahol $d(w, v)$ a w és v csúcsok közötti legrövidebb út hossza. Azt mondjuk, hogy egy v csúcs a G gráf centruma, ha tetszőleges $w \in V$ csúcsra $e(v) \leqslant e(w)$. Adjunk hatékony algoritmust G centrumának meghatározására!

Megoldás.
Először a Floyd-Warshall algoritmussal határozzuk meg a csúcsok közötti távolságokat. Ezután az algoritmus által visszaadott távolságmátrix minden oszlopában keressük meg a maximális értéket (ezek lesznek az $e(v)$ értékek), majd válasszuk ki a kapott $e(v)$ értékek közül a legkisebbet. Az ennek megfelelő csúcs(ok bármelyike) a gráf centruma lesz.

A Floyd-Warshall algoritmus költsége $O(|V|^3)$, a mátrix oszlopaiban a maximumok kiválasztásáé összesen $O(|V|^2)$, az $e(v)$ értékek minimumának meghatározásáé pedig $O(|V|)$. Így az algoritmus teljes költsége $O(|V|^3)$.

Különböző utak száma körmentes irányított gráfokban

Feladat.
Legyen $G = (V, E)$ egy szomszédsági listákkal adott körmentes irányított gráf és $s, t \in V$ tetszőleges csúcsok. Adjunk hatékony algoritmust az s csúcsból a t csúcsba menő különböző utak számának meghatározására!
Megoldás.
Először is határozzuk meg G csúcsainak egy $(v_1, v_2, \ldots, v_n)$ topológikus rendezését. Feltehetjük, hogy $s = v_1$ és $t = v_n$, mert csak kisebbtől nagyobb sorszámú csúcs felé megy él.

Tetszőleges $v_i \in V$ csúcsra jelölje $n[v_i]$ az s csúcsból a v_i csúcsba menő különböző utak számát. Ha $i = 1$, akkor $n[v_i] = 1$. Ha pedig $i > 1$, akkor érvényes a következő rekurzió:

$$n[v_i] = \sum_{(v_j, v_i) \in E} n[v_j].$$

Ezt felhasználva sorban kiszámolhatjuk az $n[v_i]$ értékeket. Vegyük észre, hogy ha $(v_j, v_i) \in E$, akkor $j < i$, tehát $n[v_i]$ számításakor az $n[v_j]$ értékek már ismertek.

Az $n[v_i]$ érték számításakor be-fok(v_i) darab nem negatív egész számot kell összeadni. Ezeket i-re összegezve $O(|E|)$ költség adódik.

Meg kell oldani még egy szervezési problémát: adott i-re gyorsan el kell tudnunk érni a v_i csúcsba bemenő éleket. Még mielőtt hozzáfognánk az $n[v_i]$ értékek kiszámításához, a G gráf szomszédsági listáiból előállítjuk G "fordított" szomszédsági listáit. Kezdetben legyen az összes csúcshoz tartozó "fordított" szomszédsági lista üres. Ezután végigmenve az "eredeti" szomszédsági listákon, ha egy v_i csúcsnak szomszédja a v_j csúcs, akkor szúrjuk be a v_i csúcsot a v_j csúcs "fordított" szomszédsági listájának elejére. Az eljárás költsége nyilván $O(|V| + |E|)$.

Így az algoritmus teljes költsége, a topológikus rendezést is beleszámítva $O(|V| + |E|)$.

Legrövidebb utak körmentes irányított gráfokban

Feladat.
Legyen $G = (V, E)$ egy szomszédsági listákkal adott, súlyozott élű körmentes irányított gráf a w súlyfüggvénnyel. Az élsúlyok között negatív számok is

lehetnek. A gráf tetszőleges u és v csúcsaira legyen $d(u, v)$ egy legrövidebb $u \rightsquigarrow v$ irányított út hossza (ha u-ból nem vezet út v-be, akkor legyen $d(u, v) = \infty$ definíció szerint). Adjunk hatékony algoritmust a gráf egy s csúcsából a többi csúcsba menő legrövidebb utak megtalálására!

Megoldás.
Először is határozzuk meg G csúcsainak egy $(v_1, v_2, \ldots, v_n)$ topológikus rendezését. Feltehetjük, hogy $s = v_1$, mert csak kisebbtől nagyobb sorszámú csúcs felé megy él.

Jegyezzük meg, hogy egy körmentes irányított gráf bármely két csúcsa közötti legrövidebb út összes részútja is legrövidebb út a részút végpontjai között. Így a $d(s, v_i)$ távolságokra érvényes a következő rekurzió:

$$d(s, v_i) = \min\{d(s, v_j) + w(v_j, v_i) \mid (v_j, v_i) \in E\}.$$

Ezt felhasználva sorban kiszámolhatjuk a $d(s, v_i)$ távolságokat. Vegyük észre, hogy ha $(v_j, v_i) \in E$, akkor $j < i$, tehát $d(s, v_i)$ számításakor a $d(s, v_j)$ értékek már ismertek.

Az i-edik távolság számításakor be-fok(v_i) darab kéttényezős összeg közül kell a minimálisat kiválasztani. Ezeket i-re összegezve $O(|E|)$ költség adódik.

Meg kell oldani még egy szervezési problémát: adott i-re gyorsan el kell tudnunk érni a v_i csúcsba bemenő éleket. Még mielőtt hozzáfognánk a $d(s, v_i)$ távolságok kiszámításához, a G gráf szomszédsági listáiból előállítjuk G "fordított" szomszédsági listáit. Kezdetben legyen az összes csúcshoz tartozó "fordított" szomszédsági lista üres. Ezután végigmenve az "eredeti" szomszédsági listákon, ha egy v_i csúcsnak szomszédja a v_j csúcs, akkor szúrjuk be a v_i csúcsot a v_j csúcs "fordított" szomszédsági listájának elejére. Az eljárás költsége nyilván $O(|V| + |E|)$.

Így az algoritmus teljes költsége, a topológikus rendezést is beleszámítva $O(|V| + |E|)$.

Leghosszabb utak körmentes irányított gráfokban

Feladat.
Legyen $G = (V, E)$ egy szomszédsági listákkal adott, súlyozott élű körmentes irányított gráf a w súlyfüggvénnyel. Az élsúlyok között negatív számok is lehetnek. A gráf tetszőleges u és v csúcsaira legyen $l(u, v)$ egy leghosszabb $u \rightsquigarrow v$ irányított út hossza (ha u-ból nem vezet út v-be, akkor legyen $l(u, v) = -\infty$ definíció szerint). Adjunk hatékony algoritmust a gráf egy s csúcsából a többi csúcsba menő leghosszabb utak megtalálására!

Megoldás.
Először is határozzuk meg G csúcsainak egy $(v_1, v_2, \ldots, v_n)$ topológikus rendezését. Feltehetjük, hogy $s = v_1$, mert csak kisebbtől nagyobb sorszámú csúcs felé megy él.

Jegyezzük meg, hogy egy körmentes irányított gráf bármely két csúcsa közötti leghosszabb út összes részútja is leghosszabb út a részút végpontjai között. Így az $l(s, v_i)$ távolságokra érvényes a következő rekurzió:

$$l(s, v_i) = \max\{l(s, v_j) + w(v_j, v_i) \mid (v_j, v_i) \in E\}.$$

Ezt felhasználva sorban kiszámolhatjuk az $l(s, v_i)$ távolságokat. Vegyük észre, hogy ha $(v_j, v_i) \in E$, akkor $j < i$, tehát $l(s, v_i)$ számításakor az $l(s, v_j)$ értékek már ismertek.

Az i-edik távolság számításakor be-fok(v_i) darab kétténye zős összeg közül kell a maximálisat kiválasztani. Ezeket i-re összegezve $O(|E|)$ költség adódik.

Meg kell oldani még egy szervezési problémát: adott i-re gyorsan el kell tudnunk érni a v_i csúcsba bemenő éleket. Még mielőtt hozzáfognánk az $l(s, v_i)$ távolságok kiszámításához, a G gráf szomszédsági listáiból előállítjuk G "fordított" szomszédsági listáit. Kezdetben legyen az összes csúcshoz tartozó "fordított" szomszédsági lista üres. Ezután végigmenve az "eredeti" szomszédsági listákon, ha egy v_i csúcsnak szomszédja a v_j csúcs, akkor szúrjuk be a v_i csúcsot a v_j csúcs "fordított" szomszédsági listájának elejére. Az eljárás költsége nyilván $O(|V| + |E|)$.

Így az algoritmus teljes költsége, a topológikus rendezést is beleszámítva $O(|V| + |E|)$.

Dobozok

Feladat.
Van n darab dobozunk, jelölje ezeket $D_1, D_2, \ldots, D_n$. Minden $1 \leqslant i \leqslant n$ esetén legyen a D_i doboz mérete $x_i \times y_i \times z_i$. Adjunk hatékony algoritmust dobozok egy olyan leghosszabb $(D_{i_1}, D_{i_2}, \ldots, D_{i_k})$ sorozatának megtalálására, amelynél D_{i_j} belerakható $D_{i_{j+1}}$-be minden $1 \leqslant j \leqslant k-1$ esetén!

Megoldás.
Először is jegyezzük meg, hogy a D_i doboz akkor és csak akkor rakható bele a D_j dobozba, ha a következők valamelyike fennáll:

- $x_i < x_j$ és $y_i < y_j$ és $z_i < z_j$,
- $x_i < x_j$ és $y_i < z_j$ és $z_i < y_j$,

- $x_i < y_j$ és $y_i < x_j$ és $z_i < z_j$,
- $x_i < y_j$ és $y_i < z_j$ és $z_i < y_j$,
- $x_i < z_j$ és $y_i < x_j$ és $z_i < y_j$,
- $x_i < z_j$ és $y_i < y_j$ és $z_i < x_j$.

Definiáljunk egy G gráfot a következőképpen. A gráf csúcsai legyenek $D_1, D_2, \ldots, D_n$. A D_i csúcsból akkor vezessen irányított él a D_j csúcsba, ha a D_i doboz belerakható a D_j dobozba. Világos, hogy G egy körmentes irányított gráf. Csatoljunk G-hez két további csúcsot, legyenek ezek S és T. Vezessünk S-ből egy-egy irányított élt azokba a csúcsokba, amelyek be-foka 0, és vezessünk T-be egy-egy irányított élt azokból a csúcsokból, amelyek ki-foka 0. Jelölje G' az így kialakult gráfot. G' is nyilván körmentes irányított gráf.

Keressük meg G egy leghosszabb, vagyis legtöbb élből álló, S-ből T-be vezető útját. Vegyük észre, hogy ez megfelel egymásba rakott dobozok egy leghosszabb sorozatának.

A G majd a G' gráf előállításának a költsége $O(n^2)$. Egy leghosszabb S-ből T-be vezető út szintén $O(n^2)$ költséggel kereshető meg a körmentes irányított G' gráfban, hiszen G' csúcsszáma $n+2$, éleinek száma pedig $O(n^2)$. Így az algoritmus teljes költsége $O(n^2)$.

Legrövidebb végrehajtási idő

Feladat.
Képzeljük el, hogy egy összetett tevékenység a $T_1, T_2, \ldots, T_n$ részteendőkre bontható. Ezek általában nem függetlenek egymástól; gyakran vannak olyan feltételeink, hogy bizonyos teendőket csak más teendők elvégzése után hajthatunk végre. Minden tevékenységnek ismert a végrehajtási ideje. Adjunk hatékony algoritmust annak eldöntésére, hogy

- az egyes részteendőket mikor lehet legkorábban elkezdeni,
- minimálisan mennyi idő szükséges a teljes feladat elvégzéséhez.

Megoldás.
A feltételrendszer természetes módon leírható irányított gráffal. Ennek csúcsai a T_i teendők. Egy T_i csúcsból akkor vezet él egy T_j csúcsba, ha a T_i teendőt a T_j teendő előtt el kell végezni.

Csatoljunk a gráfhoz két további csúcsot, K-t és B-t, amelyek a feladat kezdetét és végét jelölik. Menjen K-ból egy-egy irányított él azokba a csúcsokba, amelyek be-foka 0, és menjen B-be egy-egy irányított él azokból a csúcsokból, amelyek ki-foka 0. A K-hoz és B-hez tartozó végrehajtási idő legyen 0.

Ha a részteendőket végre lehet hajtani a feltételeket kielégítő sorrendben, akkor így egy körmentes irányított gráfot kaptunk. Jegyezzük meg, hogy ennek a gráfnak bármely topológikus rendezésében K az első, B pedig az utolsó csúcs.

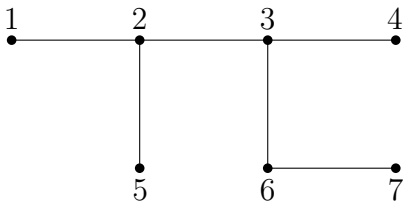
Súlyozzuk a gráf éleit a következőképpen: minden csúcsra a csúcsból induló élek súlya a csúcshoz rendelt végrehajtási idő legyen.

Hogyan határozhatjuk meg ezek után egy T_i részteendő legkorábbi kezdési idejét? Ez nem más, mint a leghosszabb $K \rightsquigarrow T_i$ út hossza. Speciálisan a teljes feladat elvégzésének minimális ideje a leghosszabb $K \rightsquigarrow B$ út hossza. Ezek az idők a körmentes irányított gráfokra vonatkozó leghosszabb út algoritmussal $O(n+m)$ költséggel megkaphatók, ahol m azon (T_i, T_j) párok száma, amelyeknél előírtuk, hogy a T_j teendőt csak a T_i teendő elvégzése után hajthatjuk végre.

Cayley tétel

Feladat.
Mutassuk meg, hogy egy $n \geqslant 2$ csúcsú teljes gráfnak n^{n-2} feszítőfája van!

Megoldás.
Az általánosság megszorítása nélkül feltehető, hogy a gráf csúcshalmaza $\{1, 2, \ldots, n\}$. Tekintsük a gráf egy T feszítőfáját. Legyen a_1 a legkisebb címkéjű levél csúcsa T-nek, és legyen b_1 az egyetlen szomszédja a_1-nek T-ben. Töröljük T-ből az a_1 csúcsot a rá illeszkedő (a_1, b_1) éllel, és jelölje T_1 az így kapott fát. Legyen a_2 a legkisebb címkéjű levél csúcsa T_1-nek, és legyen b_2 az egyetlen szomszédja a_2-nek T_1-ben. Töröljük T_1-ből az a_2 csúcsot a rá illeszkedő (a_2, b_2) éllel, és jelölje T_2 az így kapott fát. Az eljárást folytatva $n-2$ lépés után egy T_{n-2} fát kapunk, amely egyetlen (a_{n-1}, b_{n-1}) élből áll. Rendeljük hozzá a T fához a $P(T) = (b_1, b_2, \ldots, b_{n-2})$ sorozatot, ezt a T fa Prüfer-kódjának fogjuk nevezni. Például az alábbi ábrán látható fa Prüfer-kódja $(2, 3, 2, 3, 6)$.



Állítjuk, hogy a fent definiált P leképezés kölcsönösen egyértelmű megfeleltetést létesít a feszítőfák és az $\{1, 2, \ldots, n\}$ halmaz elemeiből álló $n-2$ tagú sorozatok között.

Először megmutatjuk, hogy P injektív. Legyen T egy feszítőfa, és legyen $P(T) = (b_1, b_2, \ldots, b_{n-2})$ a fa Prüfer-kódja. Elég belátni, hogy a $P(T)$ sorozat egyértelműen meghatározza az $a_1, a_2, \ldots, a_{n-1}, b_{n-1}$ pozitív egész számokat, így magát a T feszítőfát is. Kezdjük az a_1 számmal. Erre igazak a következők.

(1) Az a_1-nél kisebb címkéjű csúcsok nem levelei T-nek.

(2) Minden nem levél csúcs megjelenik a $P(T)$ sorozatban. Valóban, ha v nem levél csúcs, akkor szomszédos legalább két másik csúccsal. A $P(T)$ sorozatot előállító eljárás akkor fejeződik be, amikor már csak két T-beli csúcs marad. Ennélfogva az eljárás során legalább egy, v-vel szomszédos csúcs törlésre kerül. Ha ez az i-edik lépésben történik meg, akkor $b_i = v$.

Ez azt jelenti, hogy a_1 a legkisebb pozitív egész szám, amelyik nem jelenik meg a $P(T)$ sorozatban. Tegyük fel ezután, hogy az $a_1, a_2, \ldots, a_{i-1}$ pozitív egészek egyértelműen meghatározottak valamely $2 \leqslant i \leqslant n-2$ esetén. Törölve ezeket a csúcsokat és az $(a_1, b_1), (a_2, b_2), \ldots, (a_{i-1}, b_{i-1})$ éleket T-ből a T_{i-1} fát kapjuk. A T_{i-1} fában a_i a legkisebb címkéjű levél csúcs. Most az előbbi érvelést megismételve adódik, hogy a_i a legkisebb pozitív egész szám az $\{1, 2, \ldots, n\} \setminus \{a_1, a_2, \ldots, a_{i-1}\}$ halmazban, amelyik nem fordul elő a $b_i, b_{i+1}, \ldots, b_{n-2}$ számok között. Ennélfogva a páronként különböző $a_1, a_2, \ldots, a_{n-2}$ pozitív egész számok egyértelműen meghatározottak. Az ezek között nem szereplő $\{1, 2, \ldots, n\}$ halmazbeli két elem a_{n-1} és b_{n-1}.

Ezután belátjuk, hogy P szürjektív. A fent leírt dekódolási eljárás tetszőleges $(b_1, b_2, \ldots, b_{n-2})$ sorozatra alkalmazható, ahol a sorozat tagjai az $\{1, 2, \ldots, n\}$ halmaz elemei közül kerülnek ki. Jelölje $a_1, a_2, \ldots, a_{n-1}, b_{n-1}$ az eljárás által visszaadott számokat, amelyek szintén az $\{1, 2, \ldots, n\}$ halmaz elemei közül valók. Legyen G az a gráf, amelynek csúcshalmaza $\{1, 2, \ldots, n\}$, élei pedig $(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})$. Megmutatjuk, hogy G fa gráf. Minden $1 \leqslant i \leqslant n-1$ esetén jelölje G_i azt a gráfot, amelynek csúcshalmaza $\{1, 2, \ldots, n\}$, élei pedig $(a_i, b_i), (a_{i+1}, b_{i+1}), \ldots, (a_{n-1}, b_{n-1})$. A G_{n-1} gráfnak

egyetlen éle van, így nyilván körmentes. Tegyük fel ezután, hogy a G_{i+1} gráf körmentes valamely $1 \leqslant i \leqslant n-2$ esetén, és tekintsük a G_i gráfot. A dekódolási eljárás szerint az a_i szám különbözik mind az $a_{i+1}, a_{i+2}, \ldots, a_{n-1}$, mind pedig a $b_{i+1}, b_{i+2}, \ldots, b_{n-1}$ számoktól, ezért a_i elsőfokú csúcs G_i-ben, következésképpen G_i is körmentes. Ennélfogva $G = G_1$ körmentes gráf. Továbbá mivel G éleinek száma $n-1$, ezért G szükségképpen fa gráf, amelyre világos módon $P(G) = (b_1, b_2, \ldots, b_{n-2})$.

Az $\{1, 2, \ldots, n\}$ halmaz elemeiből álló $n-2$ tagú sorozatok száma n^{n-2}, így a feszítőfák száma is ugyanennyi.

Minimális költségű feszítőfák

Feladat.
Legyen $G = (V, E)$ egy súlyozott élű, irányítatlan, összefüggő gráf a $w\colon E \to \mathbb{R}$ súlyfüggvénnyel. Feladatunk G egy minimális költségű feszítőfájának meghatározása. Egy lehetséges megközelítés a következő. A módszer szemlélhető úgy, mintha a gráf éleit színeznénk. A gráf élei kezdetben színtelenek. Minden lépésben kiválasztunk egy még színtelen élt, és azt pirosra vagy kékre színezzük. A színezés során két szabályt használunk; egy élt csak akkor festünk be, ha valamelyik szabály ezt megengedi. A két szabályt tetszőleges sorrendben és helyeken alkalmazzuk, amíg csak lehetséges.

Kék szabály: válasszunk ki egy olyan $\emptyset \neq X \subset V$ csúcshalmazt, amiből nem vezet ki kék él, és egy legkisebb súlyú, X-ből kivezető színezetlen élt fessünk kékre.

Piros szabály: válasszunk G-ben egy olyan egyszerű kört, amiben nincs piros él, és a kör egyik legnagyobb súlyú színezetlen élét fessük pirosra.

Mutassuk meg, hogy az algoritmus befesti a G gráf minden élét, és a kék élek végül egy minimális költségű feszítőfa élei!

Megoldás.
Az állítás bizonyításánál hasznos lesz a megengedett színezés fogalma. Tekintsük a G gráf éleinek egy részleges színezését, melynél minden él piros, kék vagy színtelen. Ezt a színezést megengedettnek nevezzük, ha van G-nek olyan minimális költségű feszítőfája, amely az összes kék élt tartalmazza, és egyetlen piros élt sem tartalmaz.

Megmutatjuk, hogy az algoritmus végrehajtása során mindig megengedett színezésünk van. Ezen felül a színezéssel sosem akadunk el; végül G minden éle színes lesz. Ebből következik az állítás.

Először belátjuk, hogy a színezés mindig megengedett. Kezdetben ez nyilván igaz. Tegyük fel ezután, hogy egy megengedett színezésünk van. Legyen T egy megfelelő minimális költségű feszítőfa (amely minden kék élt tartalmaz és egyetlen pirosat sem). Tegyük fel továbbá, hogy a következő lépésben a gráf f élét festjük be. Két eset lehetséges:

(1) Kék szabályt használjuk. Ekkor f kék lesz. Ha f éle T-nek, akkor az új színezés is megengedett. Tegyük fel ezután, hogy f nem éle T-nek. Tekintsük azt az $X \subset V$ csúcshalmazt, amire a kék szabályt alkalmaztuk. A T feszítőfában van olyan út, amely f két végpontját összeköti. Ezen az úton szükségképpen van olyan f' él, ami kimegy X-ből, hiszen f kimegy X-ből. Mivel T egyetlen piros élt sem tartalmaz, ezért f' nem lehet piros, és nem lehet kék sem a kék szabály miatt (X-ből nem megy ki kék él). Így az f' él színezetlen. Másrészt szintén a kék szabály alapján $w(f') \geqslant w(f)$. Legyen T' a T-ből f' törlésével és f hozzáadásával kapott gráf. Világos, hogy T' is egy minimális költségű feszítőfa, hiszen összefüggő és $|V| - 1$ éle van, továbbá T' költsége nem haladja meg T költségét. Ebből következik, hogy az új színezés is megengedett.

(2) Piros szabályt használjuk. Ekkor f piros lesz. Ha f nem éle T-nek, akkor az új színezés is megengedett. Tegyük fel ezután, hogy f éle T-nek. Világos, hogy f törlésével T két komponensre esik szét. Tekintsük azt a kört, amelyre a piros szabályt alkalmaztuk. Ennek van olyan f-től különböző f' éle, ami a két komponens között fut. Mivel T az összes kék élt tartalmazza, ezért f' nem lehet kék, és nem lehet piros sem a piros szabály miatt (nincs a körön piros él). Így az f' él színezetlen. Másrészt szintén a piros szabály miatt $w(f') \leqslant w(f)$. Az f' él végpontjait összekötő T-beli út nyilván tartalmazza f-et. Legyen T' a T-ből f törlésével és f' hozzáadásával kapott gráf. Világos, hogy T' is egy minimális költségű feszítőfa, hiszen összefüggő és $|V| - 1$ éle van, továbbá T' költsége nem haladja meg T költségét. Ebből következik, hogy az új színezés is megengedett.

Hátra van még az állítás második felének bizonyítása. Tegyük fel tehát, hogy van még színezetlen él, legyen f egy ilyen. A színezés megengedett volta miatt a kék élek egy erdőt alkotnak, az erdő komponenseit nevezzük kék fáknak. Ha f végpontjai ugyanabban a kék fában vannak, akkor a piros szabály alkalmazható arra a körre, amelynek élei f és az f végpontjait összekötő (egyetlen) kék út élei. Ha pedig f különböző kék fákat köt össze, akkor a kék szabály alkalmazható: X legyen az egyik olyan kék fa csúcshalmaza, amelyhez f csatlakozik (ebben az esetben nem biztos, hogy f kap színt a következő lépésben). Így a színezés mindkét estben folytatható.

Minimális költségű feszítőfák élei

Feladat.
Legyen $G = (V, E)$ egy súlyozott élű, irányítatlan, összefüggő gráf, ahol az élsúlyok mind különbözők.

(A) Legyen $(S, V \backslash S)$ a gráf csúcshalmazának egy partíciója (egyik részhalmaz sem üres), és legyen $e = (v, w)$ az S és $V \setminus S$ csúcshalmazok között menő élek közül a legkisebb súlyú. Ekkor e hozzátartozik a G gráf összes minimális költségű feszítőfájához.

(B) Legyen C a gráf egy köre, és legyen $e = (v, w)$ a kör élei között a legnagyobb súlyú. Ekkor e nem tartozik hozzá a G gráf egyetlen minimális költségű feszítőfájához sem.

(C) Egy $e = (v, w)$ akkor és csak akkor nem tartozik hozzá G egyetlen minimális költségű feszítőfájához sem, ha v és w között vezet olyan út, amelyen minden él kisebb súlyú, mint e.

Megoldás.
(A) Indirekt tegyük fel, hogy létezik olyan T minimális költségű feszítőfája G-nek, amely nem tartalmazza e-t. Tekintsük a v és w csúcsok közötti T-beli utat. Mivel az út egyik végpontja S-ben, másik végpontja $V \setminus S$-ben van, ezért kell lenni olyan e' élnek az úton, amelynek végpontjaira ugyanez teljesül. Most a T-ből e hozzávételével és e' elhagyásával keletkező T' gráf szintén feszítőfája G-nek, hiszen összefüggő és $|V| - 1$ éle van, továbbá T' költsége kisebb, mint T költsége, ugyanis feltételünk szerint e súlya kisebb, mint e' súlya. Így T nem minimális költségű feszítőfája G-nek, ami ellentmondás.

Jegyezzük meg, hogy a bizonyítás során az élsúlyok különbözőségére vonatkozó feltételből csak annyit használtunk ki, hogy az S és $V \setminus S$ csúcshalmazok között menő élek között csak egy minimális súlyú van.

(B) Indirekt tegyük fel, hogy létezik olyan T minimális költségű feszítőfája G-nek, amely tartalmazza e-t. Ha a T fából töröljük e-t, akkor T két komponensre esik szét. Jelölje a két komponens csúcshalmazát S és $V \setminus S$. Ha a C körből töröljük e-t, akkor egy olyan utat kapunk, amelynek egyik végpontja S-ben, másik végpontja pedig $V \setminus S$-ben van. Világos, hogy ezen az úton kell lenni olyan e' élnek, amelynek végpontjaira ugyanez teljesül. Most a T-ből e' hozzávételével és e elhagyásával keletkező T' gráf szintén feszítőfája G-nek, hiszen összefüggő és $|V| - 1$ éle van, továbbá T' költsége kisebb, mint T költsége, ugyanis feltételünk szerint e súlya nagyobb, mint e' súlya. Így T nem minimális költségű feszítőfája G-nek, ami ellentmondás.

Jegyezzük meg, hogy a bizonyítás során az élsúlyok különbözőségére vonatkozó feltételből csak annyit használtunk ki, hogy a C kör élei között csak egy maximális súlyú van.

(C) Először tegyük fel, hogy v és w között vezet olyan út, amelyen minden él kisebb súlyú, mint e. Ezt az utat kiegészítve e-vel egy olyan kört kapunk, amelynek e a maximális súlyú éle, így az előző állítás szerint e nem tartozhat hozzá G egyetlen minimális költségű feszítőfájához sem.

Ezek után tegyük fel, hogy v és w között nem vezet olyan út, amelyen minden él kisebb súlyú, mint e. Legyen S a gráf azon csúcsainak halmaza, amelyek elérhetők v-ből olyan úton, amelyen minden él kisebb súlyú, mint e. Jegyezzük meg, hogy $v \in S$ és $w \notin S$.

Megmutatjuk, hogy az S és $V \setminus S$ csúcshalmazok között menő élek nem lehetnek e-nél kisebb súlyúak. Legyen $f = (x, y)$ olyan él, amelyre $x \in S$ és $y \in V \setminus S$. Feltételünk szerint x elérhető v-ből olyan úton, amelyen minden él kisebb súlyú, mint e. Ha most f kisebb súlyú lenne mint e, akkor y is elérhető lenne v-ből olyan úton, amelyen minden él kisebb súlyú, mint e, ellentmondva annak, hogy $y \in V \setminus S$.

Ennélfogva e az S és $V \setminus S$ csúcshalmazok között menő élek közül a legkisebb súlyú, így az első állítás szerint e hozzátartozik G összes minimális költségű feszítőfájához.

Minimális költségű feszítőfa egyértelműsége

Feladat.
Bizonyítsuk be, hogy ha egy $G = (V, E)$ súlyozott élű, irányítatlan, összefüggő gráfban az élsúlyok mind különbözők, akkor G minimális költségű feszítőfája egyértelmű!

Megoldás.
Indirekt tegyük fel, hogy T_1 és T_2 különböző minimális költségű feszítőfái G-nek. Ekkor T_1-nek van olyan e éle, amelyik nem tartozik hozzá T_2-höz. Vegyük hozzá T_2-höz az e élt. Az így kapott gráfban van kör, legyen e' ennek a legnagyobb súlyú éle. A minimális költségű feszítőfák éleiről szóló feladat második része szerint e' nem tartozhat hozzá G egyetlen minimális költségű feszítőfájához sem, ami persze ellentmondás, hiszen e' hozzátartozik T_1 és T_2 legalább egyikéhez.

Minimális költségű feszítőfák élsúly sorozatai

Feladat.
Legyen $G = (V, E)$ egy súlyozott élű, irányítatlan, összefüggő gráf a $w \colon E \to \mathbb{R}$ súlyfüggvénnyel, és legyenek T és T' minimális költségű feszítőfái G-nek. Legyenek $(e_1, e_2, \ldots, e_{n-1})$ és $(e'_1, e'_2, \ldots, e'_{n-1})$ a T és T' minimális

költségű feszítőfák éleinek az élsúlyok szerint monoton növekvően rendezett sorozatai. Mutassuk meg, hogy ekkor $w(e_i) = w(e'_i)$ minden $1 \leqslant i \leqslant n-1$ esetén!

Megoldás.
Indirekt tegyük fel, hogy léteznek olyan T és T' minimális költségű feszítőfák G-ben, amelyek éleinek az élsúlyok szerint monoton növekvően rendezett $(e_1, e_2, \ldots, e_{n-1})$ és $(e'_1, e'_2, \ldots, e'_{n-1})$ sorozataira $w(e_i) \neq w(e'_i)$ valamely $1 \leqslant i \leqslant n-1$ esetén. Legyen j a legkisebb ilyen index. Az általánosság megszorítása nélkül feltehető, hogy $w(e_j) < w(e'_j)$.

Tekintsük a G csúcsai és az $e'_1, e'_2, \ldots, e'_{j-1}$ élek által meghatározott F erdőt. Vegyük észre, hogy F bármelyik fájának a csúcsait az $e_1, e_2, \ldots, e_j$ élek közül legfeljebb annyi kötheti össze, mint ahány az $e'_1, e'_2, \ldots, e'_{j-1}$ élek közül. Valóban, legyen T^* az F erdő egy fája, és legyen V^* ennek a fának a csúcshalmaza. Mivel $e_1, e_2, \ldots, e_j$ a G gráf egy körmentes részgráfjának (feszítőfájának) élei, ezért a V^*-beli csúcsokat közülük legfeljebb $|V^*| - 1$ kötheti össze. Ugyanakkor a V^*-beli csúcsokat az $e'_1, e'_2, \ldots, e'_{j-1}$ élek közül pontosan $|V^*| - 1$ köti össze, a T^* fa élei.

Ebből következik, hogy az $e_1, e_2, \ldots, e_j$ élek között szükségképpen van olyan, amely különböző F-beli fákat köt össze, legyen e_k egy ilyen él. Tekintsük most az e_k él végpontjait összekötő T'-beli utat. Ezen az úton kell lenni olyan e'_m élnek, amely különbözik $e'_1, e'_2, \ldots, e'_{j-1}$ mindegyikétől, hiszen az e_k élt az F erdő éleihez hozzávéve nem keletkezik kör. Feltételünk szerint $w(e_k) \leqslant w(e_j) < w(e'_j) \leqslant w(e'_m)$, így a T' fához hozzávéve az e_k élt és elhagyva az e'_m élt a G gráf egy T'-nél kisebb költségű feszítőfáját kapjuk, ami ellentmondás.

Minimális költségű feszítőfa variáció

Feladat.
Egy feszítőfa költségét úgy definiáltuk, mint a fában szereplő élek súlyainak összege. Bizonyos esetekben másféle költségszámítás is értelmes lehet, például egy fa költségeként értelmezhetjük a legnagyobb súlyú élének a súlyát. Mutassuk meg, hogy ha egy $G = (V, E)$ súlyozott élű, irányítatlan, összefüggő gráfban T minimális költségű feszítőfa az előbbi értelemben, akkor minimális költségű feszítőfa az utóbbi értelemben is!

Megoldás.
Legyen T egy hagyományos értelemben vett minimális költségű feszítőfája G-nek. Indirekt tegyük fel, hogy T nem minimális költségű az új értelemben, azaz van olyan T' feszítőfája G-nek, amelyben a legnagyobb súlyú él

könnyebb, mint T legnagyobb súlyú éle. Ez azt jelenti, hogy T-nek van olyan e éle, amely nehezebb T' összes élénél. Vegyük hozzá T'-höz az e élt. Az így kapott gráfban van kör, ráadásul ennek a körnek e a legnehezebb éle. A minimális költségű feszítőfák éleiről szóló feladat második része szerint e nem tartozhat hozzá G egyetlen minimális költségű feszítőfájához sem, ami persze ellentmondás, hiszen e hozzátartozik T-hez.

Klaszterezés

Feladat.
Legyen adott (valamilyen) objektumok egy $U = \{p_1, p_2, \ldots, p_n\}$ halmaza. Tegyük fel, hogy bármely két p_i és p_j objektumnak ismerjük a $d(p_i, p_j)$ távolságát. A d távolságfüggvényről itt csak annyit teszünk fel, hogy tetszőleges p_i és p_j objektumokra

- $d(p_i, p_j) \geqslant 0$,
- $d(p_i, p_j) = 0$ akkor és csak akkor, ha $i = j$,
- $d(p_i, p_j) = d(p_j, p_i)$.

Legyen továbbá adott egy $k \leqslant n$ pozitív egész szám. Az U halmaz k nem üres részhalmazának egy $\{C_1, C_2, \ldots, C_k\}$ családját az U egy k-partíciójának nevezzük, ha

- $C_1 \cup C_2 \cup \cdots \cup C_k = U$,
- $C_i \cap C_j = \emptyset$, ha $i \neq j$.

A $\{C_1, C_2, \ldots, C_k\}$ partíció szeparáltsága legyen a

$$\min\{d(p_i, p_j) \mid p_i \in C_s,\ p_j \in C_t,\ s \neq t\}$$

mennyiség.

Adjunk hatékony algoritmust U egy olyan k-partíciójának megkeresésére, amelynek szeparáltsága a lehető legnagyobb!

Megoldás.
Tekintsük a következő súlyozott élű teljes gráfot: a gráf csúcsai legyenek az objektumok, két csúcs közötti él súlya pedig legyen a megfelelő objektumok távolsága. Indítsuk el ezen a gráfon a Kruskal algoritmust. Kezdetben n darab egy csúcsú fa van. Minden lépésben a legkisebb súlyú, kört nem okozó él hozzávételével a fák száma eggyel csökken. Amikor a fák száma eléri a k-t,

akkor álljunk meg. Kicsit más szemszögből nézve: futtassuk le a gráfon a Kruskal algoritmust, majd töröljük a kapott minimális költségű feszítőfa $k-1$ legnagyobb súlyú élét (a Kruskal algoritmus ezeket veszi hozzá a feszítőfához utoljára). Világos, hogy a kapott fák csúcshalmazai egy k-partíciót alkotnak.

Megmutatjuk, hogy az ilyen módon keletkező k-partíció szeparáltsága a lehető legnagyobb. Legyen $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ az eljárás által előállított k-partíció. Ennek szeparáltsága nyilván a Kruskal algoritmus által konstruált minimális költségű feszítőfa $(k-1)$-edik legnagyobb súlyú élének d^* súlya.

Tekintsünk ezután egy tetszőleges, $\mathcal{C}$-től különböző k-partíciót. Legyen ez mondjuk $\mathcal{C}' = \{C'_1, C'_2, \ldots, C'_k\}$. Mivel $\mathcal{C}$ és $\mathcal{C}'$ különbözők, ezért szükségképpen van olyan $C_r \in \mathcal{C}$ halmaz, amely nem részhalmaza egyetlen $\mathcal{C}'$-beli halmaznak sem. Ebben a C_r halmazban találhatók olyan p_i és p_j csúcsok, amelyek különböző $\mathcal{C}'$-beli halmazokban vannak, mondjuk $p_i \in C'_s$ és $p_j \in C'_t$, ahol $s \neq t$.

Mivel p_i és p_j ugyanabban a C_r halmazban vannak, ezért a Kruskal algoritmus még a megállítása előtt hozzávette az általa előállított erdőhöz egy p_i-t és p_j-t összekötő P út összes élét. Jegyezzük meg, hogy ezen a P úton minden él súlya kisebb vagy egyenlő, mint d^*. Másrészt $p_i \in C'_s$, míg $p_j \notin C'_s$. A p_i csúcsból indulva legyen p' az első olyan csúcs P-n, amely nincs C'_s-ben, p pedig az ezt közvetlenül megelőző csúcs. Az előbbi megjegyzéssel összhangban $d(p, p') \leqslant d^*$, így mivel p és p' a $\mathcal{C}'$ partíció különböző halmazaiban vannak, ezért $\mathcal{C}'$ szeparáltsága legfeljebb $d(p, p') \leqslant d^*$.

Az algoritmus az UNIÓ-HOLVAN adatszerkezet használatával hatékonyan megvalósítható. A halmazok legyenek az aktuális erdő komponenseinek csúcshalmazai. Az éppen vizsgált (p_i, p_j) él az erdőhöz adva pontosan akkor nem eredményez kört, ha a végpontok különböző komponensekben vannak, azaz HOLVAN$(p_i) \neq$ HOLVAN(p_j). A körmentesség tehát két HOLVAN kérdéssel ellenőrizhető. Miután a (p_i, p_j) élt hozzávettük az erdőhöz, egyesítenünk kell a p_i-t és a p_j-t tartalmazó komponenseket. Ezt egy UNIÓ művelettel érhetjük el.

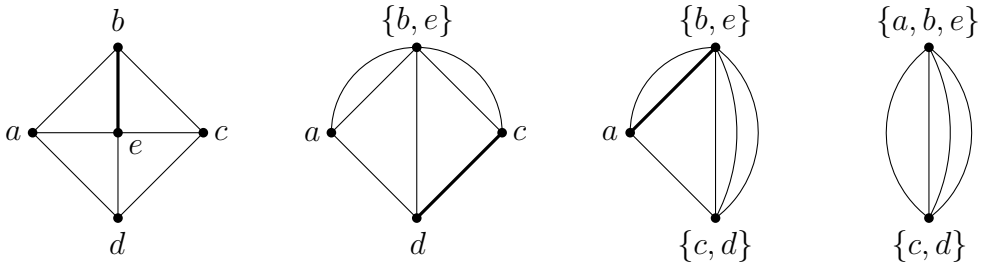
Egy él végpontjairól egyszer kérdezzük le, hogy melyik komponensben vannak. Ez $O(n^2)$ számú HOLVAN művelet, melyek összköltsége $O(n^2 \log n)$. Összesen $n-k$ élt választunk ki, ami $n-k$ számú UNIÓ műveletet jelent, ezek összköltsége $O(n)$. A kiindulási helyzetet jelentő n darab egy csúcsú fa kialakításának költsége is $O(n)$. A minimális élek kiválasztásával kapcsolatos teendők költsége $O(n^2 \log n^2) = O(n^2 \log n)$. Így a teljes költség $O(n^2 \log n)$.

Minimális vágás

Feladat.
Legyen $G = (V, E)$ egy $n \geqslant 2$ csúcsú, irányítatlan, összefüggő gráf. A gráf csúcsainak egy (A, B) partícióját, ahol A és B egyike sem üres halmaz, a gráf egy vágásának nevezzük. Egy (A, B) vágás méretén a gráf azon éleinek a számát értjük, amelyek egyik végpontja A-ban, a másik pedig B-ben van. Egy (A, B) vágást minimálisnak nevezünk, ha mérete nem haladja meg a gráf egyetlen más vágásának a méretét sem. Jegyezzük meg, hogy a G gráf egy minimális vágásának mérete megegyezik azon G-beli élek minimális számával, amelyek eltávolításával G legalább két összefüggő komponensre esik szét.

Futtassuk a G gráfon a következő algoritmust, amelyet kontrakció algoritmusnak is neveznek. A kontrakció algoritmus bemenete egy $G = (V, E)$ legalább két csúcsú, irányítatlan, összefüggő multigráf, vagyis olyan gráf, amelyben két csúcsot több (párhuzamos) él is összeköthet. Válasszuk ki véletlenszerűen (egyenletes eloszlás szerint) a G gráf egy (u, v) élét, és húzzuk össze ennek két végpontját egy új w csúcsba. Ez azt jelenti, hogy töröljük a gráfból az u és v csúcsokat az azokat összekötő élekkel együtt, majd vegyünk hozzá a gráfhoz egy új w csúcsot, amely legyen az új végpontja minden olyan élnek, amelynek eredetileg u vagy v volt az egyik végpontja (de nem mindkettő). Jelölje G' az így kapott multigráfot. Jegyezzük meg, hogy G'-ben akkor is megjelenhetnek párhuzamos élek, ha G-ben nem voltak ilyenek.

Ezután hívjuk meg a kontrakció algoritmust rekurzívan a G' gráfra. A rekurzív hívások során G' csúcsait "szupercsúcsoknak" kell tekinteni: minden w szupercsúcs megfelel a V-beli csúcsok azon $S(w)$ részhalmazának, amelyeket a w-hez vezető összehúzások során töröltünk a gráfból. A kontrakció algoritmus akkor fejeződik be, amikor G' két w_1 és w_2 szupercsúcsból áll (bizonyos számú párhuzamos éllel közöttük). Az algoritmus az $(S(w_1), S(w_2))$ vágást adja vissza.



(A) Mutassuk meg, hogy a kontrakció algoritmus a gráf egy minimális vágását adja vissza legalább $1/\binom{n}{2}$ valószínűséggel!

(B) Mutassuk meg, hogy ha a kontrakció algoritmust $\binom{n}{2} \ln n$ alkalommal, egymástól függetlenül lefuttatjuk a gráfon, akkor a visszaadott vágások közül a legkisebb értékű egy minimális vágás legalább $1 - 1/n$ valószínűséggel!

(C) Mutassuk meg, hogy egy $n \geqslant 2$ csúcsú, irányítatlan, összefüggő gráfnak legfeljebb $\binom{n}{2}$ minimális vágása létezik!

Megoldás.
(A) Legyen (A, B) a gráf egy minimális vágása. Legyen k a vágás mérete, vagyis azon élek F halmazának az elemszáma, amelyek egyik végpontja A-ban, másik végpontja pedig B-ben van. A kontrakció algoritmus pontosan akkor adja vissza az (A, B) vágást, ha egyik iterációban sem egy F-beli él két végpontját húzza össze.

Minden $1 \leqslant i \leqslant n-2$ esetén jelölje H_i azt az eseményt, hogy a kontrakció algoritmus az i-edik iterációban nem egy F-beli él két végpontját húzza össze. Célunk a $H_1 \cap H_2 \cap \cdots \cap H_{n-2}$ esemény valószínűségére alsó becslést adni. Ismert, hogy $\mathrm{Prob}[H_1 \cap H_2 \cap \cdots \cap H_{n-2}]$ felírható feltételes valószínűségek szorzataként:

$$\begin{aligned}\mathrm{Prob}[H_1 \cap H_2 \cap \cdots \cap H_{n-2}] = \mathrm{Prob}[H_1]\,\mathrm{Prob}[H_2 \mid H_1]\,\mathrm{Prob}[H_3 \mid H_1 \cap H_2] \\ \cdots \mathrm{Prob}[H_{n-2} \mid H_1 \cap H_2 \cap \cdots \cap H_{n-3}]\end{aligned}$$

Tekintsük az első iterációt. Vegyük észre, hogy G minden csúcsának a foka legalább k. Valóban, ha G valamely v csúcsának a foka kisebb lenne, mint k, akkor a $(\{v\}, V \setminus \{v\})$ vágás mérete kisebb lenne, mint k, ellentmondva az (A, B) vágás minimalitásának. Ennélfogva G éleinek száma legalább $\frac{1}{2}kn$. Ebből következik, hogy a kontrakciós algoritmus ebben az iterációban valamelyik F-beli él két végpontját legfeljebb

$$\frac{k}{\frac{1}{2}kn} = \frac{2}{n}$$

valószínűséggel húzza össze, így

$$\mathrm{Prob}[H_1] \geqslant 1 - \frac{2}{n}.$$

Legyen ezután $1 \leqslant j \leqslant n-3$, és tekintsük a $(j+1)$-edik iterációt. Most a G' gráfnak $n - j$ szupercsúcsa van. Tegyük fel, hogy a kontrakció algoritmus eddig egyetlen F-beli él két végpontját sem húzta össze. Mivel G' minden vágása egyben vágása G-nek is, ezért G' minden szupercsúcsára legalább k él illeszkedik. Ennélfogva G' éleinek száma legalább $\frac{1}{2}k(n - j)$. Ebből

következik, hogy a kontrakciós algoritmus ebben az iterációban valamelyik F-beli él két végpontját legfeljebb

$$\frac{k}{\frac{1}{2}k(n-j)} = \frac{2}{n-j}$$

valószínűséggel húzza össze, így

$$\mathrm{Prob}[H_{j+1} \mid H_1 \cap H_2 \cap \cdots \cap H_j] \geqslant 1 - \frac{2}{n-j}.$$

Mindezeket felhasználva

$$\begin{aligned}
&\mathrm{Prob}[H_1 \cap H_2 \cap \cdots \cap H_{n-2}] \\
&\qquad = \mathrm{Prob}[H_1]\,\mathrm{Prob}[H_2 \mid H_1]\,\mathrm{Prob}[H_3 \mid H_1 \cap H_2] \cdots \\
&\qquad\quad \cdots \mathrm{Prob}[H_{n-2} \mid H_1 \cap H_2 \cap \cdots \cap H_{n-3}] \\
&\qquad \geqslant \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{3}\right) \\
&\qquad = \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right) \cdots \left(\frac{1}{3}\right) \\
&\qquad = \frac{2}{n(n-1)} = \binom{n}{2}^{-1}.
\end{aligned}$$

adódik.

(B) Az előző gondolatmenettel összhangban a kontrakció algoritmus legfeljebb $1 - 1/\binom{n}{2}$ valószínűséggel ad vissza olyan vágást, amely nem minimális. Így annak a valószínűsége, hogy a kontrakció algoritmust $\binom{n}{2} \ln n$ alkalommal, egymástól függetlenül lefuttatva minden esetben olyan vágást kapunk, amely nem minimális, legfeljebb

$$\left(1 - 1/\binom{n}{2}\right)^{\binom{n}{2}\ln n} \leqslant e^{-\ln n} = 1/n.$$

Itt felhasználtuk, hogy tetszőleges $n \geqslant 2$ egész számra

$$\left(1 - 1/\binom{n}{2}\right)^{\binom{n}{2}} \leqslant e^{-1}.$$

(C) Legyen G egy $n \geqslant 2$ csúcsú, irányítatlan, összefüggő gráf, és legyenek $C_1, C_2, \ldots, C_r$ a gráf minimális vágásai. Minden $1 \leqslant i \leqslant r$ esetén jelölje M_i azt az eseményt, hogy a kontrakció algoritmus a C_i minimális vágást adja vissza, továbbá legyen $M = M_1 \cup M_2 \cup \cdots \cup M_r$.

Vegyük észre, hogy az (A) pont bizonyításánál valójában azt mutattuk meg, hogy $\mathrm{Prob}[M_i] \geqslant 1/\binom{n}{2}$ minden $1 \leqslant i \leqslant r$ esetén. Mivel tetszőleges $1 \leqslant i < j \leqslant r$ esetén az M_i és M_j események nyilvánvaló módon diszjunktak, így

$$1 \geqslant \mathrm{Prob}[M] = \mathrm{Prob}\left[\bigcup_{i=1}^{r} M_i\right] = \sum_{i=1}^{r} \mathrm{Prob}[M_i] \geqslant r/\binom{n}{2},$$

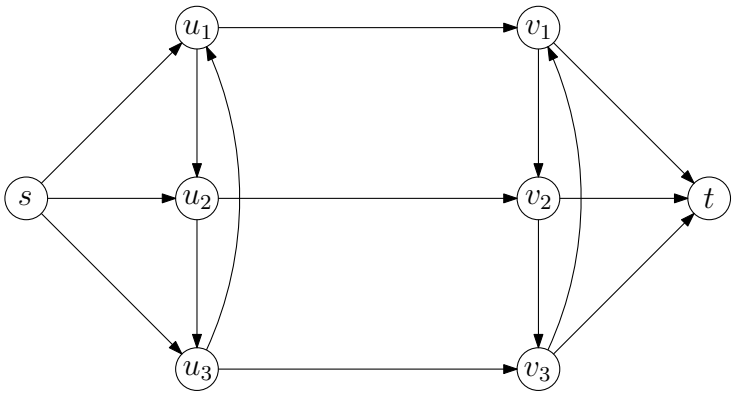
ahonnan $r \leqslant \binom{n}{2}$ adódik.

Jegyezzük meg, hogy ez a korlát éles: egy n csúcsú körnek éppen $\binom{n}{2}$ minimális vágása van, ezek mind 2 értékűek.

Folyamok és párosítások

Ford-Fulkerson algoritmus

Feladat.
Tekintsük a következő hálózatot.



Legyen az (u_1, v_1) él kapacitása 1, az (u_2, v_2) él kapacitása r, az (u_3, v_3) él kapacitása r^2, az összes többi él kapacitása pedig $2 + r$, ahol

$$r = \frac{-1 + \sqrt{5}}{2} = 0.618\ldots$$

az $x^2 + x - 1 = 0$ másodfokú egyenlet pozitív gyöke. Mutassuk meg, hogy ezen a hálózaton futtatva a Ford-Fulkerson algoritmust, az nem feltétlenül fejeződik be véges sok iterációi után!

Megoldás.
Elöször is jegyezzük meg, hogy $r^2 = 1 - r$, így $r^{n+2} = r^n - r^{n+1}$ tetszőleges n természetes számra. Jegyezzük meg azt is, hogy $0 < r < 1$ miatt az r^n sorozat szigorúan monoton csökenve tart nullához, továbbá

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r} = 2 + r.$$

A hálózatban $(\{s, u_1, u_2, u_3\}, \{v_1, v_2, v_3, t\})$ minimális vágás, hiszen ennek kapacitása $1 + r + r^2 = 2$, minden más vágás kapacitása pedig legalább $2 + r$. Így a maximális folyam minimális vágás tétel szerint a hálózat maximális folyama 2 értékű.

Futtassuk most a Ford-Fulkerson algoritmust a hálózaton a következő módon. Induljunk ki az azonosan 0 folyamból és az első iterációban növeljük a folyamot 1-gyel a reziduális hálózat

$$s \to u_1 \to v_1 \to t$$

javító útja mentén. Ezután az (u_1, v_1) él reziduális kapacitása 0, az (u_2, v_2) él reziduális kapacitása r, az (u_3, v_3) él reziduális kapacitása pedig r^2 lesz a reziduális hálózatban. A második iterációban növeljük a folyamot r^2-nal a reziduális hálózat

$$s \to u_3 \to v_3 \to v_1 \to u_1 \to u_2 \to v_2 \to t$$

javító útja mentén. Ezután az (u_1, v_1) él reziduális kapacitása r^2, az (u_2, v_2) él reziduális kapacitása $r - r^2 = r^3$, az (u_3, v_3) él reziduális kapacitása pedig 0 lesz a reziduális hálózatban. A harmadik iterációban növeljük a folyamot r^3-nal a reziduális hálózat

$$s \to u_2 \to v_2 \to v_3 \to u_3 \to u_1 \to v_1 \to t$$

javító útja mentén. Ezután az (u_1, v_1) él reziduális kapacitása $r^2 - r^3 = r^4$, az (u_2, v_2) él reziduális kapacitása 0, az (u_3, v_3) él reziduális kapacitása pedig r^3 lesz a reziduális hálózatban. A negyedik iterációban növeljük a folyamot r^4-nel a reziduális hálózat

$$s \to u_1 \to v_1 \to v_2 \to u_2 \to u_3 \to v_3 \to t$$

javító útja mentén. Ezután az (u_1, v_1) él reziduális kapacitása 0, az (u_2, v_2) él reziduális kapacitása r^4, az (u_3, v_3) él reziduális kapacitása pedig $r^3 - r^4 = r^5$ lesz a reziduális hálózatban. Világos, hogy a folyam növelését ilyen módon végtelen sokszor megismételhetjük, egyre nagyobb és nagyobb r hatványokkal. Ne feledjük, az (u_1, v_1), (u_2, v_2), (u_3, v_3) élektől különböző élek kapacitása $2 + r$, így azoknak mindig elég nagy a reziduális kapacitása a folyam növeléséhez. A folyam értéke 2-höz konvergál a megoldás elején tett második észrevétellel összhangban.

Jegyezzük meg, hogy ha a hálózathoz hozzávesszük az (s, t) élt 1 kapacitással, akkor a javító utak előbb látott megválasztásával továbbra is egy 2 értékű folyamhoz tartunk, miközben az új hálózat maximális folyama 3 értékű.

Maximális folyamok

Feladat.
Döntsük el, hogy igazak-e a következő állítások! Ha igen, bizonyítsuk be, ha nem, adjunk ellenpéldát!

(A) Ha egy hálózatban minden él kapacitása páros szám, akkor van olyan maximális folyam, hogy a hálózat minden élén páros a folyam értéke.

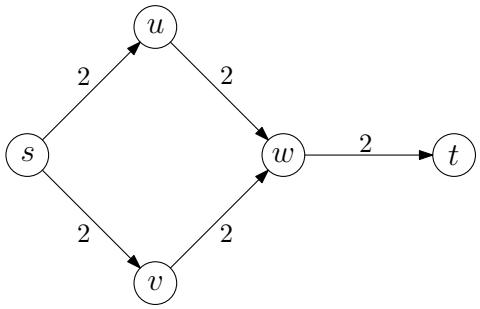
(B) Ha egy hálózatban minden él kapacitása páros szám, akkor minden maximális folyam esetén a hálózat minden élén páros a folyam értéke.

(C) Ha egy hálózatban minden él kapacitása páratlan szám, akkor van olyan maximális folyam, hogy a hálózat minden élén páratlan a folyam értéke.

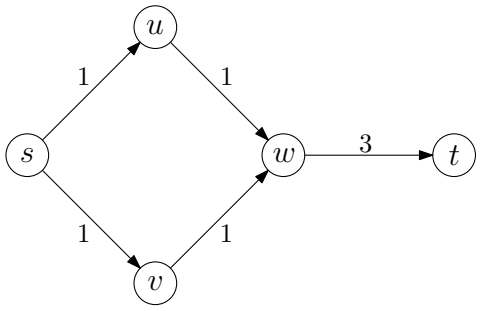
(D) Ha egy hálózatban minden él kapacitása páratlan szám, akkor minden maximális folyam esetén a hálózat minden élén páratlan a folyam értéke.

Megoldás.
(A) Igaz, a Ford-Fulkerson algoritmus egy ilyen maximális folyamot ad. Ehhez elég észrevenni, hogy esetünkben a Ford-Fulkerson algoritmus minden iterációs lépése után mind a folyamértékek, mind pedig a reziduális kapacitások páros számok.

(B) Hamis. A következő hálózatban $f(s,u) = f(s,v) = f(u,w) = f(v,w) = 1$ és $f(w,t) = 2$ egy maximális folyam.



(C) Hamis. A következő hálózatban $f(s,u) = f(s,v) = f(u,w) = f(v,w) = 1$ és $f(w,t) = 2$ az egyetlen maximális folyam.



(D) Hamis. Lásd (C).

Új maximális folyam

Feladat.
Legyen $G = (V, E)$ egy hálózat az s forrással és t nyelővel, valamint a c egészértékű kapacitásfüggvénnyel. Legyen továbbá f egy egészértékű maximális folyam a hálózaton. Tegyük fel, hogy egy adott $(u, v) \in E$ él kapacitását eggyel megnöveljük. Adjunk $O(|E| + |V|)$ költségű algoritmust az új hálózat egy maximális folyamának megkeresésére!

Megoldás.
Először megmutatjuk, hogy az új hálózatban a maximális folyamérték $|f|$ vagy $|f| + 1$. Az új hálózatban a maximális folyamérték nyilván legalább $|f|$, hiszen f egy folyam az új hálózatban is. Így elég belátni, hogy az új hálózatban a maximális folyamérték legfeljebb $|f| + 1$. A maximális folyam – minimális vágás tétel szerint van olyan (A, B) vágása a régi hálózatnak, amelynek kapacitása $|f|$. Mekkora az új hálózat (A, B) vágásának kapacitása? A régi hálózat (u, v)-n kívüli éleinek kapacitása nem változott, az (u, v) él kapacitása pedig eggyel nőtt. Ebből azonnal következik, hogy az új hálózat (A, B) vágásának kapacitása legfeljebb $|f| + 1$. A maximális folyam – minimális vágás tétel szerint ezért az új hálózatban a maximális folyamérték legfeljebb $|f| + 1$.

Ezek után az algoritmus egyszerű. Az f folyamból kiindulva hajtsuk végre a Ford-Fulkerson algoritmus egy iterációját az új hálózaton. Két dolog történhet.

- Nem találunk javító utat. Ekkor f egy maximális folyam az új hálózatban is.
- Találunk javító utat. Ekkor ennek felhasználásával egy legalább $|f|+1$ értékű folyamhoz jutunk az új hálózatban. Ez az előbbiek szerint egy maximális folyam az új hálózatban.

Az algoritmus költsége $O(|E| + |V|)$.

Maximális folyam skálázással

Feladat.
Legyen $(G = (V, E), s, t, c)$ egy hálózat az s forrással és t nyelővel, valamint a c egészértékű kapacitásfüggvénnyel. Tekintsük a Ford-Fulkerson algoritmus következő változatát (szokásos módon $c_f(P)$ a P javító úton levő élek G_f-beli reziduális kapacitásainak minimumát jelöli).

```
FordFulkersonSkálázással(G,s,t,c)
C=max{c(u,v)|(u,v) éle G-nek}
legyen f az azonosan nulla folyam
K=2^[log_2C]
while K>=1 do
   while létezik a G_f reziduális hálózatban olyan P javító út,
         amelyre c_f(P)>=K do
      növeljük f-et P mentén c_f(P)-vel
   K=K/2
return f
```

Mutassuk meg, hogy az algoritmus egy maximális folyammal tér vissza és költsége $O(|E|^2 \log C)$.

Megoldás.
Először az algoritmus helyességét látjuk be. Ez egyszerű. Az algoritmus akkor fejeződik be, ha a reziduális gráfban már nincs olyan P javító út, amelyre $c_f(P) \geqslant 1$. Mivel a $c_f(u, v)$ reziduális kapacitások mind pozitív egészek, ez azt jelenti, hogy a reziduális gráfban egyáltalán nincs javító út. Ezért az algoritmus egy maximális folyammal tér vissza.

Próbáljuk megbecsülni ezután az algoritmus költségét. A külső while ciklus iterációinak száma nyilván $O(\log C)$. De mennyi a belső while ciklus iterációinak száma adott K mellett?

Állítás. Legyen f az a folyam, amelyet adott K mellett a belső while ciklus befejezésekor látunk. Ekkor van olyan (S, T) vágás G-n, amelyre $c(S, T) < |f| + K|E|$. Következésképpen a maximális folyamérték G-n kisebb, mint $|f| + K|E|$.

Bizonyítás. Jelölje S a gráf azon csúcsainak halmazát, amelyek elérhetők G_f-ben s-ből olyan egyszerű úttal, amelyen minden él reziduális kapacitása

legalább K. Legyen továbbá $T = V \setminus S$. Világos, hogy (S, T) egy vágás G-n, máskülönben a while ciklus még nem fejeződött volna be.

Vegyük észre, hogy ha $(u, v) \in S \times T$ és $(u, v) \in E$, akkor

$$f(u, v) > c(u, v) - K,$$

ellenkező esetben (u, v) olyan éle lenne a reziduális hálózatnak, amelynek reziduális kapacitása nagyobb vagy egyenlő volna, mint K, így v is elérhető lenne G_f-ben s-ből olyan egyszerű úttal, amelyen minden él reziduális kapacitása legalább K. Hasonlóan, ha $(u, v) \in S \times T$ és $(u, v) \notin E$ de $(v, u) \in E$, akkor

$$f(u, v) > -K,$$

ellenkező esetben (u, v) ismét olyan éle lenne a reziduális hálózatnak, amelynek reziduális kapacitása nagyobb vagy egyenlő volna, mint K, így v is elérhető lenne G_f-ben s-ből olyan egyszerű úttal, amelyen minden él reziduális kapacitása legalább K. A többi $(u, v) \in S \times T$ csúcspárra $f(u, v) = 0$.

Egy kis számolás következik:

$$\begin{aligned} |f| &= \sum_{(u,v)\in S\times T} f(u,v) \\ &> \sum_{\substack{(u,v)\in S\times T \\ (u,v)\in E}} (c(u,v)-K) \quad + \sum_{\substack{(u,v)\in S\times T \\ (u,v)\notin E \text{ és } (v,u)\in E}} (-K) \\ &\geqslant c(S,T) - K|E|. \end{aligned}$$

Innen az állítás első része adódik. A második rész a maximális folyam – minimális vágás tételből következik.

Ezek után a belső while ciklus iterációinak száma adott K mellett már könnyen becsülhető. Először is jegyezzük meg, hogy adott K mellett a belső while ciklus minden iterációja során a folyamérték legalább K-val nő. A szokásos módon jelölje $|f^*|$ a maximális folyamértéket G-n.

Tekintsük a belső ciklus első végrehajtását. Ekkor

$$K = 2^{[\log_2 C]} = 2^{[\log_2 C]+1}/2 > 2^{\log_2 C}/2 = C/2,$$

így a ciklus minden iterációjában a folyamérték legalább $C/2$-vel nő. Továbbá $|f^*| \leqslant C|E|$, hiszen semelyik vágás kapacitása sem lehet nagyobb, mint $C|E|$. Ennélfogva a ciklusban az iterációk száma nem lehet több, mint $2|E|$, mivel a ciklus befejezése után a folyamérték nyilván nem nagyobb, mint $|f^*|$.

Ezek után tekintsük a belső ciklus további végrehajtásait. Az állítással összhangban adott K mellett a belső while ciklus végrehajtás előtt $|f| >$

$|f^*| - 2K|E|$ teljesül (az állítást a belső while ciklus ezt közvetlenül megelőző végrehajtására alkalmazzuk a $K' = 2K$ paraméterrel). Minden iterációban a folyamérték legalább K-val nő, végül a ciklus befejezése után a folyamérték nyilván legfeljebb $|f^*|$. Így ezekben az esetekben is az iterációk száma legfeljebb $2|E|$.

Végül nézzük a javító utak keresésének költségét. Adott K értékre a reziduális hálózatban olyan P javító utat, amelyre $c_f(P) \geqslant K$ teljesül, szélességi vagy mélységi bejárással kereshetünk, a K-nál kisebb reziduális kapacitású éleket egyszerűen figyelmen kívül hagyva. A költség $O(|V| + |E|) = O(|E|)$.

Mindezekből következik, hogy a két egymásba ágyazott while ciklus során végrehajtott műveletek költsége $O(|E|^2 \log C)$. Az egymásba ágyazott ciklusok előtti műveletek költsége $O(|E|)$, így az algoritmus teljes költsége $O(|E|^2 \log C)$.

Edmonds-Karp algoritmus

Feladat.
Legyen $(G = (V, E), s, t, c)$ egy hálózat az s forrással és t nyelővel, valamint a c kapacitásfüggvénnyel. Mutassuk meg, hogy a Ford-Fulkerson algoritmus által végrehajtott iterációk száma $|V|$ és $|E|$ polinomjával felülről becsülhető, ha mindig egy legrövidebb, vagyis egy legkevesebb élből álló javító út mentén növelünk! Emlékezzünk rá, hogy legrövidebb javító utat szélességi kereséssel találhatunk. Az algoritmus ezen változatát Edmonds-Karp algoritmusnak nevezzük.

Megoldás.
Az Edmonds-Karp algoritmus során végrehajtott iterációk számának becslésénél fontos szerepet játszanak a következő mennyiségek. Tetszőleges f folyam és $v \in V$ csúcs esetén jelölje $\delta_f(s, v)$ a G_f reziduális hálózatban a v csúcs s-től való távolságát, azaz az s-ből v-be vezető legrövidebb, azaz legkevesebb élből álló út hosszát.

Állítás. Ha a (G, s, t, c) hálózatban az Edmonds-Karp algoritmussal keresünk maximális folyamot, akkor az összes $v \in V \setminus \{s, t\}$ csúcsra a $\delta_f(s, v)$ mennyiség minden iterációs lépésben monoton nő.

Bizonyítás. Indirekt tegyük fel, hogy valamelyik iterációs lépésben valamelyik s-től és t-től különböző csúcs s-től való távolsága szigorúan monoton csökken. Legyen f az első ilyen iteráció előtti folyam, f' pedig az iteráció utáni folyam. Legyen továbbá v egy olyan s-től és t-től különböző csúcs, amelynek s-től való távolsága ebben az iterációs lépésben szigorúan monoton

csökken, tehát amelyre $\delta_{f'}(s,v) < \delta_f(s,v)$, és amelynek az ilyen tulajdonságú csúcsok közül az s-től való távolsága minimális az f'-höz tartozó rezidális hálózatban.

Tekintsünk most egy s-ből v-be vezető legrövidebb utat az f'-höz tartozó rezidális hálózatban, jelölje ennek a v-t közvetlenül megelőző csúcsát u. Ekkor

$$\delta_{f'}(s,u) = \delta_{f'}(s,v) - 1.$$

A v csúcs megválasztása miatt nyilván

$$\delta_{f'}(s,u) \geqslant \delta_f(s,u).$$

Állítjuk, hogy $(u,v) \notin E_f$, ahol E_f szokásos módon az f-hez tartozó reziduális hálózat élhalmazát jelöli. Ellenkező esetben ugyanis

$$\delta_f(s,v) \leqslant \delta_f(s,u) + 1 \leqslant \delta_{f'}(s,u) + 1 = \delta_{f'}(s,v),$$

ami ellentmond a $\delta_f(s,v) > \delta_{f'}(s,v)$ feltevésnek.

Kaptuk tehát, hogy $(u,v) \notin E_f$ és $(u,v) \in E_{f'}$. Ez csak úgy lehetséges, hogy a kérdéses iterációs lépésben az $f(u,v)$ folyamérték szigorúan monoton csökken, következésképpen az $f(v,u)$ folyamérték szigorúan monoton nő, tehát (v,u) rajta van a javító úton. Mivel az Edmonds-Karp algoritmus mindig egy legrövidebb út mentén növeli a folyamot, ezért

$$\delta_f(s,v) = \delta_f(s,u) - 1 \leqslant \delta_{f'}(s,u) - 1 = \delta_{f'}(s,v) - 2,$$

ami ellentmond a $\delta_f(s,v) > \delta_{f'}(s,v)$ feltevésnek. Így az adott tulajdonságoknak megfelelő v csúcs nem létezhet.

Állítás. Ha a (G,s,t,c) hálózatban az Edmonds-Karp algoritmussal keresünk maximális folyamot, akkor az iterációs lépések száma $O(|V||E|)$.

Bizonyítás. Kezdjük egy elnevezéssel: a G_f reziduális hálózat P javító útján egy (u,v) élt kritikusnak nevezünk, ha $c_f(u,v) = c_f(P)$. Világos, hogy miután növeljük a folyamot a P javító út mentén, a javító út minden kritikus éle kikerül a reziduális hálózatból. Az is világos, hogy minden javító úton van legalább egy kritikus él.

Tekintsük a $\bar{G} = (V, \bar{E})$ irányított gráfot, ahol

$$\bar{E} = \{(u,v) \mid (u,v) \in E \text{ vagy } (v,u) \in E\}.$$

Vizsgáljuk meg, hogy egy $(u,v) \in \bar{E}$ él hányszor lehet kritikus az Edmonds-Karp algoritmus lefutása során! Ne feledjük, a reziduális hálózatok élei mindig $\bar{G}$ élei közül valók.

Először is jegyezzük meg, hogy amikor (u, v) kritikus él, akkor

$$\delta_f(s, v) = \delta_f(s, u) + 1,$$

hiszen az Edmonds-Karp algoritmus mindig egy legrövidebb út mentén növeli a folyamot. A folyamnövelés után (u, v) kikerül a reziduális hálózatból. Ezután (u, v) csak akkor jelenhet meg ismét egy reziduális hálózatban, ha a folyamérték (u, v)-n szigorúan monoton csökken, ami viszont csak akkor lehetséges, ha (v, u)-n szigorúan monoton nő, vagyis (v, u) megjelenik egy javító úton. Legyen a folyam f' amikor ez legközelebb bekövetkezik. Ekkor

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1.$$

Az előző állítás szerint $\delta_f(s, v) \leqslant \delta_{f'}(s, v)$, ezért

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1 \geqslant \delta_f(s, v) + 1 = \delta_f(s, u) + 2.$$

Ez azt jelenti, hogy az (u, v) él kétszeri kritikussá válása között az s-ből u-ba vezető legrövidebb út hossza legalább kettővel nőtt. Kezdetben a legrövidebb út hossza legalább 0, és amíg u elérhetetlenné nem válik s-ből, a legrövidebb út hossza soha nem lehet nagyobb, mint $|V|-2$. Ezért (u, v) legfeljebb $[|V|/2]$ alkalommal lehet kritikus él.

Mivel $|\bar{E}| \leqslant 2|E|$, azért az Edmonds-Karp algoritmus lefutása során a kritikus élek teljes száma $O(|V||E|)$. Figyelembe véve, hogy minden javító úton van legalább egy kritikus él, az állítás adódik.

Ebből következik, hogy ha egy (G, s, t, c) hálózatban az Edmonds-Karp algoritmussal keresünk maximális folyamot, akkor az eljárás teljes költsége $O(|V||E|^2)$.

Hálózatok kritikus élei

Feladat.
Legyen $G = (V, E)$ egy hálózat az s forrással és t nyelővel, valamint a c kapacitásfüggvénnyel. A hálózat egy $e \in E$ élét kritikusnak nevezzük, ha az él kapacitását csökkentve a hálózat maximális folyamának értéke is csökkenni fog. Adjunk hatékony algoritmust a hálózat egy kritikus élének megkeresésére!

Megoldás.
Először határozzuk meg a hálózat egy (S, T) minimális vágását. Ehhez keressük meg a hálózat egy maximális f folyamát, majd a hálózat csúcsait aszerint

osszuk S-be, illetve T-be, hogy a G_f reziduális hálózatban elérhetők-e s-ből, vagy nem.

Állítjuk, hogy a hálózat bármely olyan (u, v) éle amelyre $u \in S$ és $v \in T$ kritikus. Valóban, ha csökkentjük az (u, v) él kapacitását, azzal csökkentjük az (S, T) vágás kapacitását is. Ez azt jelenti, hogy az új hálózatnak (amelyet az (u, v) él kapacitásának csökkentésével kapunk) van olyan vágása, amelynek kapacitása kisebb, mint az eredeti hálózat minimális vágásának kapacitása, következésképpen az új hálózatban a maximális folyamérték kisebb, mint az eredeti hálózatban a maximális folyamérték.

Maximális párosítás páros gráfokban

Feladat.
Egy $G = (V, E)$ gráfot páros gráfnak nevezünk, ha a V csúcshalmaz felosztható két diszjunkt V_1 és V_2 részre úgy, hogy minden él ezen két halmaz között fut, vagyis tetszőleges $(x, y) \in E$ esetén $x \in V_1$ és $y \in V_2$ vagy fordítva. Páros gráfokra ezzel összhangban használni fogjuk a $G = ((V_1, V_2), E)$ jelölést is.

Egy $G = ((V_1, V_2), E)$ páros gráf E élhalmazának egy E' részhalmazát G egy párosításának nevezzük, ha a $G' = ((V_1, V_2), E')$ gráfban minden csúcs foka legfeljebb 1. Egy $G = ((V_1, V_2), E)$ páros gráf egy E' párosítását maximálisnak nevezzük, ha G bármely E'' párosítására $|E''| \leqslant |E'|$.

A feladat ezek után a következő. Adott egy $G = ((V_1, V_2), E)$ páros gráf. Határozzuk meg G egy maximális párosítását!

Megoldás.
Az egyszerűség kedvéért tegyük fel, hogy G-ben nincs izolált csúcs. Készítsünk G-ből hálózatot a következőképpen.

- Vegyünk fel egy új s és egy új t csúcsot, ezek lesznek a hálózat forrása, illetve nyelője.
- Az s csúcsból vezessünk éleket V_1 csúcsaiba, V_2 csúcsaiból pedig vezessünk éleket a t csúcsba.
- Irányítsuk az E-beli éleket V_1-től V_2 felé.
- Minden él kapacitása legyen 1.

Jegyezzük meg, hogy a hálózat élszáma $|V_1| + |E| + |V_2| = O(|E|)$, hiszen G minden csúcsára illeszkedik legalább egy él.

Állítás. Az így kapott hálózatban a maximális folyam értéke megegyezik a G-beli maximális párosítások élszámával.

Bizonyítás. Legyen először f egy csupa egész értékű maximális folyam (ilyen létezik, hisz az élkapacitások mind egészek), és legyen $|f| = k$. Bármely $u \in V_1$ csúcsba legfeljebb egységnyi folyam folyhat be, ezért legfeljebb egy olyan u-ból induló $(u, v) \in E$ él lehet, amelyen $f(u, v) = 1$; a többi u-ból induló élen $f(u, v) = 0$. Hasonlóan, bármely $v \in V_2$ csúcsból legfeljebb egységnyi folyam folyhat ki, ezért legfeljebb egy olyan v-be érkező $(u, v) \in E$ él lehet, amelyen $f(u, v) = 1$; a többi v-be érkező élen $f(u, v) = 0$. Ezek szerint azok az $(u, v) \in E$ élek, amelyekre $f(u, v) = 1$, egy párosítást alkotnak G-ben. A rajtuk átmenő összes folyam értéke egyrészt nyilván megegyezik az élek számával, másrészt k-val is, hiszen ez a mennyiség az $(\{s\} \cup V_1, V_2 \cup \{t\})$ vágáson áthaladó folyam $f(\{s\} \cup V_1, V_2 \cup \{t\})$ értéke. Így van a G gráfban egy k élből álló párosítás.

Másodszor tekintsük G egy maximális párosítását, legyen ebben az élek száma k. Álljon a párosítás az $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k) \in E$ élekből, ahol $u_1, u_2, \ldots, u_k \in V_1$ és $v_1, v_2, \ldots, v_k \in V_2$. Most minden $1 \leqslant i \leqslant k$ esetén az (s, u_i, v_i, t) út mentén egységnyi értékű folyamot vihetünk a hálózatban s-ből t-be; ezek összesen egy k értékű folyamot adnak.

Innen az állítás adódik.

Ezek szerint ha a G páros gráfhoz elkészítjük a fenti hálózatot, és abban a Ford-Fulkerson algoritmussal keresünk maximális folyamot, akkor a kapott f egészértékű maximális folyamból azonnal meg tudjuk határozni G egy maximális párosítását. Egy maximális párosítás mérete nyilván $O(|V|)$, ezért $|f| = O(|V|)$, így az eljárás teljes költsége $O(|V||E|)$.

König-Hall tétel

Feladat.
Tekintsünk egy $G = ((V_1, V_2), E)$ páros gráfot, amelyre $|V_1| = |V_2|$. A gráf E élhalmazának egy E' részhalmazát G egy teljes párosításának nevezzük, ha a $G' = ((V_1, V_2), E')$ gráfban minden csúcs foka pontosan 1.

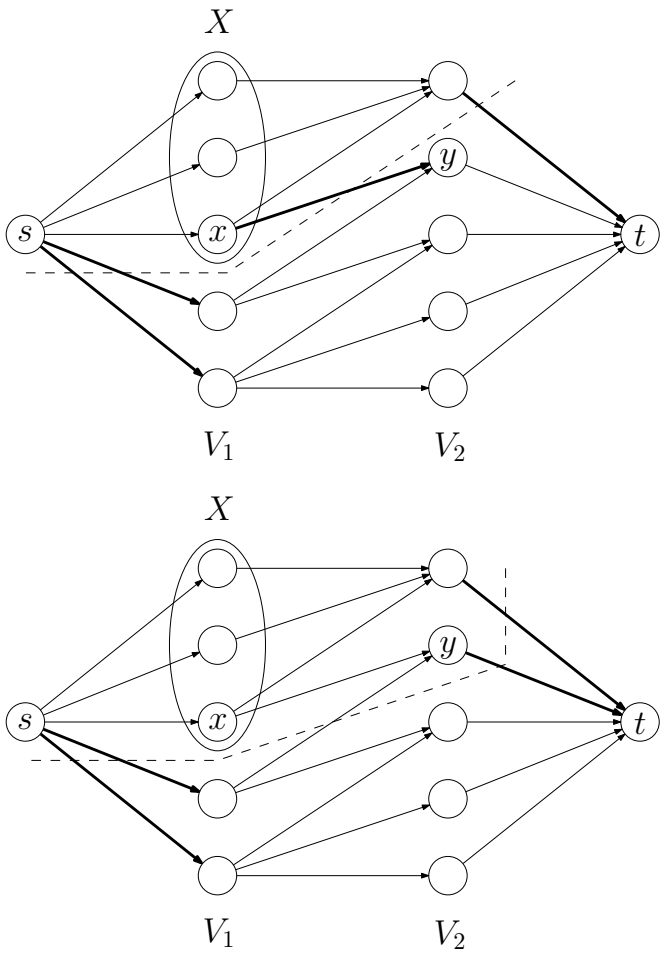
A G gráf V_1-beli csúcsainak tetszőleges X részhalmazára jelölje $\Gamma(X)$ azon V_2-beli csúcsok halmazát, amelyek szomszédosak legalább egy X-beli csúccsal. Bizonyítsuk be, hogy a G gráfban akkor és csak akkor van teljes párosítás, ha bármely $X \subseteq V_1$ esetén $|X| \leqslant |\Gamma(X)|$.

Megoldás.
A feltétel szükségessége magától értetődik. Az elégségesség bizonyításához tegyük fel, hogy a G gráfban nincs teljes párosítás. Megmutatjuk, hogy ekkor létezik olyan $X \subseteq V_1$ csúcshalmaz, amelyre $|X| > |\Gamma(X)|$. Legyen $|V_1| = |V_2| = n$.

Készítsünk G-ből hálózatot a következőképpen.

- Vegyünk fel egy új s és egy új t csúcsot, ezek lesznek a hálózat forrása, illetve nyelője.
- Az s csúcsból vezessünk éleket V_1 csúcsaiba, V_2 csúcsaiból pedig vezessünk éleket a t csúcsba.
- Irányítsuk az E-beli éleket V_1-től V_2 felé.
- Minden él kapacitása legyen 1.

Idézzük fel, hogy ebben a hálózatban a maximális folyamérték megegyezik a G-beli maximális párosítások élszámával. Feltételünk szerint G-ben nincs teljes párosítás, következésképpen a hálózatban a maximális folyamérték kisebb, mint n. Így a maximális folyam – minimális vágás tétellel összhangban van olyan (S, T) vágás a hálózatban, amelynek kapacitása kisebb, mint n. Állítjuk, hogy $X = S \cap V_1$ olyan csúcshalmaz, amelyre $|X| > |\Gamma(X)|$.



Először is vegyük észre, hogy ha valamely $y \in \Gamma(X)$ csúcs T-ben van, akkor ennek áthelyezése S-be nem növeli a vágás kapacitását. Valóban, a vágás kapacitását ugyan eggyel növeli az (y,t) él, azonban minden olyan (x,y) él, amelyre $x \in X$ a kapacitást eggyel csökkenti, és ilyen élből legalább egy mindenképpen van.

Helyezzük át az összes $y \in \Gamma(X) \cap T$ csúcsot S-be. Legyen az így kialakult vágás (S', T'). Az előbbiek szerint az (S', T') vágás kapacitása is kisebb, mint n. A hálózat V_1-ből V_2-be menő élei nyilván nem keresztezik az (S', T') vágást. Az s csúcsból induló élek közül azok keresztezik a vágást, amelyek T'-beli csúcsokhoz vezetnek, a t csúcsba érkezők közül pedig azok, amelyek S'-beli csúcsokból indulnak. Ennélfogva

$$c(S', T') = |V_1 \cap T'| + |V_2 \cap S'|,$$

Mivel $|V_1 \cap T'| = n - |X|$ és $|V_2 \cap S'| \geqslant |\Gamma(X)|$, így

$$n > c(S', T') = |V_1 \cap T'| + |V_2 \cap S'| \geqslant n - |X| + |\Gamma(X)|,$$

ahonnan $|X| > |\Gamma(X)|$ adódik.

Minimális és maximális költségű teljes párosítás

Feladat.
Tekintsünk egy $G = ((V_1, V_2), E)$ páros gráfot, amelyre $|V_1| = |V_2|$. Az egyszerűség kedvéért tegyük fel, hogy G-ben nincs izolált csúcs. A gráf E élhalmazának egy M részhalmazát G egy párosításának nevezzük, ha a $G' = ((V_1, V_2), M)$ gráfban minden csúcs foka legfeljebb 1. A gráf E élhalmazának egy M részhalmazát G egy teljes párosításának nevezzük, ha a $G' = ((V_1, V_2), M)$ gráfban minden csúcs foka pontosan 1.

Tegyük fel, hogy adott az élek halmazán egy $c\colon E \to \mathbb{R}_0^+$ nem negatív költségfüggvény. Egy M párosítás $c(M)$ költsége legyen a benne szereplő élek költségének összege.

(A) Adjunk hatékony algoritmust, amely meghatározza a G gráf egy minimális költségű teljes párosítását (ha van ilyen)!

(B) Adjunk hatékony algoritmust, amely meghatározza a G gráf egy maximális költségű teljes párosítását (ha van ilyen)!

Megoldás.
(A) Tekintsük a G gráf egy M párosítását. Készítsük el ehhez a G_M súlyozott élű, irányított gráfot a következőképpen.

- Vegyünk fel egy új s és egy új t csúcsot. Az s csúcsból vezessünk éleket V_1 azon csúcsaiba, amelyek nem végpontjai egyetlen M-beli élnek sem, valamint a t csúcsba vezessünk éleket V_2 azon csúcsaiból, amelyek nem végpontjai egyetlen M-beli élnek sem.

- Irányítsuk az E élhalmaz M-hez nem tartozó éleit V_1-től V_2 felé, M-hez tartozó éleit pedig V_2-től V_1 felé.

- Az s csúcsból induló, illetve a t csúcsba befutó élek súlya legyen 0. Ha egy $e \in E$ élt V_1-től V_2 felé irányítottunk, akkor az irányított él súlya legyen $c(e)$, míg ha V_2-től V_1 felé, akkor legyen $-c(e)$.

A G_M gráfot az M párosításhoz tartozó reziduális gráfnak nevezzük. A G_M reziduális gráf egy (u, v) irányított élének súlyát jelölje $w(u, v)$.

Tekintsünk a G_M reziduális gráfban egy s-ből t-be menő P irányított (egyszerű) utat. Egy ilyen utat javító útnak fogunk nevezni, mivel ha M-ből elhagyjuk azokat az éleket, amelyeket V_2-től V_1 felé irányítottunk és rajta vannak P-n, illetve hozzávesszük azokat az éleket, amelyeket V_1-től V_2 felé irányítottunk és rajta vannak P-n, akkor egy M elemszámánál eggyel nagyobb elemszámú M' párosítást kapunk G-ben. Jegyezzük meg, hogy itt $c(M') = c(M) + w(P)$, ahol $w(P)$ a P javító úton lévő G_M-beli irányított élek súlyainak összegét jelöli.

Algoritmusunk ezek után a következő: az üres párosításból kiindulva lépésenként növeljük a párosítást mindig egy minimális összsúlyú javító út mentén mindaddig, amíg a reziduális gráfban található javító út.

Az algoritmus helyességét több lépésben bizonyítjuk. Először azt mutatjuk meg, hogy ha a G gráfban van teljes párosítás, akkor az algoritmus teljes párosítást szolgáltat.

Állítás. Legyen M egy párosítása a G gráfnak. Ha most a G_M reziduális gráfban nincs javító út, akkor M maximális párosítása a G gráfnak.

Bizonyítás. Legyen M' a G gráf egy maximális párosítása. Tekintsük az $M^* = (M \setminus M') \cup (M' \setminus M)$ élhalmazt. Vegyük észre, hogy G bármely csúcsára legfeljebb két M^*-beli él illeszkedhet. Ebből következik, hogy az M^*-beli élek csúcsdiszjunkt utakat és köröket alkotnak G-ben, amelyek felváltva tartalmaznak M-beli és M'-beli éleket.

A körök magától értetődően páros hosszúságúak, vagyis mindkét párosításból ugyanannyi él szerepel bennük.

Foglalkozzunk ezután az utakkal. Megmutatjuk, hogy az utak első és utolsó éle nem tartozhat ugyanahhoz a párosításhoz. Ebből következik, hogy az utak is páros hosszúságúak, vagyis mindkét párosításból ugyanannyi él

szerepel bennük. Az nyilván nem lehetséges, hogy egy út első és utolsó éle M-beli legyen, hiszen ekkor M'-ből elhagyva az úton levő M'-beli éleket és hozzávéve az úton levő M-beli éleket egy M' elemszámánál nagyobb elemszámú párosítást kapnánk, ami ellentmond M' maximalitásának. De az sem lehetséges, hogy egy út első és utolsó éle M'-beli legyen. Valóban, egy ilyen útnak megfelelő G_M-beli irányított utat kiegészítve az s csúcsból az út első csúcsába futó éllel, valamint az út utolsó csúcsából a t csúcsba futó éllel G_M egy javító útjához jutnánk, ami ismét ellentmondás.

Így $|M'| = |M|$, következésképpen M szintén maximális párosítás.

A továbbiakban feltesszük, hogy a gráfban van teljes párosítás. Az algoritmus helyességének belátásához meg kell még mutatni, hogy az algoritmus által szolgáltatott teljes párosítás minimális költségű. Ehhez szükségünk lesz a következő észrevételre.

Állítás. Legyen M egy teljes párosítása a G gráfnak. Most M akkor és csak akkor minimális költségű teljes párosítás, ha nincs negatív összsúlyú irányított kör a G_M reziduális gráfban.

Bizonyítás. Először megmutatjuk, hogy ha van negatív összsúlyú irányított kör a G_M reziduális gráfban, akkor M nem minimális költségű teljes párosítás. Mivel M teljes párosítás, ezért a G_M reziduális gráfban az s csúcsból nem indul, a t csúcsba pedig nem érkezik egyetlen él sem. Legyen C egy negatív összsúlyú irányított kör a G_M reziduális gráfban. Nyilván az s és t csúcsok nincsenek rajta a C körön.

Vegyük észre, hogy ha M-ből elhagyjuk azokat az éleket, amelyeket V_2-től V_1 felé irányítottunk és rajta vannak C-n, illetve hozzávesszük azokat az éleket, amelyeket V_1-től V_2 felé irányítottunk és rajta vannak C-n, akkor ismét egy teljes párosítást kapunk. Jelölje M' ezt a teljes párosítást. Világos, hogy az M' teljes párosításra $c(M') = c(M) + w(C)$, ahol $w(C)$ a C körön lévő G_M-beli irányított élek súlyainak összegét jelöli. Mivel $w(C) < 0$, ezért $c(M') < c(M)$, amiből az állítás következik.

Ezután belátjuk, hogy ha nincs negatív összsúlyú irányított kör a G_M reziduális gráfban, akkor M minimális költségű teljes párosítás. Indirekt tegyük fel, hogy van olyan M' teljes párosítása a G gráfnak, amelyre $c(M') < c(M)$. Tekintsük az $M^* = (M \setminus M') \cup (M' \setminus M)$ élhalmazt. Vegyük észre, hogy ha G valamely csúcsára illeszkedik M^*-beli él, akkor pontosan két ilyen él illeszkedik rá. Ebből következik, hogy az M^*-beli élek csúcsdiszjunkt köröket alkotnak G-ben, amelyek felváltva tartalmaznak M-beli és M'-beli éleket. Tekintsük az ezeknek megfelelő csúcsdiszjunkt irányított köröket G_M-ben. Ezen irányított körök összsúlya $c(M') - c(M) < 0$, ami viszont csak úgy

lehetséges, hogy az irányított körök legalább egyike negatív összsúlyú, ellentmondás.

Az előző állítással összhangban az algoritmus helyességének belátásához elég megmutatni, hogy az algoritmus során kapott párosításokhoz tartozó reziduális gráfok egyike sem tartalmaz negatív összsúlyú irányított kört.

A G gráf minden $v \in V_1 \cup V_2$ csúcsához rendeljünk hozzá egy $\pi(v)$ potenciálértéket. Azt mondjuk, hogy a $\pi\colon V_1 \cup V_2 \to \mathbb{R}$ potenciálfüggvény kompatibilis a G gráf egy M párosításával, ha

- minden olyan $v \in V_1$ csúcsra, amely nem végpontja valamelyik M-beli élnek $\pi(v) = 0$,
- minden olyan (u, v) élre ($u \in V_1$ és $v \in V_2$), amely nem tartozik hozzá M-hez $\pi(u) + c(u, v) - \pi(v) \geqslant 0$,
- minden olyan (u, v) élre ($u \in V_1$ és $v \in V_2$), amely hozzátartozik M-hez $\pi(u) + c(u, v) - \pi(v) = 0$.

Ezek után legyen M egy párosítás a G gráfban, és tegyük fel, hogy létezik az M párosítással kompatibilis π potenciálfüggvény. Definiáljuk a G_M reziduális gráf tetszőleges (u, v) irányított élének a $w_\pi(u, v)$ redukált élsúlyát a következőképpen:

- ha $u = s$ vagy $v = t$, akkor legyen $w_\pi(u, v) = 0$,
- különben legyen $w_\pi(u, v) = \pi(u) + w(u, v) - \pi(v)$.

Vegyük észre, hogy a G_M reziduális gráfban a $w_\pi(u, v)$ redukált élsúlyok mind nem negatívak. Ha $u = s$ vagy $v = t$, akkor ez triviálisan igaz. Ha $u \in V_1$ és $v \in V_2$, akkor a G gráfban az (u, v) él nem tartozik hozzá M-hez, így a π potenciálfüggvénynek az M párosítással való kompatibilitása miatt $\pi(u) + c(u, v) - \pi(v) \geqslant 0$. Esetünkben

$$\pi(u) + c(u, v) - \pi(v) = \pi(u) + w(u, v) - \pi(v) = w_\pi(u, v),$$

amiből $w_\pi(u, v) \geqslant 0$ következik. Ha pedig $u \in V_2$ és $v \in V_1$, akkor a G gráfban a (v, u) él hozzátartozik M-hez, így a π potenciálfüggvénynek az M párosítással való kompatibilitása miatt $\pi(v) + c(v, u) - \pi(u) = 0$. Ebben az esetben

$$\pi(v) + c(v, u) - \pi(u) = \pi(v) - w(u, v) - \pi(u) = -w_\pi(u, v),$$

ahonnan $w_\pi(u, v) = 0$ adódik.

Tekintsük most a G_M reziduális gráf egy tetszőleges C irányított körét. Ekkor

$$w(C) = \sum_{e \in C} w(e) = \sum_{e \in C} w_\pi(e),$$

hiszen a jobb oldali összegben a potenciálértékeknek megfelelő tagok kiejtik egymást. Mivel a jobb oldali összegben szereplő redukált élsúlyok mind nem negatívak, ezért a C irányított kör is nem negatív összsúlyú.

A redukált élsúlyok bevezetésének másik előnye, hogy segítségükkel hatékonyan kereshetünk minimális összsúlyú javító utat a reziduális gráfban.

Legyen M egy párosítás a G gráfban, és tegyük fel, hogy létezik az M párosítással kompatibilis π potenciálfüggvény. Tekintsük a G_M reziduális gráfot. Dijkstra algoritmusával határozzuk meg a G_M reziduális gráfban az s csúcsból a $V_1 \cup V_2$-beli csúcsokba menő, a redukált élsúlyok szerinti legrövidebb utakat (ne feledjük, a redukált élsúlyok nem negatívak). Egy $w \in V_1 \cup V_2$ csúcsba menő, a redukált élsúlyok szerinti legrövidebb út hosszát jelölje $d_\pi(w)$.

Legyen P a G_M reziduális gráfban egy, a nem redukált élsúlyok szerinti legrövidebb javító út és legyen y a P út utolsó előtti csúcsa. Ekkor y egy olyan V_2-beli csúcs, amely nem végpontja egyetlen M-beli élnek sem. Jelölje a P út y-ig terjedő szakaszát P'. Világos, hogy P' egy s-ből y-ba menő, a nem redukált élsúlyok szerinti legrövidebb út a G_M reziduális gráfban.

Vegyük észre, hogy a G_M reziduális gráf bármely s-ből y-ba menő P^* útjára

$$w(P^*) = \sum_{e \in P^*} w(e) = \left(\sum_{e \in P^*} w_\pi(e) \right) + \pi(y) = w_\pi(P^*) + \pi(y).$$

Ebből következik, hogy ha P^* minimális összúlyú a nem redukált élsúlyok szerint, akkor minimális összsúlyú a redukált élsúlyok szerint is, és viszont.

Mivel P' egy s-ből y-ba menő, a nem redukált élsúlyok szerinti legrövidebb út a G_M reziduális gráfban, ezért az előbbiekkel összhangban a redukált élsúlyok szerint is az. Így $w(P) = w(P') = d_\pi(y) + \pi(y)$.

Ezek után a G_M reziduális gráfban egy, a nem redukált élsúlyok szerinti legrövidebb javító út a következőképpen található meg. Az összes olyan V_2-beli csúcsra, amely nem végpontja egyik M-beli élnek sem meghatározunk egy, az s csúcsból hozzá vezető, a redukált élsúlyok szerinti legrövidebb utat. Amelyik ilyen út y végpontjára $d_\pi(y)+\pi(y)$ minimális, az kiegészítve az (y, t) éllel egy, a kívánalmaknak megfelelő javító út lesz.

Egyetlen adósságunk van még: a potenciálfüggvények megadása. Kezdetben $M = \emptyset$. Egy ezzel kompatibilis potenciálfüggvény a következő: minden

$u \in V_1$ csúcsra legyen $\pi(u) = 0$, és minden $v \in V_2$ csúcsra legyen $\pi(v)$ azon G-beli élek költségének a minimuma, amelyek egyik végpontja v.

Legyen ezután M egy párosítás a G gráfban, és tegyük fel, hogy létezik az M párosítással kompatibilis π potenciálfüggvény. A fentebb leírtak szerint határozzuk meg a G_M reziduális gráf egy minimális összsúlyú P javító útját, és növeljük az M párosítást a P javító út mentén. Legyen az így kapott párosítás M'. Állítjuk, hogy a

$$\pi' \colon V_1 \cup V_2 \to \mathbb{R}, \quad v \mapsto d_\pi(v) + \pi(v)$$

potenciálfüggvény kompatibilis a G gráf M' párosításával.

Először tekintsük a G gráf egy olyan $v \in V_1$ csúcsát, amely nem végpontja egyik M'-beli élnek sem. Mivel v nem lehetett végpontja egyik M-beli élnek sem, ezért $\pi'(v) = d_\pi(v) + \pi(v) = 0 + 0 = 0$.

Foglalkozzunk ezután a G gráf éleivel. Legyen $(u, v) \in E$, ahol $u \in V_1$ és $v \in V_2$. Ha $(u, v) \in M$, akkor a G_M reziduális gráfban az u csúcsba csak a (v, u) irányított él érkezik be, így $d_\pi(u) = d_\pi(v) + w_\pi(v, u)$. Itt

$$w_\pi(v, u) = \pi(v) + w(v, u) - \pi(u) = \pi(v) - c(u, v) - \pi(u),$$

ezért

$$(d_\pi(u) + \pi(u)) + c(u, v) - (d_\pi(v) + \pi(v)) = 0,$$

azaz

$$\pi'(u) + c(u, v) - \pi'(v) = 0.$$

Ha $(u, v) \in M' \setminus M$, akkor az (u, v) irányított él rajta van a G_M reziduális gráf P minimális összsúlyú javító útján. Ebből következik, hogy $d_\pi(v) = d_\pi(u) + w_\pi(u, v)$. Itt

$$w_\pi(u, v) = \pi(u) + w(u, v) - \pi(v) = \pi(u) + c(u, v) - \pi(v),$$

ezért

$$(d_\pi(u) + \pi(u)) + c(u, v) - (d_\pi(v) + \pi(v)) = 0,$$

azaz

$$\pi'(u) + c(u, v) - \pi'(v) = 0.$$

Végül ha $(u, v) \notin M$, akkor a G_M reziduális gráf tartalmazza az (u, v) irányított élet, ezért $d_\pi(v) \leqslant d_\pi(u) + w_\pi(u, v)$. Itt

$$w_\pi(u, v) = \pi(u) + w(u, v) - \pi(v) = \pi(u) + c(u, v) - \pi(v),$$

ezért

$$(d_\pi(u) + \pi(u)) + c(u, v) - (d_\pi(v) + \pi(v)) \geqslant 0,$$

azaz

$$\pi'(u) + c(u,v) - \pi'(v) \geqslant 0.$$

Ezzel beláttuk, hogy ha $(u,v) \in M'$, akkor $\pi'(u) + c(u,v) - \pi'(v) = 0$, ha pedig $(u,v) \notin M'$, akkor $\pi'(u) + c(u,v) - \pi'(v) \geqslant 0$. Így annak bizonyítása, hogy a π' potenciálfüggvény kompatibilis az M' párosítással teljes.

Nem nehéz igazolni, hogy az algoritmus költsége $O(|V|^3)$. Ha a G gráf tartalmaz teljes párosítást, akkor az iterációk száma $|V|$, hiszen minden iterációban a párosítás elemszáma eggyel nő. Ha a G gráf nem tartalmaz teljes párosítást, akkor az iterációk száma magától értetődően kisebb, mint $|V|$.

Minden iterációban lefuttatjuk egyszer Dijkstra algoritmusát. Ennek költsége $O(|V|^2)$. Ezután jön egy legrövidebb javító út meghatározása, a párosítás növelése és a potenciálfüggvény frissítése. Mivel ezek együttes költsége is csupán $O(|V|)$, így egy iteráció költsége $O(|V|^2)$. Innen az algoritmus összköltségére vonatkozó állítás adódik.

(B) A feladat egyszerűen visszavezethető a minimális költségű teljes párosítás feladatra. A G gráf élein adott $c\colon E \to \mathbb{R}_0^+$ költségfüggvény helyett tekintsük a

$$c'\colon E \to \mathbb{R}_0^+, \quad e \mapsto C - c(e)$$

költségfüggvényt, ahol $C = \max\{c(e) \mid e \in E\}$. Ekkor G egy minimális költségű teljes párosítása a c' költségfüggvényre nézve G egy maximális költségű teljes párosítása a c költségfüggvényre nézve.

Stabil párosítás

Feladat.
Tekintsük fiúk egy F és lányok egy L halmazát. Tegyük fel, hogy a két halmaz ugyanolyan elemszámú. Most az $F \times L = \{(f,l) \mid f \in F,\ l \in L\}$ halmaz egy S részhalmazát teljes párosításnak nevezzük, ha minden F-beli fiú és minden L-beli lány pontosan egy S-beli rendezett párban fordul elő.

Minden $f \in F$ fiú rangsorolja az összes L-beli lányt; az f fiú rangsorában egy l lány pontosan akkor előz meg egy l' lányt, ha f-nek jobban tetszik l, mint l' (holtverseny nincs). A lányok ugyanígy rangsorolják a fiúkat.

Legyen S egy teljes párosítást, továbbá legyenek (f,l) és (f',l') olyan S-beli párok, amelyek esetén f-nek jobban tetszik l', mint l, és l'-nek jobban tetszik f, mint f'. Ilyenkor azt mondjuk, az (f,l') pár instabilitást jelent S-re nézve. Célunk olyan S teljes párosítás megadása, amelyre nézve nincs instabilitási tényező; egy ilyen párosítást stabilnak nevezünk.

Megoldás.
Az algoritmus szemléletesen a következőképpen írható le.

- Kezdetben senkinek nincs párja.
- Amíg van olyan fiú, akinek éppen nincs párja és még nem tett ajánlatot minden lánynak ismételjük a következőt. Vegyünk egy ilyan f fiút. Legyen l az a lány, aki az f fiúnak a legjobban tetszik azok közül, akiknek még nem tett ajánlatot. Ajánlja fel az f fiú az l lánynak, hogy legyenek egy pár. Ha az l lánynak éppen nincs párja, akkor elfogadja az ajánlatot és ettől kezdve egy párt alkotnak. Kicsit bonyolultabb a helyzet ha az l lánynak már van párja, mondjuk az f' fiú. Ha az f' fiú jobban tetszik az l lánynak, mint az f fiú, akkor l kosarat ad f-nek, a párok nem változnak. Ellenkező esetben az l lány az f' fiúnak int búcsút (ekkor is azt fogjuk mondani, hogy a lány kikosarazta a fiút) és új párja az f fiú lesz.
- Ha már nincs olyan fiú, akinek nincs párja és még nem tett ajánlatot az összes lánynak, akkor az algoritmus befejeződik. A kialakult párokat véglegesítjük; akinek ekkor nincs párja, az egyedül marad.

Az egyszerűség kedvéért legyen $F = \{1, 2, \ldots, n\}$ és $L = \{1, 2, \ldots, n\}$. A fiúk preferencialistáit egy FiúPref$[1 : n, 1 : n]$ tömbben tároljuk, itt FiúPref$[f, i]$ az a lány, aki az f fiú preferencialistáján az i-edik helyen áll. Hasonlóan, a lányok preferencialistáit egy LányPref$[1 : n, 1 : n]$ tömbben tároljuk, itt LányPref$[l, i]$ az a fiú, aki az l lány preferencialistáján az i-edik helyen áll.

Használunk még egy InvLányPref$[1 : n, 1 : n]$ tömböt, InvLányPref$[l, f]$ azt mutatja meg, hogy az f fiú hányadik helyen áll az l lány preferencialistáján. Az InvLányPref$[1 : n, 1 : n]$ tömb egyszerűen felépíthető a LányPref$[1 : n, 1 : n]$ tömbből.

Azokat a fiúkat, akiknek éppen nincs párja egy Q sorban tároljuk. Az ajánlat$[1 : n]$ tömb ajánlat$[f]$ eleme azt mutatja meg, hogy az f fiú eddig hány lánynak tett ajánlatot.

Végül a pár$[1 : n]$ tömb pár$[l]$ eleme az a fiú, aki az l lány aktuális párja. Ha az l lánynak éppen nincs párja, akkor pár$[l] = 0$.

```
StabilPárosítás(FiúPref[1:n,1:n],LányPref[1:n,1:n])
for i=1 to n do
  for l=1 to n do
    InvLányPref[l,LányPref[l,i]]=i
```

```
for l=1 to n do
  pár[l]=0
Sorinicializálás(Q)
for f=1 to n do
  ajánlat[f]=0
  Sorba(Q,f)
while Q<>EMPTYSET do
  f=Sorból(Q)
  ajánlat[f]=ajánlat[f]+1
  l=FiúPref[f,ajánlat[f]]
  if pár[l]=0
    then
      pár[l]=f
    else
      f'=pár[l]
      if InvLányPref[l,f]>InvLányPref[l,f']
        then
          pár[l]=f
          if ajánlat[f']<n then Sorba(Q,f')
        else
          if ajánlat[f]<n then Sorba(Q,f)
for l=1 to n do
  if pár[l]<>0 then
    print '(' l ',' pár[l] ')'
```

Mekkora az algoritmus költsége? Az inicializálás költsége nyilván $O(n^2)$. A while ciklus iterációinak száma legfeljebb n^2, hiszen minden iterációban egy fiú ajánlatot tesz egy olyan lánynak, akinek addig még nem tett ajánlatot és a lehetséges (fiú, lány) párok száma n^2. A while ciklus egy iterációja világos módon konstans költségű, ezért a while ciklus költsége is $O(n^2)$. Így az algoritmus teljes költsége $O(n^2)$.

Lássuk ezután az algoritmus helyességének bizonyítását. Néhány egyszerű észrevétellel kezdünk.

(1) Ha egy fiúnak végül lesz párja, akkor ő minden olyan lánynak tett ajánlatot, aki a párjánál jobban tetszik neki.

(2) Ha egy fiúnak végül nem lesz párja, akkor ő az összes lánynak tett ajánlatot.

(3) Ha egy lánynak legalább egy fiú ajánlatot tesz, akkor a lánynak végül lesz párja.

(4) Ha egy lánynak legalább egy fiú ajánlatot tesz, akkor a lánynak az a fiú lesz végül a párja, aki az ajánlattevői közül legjobban tetszik neki.

Állítás. A `StabilPárosítás` algoritmus teljes párosítást szolgáltat.

Bizonyítás. Indirekt módon bizonyítunk. Tegyük fel, hogy az állítással ellentétben valamelyik f fiúnak nem lett párja. A (2) észrevétel szerint ekkor f minden lánynak tett ajánlatot. Ez azt jelenti, hogy minden lánynak tett ajánlatot valaki, így a (3) észrevétel szerint minden lánynak lett végül párja. Ez azonban csak úgy lehet, ha az összes fiúnak is lett végül párja, hiszen a fiúk és a lányok száma ugyanannyi. Ellentmondásra jutottunk, így az állítás igaz.

Állítás. A `StabilPárosítás` algoritmus által szolgáltatott párosításra nézve nincs instabilitási tényező.

Bizonyítás. Ismét indirekt módon bizonyítunk. Tegyük fel, hogy az állítással ellentétben van olyan (f, l) páros, amely instabilitási tényező. Jelölje f-nek és l-nek a `StabilPárosítás` algoritmus által szolgáltatott párját l' és f'.

Az (1) észrevétel szerint f ajánlatot tett l-nek még l' előtt. Mivel végül l' lett f párja, így valamikor l kikosarazta őt. A (4) észrevétel szerint l-nek végül az a fiú lett a párja, aki az ajánlattevői közül a legjobban tetszett neki, így l-nek szükségképpen jobban tetszik f', mint f. Viszont ez azt jelenti, hogy az (f, l) páros nem instabilitási tényező. Ellentmondásra jutottunk, így az állítás igaz.

Vizsgáljuk meg, hogy a `StabilPárosítás` algoritmus a fiúknak vagy a lányoknak kedvez-e inkább! Mivel a lányoknak a nekik ajánlatot tevő fiúk közül a nekik legjobban tetsző lesz a párja, míg a fiúknak azon lányok közül, akiknek ajánlatot tettek a nekik legkevésbé tetsző lesz a párja, azt gondolhatnánk, hogy a lányok járnak jobban. A valóság ennek éppen az ellenkezője.

Állítás. A `StabilPárosítás` algoritmus minden fiúhoz a számára szóba jövő lányok közül a neki legjobban tetszőt párosítja. (Egy fiú számára egy lány akkor jön szóba, ha van olyan stabil párosítás, amelyben a fiú és a lány egy párt alkotnak.)

Bizonyítás. Indirekt módon bizonyítunk. Tegyük fel, hogy az állítással ellentétben van olyan fiú, akihez a `StabilPárosítás` algoritmus a számára szóba jövő lányok közül nem a neki legjobban tetszőt párosítja.

Tekintsük a legelső olyan iterációt, amikor egy fiút a számára szóba jövő lányok közül a neki legjobban tetsző kikosaraz. Legyen a fiú f, a lány pedig l. Legyen továbbá f' az a fiú, aki miatt l kikosarazta f-et. Ekkor persze l-nek jobban tetszik f', mint f. Mivel a legelső olyan iterációban vagyunk, amikor egy fiút a számára szóba jövő lányok közül a neki legjobban tetsző kikosaraz, f'-t még biztos nem kosarazta ki a számára szóba jövő lányok közül a neki legjobban tetsző. Legyen ez utóbbi lány l^*. Világos, hogy f'-nek legalább annyira tetszik l, mint l^* (itt l és l^* nem feltétlenül különböző lányok).

Legyen S egy olyan stabil párosítás, ahol f és l egy párt alkot. Ilyen stabil párosítás létezik, hisz l az f fiú számára szóba jövő lányok között van. Legyen S-ben f' párja l'. Nyilvánvaló módon f'-nek legalább annyira tetszik l^*, mint l' (itt sem feltétlenül különböző lányok l^* és l'). Mivel f'-nek legalább annyira tetszik l, mint l^*, és legalább annyira tetszik l^*, mint l', ezért f'-nek legalább annyira tetszik l, mint l' (jegyezzük meg, hogy l és l' különböző lányok).

Így az S stabil párosítás (f, l) és (f', l') párjait tekintve azt látjuk, hogy l-nek jobban tetszik f', mint f, és f'-nek jobban tetszik l, mint l', vagyis az (f', l) páros instabilitást jelent S-re nézve. Ellentmondásra jutottunk, így az állítás igaz.

Állítás. A `StabilPárosítás` algoritmus minden lányhoz a számára szóba jövő fiúk közül a neki legkevésbé tetszőt párosítja. (Egy lány számára egy fiú akkor jön szóba, ha van olyan stabil párosítás, amelyben a lány és a fiú egy párt alkotnak.)

Bizonyítás. Ismét indirekt módon bizonyítunk. Tegyük fel, hogy az állítással ellentétben van olyan S stabil párosítás, ahol valamelyik l lány rosszabbul jár, mint a `StabilPárosítás` algoritmus esetén.

Legyen l-nek a `StabilPárosítás` algoritmus által szolgáltatott párja f, az S-beli párja pedig f'. Feltételünk szerint l-nek jobban tetszik f, mint f'. Legyen továbbá az f fiú S-beli párja az l' lány. Az előző állítás szerint a `StabilPárosítás` algoritmus f-hez a számára szóba jövő lányok közül a neki legjobban tetszőt párosítja. Ebből következik, hogy f-nek jobban tetszik l, mint l'.

Így az S stabil párosítás (f', l) és (f, l') párjait tekintve azt látjuk, hogy l-nek jobban tetszik f, mint f', és f-nek jobban tetszik l, mint l', vagyis az (f, l) páros instabilitást jelent S-re nézve. Ellentmondásra jutottunk, így az állítás igaz.

Élidegen utak irányított gráfokban

Feladat.
Legyen $G = (V, E)$ izolált csúcsok nélküli irányított gráf, és $s, t \in V$ különböző csúcsok. Adjunk hatékony algoritmust maximális számú, páronként élidegen (közös él nélküli) s-ből t-be menő irányított út megkeresésére!

Megoldás.
Tekintsük a (G, s, t, c) hálózatot, ahol minden él kapacitása 1. A kapacitások egészek, így van olyan f maximális folyam, amelyre $f(u, v)$ egész minden (u, v) csúcspárra. Ebből azonnal következik, hogy ha $(u, v) \in E$, akkor $f(u, v)$ vagy 0 vagy 1. Legyen $|f| = k$. Állítjuk, hogy ekkor van G-ben k darab páronként élidegen s-ből t-be menő irányított út.

Először azt mutatjuk meg, hogy ha $k > 0$, akkor van olyan P út s-ből t-be, amelynek élein az f folyam 1 értéket vesz fel. Induljunk el s-ből egy olyan $(s, u) \in E$ élen, amelyre $f(s, u) = 1$. Az u csúcsba érve töröljük ezt az élt, majd lépjünk tovább egy olyan $(u, v) \in E$ élen, amelyre $f(u, v) = 1$, és töröljük ezt az élt is. Így folytatva sosem akadhatunk el egy t-től különböző csúcsban, hiszen a hálózat s-től és t-től különböző csúcsaiból ugyanannyi egységnyi folyamot szállító él indul ki, mint amennyi befut oda, s-ből pedig több. Előbb-utóbb t-be érünk, hiszen az élek fogynak.

Ha tehát $k > 0$, akkor nézzünk egy az előbbiek szerinti P utat. Töröljük G-ből a P éleit, és feledkezzünk meg a rajtuk átmenő egységnyi folyamról. A kapott gráfban egy $k - 1$ értékű folyam marad, amire $k - 1 > 0$ esetén ismételjük meg a fenti útkeresést. Így folytatva végül k darab páronként élidegen s-ből t-be menő irányított utat kapunk. A maximális folyam értéke tehát legfeljebb akkora, mint a páronként élidegen s-ből t-be menő irányított utak maximális elemszámú rendszereinek elemszáma.

Megfordítva, ha $P_1, P_2, \ldots, P_k$ maximális számú páronként élidegen s-ből t-be menő irányított út G-ben, akkor ezek mindegyikén egységnyi folyamot küldhetünk a forrásból a nyelőbe, ami összesen egy k értékű folyamot jelent. A maximális folyam értéke tehát legalább akkora, mint a páronként élidegen s-ből t-be menő irányított utak maximális elemszámú rendszereinek elemszáma.

Ebből következik, hogy a maximális folyam értéke megegyezik a páronként élidegen s-ből t-be menő irányított utak maximális elemszámú rendszereinek elemszámával.

A hálózatban a Ford-Fulkerson algoritmussal kereshetünk egy egészértékű maximális folyamot. Ennek költsége $O(|V||E|)$, hiszen a maximális folyam értéke nem haladhatja meg az s-ből kimenő élek számát, ami $O(|V|)$. A maximális folyam ismeretében ezek után az élidegen utak a fenti gondolatmenetet

követve szélességi vagy mélységi keresések alkalmazásával megkonstruálhatók. A költség itt is $O(|V||E|)$. Így az algoritmus összköltsége $O(|V||E|)$.

Menger tétel

Feladat.
Legyen $G = (V, E)$ irányított gráf és $s, t \in V$ különböző csúcsok. Mutassuk meg, hogy az s-ből t-be menő, páronként élidegen (közös él nélküli) irányított utak maximális száma megegyezik azon élek minimális számával, amelyek eltávolításával egyetlen s-ből t-be menő irányított út sem marad a gráfban!

Megoldás.
Ha egy $F \subseteq E$ élhalmaz eltávolításával egyetlen s-ből t-be menő irányított út sem marad a gráfban, akkor az összes s-ből t-be menő irányított úton kell lenni legalább egy F-beli élnek, így s-ből t-be menő, páronként élidegen irányított utak bármely halmazának elemszáma legfeljebb $|F|$ lehet. Ebből következik, hogy az s-ből t-be menő, páronként élidegen irányított utak maximális száma kisebb vagy egyenlő, mint azon élek minimális száma, amelyek eltávolításával egyetlen s-ből t-be menő irányított út sem marad a gráfban.

A fordított egyenlőtlenség belátásához tekintsük a (G, s, t, c) hálózatot, ahol minden él kapacitása 1. Az előző feladatnál láttuk, hogy az s-ből t-be menő, páronként élidegen irányított utak maximális száma megegyezik ebben a hálózatban a maximális folyamértékkel. Legyen ez az érték k. A maximális folyam – minimális vágás tétel szerint ekkor van olyan (S, T) vágás a hálózatban, amelynek kapacitása szintén k. Jelölje az S-ből T-be menő élek halmazát F. Most egyrészt $|F| = k$, hiszen az élek kapacitása 1, másrészt az F-beli élek eltávolításával egyetlen s-ből t-be menő irányított út sem marad a gráfban, a vágás definíciójával összhangban. Ebből következik, hogy az s-ből t-be menő, páronként élidegen irányított utak maximális száma nagyobb vagy egyenlő, mint azon élek minimális száma, amelyek eltávolításával egyetlen s-ből t-be menő irányított út sem marad a gráfban.

Innen az állítás adódik.

Laboratóriumok és kísérletek

Feladat.
Egy kutatási projektben m laboratórium vesz részt. A projekt során n különböző kísérletet kell végrehajtani. Az egyes kísérletek elég bonyolultak, emiatt egy laboratórium legfeljebb három kísérletet tud elvégezni. A laboratóriumok felszereltsége is különböző, az egyes laboratóriumok csak bizonyos

kísérletek elvégzésére képesek. Alapvető fontosságú, hogy a kapott eredmények minél megbízhatóbbak legyenek, ezért minden kísérletet d-szer akarnak végrehajtani, mindig más laboratóriumban.

Jelölje a laboratóriumokat $L_1, L_2, \ldots, L_m$, a kísérleteket pedig $K_1, K_2, \ldots, K_n$. Jelölje továbbá minden $1 \leqslant i \leqslant m$ esetén C_i azoknak a kísérleteknek a halmazát, amelyeket az L_i laboratóriumban el tudnak végezni.

Adjunk hatékony algoritmust annak eldöntésére, hogy el lehet-e osztani a feltételeknek megfelelően a kísérleteket a laboratóriumok között! Ha a válasz igenlő, akkor az algoritmus adja meg azt is, hogy az egyes laboratóriumok mely kísérleteket végezzék el!

Megoldás.
Egy $\mathcal{H}$ hálózatot készítünk, amelyben minden $1 \leqslant i \leqslant m$ esetén az L_i laboratóriumnak megfelel egy u_i csúcs és minden $1 \leqslant j \leqslant n$ esetén a K_j kísérletnek egy v_j csúcs. Ha az L_i laboratórium el tudja végezni a K_j kísérletet, vagyis $K_j \in C_i$, akkor az u_i csúcsból menjen egy irányított él a v_j csúcsba. Vegyünk fel még egy s forrást és egy t nyelőt, és minden $1 \leqslant i \leqslant m$ esetén menjen egy irányított él s-ből u_i-be, valamint minden $1 \leqslant j \leqslant n$ esetén menjen egy irányított él v_j-ből t-be. Legyen az s csúcsból induló élek kapacitása 3, legyen a t csúcsba érkező élek kapacitása d, a többi él kapacitása pedig legyen 1.

Állítás. A kísérleteket akkor és csak akkor lehet a feltételeknek megfelelően elosztani a laboratóriumok között, ha a $\mathcal{H}$ hálózatban a maximális folyamérték nd.

Jegyezzük meg, hogy a $\mathcal{H}$ hálózatban a maximális folyamérték nem lehet nagyobb, mint nd, hiszen a hálózat $(V \setminus \{t\}, \{t\})$ vágásának a kapacitása éppen ennyi.

Bizonyítás. Tegyük fel először, hogy a kísérleteket el lehet osztani a laboratóriumok között a feltételeknek megfelelő módon. Ha az L_i laboratóriumhoz hozzárendeltük a K_j kísérletet, akkor küldjünk egységnyi folyamot az (s, u_i, v_j, t) úton s-ből t-be. Ezt minden ilyen (L_i, K_j) párra elvégezve világos módon egy nd értékű folyamot kapunk, amely az összes kapacitásfeltételt kielégíti.

Megfordítva, tegyük fel, hogy a hálózaton a maximális folyamérték nd. Ekkor létezik olyan egészértékű folyam a hálózaton, amelynek értéke nd. Ha most az (u_i, v_j) élen egységnyi folyam halad át, akkor a K_j kísérletet rendeljük hozzá az L_i laboratóriumhoz. A kapacitásfeltételek miatt így minden laboratóriumhoz legfeljebb három kísérletet rendelünk hozzá és minden kísérletet pontosan d laboratóriummal végeztetünk el.

Maximális folyamot a Ford-Fulkerson algoritmussal kereshetünk. Ennek költsége $O(|f^*||E|)$, ahol $|f^*|$ a maximális folyamérték, $|E|$ pedig a hálózat élszáma. Itt $|E| = O(mn)$ nyilvánvaló módon. Másrészt a maximális folyamérték nem lehet nagyobb, mint az $(\{s\}, V \setminus \{s\})$ vágás kapacitása, ezért $|f^*| \leqslant 3m$. Így az eljárás költsége $O(m^2 n)$.

Orvosi ügyelet

Feladat.
Egy kórház a hétvégi orvosi ügyeleteket n orvossal akarja ellátni.

- Minden orvosról tudjuk, hogy mely napokon nem tud ügyeletet vállalni.
- Minden orvos legfeljebb k napot ügyelhet összesen.
- Egy orvos egy hétvégén legfeljebb egy napot ügyelhet.
- Minden hétvége minden napjára be kell osztani egy orvost.

Adjunk hatékony algoritmust annak eldöntésére, hogy van-e az adott feltételeknek eleget tevő beosztás, és ha igen, az algoritmus adjon is meg egy ilyet!

Megoldás.
Egy $\mathcal{H}$ hálózatot készítünk, amelyben az orvosoknak feleljenek meg az $u_1, u_2, \ldots, u_n$, az ellátandó napoknak pedig a $w_1, w_2, \ldots, w_m$ csúcsok. Vegyünk fel továbbá egy v_{ij} csúcsot minden olyan esetben, amikor az i-edik orvos a j-edik hétvége legalább egy napján rendelkezésre áll. Az u_i csúcsból menjen egy 1 kapacitású él a v_{ij} csúcsba, míg a v_{ij} csúcsból induljanak 1 kapacitású élek a j-edik hétvége azon napjaihoz, amelyeken az orvos rendelkezésre áll. Vegyünk fel végül egy s forrást és egy t nyelőt. Az s forrásból vezessenek k kapacitású élek az orvosokhoz, a t forrásba pedig vezessenek 1 kapacitású élek az ellátandó napokból.

Állítás. Akkor és csak akkor létezik olyan ügyeleti beosztás, amely az összes feltételt kielégíti, ha a $\mathcal{H}$ hálózatban a maximális folyamérték m.

Jegyezzük meg, hogy a hálózatban a maximális folyamérték nem lehet nagyobb, mint m, hiszen a hálózat $(V \setminus \{t\}, \{t\})$ vágásának a kapacitása éppen ennyi.

Bizonyítás. Tegyük fel először, hogy van olyan ügyeleti beosztás, amely az összes feltételt kielégíti. Ha az i-edik orvos ügyel a j-edik hétvége valamely

l napján, akkor küldjünk egységnyi folyamot az (s, u_i, v_{ij}, w_l, t) úton s-ből t-be. Ezt minden ilyen (i, l) párra elvégezve világos módon egy m értékű folyamot kapunk, amely az összes kapacitásfeltételt kielégíti.

Megfordítva, tegyük fel, hogy a hálózaton a maximális folyamérték m. Ekkor létezik olyan egészértékű folyam a hálózaton, amelynek értéke m. Ha most a (v_{ij}, w_l) élen egységnyi folyam halad át, akkor osszuk be ügyelni az i-edik orvost a j-edik hétvége l napjára. A kapacitásfeltételek miatt így minden orvos legfeljebb k napot ügyel összesen, egy hétvégén legfeljebb egy napot, és minden hétvége minden napjára be van osztva egy orvos.

Maximális folyamot a Ford-Fulkerson algoritmussal kereshetünk. Ennek költsége $O(|f^*||E|)$, ahol $|f^*|$ a maximális folyamérték, $|E|$ pedig a hálózat élszáma. Itt $|E| = O(nm)$ nyilvánvaló módon. Másrészt a maximális folyamérték nem lehet nagyobb, mint a $(V \setminus \{t\}, \{t\})$ vágás kapacitása, ezért $|f^*| \leqslant m$. Így az eljárás költsége $O(m^2 n)$.

Étterem

Feladat.
Egy étterem a város közösségi életének felpezsdítésére vasárnaponként az ebédelni érkező társaságokat úgy ülteti le, hogy egy társaság minden tagja más asztalhoz kerüljön. Az étteremben n asztal van, ezek $k_1, k_2, \ldots, k_n$ személyesek. A következő vasárnapra m társaság jelezte a jövetelét, ezek $l_1, l_2, \ldots, l_m$ tagúak. Adjunk hatékony algoritmust annak eldöntésére, hogy van-e az adott feltételnek eleget tevő ültetés, és ha igen, az algoritmus adjon is meg egy ilyet!

Megoldás.
Egy $\mathcal{H}$ hálózatot készítünk, amelyben minden $1 \leqslant i \leqslant m$ esetén a T_i társaságnak megfelel egy u_i csúcs és minden $1 \leqslant j \leqslant n$ esetén az A_j asztalnak egy v_j csúcs. Minden $1 \leqslant i \leqslant m$ és $1 \leqslant j \leqslant n$ esetén menjen egy irányított él az u_i csúcsból a v_j csúcsba. Vegyünk fel még egy s forrást és egy t nyelőt. Minden $1 \leqslant i \leqslant m$ esetén menjen egy irányított él s-ből u_i-be, valamint minden $1 \leqslant j \leqslant n$ esetén menjen egy irányított él v_j-ből t-be. Legyen az s csúcsból az u_i csúcsba vezető él kapacitása l_i minden $1 \leqslant i \leqslant m$ esetén, legyen a v_j csúcsból a t csúcsba vezető él kapacitása k_j minden $1 \leqslant j \leqslant n$ esetén, a többi él kapacitása pedig legyen 1.

Állítás. A társaságokat akkor és csak akkor lehet a feltételeknek megfelelően leültetni, ha a $\mathcal{H}$ hálózatban a maximális folyamérték $l = l_1 + l_2 + \cdots + l_m$.

Jegyezzük meg, hogy a $\mathcal{H}$ hálózatban a maximális folyamérték nem lehet nagyobb, mint l, hiszen a hálózat $(\{s\}, V \setminus \{s\})$ vágásának a kapacitása éppen ennyi.

Bizonyítás. Tegyük fel először, hogy a társaságokat le lehet ültetni a feltételeknek megfelelő módon. Ha a T_i társaság egy tagját az A_j asztalhoz ültettük, akkor küldjünk egységnyi folyamot az (s, u_i, v_j, t) úton s-ből t-be. Ezt minden társaság minden tagjára elvégezve világos módon egy l értékű folyamot kapunk, amely az összes kapacitásfeltételt kielégíti.

Megfordítva, tegyük fel, hogy a hálózaton a maximális folyamérték l. Ekkor létezik olyan egészértékű folyam a hálózaton, amelynek értéke l. Ha most az (u_i, v_j) élen egységnyi folyam halad át, akkor a T_i társaság egy tagját ültessük az A_j asztalhoz. A kapacitásfeltételek miatt így minden asztalhoz legfeljebb annyi embert ültetünk, ahány személyes az asztal, és egy társaság semelyik két tagját nem ültetjük ugyanahhoz az asztalhoz.

Maximális folyamot a Ford-Fulkerson algoritmussal kereshetünk. Ennek költsége $O(|f^*||E|)$, ahol $|f^*|$ a maximális folyamérték, $|E|$ pedig a hálózat élszáma. Itt $|E| = O(mn)$ nyilvánvaló módon. Másrészt a maximális folyamérték nem lehet nagyobb, mint az $(\{s\}, V \setminus \{s\})$ vágás kapacitása, ezért $|f^*| \leqslant l$. Így az eljárás költsége $O(lmn)$.

Ha a maximális folyamot az Edmonds-Karp algoritmussal keressük, akkor a költség $O(|V||E|^2)$, ahol $|V|$ és $|E|$ a hálózat csúcs-, illetve élszáma. Itt $|V| = O(m+n)$ és $|E| = O(mn)$, így az eljárás költségére $O((m+n)m^2n^2)$ adódik.

Páros gráfok fokszámsorozatai

Feladat.
Legyenek $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_m$ és $q_1 \geqslant q_2 \geqslant \cdots \geqslant q_n$ olyan nem negatív egész számok, amelyekre $p_1 + p_2 + \cdots + p_m = q_1 + q_2 + \cdots + q_n$ teljesül. Mutassuk meg, hogy akkor és csak akkor létezik olyan $G = ((V, W), E)$ páros gráf, amelyben $V = \{v_1, v_2, \ldots, v_m\}$ és $W = \{w_1, w_2, \ldots, w_n\}$, továbbá $\deg(v_i) = p_i$ és $\deg(w_j) = q_j$ minden $1 \leqslant i \leqslant m$, illetve minden $1 \leqslant j \leqslant n$ esetén, ha

$$\sum_{i=1}^{m} \min\{p_i, k\} \geqslant \sum_{j=1}^{k} q_j$$

fennáll minden $1 \leqslant k \leqslant n$ esetén.

Megoldás.
Készítsük el a következő $\mathcal{H}$ hálózatot.

- A hálózat csúcsai legyenek $s, v_1, v_2, \ldots, v_m, w_1, w_2, \ldots, w_n, t$, ahol s a forrás és t a nyelő.
- Az s csúcsból vezessünk irányított éleket a $v_1, v_2, \ldots, v_m$ csúcsokba; a v_i csúcsba vezető él kapacitása legyen p_i minden $1 \leqslant i \leqslant m$ esetén.
- A $w_1, w_2, \ldots, w_n$ csúcsokból vezessünk irányított éleket a t csúcsba; a w_j csúcsból induló él kapacitása legyen q_j minden $1 \leqslant j \leqslant n$ esetén.
- Minden v_i csúcsból vezessünk irányított élt minden w_j csúcsba. Ezeknek az éleknek a kapacitása legyen 1.

Legyen $V = \{v_1, v_2, \ldots, v_m\}$ és $W = \{w_1, w_2, \ldots, w_n\}$.

Állítás. Akkor és csak akkor létezik olyan $G = ((V, W), E)$ páros gráf, amelyben a fokszámok az előírt értékek, ha a $\mathcal{H}$ hálózatban a maximális folyamérték $p_1 + p_2 + \cdots + p_m$.

Jegyezzük meg, hogy a hálózatban a maximális folyamérték nem lehet nagyobb, mint $p_1 + p_2 + \cdots + p_m$, hiszen a hálózat $(\{s\}, V \cup W \cup \{t\})$ vágásának a kapacitása éppen ennyi.

Bizonyítás. Tegyük fel először, hogy van olyan $G = ((V, W), E)$ páros gráf, amely megfelel a feltételeknek. Ha most $(v_i, w_j) \in E$, akkor küldjünk egységnyi folyamot az (s, v_i, w_j, t) úton s-ből t-be. Ezt a G gráf minden (v_i, w_j) élére végrehajtva világos módon egy $p_1 + p_2 + \cdots + p_m$ értékű folyamot kapunk, amely az összes kapacitásfeltételt kielégíti.

Megfordítva, tegyük fel, hogy a hálózaton a maximális folyamérték $p_1 + p_2 + \cdots + p_m$. Ekkor létezik olyan egészértékű folyam a hálózaton, amelynek értéke $p_1 + p_2 + \cdots + p_m$. Ha most a hálózat (v_i, w_j) élen egységnyi folyam halad át, akkor legyen $(v_i, w_j) \in E$ a G gráfban. A kapacitásfeltételek miatt így egy olyan $G = ((V, W), E)$ páros gráfhoz jutunk, amelyben a fokszámok az előírt értékek.

Mivel a $\mathcal{H}$ hálózat $(\{s\}, V \cup W \cup \{t\})$ vágásának a kapacitása $p_1 + p_2 + \cdots + p_m$, ezért a maximális folyam – minimális vágás tétellel összhangban ezek után elég megmutatni, hogy a feladatban megfogalmazott

$$\sum_{i=1}^{m} \min\{p_i, k\} \geqslant \sum_{j=1}^{k} q_j$$

minden $1 \leqslant k \leqslant n$ esetén feltétel ekvivalens azzal, hogy a $\mathcal{H}$ hálózat tetszőleges vágásának a kapacitása legalább $p_1 + p_2 + \cdots + p_m$.

Tegyük fel először, hogy a $\mathcal{H}$ hálózat tetszőleges vágásának a kapacitása legalább $p_1 + p_2 + \cdots + p_m$. Legyen $1 \leqslant k \leqslant n$ és tekintsük a hálózatnak azt az (S, T) vágását, ahol $S = \{s\} \cup \{v_i \mid p_i > k\} \cup \{w_{k+1}, w_{k+2}, \ldots, w_n\}$ és $T = \{v_i \mid p_i \leqslant k\} \cup \{w_1, w_2, \ldots, w_k\} \cup \{t\}$. Ekkor

$$\sum_{i=1}^{m} p_i \leqslant c(S,T) = \sum_{p_i \leqslant k} p_i + |\{i \mid p_i > k\}| \times k + \sum_{j=k+1}^{n} q_j,$$

ahonnan

$$\sum_{j=1}^{k} q_j \leqslant \sum_{p_i \leqslant k} p_i + |\{i \mid p_i > k\}| \times k = \sum_{i=1}^{m} \min\{p_i, k\}$$

(felhasználtuk, hogy $p_1 + p_2 + \cdots + p_m = q_1 + q_2 + \cdots + q_n$).

Megfordítva, tegyük fel, hogy

$$\sum_{i=1}^{m} \min\{p_i, k\} \geqslant \sum_{j=1}^{k} q_j$$

fennáll minden $1 \leqslant k \leqslant n$ esetén. Tekintsük a hálózat egy tetszőleges (S, T) vágását és legyen $k = |T \cap W|$. Ha $k = 0$, akkor $c(S,T) \geqslant q_1 + q_2 + \cdots + q_n = p_1 + p_2 + \ldots + p_m$ triviálisan igaz, ezért tegyük fel, hogy $k \geqslant 1$. Most

$$\begin{aligned}
c(S,T) &= \sum_{\{i \mid v_i \in T \cap V\}} p_i + |\{i \mid v_i \in S \cap V\}| \times k + \sum_{\{j \mid w_j \in S \cap W\}} q_j \\
&\geqslant \sum_{\{i \mid v_i \in T \cap V\}} p_i + \sum_{\{i \mid v_i \in S \cap V\}} \min\{p_i, k\} + \sum_{\{j \mid w_j \in S \cap W\}} q_j \\
&\geqslant \sum_{\{i \mid v_i \in T \cap V\}} p_i + \sum_{j=1}^{k} q_j - \sum_{\{i \mid v_i \in T \cap V\}} \min\{p_i, k\} + \sum_{\{j \mid w_j \in S \cap W\}} q_j \\
&\geqslant \sum_{\{i \mid v_i \in T \cap V\}} p_i + \sum_{\{j \mid w_j \in T \cap W\}} q_j - \sum_{\{i \mid v_i \in T \cap V\}} p_i + \sum_{\{j \mid w_j \in S \cap W\}} q_j \\
&= \sum_{j=1}^{n} q_j = \sum_{i=1}^{m} p_i
\end{aligned}$$

(a második egyenlőtlenségnél használtuk ki a feltételt, a harmadiknál pedig azt, hogy $q_1 \geqslant q_2 \geqslant \cdots \geqslant q_n$).

Innen az állítás adódik.

Projektek és beruházások

Feladat.
Egy cég a $P_1, P_2, \ldots, P_n$ projektek végrehajtására kap megbízási ajánlatot. A megbízó a P_i projekt végrehajtásáért p_i összeget fizetne. A projektek végrehajtásához különböző beruházások szükségesek, jelölje ezeket $B_1, B_2, \ldots, B_m$, a P_i projekthez pontosan az $R_i \subseteq \{B_1, B_2, \ldots, B_m\}$ beruházások. A cégnek b_j összegbe kerül a B_j beruházás elvégzése.

Adjunk hatékony algoritmust annak eldöntésére, hogy a cég mely projektek végrehajtását vállalja el, ha a bevételt maximalizálni akarja!

Megoldás.
Az egyszerűség kedvéért tegyük fel, hogy minden projekthez végre kell hajtani legalább egy beruházást, és nincs olyan beruházás, amely egyetlen projekthez sem szükséges. Készítsük el a következő hálózatot.

- A hálózat csúcsai legyenek $s, B_1, B_2, \ldots, B_m, P_1, P_2, \ldots, P_n, t$, ahol s a forrás és t a nyelő.
- Az s csúcsból vezessünk irányított éleket a $B_1, B_2, \ldots, B_m$ csúcsokba; a B_i csúcsba vezető él kapacitása legyen b_i minden $1 \leqslant i \leqslant m$ esetén.
- A $P_1, P_2, \ldots, P_n$ csúcsokból vezessünk irányított éleket a t csúcsba; a P_i csúcsból induló él kapacitása legyen p_i minden $1 \leqslant i \leqslant n$ esetén.
- Egy B_i csúcsból vezessünk irányított élt egy P_j csúcsba akkor és csak akkor, ha $B_i \in R_j$. Egy ilyen él kapacitása legyen $C + 1$, ahol

$$C = \sum_{k=1}^{n} p_k.$$

Az algoritmus ezek után egyszerű. Határozzuk meg a hálózat egy (S, T) minimális vágását, és vállaljuk el azokat a projekteket, amelyek T-ben vannak. Egy (S, T) minimális vágást a következőképpen találhatunk. Keressük meg a hálózat egy maximális f folyamát, majd a hálózat csúcsait aszerint osszuk S-be, illetve T-be, hogy a G_f reziduális hálózatban elérhetők-e s-ből, vagy nem. Ha a maximális folyamot az Edmonds-Karp algoritmussal keressük, akkor az algoritmus költsége $O((m+n)(m+n+r)^2)$, ahol

$$r = \sum_{k=1}^{n} |R_k|.$$

Lássuk az algoritmus helyességének a bizonyítását. Tekintsük először projekteknek és beruházásoknak egy olyan

$$\{P_{i_1}, P_{i_2}, \ldots, P_{i_h}, B_{j_1}, B_{j_2}, \ldots, B_{j_l}\}$$

halmazát, amelyben minden projekt mellett az ahhoz szükséges beruházások is szerepelnek. Tekintsük továbbá azt az (S', T') vágást, ahol $T' = \{B_{j_1}, B_{j_2}, \ldots, B_{j_l}, P_{i_1}, P_{i_2}, \ldots, P_{i_h}, t\}$.

A hálózat azon élei, amelyek beruházásokból vezetnek projektekbe, nem keresztezik a vágást, hiszen T'-ben az összes olyan beruházás benne van, amely valamelyik T'-beli projekthez szükséges.

Az s csúcsból induló élek közül azok keresztezik a vágást, amelyek T'-ben lévő beruházásokhoz vezetnek. Ezek a

$$\sum_{k=1}^{l} b_{j_k}$$

mennyiséggel járulnak hozzá a vágás kapacitásához.

A t csúcsba érkező élek közül azok keresztezik a vágást, amelyek nem T'-ben lévő projektekből indulnak. Ezek a

$$C - \sum_{k=1}^{h} p_{i_k}$$

mennyiséggel járulnak hozzá a vágás kapacitásához.

Az (S', T') vágás kapacitása ezen két mennyiség összege:

$$\sum_{k=1}^{l} b_{j_k} + C - \sum_{k=1}^{h} p_{i_k} = C - \left(\sum_{k=1}^{h} p_{i_k} - \sum_{k=1}^{l} b_{j_k} \right).$$

Vegyük észre, hogy a jobb oldalon a zárójeles mennyiség a cég bevétele abban az esetben, amikor a $P_{i_1}, P_{i_2}, \ldots, P_{i_h}$ projekteket teljesíti és a $B_{j_1}, B_{j_2}, \ldots, B_{j_l}$ beruházásokat hajtja végre. Így a bevétel akkor lesz maximális, amikor az (S', T') vágás kapacitása a lehető legkisebb.

A bizonyítás teljessé tételéhez ezek után elég megmutatni, hogy a hálózat egy (S, T) minimális vágására is teljesül, hogy T a benne levő projektek mellett az azokhoz szükséges beruházásokat is tartalmazza. Legyen (S, T) a hálózat egy minimális vágása. Mivel az

$$(\{s, B_1, B_2, \ldots, B_m, P_1, P_2, \ldots, P_n\}, \{t\})$$

vágás kapacitása C, ezért az (S, T) minimális vágás kapacitása sem lehet ennél nagyobb. Ebből viszont következik, hogy az (S, T) vágást nem keresztezheti egyetlen beruházásból projektbe vezető él sem, hiszen ezek kapacitása $C + 1$. Így ha valamelyik projekt benne van T-ben, akkor az ehhez szükséges beruházások is mind benne vannak T-ben.

Hálózatok további korlátokkal

Feladat.
Tekintsünk egy (G, s, t, c) hálózatot. Az egyszerűség kedvéért tegyük fel, hogy a hálózat bármely u és v csúcsa között legfeljebb csak az egyik irányba megy él. Speciálisan az s forrásba nem érkezik és a t nyelőből nem indul egyetlen él sem. Tegyük fel továbbá, hogy s-ből t-be nem megy közvetlen él.

Jegyezzük meg, hogy az a feltétel, mely szerint két csúcs között csak az egyik irányban mehet él, igazából nem jelent megszorítást. Ha (u, v) és (v, u) is éle a hálózatnak, akkor az utóbbit helyettesíthetjük egy (v, v^*), (v^*, u) élpárral, ahol v^* egy új csúcs. A (v, v^*) és (v^*, u) élekhez ugyanazt a kapacitást rendeljük, mint ami a (v, u) élnél szerepelt.

Tegyük fel, hogy a $c(u, v)$ kapacitások mellett, amelyek felülről korlátozzák az éleken áthaladó folyamértékeket, alsó korlátjaink is vannak az $f(u, v)$ mennyiségekre. Egy ilyen hálózatot formálisan egy (G, s, t, c, k) ötössel írhatunk le. Ebben az első négy összetevő értelmezése a régi. Az utolsó tag, a k egy a G élein értelmezett, nem negatív értékű függvény, amely az élekhez rendelt alsó korlátokat mondja meg. A korábbi folyam definíció kiegészítéseként megköveteljük, hogy G minden (u, v) élére $k(u, v) \leqslant f(u, v)$ is teljesüljön.

Adjunk hatékony algoritmust annak eldöntésére, hogy létezik-e legfeljebb d értékű folyam egy $\mathcal{H} = (G, s, t, c, k)$ hálózathoz, ahol d adott pozitív valós szám!

Megoldás.
Tekintsünk egy $\mathcal{H} = (G, s, t, c, k)$ hálózatot. Ehhez készítünk egy alsó korlátok nélküli $\mathcal{H}'$ hálózatot a következőképpen. Vegyünk fel a $G = (V, E)$ gráf csúcsai mellé egy S új forrást és egy T új nyelőt. A $\mathcal{H}'$ hálózatot csúcshalmaza $V \cup \{S, T\}$ lesz. A G éleit megtartjuk, de új kapacitásokat rendelünk hozzájuk. Az $(u, v) \in E$ él kapacitása a $\mathcal{H}'$ hálózatban legyen

$$c'(u, v) = c(u, v) - k(u, v).$$

Feltehetjük, hogy $c'(u, v) \geqslant 0$, ellenkező esetben nyilván nem létezhet folyam a $\mathcal{H}$ hálózatban. Adjunk még $\mathcal{H}'$-höz minden $v \in V$ csúcsra egy (S, v) és egy (v, T) élt, amelyek kapacitása legyen

$$c'(S, v) = \sum_{(u,v)\in E} k(u, v) \quad \text{és} \quad c'(v, T) = \sum_{(v,w)\in E} k(v, w),$$

vagyis a v-be érkező, illetve a v-ből kimenő G-beli élek alsó korlátjainak összege. Jegyezzük meg, hogy $c'(S, s) = 0$ és $c'(t, T) = 0$. Végül adjuk még $\mathcal{H}'$-höz a (t, s) élt, amelynek kapacitása legyen d.

Állítás. A $\mathcal{H} = (G, s, t, c, k)$ hálózatban akkor és csak akkor létezik legfeljebb d értékű folyam, ha a $\mathcal{H}'$ hálózatban a maximális folyamérték

$$\sum_{(u,v)\in E} k(u, v).$$

Megjegyezzük, hogy a $\mathcal{H}'$-re vonatkozó állítás egyenértékű azzal, hogy egy maximális folyam esetén az (S, v) és (v, T) éleken áthaladó folyamértékek megegyeznek az élek kapacitásaival minden $v \in V$ csúcsra, ugyanis az $(\{S\}, V \cup \{T\})$ és az $(\{S\} \cup V, \{T\})$ vágások kapacitása éppen

$$\sum_{(u,v)\in E} k(u, v).$$

Azt is jegyezzük meg, hogy az Edmonds-Karp algoritmus felhasználásával $O(|V||E|^2)$ költséggel meghatározható a maximális folyamérték a $\mathcal{H}'$ hálózaton.

Bizonyítás. Tegyük fel először, hogy $\mathcal{H}'$-ben a maximális folyamérték

$$\sum_{(u,v)\in E} k(u, v).$$

Legyen f egy maximális folyam $\mathcal{H}'$-ben. Ebből egy g folyamot konstruálunk $\mathcal{H}$-ban. Módosítjuk f-t. Egymás után véve az $(u, v) \in E$ éleket

- vonjunk le $k(u, v)$-t $f(u, T)$-ből és $f(S, v)$-ből,
- adjunk hozzá $k(u, v)$-t $f(u, v)$-hez.

A módosítások során a folyamokra vonatkozó "megmaradási" feltételt a G összes csúcsában megtartottuk: egy csúcsból kimenő, illetve oda beérkező összes folyamból mindig ugyanannyit vontunk le az egyik élen, mint amennyit hozzáadtunk egy másikon.

A módosítások után az S-hez és T-hez illeszkedő éleken áramló folyamérték 0, így ezeket figyelmen kívül hagyhatjuk. Ebből következik, hogy a G élein értelmezett

$$g(u, v) = f(u, v) + k(u, v)$$

függvény kielégíti a $\mathcal{H}$-beli folyamokra vonatkozó "megmaradási" feltételt.

Tekintsük most G egy tetszőleges (u, v) élét. A

$$0 \leqslant f(u, v) \leqslant c'(u, v) = c(u, v) - k(u, v)$$

egyenlőtlenségekből adódik, hogy

$$k(u,v) \leqslant f(u,v) + k(u,v) \leqslant c(u,v).$$

Ám itt a középen álló mennyiség nem más, mint $g(u,v)$.

Így g tényleg egy folyam a $\mathcal{H} = (G,s,t,c,k)$ hálózaton. A g folyam értéke világos módon megegyezik a $\mathcal{H}'$ hálózat f folyamának a (t,s) élen átfolyó értékével. Mivel $c'(t,s) = d$, ezért $|g| \leqslant d$.

A megfordítást úgy bizonyítjuk, hogy egy legfeljebb d értékű $\mathcal{H}$-beli g folyamból egy alkalmas f-et konstruálunk. Ezt lényegében az előző eljárás fordítottjával tehetjük meg. Először egy f' függvényt definiálunk a $\mathcal{H}'$ élein. Legyen

- $f'(t,s) = |g|$,
- $f'(u,v) = g(u,v)$ minden $(u,v) \in E$ esetén,
- $f'(S,v) = f'(v,T) = 0$ minden $v \in V$ esetén.

Az így kapott f' függvény nyilván kielégíti a $\mathcal{H}'$-beli folyamokra vonatkozó "megmaradási" feltételt. Ezután módosítjuk f'-t. Egymás után véve az $(u,v) \in E$ éleket

- adjunk $k(u,v)$-t $f'(u,T)$-hez és $f'(S,v)$-hez,
- vonjunk le $k(u,v)$-t $f'(u,v)$-ből.

Az előzőekhez hasonló módon látható, hogy az eredményül kapott f függvény egy folyam a $\mathcal{H}'$ hálózatban, amelyre

$$|f| = \sum_{(u,v)\in E} k(u,v),$$

így maximális. Ezzel a bizonyítás teljes.

Jegyezzük meg, hogy a $\mathcal{H}$ hálózatban

$$d = \sum_{(v,t)\in E} c(v,t)$$

felső korlát bármely folyam értékére, így ezzel a d-vel futtatva az algoritmust eldönthetjük, hogy létezik-e folyam $\mathcal{H}$-ban.

És még egy fontos észrevétel. Tegyük fel, hogy a $\mathcal{H} = (G,s,t,c,k)$ hálózatban a kapacitások és az alsó korlátok mind nem negatív egész számok. Legyen továbbá d szintén nem negatív egész. Ha most $\mathcal{H}$-ban létezik olyan

g folyam, amelyre $|g| \leqslant d$, akkor létezik olyan g^* folyam is, amelyre $|g^*| \leqslant d$ szintén teljesül, ezen felül a $g^*(u, v)$ értékek mind egészek.

Esetünkben a $\mathcal{H}$ hálózathoz konstruált $\mathcal{H}'$ hálózat kapacitásai mind egészek lesznek. Ebben létezik egészértékű maximális folyam, amelynek értéke a feltétel miatt

$$\sum_{(u,v)\in E} k(u, v).$$

Egy ilyen egészértékű maximális folyamból konstruált g^* folyam pedig szintén egészértékű lesz.

Piackutatás

Feladat.
Egy vállalat k féle terméket állít elő, jelölje ezeket $P_1, P_2, \ldots, P_k$, amelyeket saját maga értékesít. A vállalatnak hosszú időre visszamenőleg rendelkezésére áll, hogy a vásárlóik mikor, mely termékeket vásárolták meg (törzsvásárlói kártya). A vállalat felmérést szeretne végezni, hogy mely termékekkel vannak a vásárlóik leginkább megelégedve. Ebből a célból kiválasztanak n vásárlót, jelölje őket $V_1, V_2, \ldots, V_n$, akiknek személyre szóló kérdőíveket küldenek. A kérdőívek összeállításának szempontjai a következők:

- A kérdőívek minden vásárlótól csak az általa megvásárolt termékek felől érdeklődnek.
- Minden $1 \leqslant i \leqslant n$ esetén a V_i vásárlónak küldött kérdőív legalább c_i és legfeljebb c'_i termék felől érdeklődik.
- Minden $1 \leqslant j \leqslant k$ esetén a P_j termékről a kérdőívek legalább r_j és legfeljebb r'_j különböző vásárlótól érdeklődnek.

Adjunk hatékony algoritmust annak eldöntésére, hogy meg lehet-e szerkeszteni a kérdőíveket az adott szempontok szerint, és ha igen, az algoritmus szerkessze is meg a ezeket!

Megoldás.
Egy $\mathcal{H}$ hálózatot készítünk, amelyben minden $1 \leqslant i \leqslant n$ esetén a V_i vásárlónak megfelel egy v_i csúcs és minden $1 \leqslant j \leqslant k$ esetén a P_j terméknek egy p_j csúcs. Ha a V_i vásárló vásárolt a P_j termékből, akkor a v_i csúcsból menjen egy irányított él a p_j csúcsba. Ezeknek az éleknek a kapacitása legyen 1, az alsó korlátjuk pedig 0. Vegyünk fel még egy s forrást és egy t nyelőt. Minden $1 \leqslant i \leqslant n$ esetén menjen egy irányított él s-ből v_i-be, amelynek kapacitása

c'_i, alsó korlátja pedig c_i, valamint minden $1 \leqslant j \leqslant k$ esetén menjen egy irányított él p_j-ből t-be, amelynek kapacitása r'_j, alsó korlátja pedig r_j.

Állítás. A kérdőíveket akkor és csak akkor lehet az adott szempontok szerint megszerkeszteni, ha a $\mathcal{H}$ hálózatban létezik folyam.

Bizonyítás. Tegyük fel először, hogy a kérdőíveket meg lehet szerkeszteni az adott szempontok szerint. Ha a V_i vásárlótól érdeklődünk a P_j termék felől, akkor küldjünk egységnyi folyamot az (s, v_i, p_j, t) úton s-ből t-be. Ezt minden ilyen (V_i, P_j) párra elvégezve világos módon egy olyan folyamot kapunk, amely az összes kapacitás és alsó korlát feltételt kielégíti.

Megfordítva, tegyük fel, hogy a hálózatban létezik folyam. Ekkor létezik egészértékű folyam is a hálózaton. Ha most a (v_i, p_j) élen egységnyi folyam halad át, akkor a P_j termék felől érdeklődjünk a V_i vásárlótól. A kapacitás és alsó korlát feltételek miatt így az összes szempontnak megfelelő kérdőívekhez jutunk.

Repülőgépek ütemezése

Feladat.
Tegyük fel, hogy egy légitársaság a $J_1, J_2, \ldots, J_m$ járatokat szeretné üzemeltetni, és d darab azonos típusú repülőgépe van erre a célra. Minden (J_i, J_k) járatpárra ismert, hogy van-e elég idő arra, hogy J_i teljesítése után egy gép felkészíthető a J_k repülésre. Ha egy (J_i, J_k) párra igen a válasz, akkor röviden azt fogjuk mondani, hogy J_k követheti J_i-t.

Adjunk hatékony algoritmust annak eldöntésére, hogy üzemeltethető-e az összes járat legfeljebb d repülőgéppel!

Megoldás.
Egy $\mathcal{H}$ hálózatot készítünk, amelyben minden $1 \leqslant i \leqslant m$ esetén a J_i járatnak két csúcs, v_i és v'_i, valamint a (v_i, v'_i) él felel meg. Ha J_k követheti J_i-t, akkor menjen egy irányított él v'_i-ből v_k-ba. Vegyünk fel még egy s forrást és egy t nyelőt, és minden $1 \leqslant i \leqslant m$ esetén menjen egy-egy irányított él s-ből v_i-be, illetve v'_i-ből t-be. Az összes él kapacitása legyen 1, a (v_i, v'_i) élek alsó korlátja szintén 1, a többi él alsó korlátja pedig 0.

Állítás. A $J_1, J_2, \ldots, J_m$ járatok akkor és csak akkor üzemeltethetők d repülőgéppel, ha a $\mathcal{H}$ hálózatban létezik olyan g folyam, amelyre $|g| \leqslant d$.

Bizonyítás. Tegyük fel először, hogy létezik olyan folyam a hálózaton, amelynek értéke kisebb vagy egyenlő, mint d. A kapacitások és korlátok

egész volta miatt ezen folyamok között van olyan, amely minden élen egész értéket vesz fel. Jelöljön g egy ilyen folyamot. Szintén a kapacitások és korlátok figyelembe vételével adódik, hogy g értéke minden élen 0 vagy 1.

Tekintsük azt a részgráfot, amelynek élei az egységnyi folyamot szállító élek. Ez felbontható legfeljebb d darab

$$(s, v_{i_1}, v'_{i_1}, v_{i_2}, v'_{i_2}, \ldots, v_{i_l}, v'_{i_l}, t)$$

út egyesítésére. Valóban, tekintsünk egy olyan (s, v_i) élt, amelyen elindul egy egység folyam. Ezt a (v_i, v'_i) él szállítja tovább, majd a folyam valamelyik v'_i-ből induló élen folytatja útját, míg végül t-be jut. Eközben egy s-ből t-be menő utat járunk be. Ezt megismételve minden olyan (s, v_j) élre, amelyen elindul egy egység folyam, végül a kívánt s-ből t-be menő utakat kapjuk.

Két ilyen útnak s-en és t-n kívül nyilván nincs közös csúcsa. Egy ilyen út létezése azt jelenti, hogy a $J_{i_1}, J_{i_2}, \ldots, J_{i_l}$ járatok egy géppel teljesíthetők. A (v_i, v'_i) élekre kiszabott alsó korlát feltételek miatt minden (v_i, v'_i) él rajta van valamelyik ilyen úton. Tehát legfeljebb d repülőgéppel tényleg minden járat lebonyolítható.

Megfordítva, tegyük fel, hogy van egy legfeljebb d repülőgépet használó, minden járatot teljesítő ütemezésünk. Nézzük ebben egy gép egymás utáni feladatait. Ha ezek a $J_{i_1}, J_{i_2}, \ldots, J_{i_l}$ járatok (ebben a sorrendben), akkor az

$$(s, v_{i_1}, v'_{i_1}, v_{i_2}, v'_{i_2}, \ldots, v_{i_l}, v'_{i_l}, t)$$

út éleit a hálózat tartalmazza. Ezen az úton egységnyi folyamot küldhetünk s-ből t-be. Ezt minden repülőgépre elvégezve egy legfeljebb d értékű g folyamot kapunk.

A kapacitásfeltételek teljesülnek, mert az így kapott utaknak nincs közös élük, hiszen minden járat egy repülőgéphez tartozik. A (v_i, v'_i) élekre kiszabott alsó korlát feltételek sem sérülnek, mert az ütemezésben minden járat teljesítve van. Így g eleget tesz a követelményeknek.

NP-teljesség és közelítő algoritmusok

Gráfok színezése

Feladat.
Egy gráfot k színnel színezhetőnek nevezünk, ha a csúcsaihoz i egészeket (színeket) rendelhetünk úgy, hogy $1 \leqslant i \leqslant k$, és ha két csúcsot él köt össze, akkor a hozzájuk rendelt színek különbözők.

(A) A 3-SZÍNEZÉS nyelv álljon azokból a gráfokból, amelyek 3 színnel színezhetők. Mutassuk meg, hogy a 3-SZÍNEZÉS nyelv NP-teljes!

(B) Legyen $k \geqslant 4$ egész szám. A k-SZÍNEZÉS nyelv álljon azokból a gráfokból, amelyek k színnel színezhetők. Mutassuk meg, hogy a k-SZÍNEZÉS nyelv NP-teljes!

(C) A SZÍNEZÉS nyelv álljon azokból azokból a (G, k) párokból (G egy gráf, k pedig egy pozitív egész), amelyekre teljesül, hogy a G gráf k színnel színezhető. Mutassuk meg, hogy a SZÍNEZÉS nyelv NP-teljes!

Megoldás.
(A) Nyilvánvaló, hogy 3-SZÍNEZÉS $\in$ NP. Az NP-teljesség belátásához a 3-SAT nyelvet fogjuk visszavezetni a 3-SZÍNEZÉS nyelvre. Legyen $\phi = \phi_1 \wedge \cdots \wedge \phi_m$ egy n változós 3-normálforma. Legyenek a ϕ-beli változók $x_1, x_2, \ldots, x_n$. Feltehető, hogy minden elemi diszjunkcióban pontosan 3 különböző változó szerepel (erre a bizonyítás végén még visszatérünk).

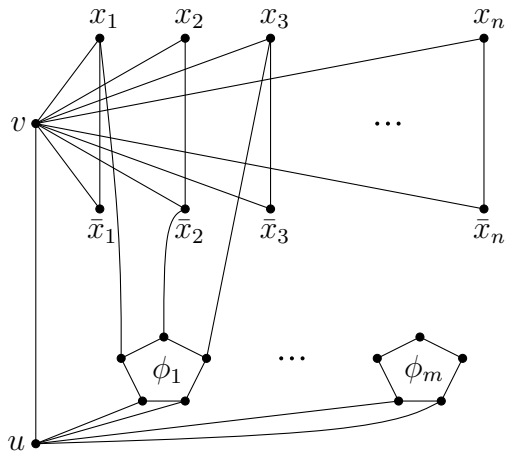
Definiálunk egy G gráfot a következőképpen. Kezdjük a G gráf csúcsaival.

- Van két különálló csúcs, u és v.
- A ϕ-beli x_i változóknak és negáltjaiknak is megfelel egy-egy csúcs, ezek jele x_i illetve $\bar{x}_i$, $1 \leqslant i \leqslant n$.
- Végül a ϕ_j elemi diszjunkcióknak öt csúcs felel meg: ezek $\phi_j^1, \ldots, \phi_j^5$, $1 \leqslant j \leqslant m$.

A G gráf élei jönnek.

- Először is $(u, v) \in E(G)$, továbbá $(v, x_i), (v, \bar{x}_i), (x_i, \bar{x}_i) \in E(G)$ minden $1 \leqslant i \leqslant n$ esetén.
- Ezen kívül $(u, \phi_j^4), (u, \phi_j^5) \in E(G)$ minden $1 \leqslant j \leqslant m$ esetén, valamint $(\phi_j^k, \phi_j^{k+1}) \in E(G)$ minden $1 \leqslant j \leqslant m$ és $1 \leqslant k \leqslant 5$ esetén (itt a felső index ciklikusan fut, 5 után 1 jön).
- Végül minden $1 \leqslant j \leqslant m$, $1 \leqslant i \leqslant n$ és $1 \leqslant k \leqslant 3$ esetén $(\phi_j^k, x_i) \in E(G)$, ha a ϕ_j elemi diszjunkció k-adik tagja x_i, és hasonlóan, minden $1 \leqslant j \leqslant m$, $1 \leqslant i \leqslant n$ és $1 \leqslant k \leqslant 3$ esetén $(\phi_j^k, \bar{x}_i) \in E(G)$, ha a ϕ_j elemi diszjunkció k-adik tagja $\bar{x}_i$.

A következő ábra azt az esetet szemlélteti, amikor $\phi_1 = x_1 \vee \bar{x}_2 \vee x_3$.



A G gráf mérete lineáris ϕ méretében, és az is látható, hogy polinom időben felépíthető ϕ ismeretében. Igazolnunk kell még, hogy $G \in$ 3-Színezés pontosan akkor áll fenn, ha $\phi \in$ 3-SAT.

Tegyük fel először, hogy $G \in$ 3-Színezés, és tekintsük a gráf egy 3-színezését. Nevezzük v színét pirosnak, u színét sárgának, a harmadik szín pedig legyen zöld. Vegyük észre, hogy egy literál színe csak zöld vagy sárga lehet a v-vel összekötő él miatt. A ϕ formula ℓ literáljának adjuk az igaz értéket, ha G-ben az ℓ csúcs zöld, illetve a hamis értéket, ha az ℓ csúcs sárga. Ez megfelel az x_i változók egy kiértékelésének, hiszen az $(x_i, \bar{x}_i)$ él megléte miatt az x_i és $\bar{x}_i$ csúcsok közül az egyik sárga, a másik zöld minden $1 \leqslant i \leqslant n$ esetén. Megmutatjuk, hogy így ϕ egy kielégítő kiértékelése adódik. Ez nyilván egyenértékű azzal, hogy mindegyik ϕ_j-ben van zöld literál. Ez viszont igaz, ellenkező esetben ugyanis a ϕ_j-hez tartozó ötszög csúcsaira csak

két szín lenne megengedett, a piros és a zöld, ami nyilván kevés az ötszög kiszínezéséhez.

Ezután tegyük fel, hogy $\phi \in$ 3-SAT és tekintsük ϕ egy kielégítő kiértékelését. Akár az előbb, feleltessük meg a literálok igaz értékének a zöld színt, hamis értékének pedig a sárga színt. Színezzük v-t pirosra, u-t pedig sárgára. Az eddig kapott részleges színezéssel nincs baj, már csak a ϕ_j elemi diszjunkciókhoz tartozó ötszögeket kell kiszínezni. Egy ilyen ötszög minden csúcsára tiltott a zöld és sárga színek egyike. Az "alsó" két csúcsra a sárga, a "felső" csúcsokra viszont nem mindenütt a sárga, hiszen ϕ_j-ben van zöld literál. Alapvetően négy eset van: a felső három csúcs közül egyre, két szomszédosra, két nem szomszédosra, illetve mindháromra tiltott a zöld szín. Könnyű ellenőrizni, hogy mind a négy esetben az ötszög kiszínezhető.

A bizonyítás teljessé tételéhez megmutatjuk, hogy tetszőleges n változós, m elemi diszjunkcióból álló 3-normálformához létezik olyan, vele ekvivalens, legfeljebb $n + 4$ változós, legfeljebb $m + 8$ elemi diszjunkcióból álló 3-normálforma, amelynek minden elemi diszjunkciójában pontosan 3 különböző változó szerepel.

Legyen ϕ egy n változós, m elemi diszjunkcióból álló 3-normálforma. Feltehető, hogy ϕ minden elemi diszjunkciójában egy literál csak egyszer fordul elő és semelyik elemi diszjunkcióban nem fordul elő egyszerre egy változó és annak negáltja is. (Ha ϕ valamely elemi diszjunkciójában ugyanaz a literál többször is előfordul, akkor ezen előfordulások közül csak egyet meghagyva, illetve ha valamely elemi diszjunkcióban egy változó és annak negáltja is előfordul, akkor azt az elemi diszjunkciót elhagyva nyilván egy ekvivalens 3-normálformát kapunk.) Vegyünk fel négy új változót, legyenek ezek z_1, z_2, z_3, z_4. Azokat a ϕ-beli elemi diszjunkciókat, amelyekben egy változó szerepel, egészítsük ki z_1-gyel és z_2-vel, azokat az elemi diszjunkciókat pedig, amelyekben két változó szerepel, egészítsük ki z_1-gyel. Ezen kívül egészítsük még ki ϕ-t a

$$\begin{gathered}
\bar{z}_1 \vee z_3 \vee z_4, \\
\bar{z}_1 \vee \bar{z}_3 \vee z_4, \\
\bar{z}_1 \vee z_3 \vee \bar{z}_4, \\
\bar{z}_1 \vee \bar{z}_3 \vee \bar{z}_4, \\
\bar{z}_2 \vee z_3 \vee z_4, \\
\bar{z}_2 \vee \bar{z}_3 \vee z_4, \\
\bar{z}_2 \vee z_3 \vee \bar{z}_4, \\
\bar{z}_2 \vee \bar{z}_3 \vee \bar{z}_4
\end{gathered}$$

elemi diszjunkciókkal. Jelölje ϕ^* az így kapott 3-normálformát. Vegyük észre,

hogy a nyolc új elemi diszjunkció bármely kielégítő kiértékelésében z_1 és z_2 szükségképpen hamis értékű. Innen egyszerűen adódik, hogy ϕ akkor és csak akkor elégíthető ki, ha ϕ^* is kielégíthető.

(B) Nyilvánvaló, hogy k-SZÍNEZÉS $\in$ NP. Az NP-teljesség belátásához a 3-SZÍNEZÉS nyelvet fogjuk visszavezetni a k-SZÍNEZÉS nyelvre.

Legyen G tetszőleges gráf. Vegyünk hozzá G-hez $k-3$ új csúcsot, és kössünk össze ezeket egymással, valamint G csúcsaival. Az így kapott gráfot jelölje G'. A G' gráf mérete nyilván lineáris G méretében, és az is látható, hogy G' polinom időben felépíthető G ismeretében. Vegyük észre továbbá, hogy a G gráf akkor és csak akkor 3 színnel színezhető, ha a G' gráf k színnel színezhető.

(C) Nyilvánvaló, hogy SZÍNEZÉS $\in$ NP. Az NP-teljesség belátásához pedig elég megjegyezni, hogy a 3-SZÍNEZÉS nyelv triviális módon visszavezethető a SZÍNEZÉS nyelvre.

Klaszterezés variáció

Feladat.
Legyen adott (valamilyen) objektumok egy $U = \{p_1, p_2, \ldots, p_n\}$ halmaza. Tegyük fel, hogy bármely két p_i és p_j objektumnak ismerjük a $d(p_i, p_j)$ távolságát. A d távolságfüggvényről itt csak annyit teszünk fel, hogy tetszőleges p_i és p_j objektumokra

- $d(p_i, p_j) \geqslant 0$,
- $d(p_i, p_j) = 0$ akkor és csak akkor, ha $i = j$,
- $d(p_i, p_j) = d(p_j, p_i)$.

Legyen továbbá adott egy $k \leqslant n$ pozitív egész szám. Az U halmaz k nem üres részhalmazának egy $\{C_1, C_2, \ldots, C_k\}$ családját az U egy k-partíciójának nevezzük, ha

- $C_1 \cup C_2 \cup \cdots \cup C_k = U$,
- $C_i \cap C_j = \emptyset$, ha $i \neq j$.

A KLASZTEREZÉS nyelv álljon azokból az (U, d, k, B) rendszerekből, amelyekre az objektumok $U = \{p_1, p_2, \ldots, p_n\}$ halmazának van olyan k-partíciója, hogy a partíció egyik halmazában sem haladja meg az objektumok távolsága a B pozitív egész értéket. Mutassuk meg, hogy a KLASZTEREZÉS nyelv NP-teljes!

Megoldás.
Nyilvánvaló, hogy KLASZTEREZÉS $\in$ NP. Az NP-teljesség belátásához a SZÍNEZÉS nyelvet fogjuk visszavezetni a KLASZTEREZÉS nyelvre. Legyen G tetszőleges gráf a $\{v_1, v_2, \ldots, v_n\}$ csúcshalmazon, $k \leqslant n$ pedig egy pozitív egész szám (ha $k > n$, akkor $(G, k) \in$ SZÍNEZÉS triviálisan teljesül).

Definiálunk egy (U, d, k, B) rendszert a következőképpen. Legyen $U = V(G)$. Legyen továbbá tetszőleges v_i és v_j objektumokra

$$d(v_i, v_j) = \begin{cases} 0 & \text{ha } i = j, \\ 1 & \text{ha } i \neq j \text{ és } (v_i, v_j) \notin E(G), \\ 2 & \text{ha } i \neq j \text{ és } (v_i, v_j) \in E(G). \end{cases}$$

Végül legyen $B = 1$.

Az (U, d, k, B) rendszer mérete nyilván polinomiális G méretében, és az is világos, hogy (U, d, k, B) polinom időben felépíthető G ismeretében. Megmutatjuk, hogy a G gráf akkor és csak akkor k színnel színezhető, ha az objektumoknak létezik olyan k-partíciója, hogy a partíció egyik halmazában sem haladja meg az objektumok távolsága az 1 értéket.

Tegyük fel először, hogy a G gráf k színnel színezhető. Ekkor persze van olyan színezése is G-nek, amelynél pontosan k színt használunk. Definiáljuk a k-partíciót a következőképpen: két objektum akkor és csak akkor legyen egy halmazba, ha a nekik megfelelő csúcsok azonos színűek. Mivel a gráfban két azonos színű csúcs nem lehet szomszédos, ezért a partíció bármely halmazában az objektumok távolsága 1.

Tegyük fel ezután, hogy az objektumoknak létezik olyan k-partíciója, hogy a partíció egyik halmazában sem haladja meg az objektumok távolsága az 1 értéket. Definiáljuk a k-színezést a következőképpen: két csúcs akkor és csak akkor legyen azonos színű, ha a nekik megfelelő objektumok egy halmazban vannak. Ez valóban egy k-színezése a gráfnak: nem lehet a gráfban két azonos színű szomszédos csúcs, ellenkező esetben az ezeknek megfelelő objektumok a partíció egy halmazában lennének, miközben távolságuk 2.

Független csúcshalmaz

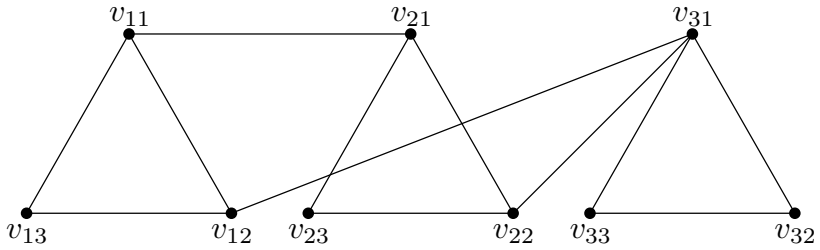
Feladat.
(A) A G irányítatlan gráf csúcsainak egy S részhalmazát független halmaznak nevezzük, ha semelyik két S-beli csúcs között nem fut él G-ben. A FÜGGETLEN CSÚCSHALMAZ nyelv álljon azokból a (G, k) párokból (G egy gráf, k pedig egy pozitív egész), amelyekre teljesül, hogy a G gráfnak van legalább k elemű független csúcshalmaza. Mutassuk meg, hogy a FÜGGETLEN CSÚCSHALMAZ nyelv NP-teljes!

(B) A G irányítatlan gráf csúcsainak egy T részhalmazát klikknek nevezzük, ha bármely két T-beli csúcs között fut él G-ben. A KLIKK nyelv álljon azokból a (G, k) párokból (G egy gráf, k pedig egy pozitív egész), amelyekre teljesül, hogy a G gráf tartalmaz legalább k elemű klikket. Mutassuk meg, hogy a KLIKK nyelv NP-teljes!

(C) A G irányítatlan gráf csúcsainak egy U részhalmazát éllefogó halmaznak nevezzük, ha G bármely élének van U-beli végpontja. Az ÉLLEFOGÓ CSÚCSHALMAZ nyelv álljon azokból a (G, k) párokból (G egy gráf, k pedig egy pozitív egész), amelyekre teljesül, hogy a G gráfnak van legfeljebb k elemű éllefogó csúcshalmaza. Mutassuk meg, hogy az ÉLLEFOGÓ CSÚCSHALMAZ nyelv NP-teljes!

Megoldás.
(A) Nyilvánvaló, hogy FÜGGETLEN CSÚCSHALMAZ $\in$ NP. Az NP-teljesség belátásához a 3-SAT nyelvet fogjuk visszavezetni a FÜGGETLEN CSÚCSHALMAZ nyelvre. Legyen $\phi = \phi_1 \wedge \cdots \wedge \phi_m$ egy n változós 3-normálforma. Feltehető, hogy minden elemi diszjunkcióban pontosan 3 különböző változó szerepel (ld. a gráfok 3-színezéséről szóló feladat megoldásának végét).

Definiálunk egy G gráfot a következőképpen. Minden $1 \leqslant i \leqslant m$ esetén a ϕ_i elemi diszjunkciónak három csúcs fog megfelelni, az első literálnak v_{i1}, a másodiknak v_{i2}, a harmadiknak pedig v_{i3}. Így $|V(G)| = 3m$. Minden $1 \leqslant i \leqslant m$ esetén legyen $(v_{i1}, v_{i2}), (v_{i2}, v_{i3}), (v_{i3}, v_{i1}) \in E(G)$. Legyen továbbá minden $1 \leqslant i < j \leqslant m$, valamint $1 \leqslant k \leqslant 3$ és $1 \leqslant l \leqslant 3$ esetén $(v_{ik}, v_{jl}) \in E(G)$ ha a ϕ_i elemi diszjunkció k-adik literálja és a ϕ_j elemi diszjunkció l-edik literálja egymás negáltjai.



$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_4) \wedge (\overline{x}_2 \vee x_3 \vee x_4)$$

A G gráf mérete nyilván polinomiális ϕ méretében, és az is látható, hogy G polinom időben felépíthető ϕ ismeretében. Igazolnunk kell még, hogy G-nek akkor és csak akkor van legalább m elemű független csúcshalmaza, ha ϕ kielégíthető.

Először tegyük fel, hogy ϕ kielégíthető és tekintsük ϕ egy kielégítő kiértékelését. Ekkor minden $1 \leqslant i \leqslant m$ esetén létezik olyan $1 \leqslant k \leqslant 3$ egész, hogy

a ϕ_i elemi diszjunkció k-adik literáljának értéke igaz. Álljon S az ezeknek megfelelő v_{ik} csúcsokból. Az S halmaz elemszáma nyilván m. Állítjuk, hogy S független csúcshalmaz G-ben. Valóban, ha futna él S valamely két v_{ik} és v_{jl} csúcsa között, akkor a két csúcsnak megfelelő literálok egymás negáltjai volnának, így nem lehetne mindkettő értéke igaz.

Tegyük fel ezután, hogy G-nek van egy legalább m elemű S független csúcshalmaza. Vegyük észre, hogy ekkor S pontosan m csúcsból áll; az elemi diszjunkcióknak megfelelő háromszögek mindegyikéből egy-egy csúcsot tartalmaz. Tekintsük a ϕ formula változóit. Ha valamely $1 \leqslant h \leqslant n$ esetén sem x_h sem pedig $\overline{x}_h$ nem fordul elő az S-beli csúcsoknak megfelelő literálok között, akkor adjuk x_h-nak az igaz értéket. Ellenkező esetben x_h és $\overline{x}_h$ közül pontosan az egyik fordul elő az S-beli csúcsoknak megfelelő literálok között. Valóban, ha valamely $1 \leqslant h \leqslant n$ esetén x_h és $\overline{x}_h$ is előfordulna az S-beli csúcsoknak megfelelő literálok között, akkor a kérdéses csúcsok között él futna, ellentmondva S független voltának. Ha valamely $1 \leqslant h \leqslant n$ esetén x_h előfordul az S-beli csúcsoknak megfelelő literálok között, akkor adjuk x_h-nak az igaz értéket, ha pedig $\overline{x}_h$ fordul elő, akkor adjuk x_h-nak a hamis értéket. Világos, hogy ilyen módon ϕ egy kielégítő kiértékelése adódik.

(B) Nyilvánvaló, hogy Klikk $\in$ NP. Az NP-teljesség belátásához a Független Csúcshalmaz nyelvet vezetjük vissza a Klikk nyelvre.

Jegyezzük meg, hogy egy G gráf csúcsainak valamely X részhalmaza akkor és csak akkor független G-ben, ha X klikk a G gráf $\overline{G}$ komplementer gráfjában (a $\overline{G}$ komplementer gráf csúcsai ugyanazok, mint a G gráf csúcsai, továbbá (u, v) pontosan akkor éle $\overline{G}$-nek, ha (u, v) nem éle G-nek). Ebből azonnal adódik, hogy egy gráfnak akkor és csak akkor van legalább k elemű független csúcshalmaza, ha komplementer gráfja tartalmaz legalább k elemű klikket. (Az nyilvánvaló, hogy a komplementer gráf polinom időben felépíthető a gráf ismeretében.)

(C) Nyilvánvaló, hogy Éllefogó Csúcshalmaz $\in$ NP. Az NP-teljesség belátásához a Független Csúcshalmaz nyelvet vezetjük vissza az Éllefogó Csúcshalmaz nyelvre.

Állítás. Legyen $G = (V, E)$ tetszőleges irányítatlan gráf. A gráf csúcsainak egy S részhalmaza akkor és csak akkor független halmaz, ha $V \setminus S$ éllefogó halmaz.

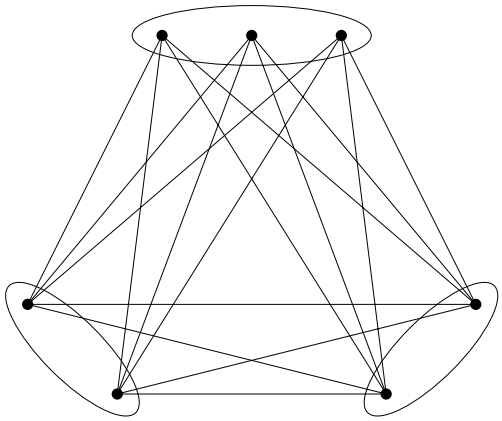
Bizonyítás. Először tegyük fel, hogy S független halmaz G-ben. Tekintsük egy tetszőleges $e = (u, v)$ élét G-nek. Mivel S független, nem lehetséges, hogy u és v is hozzátartozik S-hez, következésképpen u és v legalább egyike $V \setminus S$-ben van. Ez G minden élére teljesül, így $V \setminus S$ éllefogó halmaz G-ben.

Megfordítva, tegyük fel, hogy $V \setminus S$ éllefogó halmaz G-ben. Tekintsük S valamely két különböző u és v csúcsát. Ha ezek között egy e él futna G-ben, akkor e-nek nem lenne $V \setminus S$-beli végpontja, ellentmondás. Ebből következik, hogy semelyik két S-beli csúcs között nem fut él G-ben, így S független halmaz.

Innen azonnal adódik, hogy egy n csúcsú gráfnak akkor és csak akkor van legalább k elemű független csúcshalmaza, ha a gráfnak van legfeljebb $n - k$ elemű éllefogó csúcshalmaza.

Turán tétel

Feladat.
Legyenek $r \leqslant n$ pozitív egész számok. Definiáljuk a $T_r(n)$ Turán gráfot a következőképpen. Osszuk el maradékosan az n számot az r számmal, azaz legyen $n = rq + m$, ahol $0 \leqslant m < r$. A $T_r(n)$ Turán gráfnak n csúcsa van, amelyeket r osztályba sorolunk; ezek közül m osztály $q + 1$ csúcsból áll, a többi $r - m$ osztály pedig q csúcsból. A gráf két csúcsa akkor és csak akkor van összekötve, ha különböző osztályba tartoznak.



$T_3(7)$

Mutassuk meg, hogy ha egy n csúcsú gráf nem tartalmaz $r + 1$ csúcsú teljes gráfot, akkor élszáma nem haladja meg a $T_r(n)$ Turán gráf élszámát!

Megoldás.
Kezdjük egy definícióval. Egy gráfot r-osztályúnak vagy r-kromatikusnak nevezünk, ha csúcsai r darab nem üres osztályba sorolhatók úgy, hogy az egy osztályba tartozó csúcsok között nem fut él. Jegyezzük meg, hogy egy r-osztályú gráf nem tartalmazhat $r + 1$ csúcsú teljes gráfot.

Először megmutatjuk, hogy az n csúcsú, r-osztályú gráfok közül a $T_r(n)$ Turán gráfnak van a legtöbb éle. Legyen G egy maximális élszámú n csúcsú, r-osztályú gráf. Világos, hogy G bármely két különböző osztályba tartozó csúcsa között fut él, kérdés az osztályok mérete. Jelölje az osztályok méretét $n_1, n_2, \ldots, n_r$. Ha minden $1 \leqslant i < j \leqslant r$ esetén $|n_i - n_j| \leqslant 1$, akkor készen vagyunk; a G gráf ekkor izomorf a $T_r(n)$ Turán gráffal. Tegyük fel ezért, hogy valamely $1 \leqslant i < j \leqslant r$ esetén $|n_i - n_j| \geqslant 2$, mondjuk $n_i - n_j \geqslant 2$. Tekintsük most azt az n csúcsú, r-osztályú G^* gráfot, amelyben az osztályok mérete

$$n_1, \ldots, n_{i-1}, n_i - 1, n_{i+1}, \ldots, n_{j-1}, n_j + 1, n_{j+1}, \ldots, n_r,$$

és amelynek bármely két különböző osztályba tartozó csúcsa között fut él. Egy kis számolás következik:

$$|E(G^*)| - |E(G)| = (n_i - 1)(n_j + 1) - n_i n_j = n_i - n_j - 1 > 0.$$

Ez viszont ellentmond annak, hogy G maximális élszámú az n csúcsú, r-osztályú gráfok között. Innen az állítás adódik.

Ezután megmutatjuk, hogy bármely n csúcsú, $r+1$ csúcsú teljes gráfot nem tartalmazó G gráfhoz létezik olyan n csúcsú, r-osztályú H gráf, amelyre $V(H) = V(G)$, és tetszőleges $v \in V(H)$ csúcsra $\deg_G(v) \leqslant \deg_H(v)$. Ekkor a H gráfra $|E(H)| \geqslant |G(H)|$, hiszen egy gráfban a fokszámok összege az élszám kétszerese. Az előző állítás szerint H élszáma nem haladhatja meg a $T_r(n)$ Turán gráf élszámát, így ugyanez teljesül a G gráfra is.

Az állítást r szerinti teljes indukcióval bizonyítjuk. Az állítás $r = 1$ esetén triviális; ekkor G üres gráf. Legyen ezután $r \geqslant 2$, és tekintsünk egy n csúcsú, $r+1$ csúcsú teljes gráfot nem tartalmazó G gráfot. Jelölje a G-beli maximális fokszámot Δ, és legyen $w \in V(G)$ olyan, hogy $\deg_G(w) = \Delta$. Legyen $V_1 = \{u \in V(G) \mid (u, w) \in E(G)\}$, és $V_2 = V(G) \setminus V_1$. Jegyezzük meg, hogy $w \in V_2$.

Tekintsük most a G gráfnak a V_1 csúcshalmaz által feszített részgráfját. Jelölje ezt a gráfot G'. Vegyük észre, hogy a G' gráf nem tartalmazhat r csúcsú teljes gráfot, hiszen ennek csúcsai w-vel együtt egy $r+1$ csúcsú teljes gráfot feszítenének ki G-ben. Mivel G' nem tartalmaz r csúcsú teljes gráfot, az indukciós feltevés szerint létezik olyan $(r-1)$-osztályú H' gráf, amelyre $V(H') = V(G')$, és tetszőleges $v \in V(H')$ csúcsra $\deg_{G'}(v) \leqslant \deg_{H'}(v)$.

Ezek után a H gráfot a következőképpen konstruáljuk meg a G gráfból. Vegyük a V_1 csúcshalmazon a G' gráf helyett a H' gráfot, majd kössük össze V_1 minden csúcsát V_2 minden csúcsával, végül hagyjuk el a V_2-beli csúcsok között futó éleket. Nyilvánvaló, hogy a H gráf r-osztályú, hiszen úgy kaptuk, hogy az $(r-1)$-osztályú H' gráfhoz csatoltunk még egy osztályt, V_2-t.

Belátjuk, hogy $\deg_G(v) \leqslant \deg_H(v)$ tetszőleges $v \in V(H)$ csúcsra. Ha $v \in V_2$, akkor $\deg_H(v) = |V_1| = \Delta$, miközben $\deg_G(v) \leqslant \Delta$ definíció szerint. Ha pedig $v \in V_1$, akkor

$$\deg_H(v) = \deg_{H'}(v) + |V_2| \geqslant \deg_{G'}(v) + |V_2| \geqslant \deg_G(v).$$

Ezzel a bizonyítás teljes.

Egyszerű számolással adódik, hogy a $T_r(n)$ Turán gráf élszáma

$$\binom{n}{2} - m\binom{q+1}{2} - (r-m)\binom{q}{2} = \frac{n^2 - m^2}{2}\left(1 - \frac{1}{r}\right) + \binom{m}{2}.$$

Hamilton kör

Feladat.
(A) A G irányított gráf egy irányított köre Hamilton kör, ha abban G minden csúcsa pontosan egyszer szerepel. Az Irányított Hamilton Kör nyelv álljon azokból az irányított gráfokból, amelyek tartalmaznak Hamilton kört. Mutassuk meg, hogy az Irányított Hamilton Kör nyelv NP-teljes!

(B) A G irányított gráf egy irányított útja Hamilton út, ha abban G minden csúcsa pontosan egyszer szerepel. Az Irányított Hamilton Út nyelv álljon azokból az irányított gráfokból, amelyek tartalmaznak Hamilton utat. Mutassuk meg, hogy az Irányított Hamilton Út nyelv NP-teljes!

(C) A G irányítatlan gráf egy köre Hamilton kör, ha abban G minden csúcsa pontosan egyszer szerepel. A Hamilton Kör nyelv álljon azokból az irányítatlan gráfokból, amelyek tartalmaznak Hamilton kört. Mutassuk meg, hogy a Hamilton Kör nyelv NP-teljes!

(D) A G irányítatlan gráf egy útja Hamilton út, ha abban G minden csúcsa pontosan egyszer szerepel. A Hamilton Út nyelv álljon azokból az irányítatlan gráfokból, amelyek tartalmaznak Hamilton utat. Mutassuk meg, hogy a Hamilton Út nyelv NP-teljes!

Megoldás.
(A) Nyilvánvaló, hogy Irányított Hamilton Kör $\in$ NP. Az NP-teljesség belátásához a 3-Sat nyelvet fogjuk visszavezetni az Irányított Hamilton Kör nyelvre. Legyen $\phi = \phi_1 \wedge \cdots \wedge \phi_m$ egy n változós 3-normálforma. Legyenek a ϕ-beli változók $x_1, x_2, \ldots, x_n$. Feltehető, hogy minden elemi diszjunkcióban pontosan 3 különböző változó szerepel (ld. a gráfok 3-színezéséről szóló feladat megoldásának végét).

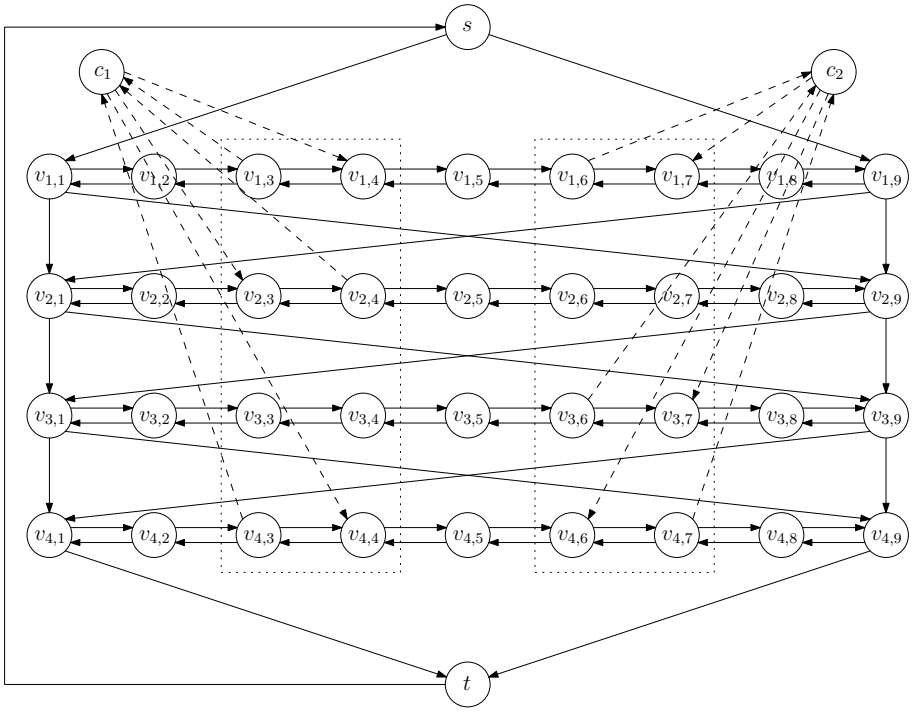
Definiálunk egy G irányított gráfot a következőképpen. Minden $1 \leqslant i \leqslant n$ esetén az x_i változónak $3m+3$ csúcs fog megfelelni: $v_{i,1}, v_{i,2}, \ldots, v_{i,3m+3}$. Minden $1 \leqslant i \leqslant n$ és minden $1 \leqslant j \leqslant 3m+2$ esetén vezessen él mindkét irányba a $v_{i,j}$ és a $v_{i,j+1}$ csúcsok között. Ez azt jelenti, hogy minden $1 \leqslant i \leqslant n$ esetén a $v_{i,1}, v_{i,2}, \ldots, v_{i,3m+3}$ csúcsokon végig tudunk lépkedni $v_{i,1}$-től $v_{i,3m+3}$-ig és $v_{i,3m+3}$-tól $v_{i,1}$-ig is.

Vegyünk fel még két további csúcsot, legyenek ezek s és t, és vezessünk éleket s-ből $v_{1,1}$-be és $v_{1,3m+3}$-ba, vezessünk éleket t-be $v_{n,1}$-ből és $v_{n,3m+3}$-ból, továbbá vezessünk egy élt t-ből s-be.

Ezen kívül minden $1 \leqslant i \leqslant n-1$ esetén vezessünk éleket $v_{i,1}$-ből $v_{i+1,1}$-be és $v_{i+1,3m+3}$-ba, valamint $v_{i,3m+3}$-ból $v_{i+1,1}$-be és $v_{i+1,3m+3}$-ba.

Tekintsük ezután az elemi diszjunkciókat. Minden $1 \leqslant k \leqslant m$ esetén a ϕ_k elemi diszjunkciónak feleljen meg egy c_k csúcs. Legyen ℓ a ϕ_k elemi diszjunkció egy literálja. Ha $\ell = x_i$ akkor vezessünk egy-egy élt $v_{i,3k}$-ból c_k-ba és c_k-ból $v_{i,3k+1}$-be, ha pedig $\ell = \overline{x}_i$, akkor vezessünk egy-egy élt $v_{i,3k+1}$-ből c_k-ba és c_k-ból $v_{i,3k}$-ba.

A következő ábrán az $(x_1 \vee \overline{x}_2 \vee x_4) \wedge (x_1 \vee x_3 \vee \overline{x}_4)$ normálformának megfelelő irányított gráf látható.



A G gráf mérete nyilván polinomiális ϕ méretében, és az is látható, hogy a G gráf polinom időben felépíthető ϕ ismeretében. Megmutatjuk, hogy G-ben akkor és csak akkor van irányított Hamilton kör, ha ϕ kielégíthető.

Először tegyük fel, hogy G-ben van egy H irányított Hamilton kör. Legyen $1 \leqslant k \leqslant m$, és tekintsük a c_k csúcsot. A c_k csúcson szükségképpen áthalad a H irányított Hamilton kör. Két lehetőség van:

- A H irányított Hamilton kör c_k-t megelőző csúcsa $v_{i,3k}$ valamely $1 \leqslant i \leqslant n$ esetén. Ekkor a H irányított Hamilton kör c_k-t követő csúcsa szükségképpen $v_{i,3k+1}$, ellenkező esetben H elakadna a $v_{i,3k-1}$ vagy a $v_{i,3k+1}$ csúcsban, attól függően, hogy a $v_{i,3k}$ csúcsba a $v_{i,3k+1}$ vagy a $v_{i,3k-1}$ csúcsból érkeztünk; ez nem lehetséges.

- A H irányított Hamilton kör c_k-t megelőző csúcsa $v_{i,3k+1}$ valamely $1 \leqslant i \leqslant n$ esetén. Ekkor a H irányított Hamilton kör c_k-t követő csúcsa szükségképpen $v_{i,3k}$, ellenkező esetben H elakadna a $v_{i,3k}$ vagy a $v_{i,3k+2}$ csúcsban, attól függően, hogy a $v_{i,3k+1}$ csúcsba a $v_{i,3k+2}$ vagy a $v_{i,3k}$ csúcsból érkeztünk; ez nem lehetséges.

Ha a H irányított Hamilton kör c_k-t megelőző csúcsa $v_{i,3k}$, akkor helyettesítsük H-n a $(v_{i,3k}, c_k, v_{i,3k+1})$ irányított utat a $(v_{i,3k}, v_{i,3k+1})$ irányított éllel. Ha a H irányított Hamilton kör c_k-t megelőző csúcsa $v_{i,3k+1}$, akkor helyettesítsük H-n a $(v_{i,3k+1}, c_k, v_{i,3k})$ irányított utat a $(v_{i,3k+1}, v_{i,3k})$ irányított éllel. Ezt a módosítást minden $1 \leqslant k \leqslant m$ esetén elvégezve a G gráf egy H' irányított körét kapjuk, amely nem megy át a $c_1, c_2, \ldots, c_m$ csúcsokon.

Tekintsük azt a G' gráfot, amelyet a G gráfból a $c_1, c_2, \ldots, c_m$ csúcsok és a rájuk illeszkedő élek elhagyásával kapunk. Világos, hogy H' ekkor a G' gráf egy irányított Hamilton köre. Mivel a t csúcsból egyetlen él indul ki, ezért a H' irányított Hamilton körön szükségképpen t-ből s-be megyünk a (t, s) irányított élen. Ezután minden $1 \leqslant i \leqslant n$ esetén a $v_{i,1}, v_{i,2}, \ldots, v_{i,3m+3}$ csúcsokon $v_{i,1}$-től $v_{i,3m+3}$-ig vagy $v_{i,3m+3}$-tól $v_{i,1}$-ig lépkedünk végig. Végül ismét a t csúcsba érünk.

Minden $1 \leqslant i \leqslant n$ esetén a fenti két lehetőség természetes módon megfeleltethető az x_i logikai változó két lehetséges értékének: ha a H' irányított Hamilton kör a $v_{i,1}, v_{i,2}, \ldots, v_{i,3m+3}$ csúcsokon $v_{i,1}$-től $v_{i,3m+3}$-ig lépked végig, akkor legyen x_i értéke igaz, ha pedig $v_{i,3m+3}$-tól $v_{i,1}$-ig, akkor legyen x_i értéke hamis.

Minden $1 \leqslant k \leqslant m$ esetén tekintsük a ϕ_k elemi diszjunkciót, illetve az annak megfelelő c_k csúcsot G-ben. Két lehetőség van:

- A H irányított Hamilton körön $v_{i,3k}$-ból megyünk c_k-ba majd onnan $v_{i,3k+1}$-be valamely $1 \leqslant i \leqslant n$ esetén. Ekkor egyrészt x_i igaz értékű, másrészt ϕ_k-ban szerepel az x_i literál.

- A H irányított Hamilton körön $v_{i,3k+1}$-ből megyünk c_k-ba majd onnan $v_{i,3k}$-ba valamely $1 \leqslant i \leqslant n$ esetén. Ekkor egyrészt x_i hamis értékű, másrészt ϕ_k-ban szerepel az $\bar{x}_i$ literál.

Mindkét esetben van ϕ_k-ban igaz értékű literál, következésképpen ϕ_k értéke igaz. Innen ϕ kielégíthetősége adódik.

Tegyük fel ezután, hogy ϕ kielégíthető és tekintsük ϕ egy kielégítő kiértékelését. Tekintsük azt a G' gráfot, amelyet a G gráfból a $c_1, c_2, \ldots, c_m$ csúcsok és a rájuk illeszkedő élek elhagyásával kapunk. Haladjunk végig a G' gráf csúcsain a következő sorrendben. Induljunk el az s csúcsból. Ezután haladjunk végig a $v_{1,1}, v_{1,2}, \ldots, v_{1,3m+3}$ csúcsokon $v_{1,1}$-től $v_{1,3m+3}$-ig ha az x_1 változó értéke igaz, illetve $v_{1,3m+3}$-tól $v_{1,1}$-ig ha az x_1 változó értéke hamis. Ezt követően haladjunk végig a $v_{2,1}, v_{2,2}, \ldots, v_{2,3m+3}$ csúcsokon $v_{2,1}$-től $v_{2,3m+3}$-ig ha az x_2 változó értéke igaz, illetve $v_{2,3m+3}$-tól $v_{2,1}$-ig ha az x_2 változó értéke hamis, és így tovább. Végül a t csúcsba érkezünk, ahonnan a (t, s) irányított élen ismét az s csúcsba juthatunk. Ilyen módon a G' gráf egy H' irányított Hamilton körét kapjuk. Világos, hogy H' irányított kör a G gráfban, amely nem megy át a $c_1, c_2, \ldots, c_m$ csúcsokon.

Legyen $1 \leqslant k \leqslant m$, és tekintsük a ϕ_k elemi diszjunkciót. A ϕ_k elemi diszjunkció legalább egy ℓ literálja szükségképpen igaz értékű.

- Ha $\ell = x_i$, akkor egyrészt a G gráfban él vezet $v_{i,3k}$-ból c_k-ba és c_k-ból $v_{i,3k+1}$-be, másrészt a G gráf H' irányított körén a $v_{i,3k}$ csúcs után a $v_{i,3k+1}$ csúcs következik. Helyettesítsük most a G gráf H' irányított körén a $(v_{i,3k}, v_{i,3k+1})$ irányított élt a $(v_{i,3k}, c_k, v_{i,3k+1})$ irányított úttal.

- Ha $\ell = \bar{x}_i$, akkor egyrészt a G gráfban él vezet $v_{i,3k+1}$-ből c_k-ba és c_k-ból $v_{i,3k}$-ba, másrészt a G gráf H' irányított körén a $v_{i,3k+1}$ csúcs után a $v_{i,3k}$ csúcs következik. Helyettesítsük most a G gráf H' irányított körén a $(v_{i,3k+1}, v_{i,3k})$ irányított élt a $(v_{i,3k+1}, c_k, v_{i,3k})$ irányított úttal.

Ezt a módosítást minden $1 \leqslant k \leqslant m$ esetén elvégezve nyilvánvaló módon a G gráf egy H irányított Hamilton körét kapjuk.

(B) Nyilvánvaló, hogy Irányított Hamilton Út $\in$ NP. Az NP-teljesség belátásához az Irányított Hamilton Kör nyelvet fogjuk visszavezetni az Irányított Hamilton Út nyelvre. Legyen G egy irányított gráf.

Definiálunk egy G' irányított gráfot a következőképpen. Tekintsük a G gráf egy tetszőleges v csúcsát. Cseréljük le a v csúcsot két új v' és v'' csúcsra. Cseréljük le továbbá a G gráf mindegyik v-be érkező (u, v) élét egy új (u, v'') élre és mindegyik v-ből induló (v, w) élét egy új (v', w) élre.

A G' gráf mérete nyilván lineáris G méretében, és az is látható, hogy G' polinom időben felépíthető G ismeretében. Megmutatjuk, hogy G'-ben akkor és csak akkor van irányított Hamilton út, ha G-ben van irányított Hamilton kör.

Először tegyük fel, hogy G'-ben van egy H' irányított Hamilton út. A H' irányított Hamilton út szükségképpen v'-ben kezdődik, hiszen v'-be nem

érkezik él, és v''-ben végződik, hiszen v''-ből nem indul él. A H' Hamilton út két végpontját "egyesítve" a G gráf egy irányított Hamilton köréhez jutunk.

Tegyük fel ezután, hogy G-ben van egy H irányított Hamilton kör. A H irányított Hamilton kört a v csúcsnál "megszakítva" a G' gráf egy v'-ből induló és v''-be érkező irányított Hamilton útjához jutunk.

(C) Nyilvánvaló, hogy HAMILTON KÖR $\in$ NP. Az NP-teljesség belátásához az IRÁNYÍTOTT HAMILTON KÖR nyelvet fogjuk visszavezetni a HAMILTON KÖR nyelvre. Legyen G egy irányított gráf.

Definiálunk egy G' irányítatlan gráfot a következőképpen. A G gráf minden v_i csúcsának feleljen meg három csúcs G'-ben, jelölje ezeket $v_i^{\text{be}}, v_i^*, v_i^{\text{ki}}$. Így $|V(G')| = 3|V(G)|$. A G gráf minden v_i csúcsára legyen $(v_i^{\text{be}}, v_i^*) \in E(G')$ és $(v_i^*, v_i^{\text{ki}}) \in E(G')$. Legyen továbbá $(v_i^{\text{ki}}, v_j^{\text{be}}) \in E(G')$ minden $(v_i, v_j) \in E(G)$ esetén. Így $|E(G')| = 2|V(G)| + |E(G)|$. A G' gráf mérete lineáris G méretében, és az is látható, hogy G' polinom időben felépíthető G ismeretében. Megmutatjuk, hogy G'-ben akkor és csak akkor van Hamilton kör, ha G-ben van irányított Hamilton kör.

Először tegyük fel, hogy G'-ben van egy H' Hamilton kör. A H' Hamilton kör felbontható $(v_i^*, v_i^{\text{ki}}, v_j^{\text{be}}, v_j^*)$ alakú utakra; feleltessük meg minden ilyen útnak a (v_i, v_j) irányított élt G-ben. Ezek az élek együttesen G egy irányított Hamilton körét alkotják.

Tegyük fel ezután, hogy G-ben van egy H irányított Hamilton kör. A H irányított Hamilton kör minden (v_i, v_j) irányított élének feleltessük meg G'-ben a $(v_i^*, v_i^{\text{ki}}, v_j^{\text{be}}, v_j^*)$ utat. Ezek az utak együttesen G' egy Hamilton körét alkotják.

(D) Nyilvánvaló, hogy HAMILTON ÚT $\in$ NP. Az NP-teljesség belátásához a HAMILTON KÖR nyelvet fogjuk visszavezetni a HAMILTON ÚT nyelvre. Legyen G egy irányítatlan gráf.

Definiálunk egy G' irányítatlan gráfot a következőképpen. Tekintsük a G gráf egy tetszőleges v csúcsát. Vegyünk fel három új csúcsot, legyenek ezek u, w és z, és vezessünk éleket u és w között, w és v szomszédai között, valamint v és z között.

A G' gráf mérete nyilván lineáris G méretében, és az is látható, hogy G' polinom időben felépíthető G ismeretében. Megmutatjuk, hogy G'-ben akkor és csak akkor van Hamilton út, ha G-ben van Hamilton kör.

Először tegyük fel, hogy G'-ben van egy H' Hamilton út. A H' Hamilton út egyik végpontja az u csúcs, ezután következik a w csúcs, majd v valamelyik t szomszédja. A H' Hamilton út másik végpontja a z csúcs, ezt a v csúcs követi. Világos, hogy ha a H' Hamilton út elejéről és végéről töröjük az u és w, valamint a z csúcsokat az (u, w), (w, t) és (v, z) élekkel együtt, majd hozzávesszük a (t, v) élt, akkor a G gráf egy Hamilton köréhez jutunk.

Tegyük fel ezután, hogy G-ben van egy H Hamilton kör. Világos, hogy ha a H Hamilton körből töröljük valamelyik v-re illeszkedő (v,t) élt, akkor a kapott Hamilton utat az u, w és z csúcsokkal, valamint az (u,w), (w,t) és (v,z) élekkel kiegészítve a G' gráf egy Hamilton útjához jutunk.

Rédei tétel

Feladat.
Mutassuk meg, hogy ha egy irányított gráf bármely két csúcsa között pontosan az egyik irányba megy él (egy ilyen gráfot turnamentnek nevezünk), akkor a gráf tartalmaz irányított Hamilton utat!

Megoldás.
Az állítást a turnament csúcsszámára vonatkozó teljes indukcióval bizonyítjuk. Tekintsünk egy n csúcsú G turnamentet. Ha $n = 1$ vagy $n = 2$, akkor az állítás triviálisan igaz. Legyen ezután $n \geqslant 3$ és legyen v a turnament egy tetszőleges csúcsa. Tekintsük azt a G' turnamentet, amelyet G-ből a v csúcs (és a rá illeszkedő élek) elhagyásával kapunk. Legyen $V(G') = \{v_1, v_2, \ldots, v_{n-1}\}$. Az indukciós feltevés szerint G' tartalmaz irányított Hamilton utat. Az általánosság megszorítása nélkül feltehetjük, hogy

$$(v_1, v_2, \ldots, v_{n-1})$$

egy ilyen irányított Hamilton út G'-ben, vagyis $(v_i, v_{i+1}) \in E(G')$ minden $1 \leqslant i \leqslant n-2$ esetén. Helyezzük most vissza az előbb eltávolított v csúcsot a rá illeszkedő élekkel együtt.

Ha $(v, v_1) \in E(G)$, akkor készen vagyunk, hiszen ekkor

$$(v, v_1, v_2, \ldots, v_{n-1})$$

egy irányított Hamilton út G-ben. Hasonlóan, ha $(v_{n-1}, v) \in E(G)$, akkor is készen vagyunk, hiszen ekkor

$$(v_1, v_2, \ldots, v_{n-1}, v)$$

egy irányított Hamilton út G-ben.

Tegyük fel ezért, hogy $(v, v_1) \notin E(G)$ és $(v_{n-1}, v) \notin E(G)$. Ekkor persze $(v_1, v) \in E(G)$ és $(v, v_{n-1}) \in E(G)$. Legyen $2 \leqslant i \leqslant n-1$ az a legkisebb index, amelyre $(v, v_i) \in E(G)$. Ekkor $(v_{i-1}, v) \in E(G)$ nyilvánvaló módon. Így ebben az esetben

$$(v_1, \ldots, v_{i-1}, v, v_i, \ldots, v_{n-1})$$

egy irányított Hamilton út G-ben.

Dirac tétel

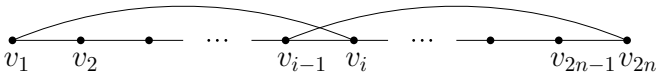
Feladat.
Mutassuk meg, hogy ha egy $2n$ csúcsú (irányítatlan) gráfban minden csúcs foka legalább n, akkor a gráf tartalmaz Hamilton kört!

Megoldás.
Indirekt tegyük fel, hogy van olyan $2n$ csúcsú gráf, amelyben minden csúcs foka legalább n, és a gráf nem tartalmaz Hamilton kört. Legyen G egy maximális élszámú ilyen gráf. Legyen $V(G) = \{v_1, v_2, \ldots, v_{2n}\}$ és tegyük fel, hogy $(v_1, v_{2n}) \notin E(G)$. Mivel G maximális élszámú azon $2n$ csúcsú gráfok között, amelyekben minden csúcs foka legalább n, és amelyek nem tartalmaznak Hamilton kört, ezért ha hozzávesszük G-hez a (v_1, v_{2n}) élt, a kapott gráf már szükségképpen tartalmaz Hamilton kört. Ebből következik, hogy G tartalmaz egy v_1-et és v_{2n}-et összekötő Hamilton utat. Az általánosság megszorítása nélkül feltehetjük, hogy $(v_1, v_2, \ldots, v_{2n-1}, v_{2n})$ egy ilyen Hamilton út G-ben, vagyis $(v_i, v_{i+1}) \in E(G)$ minden $1 \leqslant i \leqslant 2n-1$ esetén.

Tekintsük a v_1 csúcs, illetve a v_{2n} csúcs szomszédjait. Vegyük észre, hogy ha valamely $2 \leqslant i \leqslant 2n-1$ esetén a v_i csúcs szomszédja a v_1 csúcsnak, akkor a v_{i-1} csúcs nem lehet szomszédja a v_{2n} csúcsnak, ellenben

$$(v_1, v_2, \ldots, v_{i-1}, v_{2n}, v_{2n-1}, \ldots, v_i, v_1)$$

egy Hamilton köre lenne G-nek.



Ebből következik, hogy

$$\deg(v_{2n}) \leqslant 2n - 1 - \deg(v_1),$$

vagyis

$$\deg(v_{2n}) + \deg(v_1) \leqslant 2n - 1,$$

ami ellentmondás, hiszen feltételünk szerint G minden csúcsának foka legalább n.

Utazó ügynök probléma

Feladat.
Legyen adott városok egy $U = \{v_1, v_2, \ldots, v_n\}$ halmaza. Tegyük fel, hogy bármely két v_i és v_j városnak ismerjük a $d(v_i, v_j)$ távolságát. A d távolságfüggvényről csupán annyit teszünk fel, hogy tetszőleges v_i és v_j városokra $d(v_i, v_j)$ nem negatív egész szám. Nem követeljük meg, hogy d szimmetrikus legyen, tehát előfordulhat, hogy bizonyos v_i és v_j városokra $d(v_i, v_j) \neq d(v_j, v_i)$. Nem követeljük meg azt se, hogy d-re teljesüljön a háromszög-egyenlőtlenség, tehát előfordulhat, hogy bizonyos v_i, v_j és v_k városokra

$$d(v_i, v_j) + d(v_j, v_k) < d(v_i, v_k).$$

Legyen továbbá adott egy D pozitív egész szám. Az Utazó Ügynök nyelv álljon azokból az (U, d, D) rendszerekből, amelyekre a városok $U = \{v_1, v_2, \ldots, v_n\}$ halmazának létezik olyan $(v_{i_1}, v_{i_2}, \ldots, v_{i_n})$ permutációja, hogy

$$d(v_{i_1}, v_{i_2}) + d(v_{i_2}, v_{i_3}) + \cdots + d(v_{i_{n-1}}, v_{i_n}) + d(v_{i_n}, v_{i_1}) \leqslant D.$$

Mutassuk meg, hogy az Utazó Ügynök nyelv NP-teljes!

Megoldás.
Nyilvánvaló, hogy Utazó Ügynök $\in$ NP. Az NP-teljesség belátásához az Irányított Hamilton Kör nyelvet fogjuk visszavezetni az Utazó Ügynök nyelvre. Legyen G egy irányított gráf a $\{v_1, v_2, \ldots, v_n\}$ csúcshalmazon.

Definiálunk egy (U, d, D) rendszert a következőképpen. Legyen $U = V(G)$. Legyen továbbá tetszőleges v_i és v_j városokra

$$d(v_i, v_j) = \begin{cases} 0 & \text{ha } i = j, \\ 1 & \text{ha } i \neq j \text{ és } (v_i, v_j) \in E(G), \\ 2 & \text{ha } i \neq j \text{ és } (v_i, v_j) \notin E(G). \end{cases}$$

Végül legyen $D = n$.

Az (U, d, D) rendszer mérete nyilván polinomiális G méretében, és az is világos, hogy (U, d, D) polinom időben felépíthető G ismeretében. Megmutatjuk, hogy G-ben akkor és csak akkor van irányított Hamilton kör, ha a városoknak létezik olyan $(v_{i_1}, v_{i_2}, \ldots, v_{i_n})$ permutációja, hogy

$$d(v_{i_1}, v_{i_2}) + d(v_{i_2}, v_{i_3}) + \cdots + d(v_{i_{n-1}}, v_{i_n}) + d(v_{i_n}, v_{i_1}) \leqslant n.$$

Ha létezik egy $(v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1})$ Hamilton kör G-ben, akkor

$$d(v_{i_1}, v_{i_2}) + d(v_{i_2}, v_{i_3}) + \cdots + d(v_{i_{n-1}}, v_{i_n}) + d(v_{i_n}, v_{i_1}) = n$$

nyilvánvaló módon.

Tegyük fel ezután, hogy a városoknak létezik olyan $(v_{i_1}, v_{i_2}, \ldots, v_{i_n})$ permutációja, hogy

$$d(v_{i_1}, v_{i_2}) + d(v_{i_2}, v_{i_3}) + \cdots + d(v_{i_{n-1}}, v_{i_n}) + d(v_{i_n}, v_{i_1}) \leqslant n.$$

Itt a bal oldal egy n tagú összeg, amelyben minden tag legalább 1. Ez csak úgy lehetséges, hogy minden tag 1, ami viszont azt jelenti, hogy

$$(v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), \ldots, (v_{i_{n-1}}, v_{i_n}), (v_{i_n}, v_{i_1}) \in E(G),$$

következésképpen $(v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1})$ Hamilton kör G-ben.

3-dimenziós párosítás

Feladat.
Legyenek X, Y, Z páronként diszjunkt, azonos méretű véges halmazok és legyen $S \subseteq X \times Y \times Z$. Azt mondjuk, hogy egy $T \subseteq S$ halmaz 3-dimenziós párosítás, ha $X \cup Y \cup Z$ minden eleme T pontosan egy hármasában fordul elő. A 3-Dimenziós Párosítás nyelv álljon azokból az (X, Y, Z, S) rendszerekből, amelyeknél S-ből kiválasztható 3-dimenziós párosítás. Mutassuk meg, hogy a 3-Dimenziós Párosítás nyelv NP-teljes!

Megoldás.
Nyilvánvaló, hogy 3-Dimenziós Párosítás $\in$ NP. Az NP-teljesség belátásához a 3-Sat nyelvet fogjuk visszavezetni a 3-Dimenziós Párosítás nyelvre. Legyen $\phi = \phi_1 \wedge \cdots \wedge \phi_m$ egy n változós 3-normálforma. Legyenek a ϕ-beli változók $x_1, x_2, \ldots, x_n$. Feltehető, hogy minden elemi diszjunkcióban pontosan 3 különböző változó szerepel (ld. a gráfok 3-színezéséről szóló feladat megoldásának végét).

Definiálunk egy (X, Y, Z, S) rendszert a következőképpen. Először is minden $1 \leqslant i \leqslant n$ esetén definiálunk két darab $2m$ elemű halmazt:

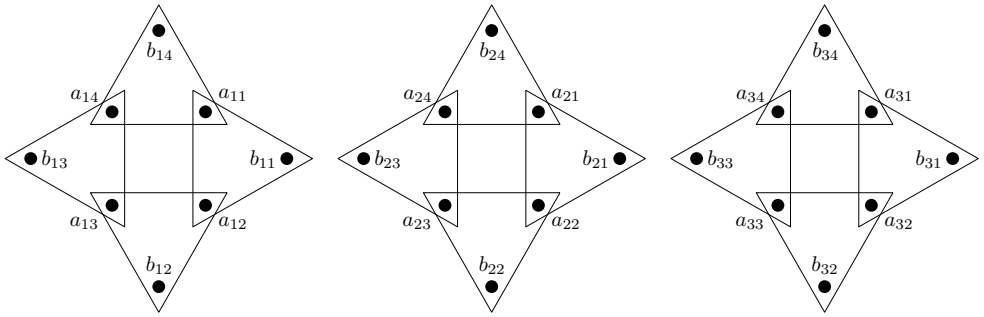
$$A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,2m}\},$$
$$B_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,2m}\};$$

ezek az x_i változónak fognak megfelelni.

Ezen kívül minden $1 \leqslant i \leqslant n$ esetén definiálunk $2m$ darab hármast:

$$\begin{aligned}
&(a_{i,2}, a_{i,1}, b_{i,1}),\\
&(a_{i,2}, a_{i,3}, b_{i,2}),\\
&(a_{i,4}, a_{i,3}, b_{i,3}),\\
&(a_{i,4}, a_{i,5}, b_{i,4}),\\
&\qquad\vdots\\
&(a_{i,2m}, a_{i,2m-1}, b_{i,2m-1}),\\
&(a_{i,2m}, a_{i,1}, b_{i,2m}),
\end{aligned}$$

jelölje ezeket $t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}, \ldots, t_{i,2m-1}, t_{i,2m}$. Egy $t_{i,j}$ hármast párosnak fogunk nevezni, ha j páros, illetve páratlannak, ha j páratlan. Analóg módon beszélni fogunk a $b_{i,j}$ elemek páros, illetve páratlan voltáról.



Ezek után minden $1 \leqslant k \leqslant m$ esetén definiálunk egy $P_k = \{p_k, p'_k\}$ halmazt, ez a ϕ_k elemi diszjunkciónak fog megfelelni. Minden $1 \leqslant k \leqslant m$ esetén definiálunk három új hármast, a ϕ_k elemi diszjunkció mindhárom literáljához egyet-egyet. Legyen ℓ a ϕ_k elemi diszjunkció egy literálja. Ha $\ell = x_i$ akkor legyen a hármas $(p_k, p'_k, b_{i,2k})$, ha pedig $\ell = \overline{x}_i$, akkor legyen a hármas $(p_k, p'_k, b_{i,2k-1})$. Jegyezzük meg, hogy a $b_{i,2k}$ és $b_{i,2k-1}$ elemek csak a ϕ_k elemi diszjunkcióhoz tartozó hármasokban fordulhatnak elő.

Végül minden $1 \leqslant h \leqslant (n-1)m$ esetén definiálunk egy $Q_h = \{q_h, q'_h\}$ halmazt és minden $1 \leqslant h \leqslant (n-1)m$, továbbá minden $1 \leqslant i \leqslant n$ és minden $1 \leqslant j \leqslant 2m$ esetén egy $(q_h, q'_h, b_{i,j})$ hármast. Ezek szerepére a bizonyítás végén derül majd fény.

Álljon az X halmaz a p_k elemekből, a q_h elemekből, valamint azon $a_{i,j}$ elemekből, amelyekre j páros. Álljon az Y halmaz a p'_k elemekből, a q'_h elemekből, valamint azon $a_{i,j}$ elemekből, amelyekre j páratlan. Végül álljon a Z halmaz a $b_{i,j}$ elemekből. Legyen továbbá S a fent definiált rendezett hármasok halmaza.

Az (X, Y, Z, S) rendszer mérete nyilván polinomiális ϕ méretében, és az is látható, hogy az (X, Y, Z, S) rendszer polinom időben felépíthető ϕ ismeretében. Megmutatjuk, hogy S-ből akkor és csak akkor választható ki 3-dimenziós párosítás ha ϕ kielégíthető.

Először tegyük fel, hogy S-ből kiválasztható 3-dimenziós párosítás. Legyen T egy ilyen 3-dimenziós párosítás. Minden $1 \leqslant i \leqslant n$ esetén tekintsük az A_i halmazt. Vegyük észre, hogy az A_i halmazt csak úgy lehet S-beli hármasokkal egyrétűen lefedni, ha vagy az összes páros vagy az összes páratlan $t_{i,j}$ hármast felhasználjuk. Ezért T vagy az összes páros, vagy az összes páratlan $t_{i,j}$ hármast tartalmazza. A két lehetőség természetes módon megfeleltethető az x_i logikai változó két lehetséges értékének: ha T a páros $t_{i,j}$ hármasokat tartalmazza, akkor legyen x_i értéke hamis, ha pedig a páratlanokat, akkor legyen x_i értéke igaz. Jegyezzük meg, hogy a T-beli $t_{i,j}$ hármasok az első esetben lefedetlenül hagyják a páratlan $b_{i,j}$ elemeket, míg a másodikban a páros $b_{i,j}$ elemeket.

Minden $1 \leqslant k \leqslant m$ esetén tekintsük a ϕ_k elemi diszjunkciót. A T-beli hármasok közül pontosan egy tartalmazza a P_k halmazt. Ennek a hármasnak a harmadik eleme $b_{i,2k}$ vagy $b_{i,2k-1}$ valamely $1 \leqslant i \leqslant n$ esetén.

- Ha a harmadik elem $b_{i,2k}$, akkor egyrészt ϕ_k-ban előfordul az x_i literál, másrészt az x_i változó értéke igaz, mivel $b_{i,2k}$-t nem a T-beli $t_{i,j}$ hármasok fedik le.

- Ha a harmadik elem $b_{i,2k-1}$, akkor egyrészt ϕ_k-ban előfordul az $\overline{x}_i$ literál, másrészt az x_i változó értéke hamis, mivel $b_{i,2k-1}$-t nem a T-beli $t_{i,j}$ hármasok fedik le.

Így mindkét esetben van ϕ_k-ban igaz értékű literál, következésképpen ϕ_k értéke igaz. Innen ϕ kielégíthetősége adódik.

Tegyük fel ezután, hogy ϕ kielégíthető és tekintsük ϕ egy kielégítő kiértékelését. Minden $1 \leqslant i \leqslant n$ esetén tartozzanak hozzá T-hez a páros $t_{i,j}$ hármasok, ha az x_i változó értéke hamis, illetve a páratlan $t_{i,j}$ hármasok, ha az x_i változó értéke igaz.

Minden $1 \leqslant k \leqslant m$ esetén legyen ℓ a ϕ_k elemi diszjunkció egy igaz értékű literálja. Ha $\ell = x_i$ valamely $1 \leqslant i \leqslant n$ esetén, akkor tartozzon hozzá T-hez a $(p_k, p'_k, b_{i,2k})$ hármas, ha pedig $\ell = \overline{x}_i$ valamely $1 \leqslant i \leqslant n$ esetén, akkor tartozzon hozzá T-hez a $(p_k, p'_k, b_{i,2k-1})$ hármas.

Végül minden $1 \leqslant h \leqslant (n-1)m$ esetén tartozzon hozzá T-hez a $(q_h, q'_h, b_{i,j})$ hármas; ezekben a hármasokban az az $(n-1)m$ darab $b_{i,j}$ elem szerepel, amelyek nem fordultak elő egyik eddigi hármasban sem.

Világos, hogy az így kapott T egy 3-dimenziós párosítás.

Részhalmaz összeg

Feladat.
A RÉSZHALMAZ ÖSSZEG nyelv álljon azokból a $(w_1, w_2, \ldots, w_n, W)$ rendszerekből ($w_1, w_2, \ldots, w_n, W$ természetes számok), amelyekre teljesül, hogy

$$\sum_{i \in I} w_i = W$$

valamilyen $I \subseteq \{1, 2, \ldots, n\}$ indexhalmaz esetén. Mutassuk meg, hogy a RÉSZHALMAZ ÖSSZEG nyelv NP-teljes!

Megoldás.
Nyilvánvaló, hogy RÉSZHALMAZ ÖSSZEG $\in$ NP. Az NP-teljesség belátásához a 3-DIMENZIÓS PÁROSÍTÁS nyelvet fogjuk visszavezetni a RÉSZHALMAZ ÖSSZEG nyelvre. Legyenek

$$X = \{x_1, x_2, \ldots, x_n\},$$
$$Y = \{y_1, y_2, \ldots, y_n\},$$
$$Z = \{z_1, z_2, \ldots, z_n\}$$

páronként diszjunkt n elemű halmazok, továbbá legyen

$$T = \{t_1, t_2, \ldots, t_m\}$$

$X \times Y \times Z$-beli rendezett hármasok egy m elemű halmaza.

Minden $1 \leqslant i \leqslant m$ esetén a $t_i = (x_p, y_q, z_r)$ rendezett hármasnak feleltessük meg a

$$w_i = (m+1)^{p-1} + (m+1)^{n+q-1} + (m+1)^{2n+r-1}$$

pozitív egész számot és legyen

$$W = 1 + (m+1) + (m+1)^2 + \cdots + (m+1)^{3n-1}.$$

Megjegyezzük, hogy minden $1 \leqslant i \leqslant m$ esetén w_i éppen a t_i rendezett hármas bitvektora $m+1$ alapú számrendszerben felírt számként tekintve.

A $(w_1, w_2, \ldots, w_n, W)$ rendszer mérete nyilván polinomiális az (X, Y, Z, T) rendszer méretében, és az is világos, hogy $(w_1, w_2, \ldots, w_n, W)$ polinom időben felépíthető (X, Y, Z, T) ismeretében.

Megmutatjuk, hogy T-ből akkor és csak akkor választható ki 3-dimenziós párosítás, ha

$$\sum_{i \in I} w_i = W$$

valamilyen $I \subseteq \{1, 2, \ldots, m\}$ indexhalmaz esetén.

Először tegyük fel, hogy T-ből kiválasztható 3-dimenziós párosítás. Legyen

$$\{t_{i_1}, t_{i_2}, \ldots, t_{i_n}\}$$

egy ilyen 3-dimenziós párosítás. Ekkor a

$$w_{i_1} + w_{i_2} + \cdots + w_{i_n}$$

összeget $m + 1$ alapú számrendszerben felírva mind a $3n$ helyiértéken 1 áll, így az összeg W.

Tegyük fel ezután, hogy

$$\sum_{i \in I} w_i = W$$

valamilyen $I \subseteq \{1, 2, \ldots, m\}$ indexhalmaz esetén. Legyen

$$I = \{i_1, i_2, \ldots, i_k\}$$

egy ilyen indexhalmaz. Mivel minden $1 \leqslant j \leqslant k$ esetén a w_{i_j} számot $m + 1$ alapú számrendszerben felírva pontosan 3 helyiértéken áll 1 és mivel az $m+1$ alapú számrendszerbeli összeadásnál nem lehet átvitel (összesen m számunk van), ezért $k = n$. Ebből következik, hogy a $3n$ helyiérték mindegyikén pontosan egy w_{i_j} szám esetén áll 1 az adott helyiértéken, így

$$\{t_{i_1}, t_{i_2}, \ldots, t_{i_k}\}$$

egy 3-dimenziós párosítás.

Még egy ütemezési probléma

Feladat.
Tegyük fel, hogy adottak bizonyos feladatok, amelyek végrehajtása egy közös erőforrás használatával lehetséges csak. Az erőforrás egyszerre egy feladattal tud foglalkozni, és ha valamelyiket elkezdte, akkor azt addig nem hagyja abba, amíg teljesen el nem készült vele. Minden feladathoz adott egy r érkezési idő, amikor a feladat végrehajtását legkorábban el lehet kezdeni, a feladat t végrehajtási ideje, és egy d határidő, amelyre a feladat végrehajtásával el kell készülni. Tegyük fel, hogy az r, t, d mennyiségek természetes számok. Az ÜTEMEZÉS nyelv álljon azokból a $((r_1, t_1, d_1), (r_2, t_2, d_2), \ldots, (r_n, t_n, d_n))$ rendszerekből, amelyekre teljesül, hogy az összes feladat elvégezhető a kezdési és a befejezési időkre vonatkozó feltételek betartásával. Mutassuk meg, hogy az ÜTEMEZÉS nyelv NP-teljes!

Megoldás.
Nyilvánvaló, hogy ÜTEMEZÉS $\in$ NP. Az NP-teljesség belátásához a RÉSZHALMAZ ÖSSZEG nyelvet fogjuk visszavezetni az ÜTEMEZÉS nyelvre. Legyenek $w_1, w_2, \ldots, w_n$ és $W \leqslant w_1 + w_2 + \ldots + w_n$ természetes számok (ha $W > w_1 + w_2 + \ldots + w_n$, akkor $(w_1, w_2, \ldots, w_n, W) \notin$ RÉSZHALMAZ ÖSSZEG triviálisan módon).

Minden $1 \leqslant i \leqslant n$ esetén legyen f_i olyan feladat, melynek érkezési ideje 0, végrehajtási ideje w_i, határideje pedig $w_1 + w_2 + \ldots + w_n + 1$. Legyen továbbá f_{n+1} olyan feladat, melynek érkezési ideje W, végrehajtási ideje 1, határideje pedig $W + 1$.

Az $((r_1, t_1, d_1), (r_2, t_2, d_2), \ldots, (r_{n+1}, t_{n+1}, d_{n+1}))$ rendszer mérete nyilván polinomiális a $(w_1, w_2, \ldots, w_n, W)$ rendszer méretében, és az is világos, hogy $((r_1, t_1, d_1), (r_2, t_2, d_2), \ldots, (r_{n+1}, t_{n+1}, d_{n+1}))$ polinom időben felépíthető $(w_1, w_2, \ldots, w_n, W)$ ismeretében.

Megmutatjuk, hogy az $f_1, f_2, \ldots, f_n, f_{n+1}$ feladatok mindegyike akkor és csak akkor végezhető el a kezdési és a befejezési időkre vonatkozó feltételek betartásával ha

$$\sum_{i \in I} w_i = W$$

valamilyen $I \subseteq \{1, 2, \ldots, n\}$ indexhalmaz esetén.

Először tegyük fel, hogy az $f_1, f_2, \ldots, f_n, f_{n+1}$ feladatok mindegyike elvégezhető a kezdési és a befejezési időkre vonatkozó feltételek betartásával. Tekintsük a feladatoknak egy a feltételeknek megfelelő ütemezését. Ebben az ütemezésben nincs holtidő, hiszen a feladatok elvégzésére rendelkezésre álló idő megegyezik a végrehajtási idők összegével. Az f_{n+1} feladat kezdési ideje itt nyilván W. Legyenek $f_{i_1}, f_{i_2}, \ldots, f_{i_k}$ azok a feladatok, amelyek f_{n+1} elé vannak ütemezve. Ekkor

$$w_{i_1} + w_{i_2} + \cdots + w_{i_k} = W$$

világos módon.

Tegyük fel ezután, hogy

$$\sum_{i \in I} w_i = W$$

valamilyen $I \subseteq \{1, 2, \ldots, m\}$ indexhalmaz esetén. Legyen

$$I = \{i_1, i_2, \ldots, i_k\}$$

egy ilyen indexhalmaz. Ütemezzük most a feladatokat úgy, hogy f_{n+1} kezdési ideje W legyen, a $f_{i_1}, f_{i_2}, \ldots, f_{i_k}$ feladatok f_{n+1} előtt jöjjenek, a többi pedig f_{n+1} után. Ez az ütemezés nyilván megfelel a kezdési és a befejezési időkre vonatkozó feltételeknek.

Minimális összsúlyú éllefogó csúcshalmaz

Feladat.
Tekintsünk egy $G = (V, E)$ irányítatlan gráfot. A G gráf csúcsainak egy U részhalmazát éllefogó halmaznak nevezzük, ha G bármely élének van U-beli végpontja. Tegyük fel, hogy adott a csúcsok halmazán egy $w\colon V \to \mathbb{R}_0^+$ nem negatív súlyfüggvény. Egy U éllefogó halmaz $w(U)$ súlya legyen a benne szereplő csúcsok súlyainak összege. Adjuk meg a G gráf egy minimális összsúlyú éllefogó csúcshalmazát!

Megoldás.
Sajnos a feladat megoldására nem ismert hatékony eljárás! Az alábbi algoritmus egy olyan éllefogó csúcshalmazt szolgáltat, amelynek összsúlya legfeljebb kétszerese az optimális éllefogó csúcshalmaz összsúlyának. Egy ilyen algoritmust közelítő algoritmusnak nevezünk.

A G gráf minden $e \in E$ éléhez rendeljünk hozzá egy $\pi(e)$ nem negatív potenciálértéket. Azt mondjuk, hogy a $\pi\colon E \to \mathbb{R}_0^+$ potenciálfüggvény kompatibilis a w súlyfüggvénnyel, ha tetszőleges $v \in V$ csúcsra

$$\sum_{(u,v)\in E} \pi(u, v) \leqslant w(v),$$

azaz a v csúcsra illeszkedő élek potenciálértékeinek összege nem haladja meg a v csúcs súlyát. Ha a v csúcsra illeszkedő élek potenciálértékeinek összege megegyezik a v csúcs súlyával, akkor v-t telített csúcsnak nevezzük.

Kezdetben minden $e \in E$ élre legyen $\pi(e) = 0$ és legyen $U = \emptyset$. Ezek után amíg van olyan él, amelynek egyik végpontja sem telített csúcs, ismételjük a következőt. Válasszunk egy ilyen e élt, és növeljük a $\pi(e)$ potenciálértéket, amíg e legalább egyik végpontja telített csúccsá nem válik. Amelyik végpont így telített csúccsá vált, azt vegyük hozzá az U halmazhoz (ha e mindkét végpontja telített csúccsá vált, akkor mindkettőt hozzávesszük U-hoz).

Világos, hogy az algoritmus befejeződése után a π potenciálfüggvény kompatibilis a w súlyfüggvénnyel. Megmutatjuk, hogy a kapott U halmaz éllefogó csúcshalmaz. Indirekt tegyük fel, hogy ez nem igaz, vagyis van olyan (u, v) él, amelynek egyik végpontja sem tartozik hozzá U-hoz. Ekkor u és v nem telített csúcsok. Ez viszont lehetetlen, hiszen az algoritmus nem fejeződik be addig, amíg van olyan él, amelynek egyik végpontja sem telített csúcs.

Mivel minden $v \in U$ csúcs telített, ezért

$$w(U) = \sum_{v\in U} w(v) = \sum_{v\in U} \sum_{(u,v)\in E} \pi(u, v).$$

Itt a jobb oldali kettős összegben minden él legfeljebb kétszer fordulhat elő, így

$$\sum_{v\in U}\sum_{(u,v)\in E}\pi(u,v)\leqslant 2\sum_{e\in E}\pi(e).$$

Innen

$$w(U)\leqslant 2\sum_{e\in E}\pi(e)$$

adódik.

Ezek után tekintsünk egy optimális U^* éllefogó csúcshalmazt. A π potenciálfüggvénynek a w súlyfüggvénnyel való kompatibilitása miatt tetszőleges $v\in U^*$ csúcsra

$$\sum_{(u,v)\in E}\pi(u,v)\leqslant w(v).$$

Összegezzük ezeket az egyenlőtlenségeket az összes $v\in U^*$ csúcsra:

$$\sum_{v\in U^*}\sum_{(u,v)\in E}\pi(u,v)\leqslant\sum_{v\in U^*}w(v)=w(U^*).$$

Mivel U^* éllefogó csúcshalmaz, ezért a bal oldalon levő kettős összegben minden él legalább egyszer előfordul, így

$$\sum_{e\in E}\pi(e)\leqslant\sum_{v\in U^*}\sum_{(u,v)\in E}\pi(u,v).$$

Innen

$$\sum_{e\in E}\pi(e)\leqslant w(U^*)$$

adódik.

Összevetve a két végső egyenlőtlenséget

$$w(U)\leqslant 2\sum_{e\in E}\pi(e)\leqslant 2w(U^*)$$

következik.

Terheléselosztás

Feladat.
Tegyük fel, hogy adottak az $a_1,a_2,\ldots,a_n$ feladatok, amelyek végrehajtásához a $b_1,b_2,\ldots,b_m$ erőforrások állnak rendelkezésre. Egy erőforrás egyszerre egy feladattal tud foglalkozni, és ha valamelyiket elkezdte, akkor azt addig nem

hagyja abba, amíg teljesen el nem készült vele. Minden a_i feladathoz adott a t_i végrehajtási ideje.

Minden a_i feladatot rendeljük hozzá valamelyik b_j erőforráshoz, ezen fogjuk a feladatot elvégezni. Egy erőforráshoz természetesen több feladatot is hozzárendelhetünk. Minden $1 \leqslant j \leqslant m$ esetén jelölje A_j azon i indexek halmazát, amelyekre az a_i feladatot a b_j erőforráshoz rendeltük. Jelölje továbbá

$$T_j = \sum_{i \in A_j} t_i$$

a b_j erőforrás terhelését, vagyis azt az időt, amely a b_j erőforráshoz rendelt feladatok elvégzéséhez szükséges. Adjunk meg egy olyan hozzárendelést, amelynél a $T = \max\{T_j \mid 1 \leqslant j \leqslant m\}$ maximális terhelés a lehető legkisebb!

Megoldás.
Sajnos a feladat megoldására nem ismert hatékony eljárás! A következő algoritmus egy olyan hozzárendelést szolgáltat, amelynél a maximális terhelés legfeljebb másfélszerese az optimális megoldáshoz tartozó maximális terhelésnek. Egy ilyen algoritmust közelítő algoritmusnak nevezünk.

Tegyük fel, hogy a feladatok a végrehajtási idejük szerint monoton csökkenően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$t_1 \geqslant t_2 \geqslant \cdots \geqslant t_n.$$

A feladatokat ebben a sorrendben rendeljük hozzá az erőforrásokhoz. Minden $1 \leqslant i \leqslant n$ esetén az a_i feladatot ahhoz a b_j erőforráshoz rendeljük, amelynek addigi terhelése (az addig hozzárendelt feladatok végrehajtási idejének összege) a legkisebb. Jegyezzük meg, hogy kupac adatszerkezettel ez hatékonyan megvalósítható.

Jelölje T az algoritmusunk által előállított hozzárendelésnél a maximális terhelést, T^* pedig egy optimális hozzárendelésnél a maximális terhelést. Megmutatjuk, hogy $T \leqslant \frac{3}{2}T^*$.

Ha $n \leqslant m$, akkor az állítás triviálisan igaz; ebben az esetben algoritmusunk egy optimális hozzárendelést állít elő. Tegyük fel ezért, hogy $n > m$. Legyen b_k az az erőforrás, amelynek az algoritmusunk által előállított hozzárendelésnél a legnagyobb a terhelése, és legyen a_l az a feladat, amelyet az algoritmusunk utoljára rendelt ehhez az erőforráshoz. Ha algoritmusunk a b_k erőforráshoz csak az a_l feladatot rendelte hozzá, akkor az állítás megint csak triviálisan igaz; ebben az esetben is algoritmusunk egy optimális hozzárendelést állít elő. Tegyük fel ezért, hogy algoritmusunk a b_k erőforráshoz legalább két feladatot rendelt hozzá. Mivel algoritmusunk az első m feladatot különböző erőforrásokhoz rendeli, ezért $l \geqslant m + 1$.

Tekintsük azt a pillanatot, amikor algoritmusunk hozzárendeli az a_l feladatot a b_k erőforráshoz. Ekkor a b_k erőforrásnak a legkisebb a terhelése. A b_k erőforrás terhelése ebben a pillanatban $T_k - t_l$, amiből következik, hogy a többi erőforrás terhelése is legalább ennyi. Az erőforrások terhelése a hátralevő időben nyilván nem csökken, így algoritmusunk befejezésekor $T_j \geqslant T_k - t_l$ minden $1 \leqslant j \leqslant m$ esetén. Adjuk össze ezeket az egyenlőtlenségeket:

$$\sum_{j=1}^{m} T_j \geqslant m(T_k - t_l),$$

illetve átrendezve

$$T_k - t_l \leqslant \frac{1}{m} \sum_{j=1}^{m} T_j.$$

Az erőforrások terhelésének összege nyilván megegyezik a feladatok végrehajtási idejének összegével:

$$\sum_{j=1}^{m} T_j = \sum_{i=1}^{n} t_i.$$

Behelyettesítünk az előző egyenlőtlenségbe:

$$T_k - t_l \leqslant \frac{1}{m} \sum_{i=1}^{n} t_i.$$

Vegyük szemügyre a jobb oldalt. Ez a végrehajtási idők összegének az m-ed része. Világos, hogy akármilyen módon is rendeljük hozzá a feladatokat az erőforrásokhoz, mindig lesz olyan erőforrás, amelynek terhelése legalább ennyi. Ebből következik, hogy

$$T^* \geqslant \frac{1}{m} \sum_{i=1}^{n} t_i.$$

Ezt az egyenlőtlenséget az előzővel kombinálva kapjuk, hogy

$$T_k - t_l \leqslant T^*.$$

Tekintsük ezután az első $m+1$ feladatot. Mivel a feladatok a végrehajtási idejük szerint csökkenően rendezettek, ezeknek a feladatoknak a végrehajtási ideje legalább t_l. Mivel csak m erőforrás áll rendelkezésre, akármilyen módon is rendeljük hozzá a feladatokat az erőforrásokhoz, mindig lesz az első $m+1$ feladat között két olyan, amelyeket ugyanahhoz az erőforráshoz rendeltünk. Ennek az erőforrásnak a terhelése világos módon legalább $2t_l$. Ebből következik, hogy

$$T^* \geqslant 2t_l,$$

illetve átrendezve

$$t_l \leqslant \tfrac{1}{2}T^*.$$

Adjuk össze a kapott két egyenlőtlenséget:

$$(T_k - t_l) + t_l \leqslant T^* + \tfrac{1}{2}T^*,$$

azaz

$$T_k \leqslant \tfrac{3}{2}T^*.$$

Innen $T = T_k$ felhasználásával az állítás adódik.

Kifinomultabb elemzéssel megmutatható, hogy az algoritmusunk által előállított hozzárendelésre $T \leqslant \frac{4}{3}T^*$ is teljesül.

Kiszolgáló egységek telepítése

Feladat.
Adott egy n elemű $P = \{p_1, p_2, \ldots, p_n\}$ pontrendszer a síkon. Tegyük fel, hogy a sík bármely két x és y pontjának ismerjük a $d(x, y)$ távolságát. A d távolságfüggvényről itt csak annyit teszünk fel, hogy

- a sík tetszőleges x és y pontjára $d(x, y) \geqslant 0$,
- a sík tetszőleges x és y pontjára $d(x, y) = 0$ akkor és csak akkor, ha $x = y$,
- a sík tetszőleges x és y pontjára $d(x, y) = d(y, x)$,
- a sík tetszőleges x, y és z pontjára $d(x, y) + d(y, z) \geqslant d(x, z)$.

A szokásos euklideszi távolságfüggvény megfelel ezeknek a feltételeknek, de szóba jöhet más távolságfüggvény is.

Adott továbbá egy $k < n$ pozitív egész szám. Egy k elemű $C = \{c_1, c_2, \ldots, c_k\}$ síkbeli pontrendszer fedési sugara legyen az

$$r(C) = \max_{1 \leqslant i \leqslant n}(\min_{1 \leqslant j \leqslant k} d(p_i, c_j))$$

mennyiség. Adjunk meg egy olyan k elemű C pontrendszert a síkon, amelynek $r(C)$ fedési sugara a lehető legkisebb!

A feladat szemléletesen a következőképpen fogalmazható meg. Tegyük fel, hogy egy áruházlánc k üzlet nyitását tervezi egy olyan régióban, ahol n kistelepülés található. Az üzletek helyét úgy szeretnék meghatározni, hogy

a települések lakóinak a lehető legkevesebbet kelljen autózni a hozzájuk legközelebbi üzletig. Hol legyenek az üzletek?

Megoldás.
Sajnos a feladat megoldására nem ismert hatékony eljárás! A következő algoritmus egy olyan k elemű C pontrendszert szolgáltat, amelynek fedési sugara legfeljebb kétszerese az optimális megoldás fedési sugarának. Egy ilyen algoritmust közelítő algoritmusnak nevezünk.

Bevezetünk egy jelölést. A sík egy x pontjára és egy Y nem üres pontrendszerére legyen

$$d(x, Y) = \min\{d(x, y) \mid y \in Y\}.$$

Az algoritmus ezek után a következő.

```
KiszolgálóEgységekTelepítése(P,k)
Legyen p tetszőleges P-beli pont
C={p}
while |C|<k do
   legyen p olyan P-beli pont, amelyre d(p,C) maximális
   C=C∪{p}
return C
```

Az algoritmus érdekes vonása, hogy C pontjait a P-beli pontok közül választja ki.

Legyen $C^* = \{c_1, c_2, \ldots, c_k\}$ egy optimális megoldása a feladatnak és legyen $r = r(C^*)$. Indirekt tegyük fel, hogy az algoritmus által előállított C pontrendszerre $r(C) > 2r$. Ekkor létezik olyan $p_{i_0} \in P$ pont, amely C minden pontjától nagyobb, mint $2r$ távolságra van, vagyis amelyre $d(p_{i_0}, C) > 2r$.

Minden $1 \leqslant j \leqslant k$ esetén jelölje p_{i_j} azt a P-beli pontot, amelyet az algoritmus a j-edik lépésben vett hozzá a C halmazhoz. Most minden $2 \leqslant j \leqslant k$ esetén

$$d(p_{i_j}, \{p_{i_1}, \ldots, p_{i_{j-1}}\}) \geqslant d(p_{i_0}, \{p_{i_1}, \ldots, p_{i_{j-1}}\}) \geqslant d(p_{i_0}, C) > 2r.$$

Ennélfogva a $\{p_{i_1}, \ldots, p_{i_k}\}$ halmaz bármely két pontja nagyobb, mint $2r$ távolságra van egymástól. Sőt ennél több is igaz, a $\{p_{i_0}, p_{i_1}, \ldots, p_{i_k}\}$ halmaz bármely két pontja nagyobb, mint $2r$ távolságra van egymástól.

Tekintsük ezután a C^* pontrendszert. Mivel $r(C^*) = r$, ezért minden $0 \leqslant j \leqslant k$ esetén létezik olyan C^*-beli pont, amely a p_{i_j} ponttól legfeljebb r távolságra van. Másrészt semelyik $c_s \in C^*$ pont nem lehet két különböző $\{p_{i_0}, p_{i_1}, \ldots, p_{i_k}\}$-beli, mondjuk p_{i_g} és p_{i_h} ponttól legfeljebb r távolságra,

ellenkező esetben

$$2r = r + r \geqslant d(p_{i_g}, c_s) + d(p_{i_h}, c_s) \geqslant d(p_{i_g}, p_{i_h}) > 2r$$

teljesülne, ami nyilván lehetetlen.

Ebből következik, hogy $|C^*| \geqslant |\{p_{i_0}, p_{i_1}, \ldots, p_{i_k}\}| = k + 1$, ellentmondás.

Irodalomjegyzék

[1] Aho, A. V., Hopcroft, J. E., Ullman, J. D.: *Számítógépalgoritmusok tervezése és analízise.* Műszaki Könykiadó, 1982.

[2] Ahuja, R. K., Magnanti, T. L., Orlin, J. B.: *Network flows: theory, algorithms, and applications.* Prentice Hall, 1993.

[3] Arora, S., Barak, B.: *Computational complexity: a modern approach.* Cambridge University Press, 2009.

[4] Ausiello, G., Petreschi, R. (eds): *The power of algorithms.* Springer, 2013.

[5] Backhouse, R.: *Algorithmic problem solving.* Wiley, 2011.

[6] Baker, K. R., Trietsch, D.: *Principles of sequencing and scheduling.* Wiley, 2009.

[7] Baase, S., Van Gelder, A.: *Computer algorithms: introduction to design and analysis.* Third edition. Addison-Wesley, 2000.

[8] Brass, P.: *Advanced data structures.* Cambridge University Press, 2008.

[9] Brassard, G., Bratley, P.: *Fundamentals of algorithmics.* Addison-Wesley, 1996.

[10] Cormen, T. H.: *Algorithms unlocked.* MIT Press, 2013.

[11] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: *Új algoritmusok.* Scolar Kiadó, 2003.

[12] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: *Introduction to algorithms.* Third edition. MIT Press, 2009.

[13] Crochemore, M., Hancart, C., Lecroq, T.: *Algorithms on strings.* Cambridge University Press, 2007.

[14] Dasgupta, S., Papadimitriou, C., Vazirani, U.: *Algorithms.* McGraw-Hill, 2006.

[15] Edmonds, J.: *How to think about algorithms.* Cambridge University Press, 2008.

[16] Even, S.: *Graph algorithms.* Second edition. Cambridge University Press, 2012.

[17] Forišek, M., Steinová, M.: *Explaining algorithms using metaphores.* Springer, 2013.

[18] Garey, M. R., Johnson, D. S.: *Computers and intractability: a guide to the theory of NP-completeness.* W. H. Freeman, 1979.

[19] Gibbons, A.: *Algorithmic graph theory.* Cambridge University Press, 1985.

[20] Goldreich, O.: *Computational complexity: a conceptual perspective.* Cambridge University Press, 2008.

[21] Goldreich, O.: *P, NP and NP-completeness.* Cambridge University Press, 2010.

[22] Goodrich, M. T., Tamassia, R.: *Algorithm design and applications.* Wiley, 2014.

[23] Horowitz, E., Sahni, S., Rajasekaran, S.: *Computer algorithms.* Second edition. Silicon Press, 2008.

[24] Hromkovič, J.: *Algorithmics for hard problems.* Second edition. Springer, 2004.

[25] Hromkovič, J.: *Design and analysis of randomized algorithms.* Springer, 2005.

[26] Hromkovič, J.: *Algorithmic adventures.* Springer, 2009.

[27] Imreh B., Imreh Cs.: *Kombinatorikus optimalizálás.* Novadat, 2005.

[28] Johnsonbaugh, R., Schaefer, M.: *Algorithms.* Addison Wesley, 2004.

[29] Juhász T., Tóth B.: *Programozási ismeretek versenyzőknek.* Műszaki Könyvkiadó, 2015.

[30] Jungnickel, D.: *Graphs, networks and algorithms.* Fourth edition. Springer, 2013.

[31] Katona Gy., Recski A., Szabó Cs.: *A számítástudomány alapjai.* Ötödik kiadás. Typotex, 2006.

[32] Kleinberg, R., Tardos, É.: *Algorithm design.* Addison-Wesley, 2006.

[33] Knuth, D. E.: *A számítógépprogramozás művészete 1. - Alapvető algoritmusok.* Műszaki Könykiadó, 1987.

[34] Knuth, D. E.: *A számítógépprogramozás művészete 2. - Szeminumerikus algoritmusok.* Műszaki Könykiadó, 1987.

[35] Knuth, D. E.: *A számítógépprogramozás művészete 3. - Keresés és rendezés.* Műszaki Könykiadó, 1988.

[36] Knuth, D. E.: *The art of computer programming, Volume 4A - Combinatorial algorithms, Part 1.* Addison-Wesley, 2011.

[37] Kozen, D. C.: *The design and analysis of algorithms.* Springer, 1992.

[38] Lawler, L. L.: *Kombinatorikus optimalizálás: hálózatok és matroidok.* Műszaki Könykiadó, 1982.

[39] Levitin, A. V.: *Introduction to the design and analysis of algorithms.* Third edition. Addison-Wesley, 2011.

[40] Levitin, A., Levitin, M.: *Algorithmic puzzles.* Oxford University Press, 2011.

[41] Lew, A., Mauch, H.: *Dynamic programming: a computational tool.* Springer, 2007.

[42] Lovász L., Gács P.: *Algoritmusok.* Harmadik kiadás. Tankönyvkiadó, 1989.

[43] Manber, U.: *Introduction to algorithms: a creative approach.* Addison Wesley, 1989.

[44] McConnell, J. J.: *The analysis of algorithms: an active learning approach.* Second edition. Jones and Bartlett Publishers, 2008.

[45] Mehlhorn, K., Sanders, P.: *Algorithms and data structures: the basic toolbox.* Springer, 2008.

[46] Motwani, R., Raghavan, P.: *Randomized algorithms.* Cambridge University Press, 1995.

[47] Neapolitan, R. E.: *Foundations of algorithms.* Fifth edition. Jones & Bartlett Learning, 2013.

[48] Nievergelt, J., Farrar, J. C., Reingold, E. M.: *Matematikai problémák megoldásainak számítógépes módszerei.* Műszaki Könyvkiadó, 1977.

[49] Papadimitriou, C.: *Számítási bonyolultság.* Novadat, 1999.

[50] Parberry, I.: *Problems on algorithms.* Prentice Hall, 1995.

[51] Pinedo, M. L.: *Scheduling: theory, algorithms, and systems.* Fifth edition. Springer, 2016.

[52] Rónyai L., Ivanyos G., Szabó R.: *Algoritmusok.* Typotex, 1999.

[53] Sedgewick, R., Flajolet, P.: *An introduction to the analysis of algorithms.* Second edition, Addison-Wesley, 2013.

[54] Sedgewick, R., Wayne, K.: *Algorithms.* Fourth edition, Addison-Wesley, 2011.

[55] Shen, A.: *Algorithms and programming: problems and solutions.* Second edition, Springer, 2010.

[56] Sierksma, G., Ghosh, D.: *Networks in action: text and computer exercises in network optimization* Springer, 2010.

[57] Sipser, M.: *Introduction to the theory of computation.* Second edition. Thomson Course Technology, 2006.

[58] Skiena, S. S.: *The algorithm design manual.* Second edition. Springer, 2008.

[59] Skiena, S. S., Revilla, M. A.: *Programming challenges: the programming contest training manual.* Springer, 2003.

[60] Stephen, R.: *Essential algorithms.* Wiley, 2013.

[61] Tarjan, R. E.: *Data structures and network algorithms.* SIAM, 1983.

[62] Tasnádi A.: *Számítástudomány gazdaságinformatikusoknak.* Aula, 2006.

[63] Valiente, G.: *Algorithms on trees and graphs.* Springer, 2002.

[64] Vazirani, U.: *Approximation algorithms.* Springer, 2001.

[65] Vöcking, B. et al (eds): *Algorithms unplugged.* Springer, 2010.

[66] Wilf, H. S.: *Algorithms and complexity.* Second edition. A K Peters, 2002.

[67] Vrajitoru, D., Knight, W.: *Practical analysis of algorithms.* Springer, 2014.

[68] Williamson, D. P., Shmoys, D. B.: *The design of approximation algorithms.* Cambridge University Press, 2011.

[69] Wu, Y., Wang, J.: *Data structure practice for collegiate programming contests and education.* CRC Press, 2016.