Algoritmizálás

Horváth Gyula
Szegedi Tudományegyetem
Természettudományi és Informatikai Kar
horvath@inf.u-szeged.hu

2. Kombinatorikus feladatok

2.1. Feladat: Ládák pakolása.

Egy vállalat udvarán egyetlen sorban vannak az elszállításra várakozó üres ládák. Három különböző típusú láda van, jelölje ezeket A, B és C. Minden láda a felső oldalán nyitott kocka alakú. Az A-típusú láda a legnagyobb és a C-típusú a legkisebb. Tehát minden C-típusú láda belerakható A-típusú és B-típusú ládába, minden B-típusú belerakható A-típusúba és A-típusúba belerakható B-típusú, majd ebbe egy C-típusú. Az a cél, hogy a ládákat úgy pakoljuk össze, hogy a lehető legkevesebb összepakolt láda legyen. A pakolást olyan robot végzi, amely a ládasor felett tud mozogni mindkét irányban, de ládát csak balról jobbra mozogva tud szállítani.
Írjon programot, amely megadja, hogy legkevesebb hány ládába lehet összepakolni a ládasort!

Bemenet

A `pakol.be` szöveges állomány első sorában a ládák n ($1 \leq n \leq 10000$) száma van. A második sor pontosan n karaktert tartalmaz (szóközök nélkül), a ládasor leírását. Minden karakter vagy 'A', vagy 'B' vagy 'C'.

Kimenet

A `pakol.ki`szöveges állomány első és egyetlen sorába azt a legkisebb K számot kell írni, amelyre a bemeneti ládasor összepakolható K ládába!

Példa bemenet és kimenet

bemenet	**kimenet**
11 CABCACCBACB	5

Megoldás

Jelölje $S = \langle x_1, x_2, \ldots, x_n \rangle$ az elpakolandó ládasort, és $M(S)$ a megoldás értékét az S sorozatra. $M(S)$ kiszámítását vezessük vissza ugyanilyen, de kisebb méretű probléma (sorozat) megoldására.
Oldjuk meg a probléma egyszerűbb változatát, amikor csak kétféle betű fordul elő a sorozatban, mondjuk 'A' és 'B'. Ekkor a megoldás egyszerű: balról-jobbra haladva, ha 'A' betű esetén maradt el 'B', amit nem raktunk bele 'A'-ba, akkor azt rakjuk bele az aktuális 'A'-ba. Ha az aktuális betű 'B', akkor növeljük az nB számláló értékét, amelynek értéke az olyan 'B' betűk száma, amelyet nem raktunk bele 'A'-ba.

```
1  nB:=0; {azon B–k száma, amelyeket nem raktunk bele A–ba}
2  m:=n;  {a megoldás}
3  for i:=1 to n do
4    case X[i] of
5      'A': if nB>o then dec(m);
6      'B': inc(nB);
7    end{case};
```

Tegyük fel, hogy mindhárom betű előfordul a sorozatban, és tekintsük az 'A', 'B' és 'C' betű első előfordulását az S sorozatban. A betűk sorrendjét tekintve Az alábbi három eset lehetséges:

```
      1.                    2.                    3.
C ... B ... A ...     C ... A ... B ...     A ... B ... C ...
B ... C ... A ...     B ... A ... C ...     A ... C ... B ...
```

Ekkor a megoldás $1+M(\overline{S})$, ahol $\overline{S}$ a három esetben a következő:
1. eset: Ekkor A-ba berakható B, majd ebbe C, tehát $\overline{S}$-t az S-ből úgy kapjuk, hogy töröljük 'A', 'B' és 'C' betű első előfordulását.
2. eset: Ekkor a sorozat első betűje ('C' vagy 'B') rakható az első A-ba és $\overline{S}$-t az S-ből úgy kapjuk, hogy töröljük az első betűt.
3. eset: Az első A-ba nem tudunk rakni, és $\overline{S}$-t az S-ből úgy kapjuk, hogy töröljük az első betűt.
Kivitelezés. A törléseket nem kell ténylegesen elvégezni, ha balról-jobbra vizsgáljuk a sorozatot, és számítjuk, hogy hány olyan elemet hagytunk el eddig, amely berakható másikba. Ezeket az nB, nC, nBC változókban számítjuk.

```
1  program Pakol;
2  Var
3    n:Word;          { a ládák száma }
4    BeF,KiF:Text;    { Be–Ki állományok }
5    nB,              { nB db B láda marad el }
6    nC,              { nC db C láda marad el }
7    nBC,             { nBC db B–benC láda marad el }
8    atPakolt:Word;   { ennyit pakoltunk be másikba}
9    k,i:Word;
10   X:char;
11
12 begin{}
13   Assign(BeF, 'pakol.be'); Reset(BeF);
14   Assign(Kif, 'pakol.ki'); Rewrite(KiF);
15
16   ReadLn(BeF,n);
17
18   atPakolt:=0;
19   nB:=0; nC:=0; nBC:=0;

20   for i:=1 to n do begin
21     read(BeF,X);
22     case X of
23       'A': if (nB>0)and(nC>0) then begin
24              Dec(nB); Dec(nC);
25              Inc(atPakolt,2);
26            end else if nBC>0 then begin
27              Dec(nBC);
28              Inc(atPakolt);
29            end else if nB>0 then begin
30              Dec(nB);
31              Inc(atPakolt);
32            end  else if nC>0 then begin
33              Dec(nC);
34              Inc(atPakolt);
35            end;
36       'B': if nC>0 then begin
37              Dec(nC);
38              Inc(atPakolt);
39              Inc(nBC)
40            end else
41              Inc(nB)
42       'C': Inc(nC)
43       end{case};
```

```
44    end{for i->n};
45
46    WriteLn(KiF,n-atPakolt);
47
48    Close(BeF);
49    Close(KiF);
50  end.
```

2.2. Feladat: Kód

A processzorgyártó cégek megállapodtak abban, hogy milyen rendszert alkalmaznak az általuk gyártott processzorok egyedi azonosítására. Minden cég kap egy betűkészletet, és ezekből kell az azonosító kódot képeznie, úgy, hogy minden betű meghatározott számszor szerepeljen az azonosítóban. Például egy cég azt kapta, hogy minden azonosítója pontosan 3 db 'a' betűt, 2 db 'b'betűt és 1 db 'c' betűt tartalmazhat.
Készítsen olyan programot, amely adott kódra meghatározza a lexikografikus (ábécé szerinti) sorrendben rákövetkező szabályos kódot, ha van rákövetkező!

Bemenet

A `kod.be` szöveges állomány első sorában a kódok m száma $(1 \leq m \leq 1000)$ van. A további m sorban az egyes kódok találhatók. Minden kód legfeljebb 200 betűből állhat, csak az angol ábécé kis betűit tartalmazhatja.

Kimenet

A `kod.ki` szöveges állományba pontosan m sort kell írni! Az i-edik sorba a bemeneti állomány $i+1$-edik sorában lévő kód rákövetkezőjét kell írni, ha nincs rákövetkezője, akkor a `NINCS` szót.

Példa bemenet és kimenet

bemenet

```
2
abaacb
cbbaaa
```

kimenet

```
ababac
NINCS
```

Megoldás

A megoldás elemzése. Tegyük fel, hogy a $K = \langle k_1, \ldots, k_n \rangle$ bemenetre a megoldás a $\overline{K} = \langle \overline{k_1}, \ldots, \overline{k_n} \rangle$ karaktersorozat. A lexikografikus rendezés definíciója szerint K akkor és csak akkor előzi meg a rendezésben a $\overline{K}$-t, ha van olyan $1 \leq i \leq n$ index, hogy $k_j = \overline{k_j}$ ha $j < i$ és $k_i < \overline{k_i}$. Mivel minden kódban minden betű ugyanannyiszor fordul elő, ezért $\overline{k_i} = k_j$ valamely $j > i$ indexre. Tehát

$$k_i \quad < \quad \max\{k_j : i < j \leq n\} \tag{1}$$

Ha több ilyen i index lenne, akkor a legnagyobb olyan i-t kell venni, amelyre teljesül az (1) képlet.

$$\overline{k_i} \quad = \quad \min\{k_j : i < j \leq n \wedge k_i < k_j\} \tag{2}$$

Legyen j az az index, amelyre a (2) formulában a minimum adódik. Nyilvánvaló, hogy a $\langle \overline{k_{i+1}}, \ldots, \overline{k_n} \rangle$ sorozat a $\{k_i, \ldots, k_n\} - \{k_j\}$ karakterekből képezhető lexikografikus rendezés szerinti első kell legyen. Tehát a megoldás:
$Kovet(K) = \langle k_1, \ldots, k_{i-1} \rangle k_j \langle \overline{k_{i+1}}, \ldots, \overline{k_n} \rangle$, ahol $\langle \overline{k_{i+1}}, \ldots, \overline{k_n} \rangle$ a $\{k_i, \ldots, k_n\} - \{k_j\}$ betűkből képzett lexikografikus rendezés szerinti első szó.
A megoldás kiszámítása.
Az (1) feltételt teljesítő legnagyobb i kiszámítása egyszerű.
Ez után a (2) képlet szerinti j kiszámítható lenne a $K[i..n]$ sorozaton végigmenve. A $\overline{k_{i+1}}, \ldots, \overline{k_n}$ sorozat kiszámítható lenne a $k_{i+1}, \ldots, k_n$ sorozat rendezésével, számláló rendezést alkalmazva. Azonban, ha tovább elemezzük a $\overline{k_i}, \ldots, \overline{k_n}$ sorozatot, akkor egyszerűbb megoldáshoz juthatunk. Ha i a legnagyobb olyan index, amelyre teljesül az (1) feltétel és j amelyre teljesül (2), akkor

$$k_i < k_{i+1} \geq \cdots \geq k_j \geq \cdots \geq k_n \tag{3}$$

$$k_i < k_j \tag{4}$$

$$k_i \geq k_{j+1} \tag{5}$$

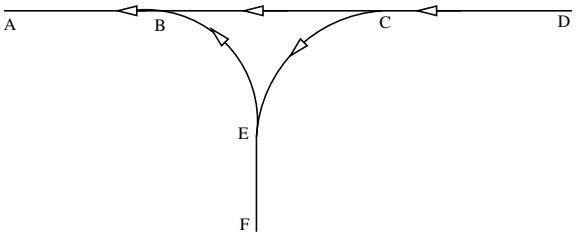
Ugyanis bármely $j > i$-re nem lehet $k_j < k_{j+1}$, mert i a legnagyobb olyan index, hogy van x_i-nél nagyobb tőle jobbra. Továbbá, $k_i \geq k_{i+1}$ sem lehet, mert akkor i nem a legnagyobb olyan index lenne, amelyre (1) teljesül. Tehát a $\overline{k_{i+1}}, \ldots, \overline{k_n}$ sorozat a $k_{i+1}, \ldots, k_n$ sorozat fordított sorrenden, de k_j helyébe k_i-t írva.

```
1  program Kod;
2  const
3    MaxN=200;
4  var
5    K,          {a bementi sorozat}
6    KK: String;{a kimeneti sorozat}
7    M:Word;     {a bemeneti sorozatok száma}
8    N:Word;     {a bemenet sorozat hossza}
9    Van:Boolean;
10   BeF,        {a bemeneti állomány }
11   KiF:Text;   {a bemeneti állomány }
12   t:Word;     {a tesztesetek sz ma}

13 procedure Szamit;
14 {Global: K, KK}
15 var
16   i,j:word;
17   x:char;
18 begin
19   N:=Length(K);
20   i:=N-1;
21   Van:=false;
22   while (i>0) and (K[i] >= K[i+1]) do
23     Dec(i);
24   Van:= i>0;
25   if i=0 then Exit;
26   KK:=K;
27   x:=K[i];
28   for j:=N downto i+1 do begin
29      if K[j]>X then begin
30        KK[i]:=K[j];
31        KK[i+N-j+1]:=K[i];
32        X:='z';
33      end else
34        KK[i+N-j+1]:=K[j];
35   end;
36 end{Szamit};

37 begin{Kod}
38   Assign(BeF,'kod.be'); Reset(BeF);
39   ReadLn(BeF,M);
40   Assign(KiF,'kod.ki'); Rewrite(KiF);
41   for t:=1 to M do begin
42     ReadLn(BeF,K);
43     Szamit;
44     if Van then
45       WriteLn(KiF,KK)
46     else
47       WriteLn(KiF,'NINCS')
48   end{for t};
49   Close(BeF);
50   Close(KiF);
51 end{Kod}.
```

2.3. Feladat: Vasúti kocsik rendezése.



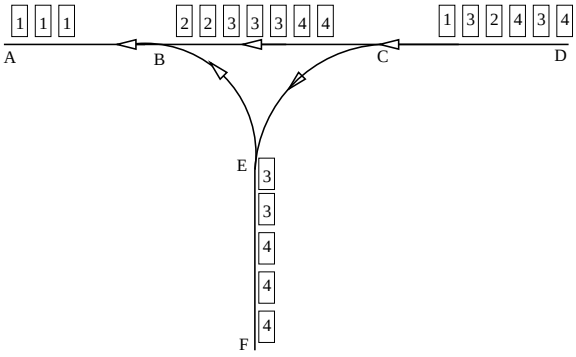
1. ábra. Az állomás vágányrendszere.

Veremsor város vasútállomásán nagy gondot okoz a szerelvények rendezése. Az állomásról továbbítandó szerelvényeket úgy kell kialakítani, hogy amikor az megérkezik a célállomásra, a szerelvény végéről mindig lekapcsolható legyen az oda továbbított kocsisor. Minden továbbítandó szerelvény négy állomást érint, ezért a rendezés előtt minden kocsit megjelölnek az 1, 2, 3 vagy 4 számokkal. A szerelvény kocsijait rendezzük át úgy, hogy a szerelvény elején legyenek az 1-essel, aztán a 2-essel, majd a 3-assal, végül a 4-essel megjelöltek. Kezdetben a kocsik az ábrán látható C-D pályaszakaszon vannak. A vasúti váltók működése csak a következő műveleteket teszi lehetővé. Az átrendezendő kocsisorból balról az első kocsit át lehet mozgatni vagy a B-C szakaszba a már ott lévő kocsik mögé, vagy az E-F szakaszba a már ott lévő kocsik elé. A B-C szakaszban lévő első kocsi átmozdítható és hozzáilleszthető az A-B szakaszon kialakítandó kocsisor végére. Hasonlóan, az E-F szakasz első (tehát az utolsónak oda érkezett) kocsija átmozdítható és hozzáilleszthető az A-B szakaszon kialakítandó kocsisor végére.
A feladat annak eldöntése, hogy egy adott kocsisor rendezhető-e a megengedett mozgatásokkal?

Megoldás

Jelölje $K = (k_1, \ldots, k_n)$ a bemeneti kocsisor címkéinek sorozatát ($k_i \in \{1,2,3,4\}$). Tegyük fel, hogy a bemenet rendezhető és a rendezés során az első $i-1$ kocsit már átmozgattuk a C-D szakaszról, és $k_i = 1$. Ekkor az alábbi 3 feltétel mindegyikének igaznak kell lenni:

- Az A-B szakaszon csak 1-el címkézett kocsi van.
- A B-C szakaszon (sor) lévő kocsik címkéi balról jobbra monoton nemcsökkenőek.
- Az E-F szakaszon (verem) lévő kocsik címkéi felülről lefelé monoton nemcsökkenőek.



2. ábra.

Ekkor az i-edik kocsi ($k_i = 1$) a vermen keresztül helyére rakható. Ha $k_i = 1$ és ez az utolsó 1-es a bemenetben, akkor a rendezés folytatható a következőképpen: az utolsó 1-est kivisszük a vermen át a helyére, majd a veremből minden 2-est kiviszünk a helyére, azaz az A-B szakasz végére, majd a C-D szakaszon várakozó minden kocsira: ha az 2-es, akkor a vermen át átvisszük az A-B szakasz végére, ha 3-as, akkor a verembe viszünk (ott az utolsó elem nem 2-es, tehát továbbra is monoton lesz), ha 4-es, akkor a sor végére viszünk (ami továbbra is monoton lesz), végül a sorból és a veremből a kocsikat összefésülve kivisszük a helyükre.
Állítás: A bemenet akkor és csak akkor rendezhető, ha az utolsó 1-es előtti részből elhagyva az 1-eseket, az felbontható egy monoton nemcsökkenő és egy monoton nemnövekvő sorozat fésűs egyesítéseként. (A növekvő megy a sorba, a csökkenő megy a

verembe.) Hogyan dönthető el, hogy az állítás feltétele teljesül-e?
Legyen $A(i) = \{(s,v) : a(k_1,\ldots,k_i)$ sorozat elemei berakhatók a sorba és a verembe úgy, hogy a sor végén s, a verem tetején pedig v lesz $\}$.

$$A(i+1) = \{(k_i,v) : \exists (s,v) \in A(i), s \le k_i\} \bigcup \{(s,k_i) : \exists (s,v) \in A(i), k_i \le v\}$$

$A(0) = \{(2,4)\}$
A bemenet akkor és csak akkor rendezhető, ha $A(u-1) \neq \emptyset$, ahol u az utolsó 1-es pozíciója (vagy 0, ha nincs 1-es).

```
1  program Vasuti;
2  const
3   MaxN=1000;
4  type
5    KocsiSor=array[1..MaxN] of byte;
6
7  function Rendezheto(const K: Kocsisor; n:word):Boolean;
8  var
9    A,A0:array[2..4,2..4] of Boolean; {az állapot }
10   OK:Boolean;
11   x,s,v:Byte;
12   i,U1:Integer;
13 begin{Rendezheto}
14   U1:=0;
15   for i:=n downto 1 do
16     if K[i]=1 then U1:=1; {U1 az utolsó 1-es indexe}
17   if U1<=1 then begin
18     Rendezheto:=true; exit;
19   end;
20   for v:=2 to 4 do        {a kezdő állapot előállítása }
21     for s:=2 to 4 do
22       A0[s,v]:=false;
23   A0[2,4]:=true;

24   for i:=1 to U1-1 do begin
25     x:=K[i];
26     if x=1 then Continue;       {az 1-eseket kihagyjuk}
27     OK:=False;
28     for s:=2 to 4 do            {az új állapot inicializálása }
29       for v:=2 to 4 do A[s,v]:=False;
30     for s:=2 to 4 do            {A[s,v]=True <=> a K[1..i-1] sor felbontható }
31       for v:=2 to 4 do          {úgy, hogy a sor végén s, a verem tetején v van}
32           if A0[s,v] then begin
33             if (s<=x) then begin{x a sorba rakható }
34               OK:=True;
35               A[x,v]:=True;
36             end;
37             if (v>=x) then begin{x a verembe rakható }
38               OK:=True;
39               A[s,x]:=True;
40             end;
41       end{if, for s,v};
42     if not OK then Break;
43     A0:=A;
44   end{for i};
45   Rendezheto:=OK;
46 end{Rendezheto};
```

2.4. Feladat: Riadólánc készítése.

Egy osztály diákjai elhatározták, hogy riadóláncot alkotnak. Minden diák választ magának egyetlen társat (szomszédot), akinek a hozzá beérkező üzenetet közvetlenül továbbítja. Amikor egy diák megkap egy üzenetet, továbbítja szomszédjának. Riadóláncnak azt a hozzárendelést nevezzük, melyre a következők teljesülnek: Tegyük fel, hogy valaki elküld egy üzenetet a szomszédjának, aki a következő lépésben továbbítja azt, és így tovább. Az üzenet egy idő után minden diákhoz, köztük az üzenet küldőjéhez is megérkezik. Természetesen nem minden hozzárendelés riadólánc.
Írjon programot amely kiszámítja, hogy minimálisan hány módosítás szükséges a bemeneti hozzárendelés riadólánccá változtatásához és adjon is meg egy módosítást.

Bemenet

A `riado.be` szöveges állomány első sora a tanulók n ($1 < n \leq 10000$) számát tartalmazza. A tanulókat az $1,\ldots,n$ számokkal azonosítjuk. A második sor pontosan n egész számot tartalmaz egy-egy szóközzel elválasztva. A sorban az i-edik szám annak a tanulónak a sorszáma, aki az i-edik tanuló szomszédja, tehát akinek az üzenetet továbbítja az i-edik tanuló.

Kimenet

A `riado.be` szöveges állomány első sora egyetlen számot tartalmazzon, a lehető legkevesebb szükséges módosítások k számát! A következő k sor mindegyike egy-egy módosítást tartalmazzon, két egész számot egy szóközzel elválasztva: $i\,j$. Ez azt jelenti, hogy a módosítás következtében az i-edik tanuló a j-edik tanulónak fogja átadni az üzenetet. Több megoldás esetén bármelyik kiírható.

Példa bemenet és kimenet

bemenet

```
10
6 9 2 7 3 1 9 3 7 9
```

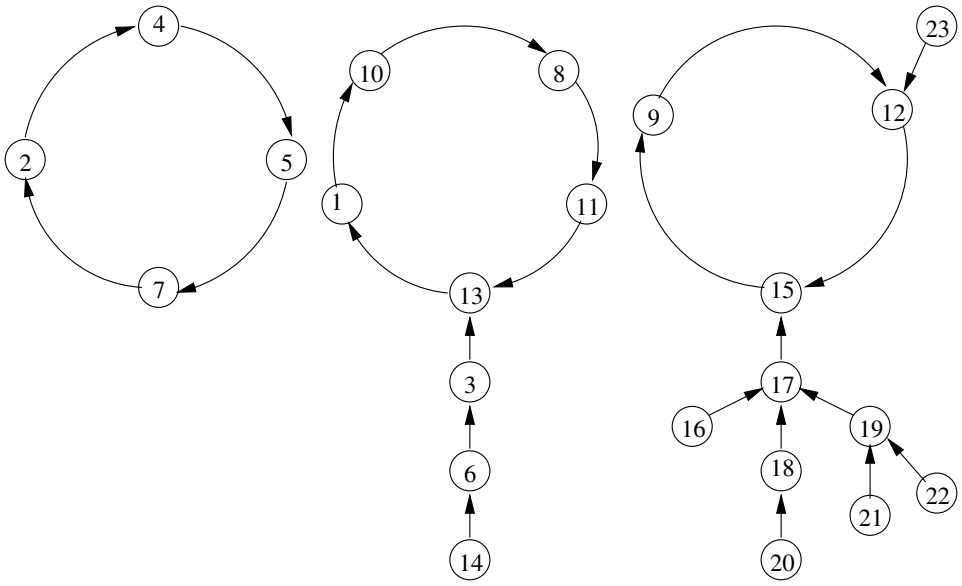
kimenet

```
5
2 4
6 10
8 5
9 1
10 8
```

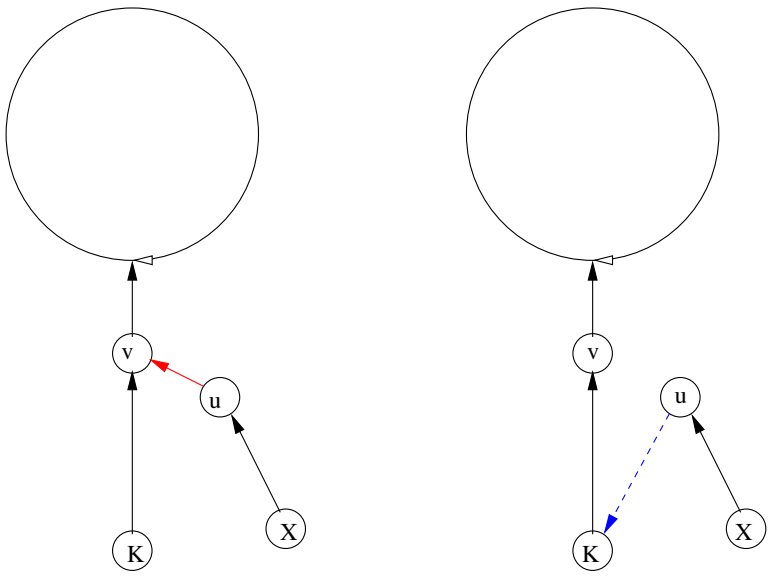
Megoldás

A tanulókat az $1..n$ számokkal azonosítva, a bement egy $F : \{1,\ldots,n\} \to \{1,\ldots,n\}$ függvény.
Tekintsük az F függvény gráfját, tehát azt a gráfot, amelynek pontjai (csúcsai) az $1..n$ számok, és irányított élei pedig az $(x,F(x))$ párok. Jelölje $F^k : 1..n \to 1..n$ az F függvény k-adik iteráltját, amelynek definíciója:
$F^0(x) = x, F^k(x) = F(F^{k-1}(x))$ha $k > 0$. Azt mondjuk, hogy x és y ugyanazon komponenshez tartozik, ha van olyan $q \geq 0$ és $r \geq 0$, hogy $F^q(x) = F^r(y)$. Egy x pont **befoka** azon y pontok száma, amelyre $F(y) = x$. A megoldás biztosan nagyobb, vagy egyenlő, mint a 0-befokú pontok száma + a farok nélküli körök száma. Megmutatjuk, hogy ennyi módosítással egyetlen körré alakítható a függvény gráfja.

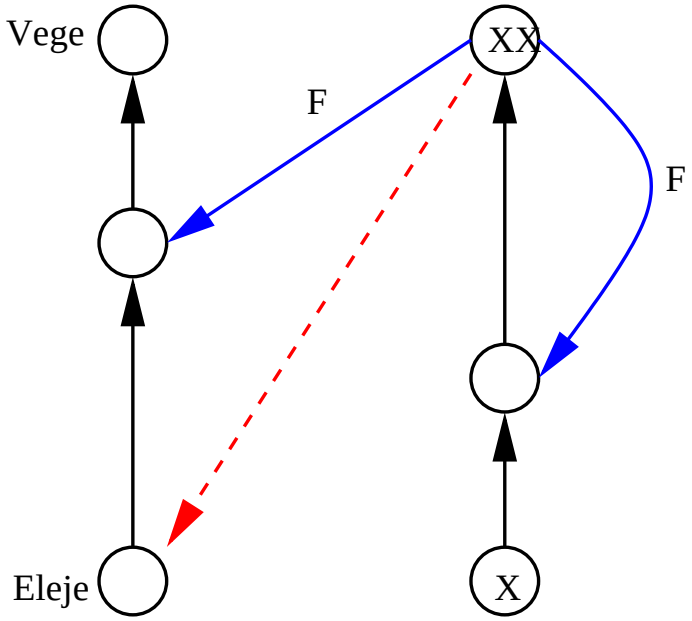
A 6. ábrán látható módosítással egy 0-befokú pont megszüntethető. Tehát ha egy komponensben m 0-befokú pont van, akkor $m-1$ módosítással olyanná alakítható, amelyben pontosan egy 0-befokú pont van.



3. ábra. Egy függvény gráfja.



4. ábra. Egy módosítással egy 0-befokú pont megszüntethető.



5. ábra. Az *Eleje* → *Vege* lánchoz kapcsoljuk az $x \to xx$ láncot. $F[xx]$ már szerepel vagy az *Eleje* → *Vege* láncban, vagy $x \to xx$ láncban.

```
1  program RiadoLanc;
2  const
3    MaxN=10000;                    {a tanulók max. száma}
4  type
5     Paletta=(Feher, Fekete);
6  var
7    N:Word;                        {a tanulók száma}
8    F:array[1..MaxN] of Word;      {a bemenet}
9    BeFok:array[1..MaxN] of Word;
10   MF:array[1..MaxN] of Word;     {a módosított függvény}
11    Szin:Array[1..MaxN] Of Paletta;
12    Modos:Word;                   {a min. módosítások száma}
13    Eleje,Vege: Word;             {a lánc eleje és vége}
14    x,xx: Word;
15    KiF:Text;                     {a kimeneti llomány}

16 procedure Beolvas;
17 {Global: N, F}
18    var i,j,k:Word;
19       BeF:Text;
20 begin{Beolvas}
21       Assign(BeF,'riado.be');
22       Assign(KiF,'riado.ki');
23       Reset(BeF); Rewrite(KiF);
24       Readln(BeF,N);
25       For i:= 1 to N do begin
26       Read(BeF,F[i]);
27       inc(BeFok[F[i]]);
28    end;
29       Close(BeF);
30 end{Beolvas};
```

```
31 begin{program}
32   Beolvas;
33   for x:=1 to N do begin          {inicializálás}
34     Szin[x]:=Feher;
35     MF[x]:=0;
36   end;
37   Eleje:=0; Vege:=0; Modos:=0;
38   for x:=1 to N do
39     if (Befok[x]=0) then begin{x-től induló lánc előállítása}
40       xx:=x;
41       Szin[xx]:=Fekete;
42       while Szin[F[xx]]=Feher do begin
43         xx:=F[xx];
44         Szin[xx]:=Fekete;
45       end;
46       if Vege=0 then begin
47         Eleje:=x; Vege:=xx {a kezdő lánc esetén}
48       end else begin        {az x-->xx lánc bekapcsolása}
49         MF[xx]:=Eleje;      {az eddigi Eleje-->Vege lánchoz}
50        inc(Modos);         {x-->xx->Eleje-->Vege}
51      end;
52       Eleje:=x;            {x lesz az új lánc kezdete}
53     end{for-if};

54   for x:=1 to N do          {az önálló körben lévők maradhattak csak ki}
55     if (Szin[x]=Feher) then begin{x-től induló lánc előállítása}
56       xx:=x;
57       Szin[xx]:=Fekete;
58       while Szin[F[xx]]=Feher do begin
59         xx:=F[xx];
60         Szin[xx]:=Fekete;
61       end;
62       if Vege=0 then
63         Vege:=xx            {a kezdő lánc esetén}
64       else if F[xx]<>x then begin{az x-->xx lánc bekapcsolása}
65         MF[xx]:=Eleje;      {az eddigi Eleje-->Vege lánchoz}
66        inc(Modos);         {x-->xx->Eleje-->Vege}
67      end;
68      Eleje:=x;            {x lesz az új lánc kezdete}
69    end{for-if};
70    if F[Vege]<>Eleje then begin{ha nem egyetlen kör a bemenet}
71       MF[Vege]:=Eleje;          {a lánc korbe zárása}
72     inc(Modos);
73    end;
74   Writeln(KiF,Modos);
75   for x:=1 to N do             {a módosítások kiíratása}
76     if MF[x]>0 then  Writeln(KiF, x,' ',MF[x]);
77   Close(KiF);
78 end.
```