

A programozási versenyek szerepe az oktatásban (Hazai és nemzetközi versenyek tapasztalatai)

Horváth Gyula
Szegedi Tudományegyetem
horvath@inf.u-szeged.hu
INFO ÉRA Konferencia
Békéscsaba

2004. november 19.

1. Készségek és képességek

- Logikus gondolkodás
- Kreativitás
- Absztrakciós készség
- Problémamegoldási készség
- Algoritmikus gondolkodás
- Komplex tevékenység szervezése
- Döntési képesség (stratégiák)
- A hatékonyság és szerepének felismerése
- Ösztönzés
- Elismert szellemi teljesítmény
- Önálló munka
 - önálló ismeretszerzés (pr. nyelv elsajátítás)
 - önálló gondolkodásra készítés

Tehetséges tanulók kiválasztása (kiválasztódása)

Nemzetközi megmérettetés

Programozás = algoritmikus problémamegoldás

A megoldandó probléma hétköznapi nyelven megfogalmazott számítási feladat.

- A feladat megértése
- Modell készítés (absztrakt feladat)
- Összefüggések kiderítése és megfogalmazása
- Algoritmus tervezés
- Helyesség igazolás
- Hatékonyság elemzés
 - futási idő becslése

– memóriaigény becslése

- Kódolás
- Tesztelés
- Véglegesítés

Versenyek

- Nemes Tihamér Országos Középiskolai Számítástechnikai Tanulmányi Verseny
- OKTV Programozás
- Közép-európai Informatikai Diákolimpia (CEOI)
- Nemzetközi Informatikai Diákolimpia (IOI)

Tehetséggondozás

NJSZT Tehetséggondozási Szakosztály tervezete (2003)

- Regionális tehetséggondozó foglalkozások szervezése (az ország 30 városában, 2 éven keresztül, évente 6-8 alkalommal a megye legtehetségesebb 9-11. osztályosai számára).
- Országos tehetséggondozó foglalkozások szervezése (az ország 2-3 városában, évente 6 alkalommal a legtehetségesebb 11-12. osztályosok számára).
- Diákolimpiai levelező felkészítés (az előző tanévben diákolimpiai válogatóversenyen szerepelt, s az adott évben még indulásra jogosult tanulók számára).
- Diákolimpiai válogatóversenyek és felkészítő táborok szervezése (más tantárgyakhoz hasonlóan a válogatóverseny és a felkészítés összekötése, 2*1 hetes felkészítéssel).

Megyei szakkörök

(2 év anyaga)

1. Programozási tételek: elemi algoritmusok áttekintése
2. Adattípusok: tömbök, halmazok
3. Programozási tételek: unió, metszet, összefuttatás, szétválogatás
4. Adattípusok: verem, sor, prioritási sor
5. Adattípusok: táblázatok
6. Rekurzio
7. Szoftverfejlesztő eszközök
8. Rendezési algoritmusok
9. Adattípusok: láncolt ábrázolás
10. Visszalépéses keresés
11. Logikai programozás alapjai
12. Funkcionális programozás alapjai
13. Kombinatorikus, logikai feladatok
14. Szoftverfejlesztő eszközök

Országos szakkör

(2 év anyaga)

1. Gráf-algoritmusok: a szélességi és mélységi bejárás
2. Gráf-algoritmusok a szélességi és mélységi bejárásra alapozva
3. Gráf algoritmusok összes útra
4. Fák, bináris fák
5. Mohó algoritmus
6. Dinamikus programozás
7. Listák, láncolt ábrázolás
8. Geometriai algoritmusok
9. Interaktív feladatok

Versenykörnyezet

Operációs rendszer a fejlesztésre (a verseny alatt)

- MS Windows (XP)
- Linux (Debian, Radhat)

Programozási nyelvek

1. FreePascal
2. GNU C
3. GNU C++

Feladat nehézségi szintek

1. lokális verseny
2. válogató (CEOI, IOI)
3. nemzetközi

Feladat típusok, megoldási módszerek

1. **Kombinatorikus**
2. **Visszalépéses keresés**
3. **Mohó algoritmusok**
4. **Dinamikus programozás**
5. **Gráf-algoritmusok**
6. **Geometriai feladatok**
7. **Interaktív feladatok**

2. Kombinatorikus feladatok

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!

Bemeneti specifikáció

A `kod.be` szöveges állomány első sorában a kódok m száma ($1 \leq m \leq 1000$) van. A további m sorban az egyes kódok találhatóak. Minden kód legfeljebb 100 betűből állhat, csak az angol ábécé kis betűit tartalmazhatja. 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

`kod.be`

2
abaacb
cbbaaa

Megoldás.

`kod.ki`

ababac
NINCS

A megoldás elemzése. Tegyük fel, hogy a $K = \langle k_1, \dots, k_n \rangle$ bemenetre a megoldás a $\bar{K} = \langle \bar{k}_1, \dots, \bar{k}_n \rangle$ karaktersorozat. A lexikografikus rendezés definíciója szerint K akkor és csak akkor előzi meg a rendezésben a \bar{K} -t, ha van olyan $1 \leq i \leq n$ index, hogy $k_j = \bar{k}_j$ ha $j < i$ és $k_i < \bar{k}_i$. Mivel minden kódban minden betű ugyanannyiszor fordul elő, ezért $\bar{k}_i = k_j$ valamely $j > i$ indexre. Tehát

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

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

$$\bar{k}_i = \min\{k_j : i < j \leq n \wedge k_i < k_j\} \quad (2)$$

Legyen j az az index, amelyre a (2) formulában a minimum adódik. Nyilvánvaló, hogy a $\langle \bar{k}_{i+1}, \dots, \bar{k}_n \rangle$ sorozat a $\{k_i, \dots, k_n\} - \{k_j\}$ karakterekből képezhető lexikografikus rendezés szerinti első kell legyen. Tehát a megoldás:

$Kovet(K) = \langle k_1, \dots, k_{i-1} \rangle k_j \langle \bar{k}_{i+1}, \dots, \bar{k}_n \rangle$, ahol $\langle \bar{k}_{i+1}, \dots, \bar{k}_n \rangle$ a $\{k_i, \dots, k_n\} - \{k_j\}$ 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 $\bar{k}_{i+1}, \dots, \bar{k}_n$ sorozat kiszámítható lenne a k_{i+1}, \dots, k_n sorozat rendezésével, számláló rendezést alkalmazva. Azonban, ha tovább elemezzük a $\bar{k}_i, \dots, \bar{k}_n$ sorozatot, akkor egyszerűbb megoldáshoz juthatunk. Ha i a legnagyobb olyan index, amelyre teljesül az (1) feltétel és j amelyre teljesül (2), akkor

$$k_i < k_{i+1} \geq \dots \geq k_j \geq \dots k_n \quad (3)$$

$$k_i < k_j \quad (4)$$

$$k_i \geq k_{j+1} \quad (5)$$

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 $\bar{k}_{i+1}, \dots, \bar{k}_n$ sorozat a k_{i+1}, \dots, k_n sorozat fordított sorrendben, de k_j helyébe k_i -t írva.

Program Kod; Var

```
K,          {a bemeneti sorozat}  
KK:String; {a kimeneti sorozat}
```

```

M:Word;    {a bemeneti sorozatok száma}
N:Word;    {a bemenet sorozat hossza}
t:Word;    {az aktuális teszteset sorszáma}
Van:Boolean;
BeF:Text;  {a bemeneti állomány }
KiF;       {a kimeneti állomány }

```

```

Procedure Szamit; Var
  i, j:Word;
  x:Char;
Begin
  N:=Length(K);
  i:=N-1;
  Van:=False;
  While (i>0) And (K[i] >= K[i+1]) Do
    Dec(i);
  Van:= i>0;
  If i=0 Then Exit;
  KK:=K;
  x:=K[i];
  For j:=N DownTo i+1 Do Begin
    If K[j]>x Then Begin
      KK[i]:=K[j];
      KK[i+N-j+1]:=K[i];
      X:='z';
    End Else
      KK[i+N-j+1]:=K[j];
  End;
End{Szamit};

```

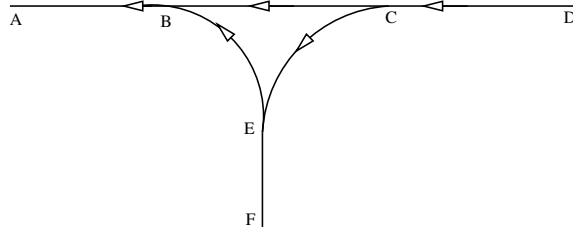
```

Begin{Kod}
  Assign(BeF, 'kod.be'); Reset(BeF);
  ReadLn(BeF, M);
  Assign(KiF, 'kod.ki'); Rewrite(KiF);
  For t:=1 To M Do Begin
    ReadLn(BeF, K);
    Szamit;
    If Van Then
      WriteLn(KiF, KK)
    Else
      WriteLn(KiF, 'NINCS')
  End{for t};
  Close(BeF);
  Close(KiF);
End{Kod}.

```

Feladat: Vasúti kocsik rendezése.

Veremser 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 kocsi megjelölnek az 1, 2, 3 vagy 4 számokkal. A szerelvény kocsijait rendezzük át úgy, hogy a szerelvény elején legyenek az 1-el, aztán a 2-vel, majd a 3-al, végül a 4-el 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 utolsó) oda



1. ábra. Az állomás vágányrendszere.

érkezett) kocsija átmozdítható és hozzáilleszhető az **A-B** szakaszon kialakítandó kocsisor végére. **Megoldás**

Jelölje $K = \langle k_1, \dots, k_n \rangle$ 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 szakasról, 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.

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 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 (tehát 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 $A(i) = \{(s, v) : \text{a } \langle k_1, \dots, k_i \rangle \text{ sorozat elemei berakhatók a sorba és a verembe úgy, hogy a sor végén } s, \text{ a verem tetején pedig } v \text{ lesz}\}$.

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

$$A(0) = \{(2, 4)\}$$

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 a bemeneti sorozatban).

```
Function Jo:Boolean; Var
  A:Array[Boolean] Of Array[2..4,2..4] Of Boolean; {az állapot }
  OK,Regi,Uj:Boolean;
  x,s,v:Byte;
  i:Integer;
Begin{Jo}
  For v:=2 To 4 Do           {a kezdő állapot előállítása }
    For s:=2 To 4 Do
      A[False,s,v]:=False;
  A[False][2,4]:=True;
  Regi:=True;
  OK:=U1=0;

  For i:=1 To U1-1 Do Begin
    x:=K[i];
    If x=1 Then Continue;   {az 1-eseket kihagyjuk}
    OK:=False;
    Uj:=Regi; Regi:=Not Uj;
    For s:=2 To 4 Do       {az új állapot inicializálása }
```

```

For v:=2 To 4 Do A[Uj][s,v]:=False;
For s:=2 To 4 Do      {A[Regi][s,v]=True <=> a K[1..i-1] sor felbontható }
For v:=2 To 4 Do      {úgy, hogy a sor végén s, a verem tetején v van}
  If A[Regi][s,v] Then Begin
    If (s<=x) Then Begin{x a sorba rakható }
      OK:=True;
      A[Uj][x,v]:=True;
    End;
    If (v>=x) Then Begin{x a verembe rakható }
      OK:=True;
      A[Uj][s,x]:=True;
    End;
  End{if, for s,v};
  If Not OK Then Break;
End{for i};
Jo:=OK;
End{Jo};

```

Feladat: Riadólánc készítése. (CEOI'95)

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 diákhhoz, köztük az üzenet küldőjéhez is megérkezik. Természetesen nem minden hozzárendelés riadólánc.

Írjunk programot amely kiszámítja, hogy minimálisan hány módosítás szükséges az input hozzárendelés riadólánccá változtatásához.

Példa bemenet és kimenet

riado.be

```

10
Anita>Peter
Andrew>Julia
David>Andrew
Natalie>Gabriella
Edith>David
Peter>Anita
Gabriella>Julius
Adam>David
Julia>Gabriella
Julius>Julia

```

riado.ki

4

Megoldás

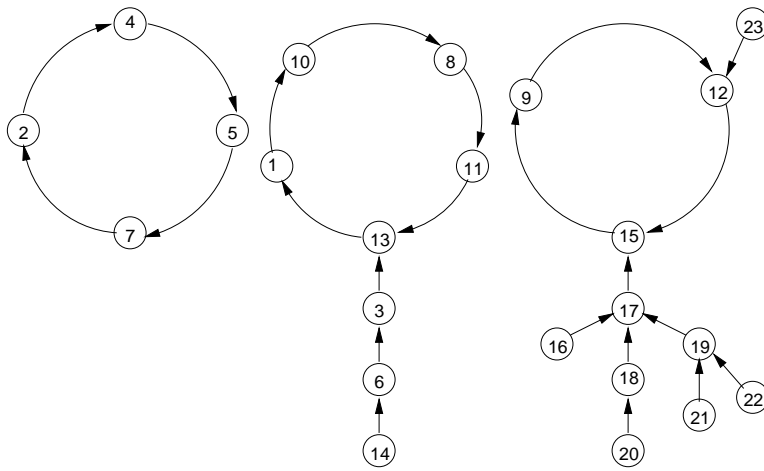
A tanulókat az $1..n$ számokkal azonosítva, a bement egy $F : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ függvény.

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 $F^k : 1..n \rightarrow 1..n$ az F függvény k -adik iteráltját, amelynek definíciója:

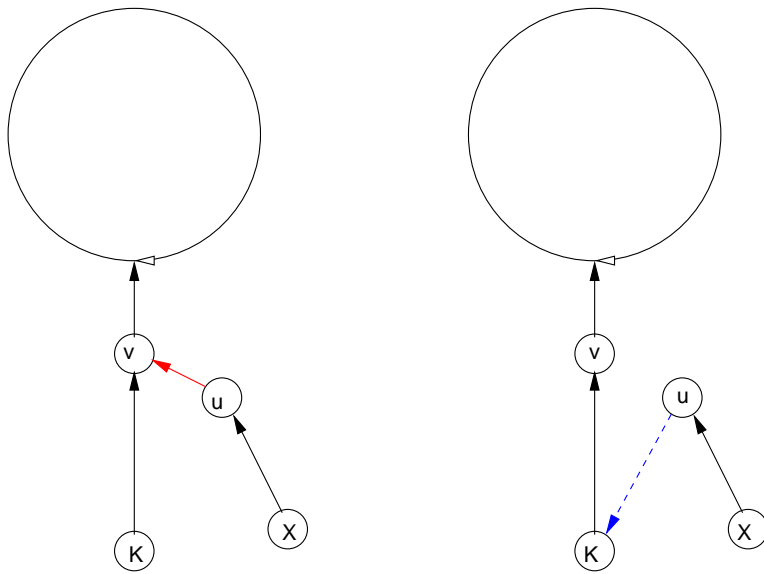
$F^0(x) = x, F^k(x) = F(F^{k-1}(x))$ ha $k > 0$. Azt mondjuk, hogy x és y ugyanazon komponenshez tartozik, ha van olyan $q \geq 0$ és $r \geq 0$, hogy $F^q(x) = F^r(y)$. Egy x pont **befoka** azon y pontok száma, amelyre $F(y) = x$.

A megoldás biztosan nagyobb, vagy egyenlő, mint a 0-befokú pontok száma + a fark nélküli körök száma. Megmutatjuk, hogy ennyi módosítással egyetlen körré alakítható a függvény gráfja. A 3. 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.

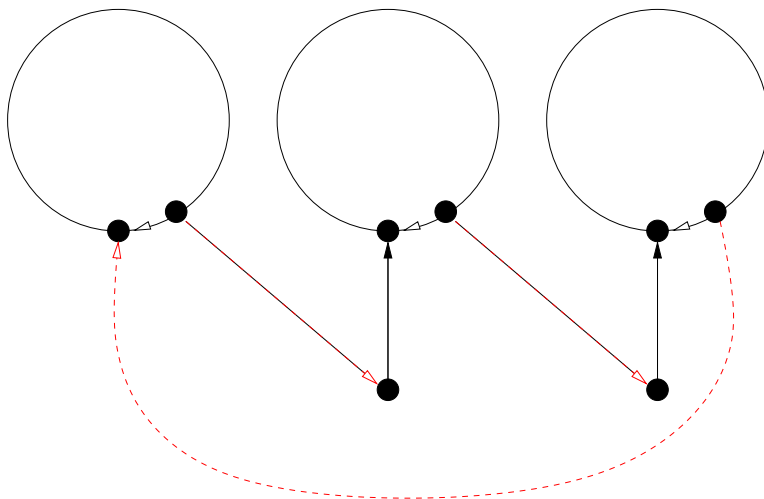
Program RiadoLanc; Const



2. ábra. Egy függvény gráfja.



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



4. ábra. A körök és egyetlen 0-befokú pontot tartalmazó komponensek összekapcsolása egyetlen körré.

```

MaxN=255;
Type
  Szinek=(Feher, Fekete, Piros);
Var
  F:Array[1..MaxN] Of Word;
  BeFok:Array[1..MaxN] Of Word;
  Szin:Array[1..MaxN] Of Szinek;
  N,Megoldas,Be0,Korszam,x,xx: Word;
  BeF,KiFf:Text;

Procedure Beolvas; {Beolvasás, az F függvény, és a BeFok előállítás.} {Global: N, F,
BeFok}
  Var i,j,k:Word;
  NevSor: Array[1..MaxN] Of String[50];
  Nev: String[20];
Begin{Beolvas}
  Assign(BeF,'input.txt');
  Assign(KiFf,'output.txt');
  Reset(BeF); Rewrite(KiFf);
  Readln(BeF,N);
  For i:= 1 To N Do BeFok[i]:=0;
  For i:= 1 To N Do ReadLn(BeF,NevSor[i]);
  For i:= 1To N Do Begin
    k:=Pos('>',NevSor[i]);
    Nev:=Copy(NevSor[i],k+1,Length(NevSor[i])-k);
    j:=1;
    While Nev<>Copy(NevSor[j],1,Length(Nev)) Do Inc(j);
    F[i]:=j;
    Inc(BeFok[j]);
  End{for i};
  Close(BeF);
End{Beolvas};

Begin{Riadolanc}
  Beolvas;
  Be0:=0; {0-befokúak száma}
  For x:=1 To N Do Szin[x]:=Feher;      {inicializálás}
  For x:=1 To N Do
    If (Szin[x]=Feher) And (BeFok[x]=0) Then Begin
      Inc(Be0);
      xx:=x;
      Repeat          {a nem körbeli pontok Feketére színezése}
        Szin[xx]:=Fekete;
        xx:=F[xx];
      Until Szin[xx]=Fekete;
    End{if};
  {Minden farok nélküli körbeli pont színe Fekete}

  Korszam:=0;
  For x:=1 To N Do
    If Szin[x]=Feher Then Begin {x körbeli pont}
      Inc(Korszam);
      xx:=x;
      Repeat
        Szin[xx]:=Piros;
        xx:=F[xx];

```

```

    Until Szin[xx]=Piros;
End{if};

If (Be0=0) And (Korszam=1) Then
    Megoldas:=0
Else
    Megoldas:=Korszam+Be0;

Writeln(KiFf,Megoldas);
Close(KiFf);

```

End.

3. Mohó módszerrel megoldható feladatok

Feladat: Fénykép probléma

Egy rendezvényre n vendéget hívtak meg. Minden vendég előre jelezte, hogy mettől meddig lesz jelen. A szervezők fényképeken akarják megörökíteni a rendezvényen résztvevőket. Azt tervezik, hogy kiválasztanak k időpontot és minden kiválasztott időpontban az akkor éppen jelenlevőkről csoportképet készítenek. Az a céljuk, hogy a lehető legkevesebb képet kelljen készíteni, de mindenki rajta legyen legalább egy képen.

Írjunk olyan programot, amely kiszámítja, hogy legkevesebb hány fényképet kell készíteni, és megadja azokat az időpontokat is amikor csoportképet kell készíteni!

Bemeneti specifikáció

A `fenykep.be` szöveges állomány első sorában a vendégek n száma van ($1 \leq n \leq 3000$). A következő n sor mindegyike két egész számot tartalmaz egy szóközzel elválasztva, egy vendég e érkezési és t távozási időpontját ($1 \leq e < t \leq 1000$). Ha egy fényképet az x időpontban készítik és $e \leq x < t$, akkor azon a fényképen rajta lesz az e időben érkező és t időben távozó vendég. A `fenykep.ki` szöveges állomány első sorába a készítendő fényképek k számát kell írni! A második sor pontosan k egész számot tartalmazzon egy-egy szóközzel elválasztva, azon időpontokat (tetszőleges sorrendben), amikor a csoportképeket készíteni kell.

Példa bemenet és kimenet

fenykep.ki

```

6
2 4
1 4
2 7
7 13
5 10
3 9

```

Megoldás.

Tehát a bemenet intervallumoknak egy

$$I = \{[e_1, t_1), \dots, [e_n, t_n)\}$$

halmaza, a kimenet pedig olyan minimális elemszámú

$$M = \{f_1, \dots, f_k\}$$

halmaz, hogy minden i -re $i = 1, \dots, n$ van olyan $f \in M$, hogy $e_i \leq f < t_i$.

Vegyük észre, hogy ha két intervallum jobb-végpontja megegyezik, $t_i = t_j$, akkor amelyik bal-végpontja kisebb, $t_i < t_j$ az elhagyható, hiszen ha $f \in [e_j, t_j)$, akkor $f \in [e_i, t_i)$.

A megoldás elemzése.

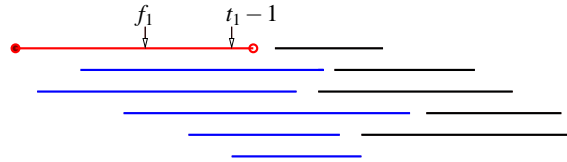
Tegyük fel, hogy az intervallumok jobb-végpontjuk szerint növekvően rendezettek, tehát $t_i < t_{i+1}$, $i = 1, \dots, n-1$ és az M megoldáshalmazra $f_1 < \dots < f_k$.

Mohó választás.

Válasszuk a megoldáshalmaz első elemének $t_1 - 1$ -et.

Megmutatjuk, hogy az optimális megoldásban f_1 helyett állhat a mohó választás, tehát $t_1 - 1$. Először is $f_1 < t_1$, mert különben az 1. intervallumba nem esne egy pontja sem az optimális megoldásnak. Továbbá, minden olyan intervallum, amelyben benne van f_1 , benne van $t_1 - 1$ is, hiszen ha $e_i \leq f_1 < t_i$.

Redukált részprobléma.



5. ábra. Mohó választás és probléma redukálás.

Töröljünk I -ből minden olyan intervallumot, amelyben benne van a $t_1 - 1$ mohó választás: $I' = I - \{[e_i, t_i) : e_i < t_1\}$. Az $M' = \{f_2, \dots, f_k\}$ ponthalmaz megoldása lesz az I' problémának. I' optimális is, mert ha lenne kevesebb pontot tartalmazó megoldása I' -nek, akkor hozzávéve $t_1 - 1$ -et, vagy f_1 -et, a kiindulási I probléma kisebb elemszámú megoldását kapnánk, mint $|M|$.

Megvalósítás

Program Fenykep; (* Bemenet : Intervallumok $\{[e_1, t_1], \dots, [e_n, t_n]\}$ halmaza.

Kimenet: Legkevesebb elemszámú olyan M halmaz, hogy minden intervallumba esik M -nek legalább egy eleme. *)

Const

MaxN=3000; { az intervallumok max. száma }
MinE=1;
MaxT=1000;

Var

N :Word; { az intervallumok száma }
Int :Array[1..MaxT] of Word; { az intervallumok: $[Int[t], i)$, ha $Int[t] > 0$ }
K:Word; { a megoldás elemszáma }
M:Array[1..MaxN] of Word; { a megoldás halmaz }
i,x:Integer;
Utolso:Integer;

Procedure Beolvas; {Global:N, Int}

Var
Bef:Text;
i,e,t:Word;
Begin
For i:=1 To MaxT Do Int[i]:=0;
Assign(Bef, 'fenykep.be'); Reset(Bef);
ReadLn(Bef, N);
For i:=1 To N Do Begin
ReadLn(Bef, e, t);
If e > Int[t] Then Int[t]:=e;
End;
Close(Bef);
End{Beolvas};

Procedure KiIr; {Global: K, M }

Var
Kif:Text;

```

    i:Word;
Begin
    Assign(KiF,' fenykep.ki'); Rewrite(KiF);
    WriteLn(Kif,K);

    For i:=1 To K Do
        Write(Kif,M[i],' ');

    WriteLn(Kif);

    Close(KiF);
End{KiIr};

Begin{Program}
    Beolvas;

    Utolso:=0;{az utolso bevalasztott pont}
    K:=0;      {a bevalasztott pontok szama}
    For x:=1 To MaxT Do
        If (Int[x]>0)And(Utolso<Int[x]) Then Begin
            Utolso:=x-1;
            Inc(K);
            M[K]:=Utolso;
        End;
    {for i};

    KiIr;
End.

```

Feladat: Darabolás

Adott egy fémrúd, amelyet megadott számú és hosszúságú darabokra kell felválni. A darabok hosszát milliméterben kifejezett értékek adják meg. Olyan vágógéppel kell a feladatot megoldani, amely egyszerre csak egy vágást tud végezni. A vágások tetszőleges sorrendben elvégezhetőek. Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen (két darabra) vágunk. A célunk optimalizálni a művelet sor teljes költségét.

Készítsünk programot amely, kiszámítja a vágási művelet sor optimális összköltségét és megad egy olyan vágási sorrendet, amely optimális költséget eredményez.

Bemeneti specifikáció

A `darabol.be.be` szöveges állomány első sora egy egész számot tartalmaz, a darabok n számát ($0 < n \leq 1000$). A második sor n darab pozitív egész számot tartalmaz egy-egy szóközzel elválasztva, a darabok hosszát. A második sorban szereplő számok nem nagyobbak, mint 1000. A `darabol.ki.be` szöveges állomány első sorába egyetlen számot, a vágási művelet sor optimális összköltségét kell írni! A további $n - 1$ sor mindegyikébe két egész számot kell írni, egy szóközzel elválasztva. Az első szám legyen az adott lépésben kettévágott rúd hossza, a második szám pedig az egyik keletkező darab hossza. Minden sor csak olyan hosszúságú darab kettévágását tartalmazhatja, amelyből a korábbi lépések során több keletkezett, mint az azóta elvégzett lépések által felhasználtak száma.

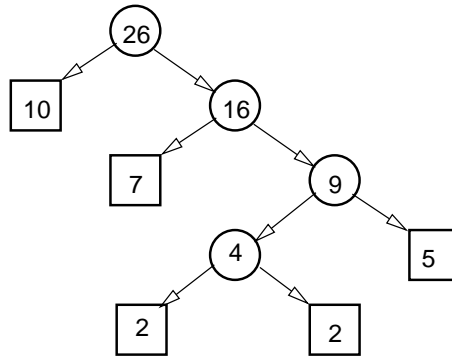
Példa bemenet és kimenet

darabol.be

```
5
2 5 2 7 10
```

darabol.ki

```
55
26 10
16 7
9 4
4 2
```



6. ábra. A példa megoldásának ábrázolása bináris fával.

Elemezzük az optimális megoldás szerkezetét. Vegyük észre, hogy minden darabolás, így az optimális is leírható egy bináris fával. A fa levelei tartalmazzák a bemenetként kapott darabok hosszait, és minden belső pontja annak a darabnak a hosszát, amelyből vágással a két fiú-pontban lévő darab keletkezett, azaz a két fiának az összegét. Példánk esetén a fa a 6. ábrán látható.

A darabolás összköltsége is kifejezhető a fával, nevezetesen, az összköltség éppen a fa belső (nem levél) pontjaiban található számok összege. Fordítva is igaz, minden ilyen fa egy darabolást ír le. A fa költségén a fa belső pontjaiban lévő számok összegét értjük. Tehát keressük az optimális megoldást, mint egy darabolási fát, tehát azt, amelynek a költsége minimális. A darabolási fa költsége kifejezhető a következőképpen. Legyenek d_1, \dots, d_n a vágandó darabok hosszai és legyen m_i a d_i darabot tartalmazó levélpont mélysége (a fa gyökerétől vett távolsága) a fában. Ezekkel a jelölésekkel a fa költsége:

$$\sum_{i=1}^n m_i * d_i$$

Az optimális fára a következő két állítás teljesül.

3.1. lemma. *A két legkisebb értéket tartalmazó levélpont mélysége a legnagyobb, és testvérek.*

Bizonyítás. *Ha az állítás nem teljesülne, akkor a két legmélyebb testvér levélpontot felcserélve a két legkisebb értéket tartalmazó levéllel, kisebb költségű fát kapnánk.* ■

3.2. lemma. *Legyen d_u és d_v a két legkisebb darab. Ha az optimális fában töröljük a d_u -t és d_v -t tartalmazó levélpontot, akkor olyan fát kapunk, amely optimális arra a bemenetre, amely d_u és d_v helyett a $d_u + d_v$ darabot tartalmazza.*

Bizonyítás. *A két levél törlésével kapott fa nyilván darabolási fa lesz a módosított bemenetre, amelynek költsége*

$$\sum_{i=1}^n m_i * d_i - (d_u + d_v)$$

Legyen F egy optimális darabolási fa a módosított bemenetre és legyen a költsége K . Ha a $d_u + d_v$ darabot tartalmazó levélponthoz hozzávesszük bal fiúként a d_u értéket tartalmazó, jobb fiúként pedig a d_v értéket tartalmazó új levelet, akkor egy olyan fát kapunk, amely darabolási fa lesz az eredeti bemenetre, költsége pedig $K + d_u + d_v$. Ez azonban nem lehet kisebb, mint az optimális darabolási fa költsége az eredeti bemenetre, tehát

$$\sum_{i=1}^n m_i * d_i \leq K + (d_u + d_v)$$

$$\sum_{i=1}^n m_i * d_i - (d_u + d_v) \leq K \leq \sum_{i=1}^n m_i * d_i - (d_u + d_v)$$

Ami az állítás bizonyítását jelenti. ■

Most már megfogalmazhatjuk a mohó stratégiánkat. Építsük fel a darabolási fát úgy, hogy lépésenként a két legkisebb értéket tartalmazó pontot egy új pont két fiává tesszük, és az új pontba a két fiúban lévő érték összegét írjuk. Az 1. Állítás igazolja a mohó választási tulajdonságot, a 2. Állítás pedig az optimális részproblémák tulajdonságot, tehát korrekt algoritmust kapunk.

Megvalósítás. A mohó választás megvalósítására prioritási sort alkalmazunk.

Procedure Darabol (Const D:Darabok;

```

                N:Word;
                Var F :Fa;
                Var Kolts:Word);
Var x,y,z,i:Word; Begin{Darabol}
  For i:=1 To N Do Begin                                {inicializálás}
    SorBa(i);
    Fa[i].bal:=0;Fa[i].jobb:=0;
  End{for i};
  For i:=1 To N-1 Do Begin{}
    x:=SorBol;          {kivesszük a prisoritási sorból}
    y:=SorBol;          {a két legkisebb elemet}
    z:=i+N;             {új pont létesítése}
    D[z]:=D[x]+D[y];    {az új pont értéke}
    SorBa(z);          {berakjuk a sorba az új fa-pontot}
    Fa[z].bal:=x;       {az új pont két fia x és y}
    Fa[z].jobb:=y;
    Kolts:=Kolts+D[z];
  End{for i};
End{Darabol};

{A feladatban megkövetelt kimenetet a fa bejárásával állítjuk elő.}
Procedure KiIr;
Var  KiF:Text;

Procedure Bejar(p:Word);
Begin{Bejar}
  If F[p].bal=0 Then Exit;
  WriteLn(KiF, D[p]:1, ' ',D[F[p].bal]:1);
  If F[p].bal<>0 Then Bejar(F[p].bal);
  If F[p].jobb<>0 Then Bejar(F[p].jobb);
End{Bejar};

Begin{KiIr}
  Assign(KiF, 'darabol.ki'); Rewrite(KiF);
  WriteLn(KiF,Kolts);
  Bejar(2*N-1);
  Close(KiF);
End{KiIr};

```

4. Dinamikus programozás

A dinamikus programozás stratégiája.

A dinamikus programozás, mint probléma-megoldási stratégia az alábbi öt lépés végrehajtását jelenti.

1. Az [optimális] megoldás szerkezetének elemzése.
2. Részproblémákra és összetevőkre bontás úgy, hogy:
 - a) Az összetevőktől való függés körmentes legyen.
 - b) Minden részprobléma [optimális] megoldása kifejezhető legyen (rekurzívan) az összetevők [optimális] megoldásaival.
3. Részproblémák [optimális] megoldásának kifejezése (rekurzívan) az összetevők [optimális] megoldásaiból.
4. Részproblémák [optimális] megoldásának kiszámítása alulról-felfelé haladva:
 - a) A részproblémák kiszámítási sorrendjének meghatározása. Olyan sorba kell rakni a részproblémákat, hogy minden p részprobléma minden összetevője (ha van) előbb szerepeljen a felsorolásban, mint p .

- b) A részproblémák kiszámítása alulról-felfelé haladva, azaz táblázat-kitöltéssel.

5. Egy [optimális] megoldás előállítás a 4. lépésben kiszámított (és tárolt) információkból.

Feladat: Tükörszó (IOI'2000)

Egy karaktersorozat tükörszónak nevezünk, ha balról-jobbra, valamint jobbról-balra olvasva megegyezik. Másképpen fogalmazva, egy S szó akkor és csak akkor tükörszó, ha vagy üres szó, vagy egybetűs, vagy az első és utolsó betűje megegyezik és ezeket elhagyva ismét tükörszót kapunk.

Írjunk olyan programot, amely kiszámítja, hogy egy adott szóból minimálisan hány betűt kell törölni, hogy tükörszót kapjunk.

Bemeneti specifikáció

A `tukorszo.be` szöveges állomány első és egyetlen sora egy S szót tartalmaz, amelynek hossza legfeljebb 5000, és S minden c karakterére: ' $a' \leq c \leq 'z'$ ' és ' $A' \leq c \leq 'Z'$ '. A `tukorszo.ki` szöveges állomány első és egyetlen sora egy m nemnegatív egész számot tartalmaz, ami a minimális törlendő karakterek száma, amellyel a bemeneti S szó tükörszová tehető.

Példa bemenet és kimenet

`tukorszo.be`

`tukorszo.ki`

eleme
Megoldás

2

Az optimális megoldás szerkezetének elemzése.

Minden S szóra jelölje $T(S)$ a probléma egy megoldását, tehát olyan tükörszót, amely S -ből a lehető legkevesebb betű törlésével kapható. Ilyen biztosan létezik, hiszen egy kivételével minden betűt törölve tükörszót kapunk. Ha S egy betűből áll, akkor maga is tükörszó, $T(S) = S$. Legyen $S = xRy$, ahol x az első, y pedig az utolsó betűje S -nek (R lehet üres szó is). Ha $x = y$, akkor $T(S) = T(R)$. Ha $x \neq y$, akkor vagy az x , vagy az y betűt biztosan törölni kell, tehát a megoldás vagy $T(xS)$, vagy $T(Sy)$. Az optimális megoldás szerkezete azt sugallja, hogy minden (i, j) , $1 \leq i \leq j \leq n$ indexpárra tekintsük azt a részproblémát, hogy az $S[i..j] = S[i]...S[j]$ szó legkevesebb hány betű törlésével tehető tükörszová. Jelölje az (i, j) részprobléma megoldását $M(i, j)$. Tehát a kitűzött feladat megoldása $M(1, n)$.

A részproblémák megoldásának kifejezése az összetevők megoldásaival.

Ha $i > j$ esetre $M(i, j)$ értékét 0-ként értelmezzük, akkor a részproblémák megoldásai között az alábbi rekurzív összefüggést lehet felírni.

$$M(i, j) = \begin{cases} 0, & \text{ha } i \leq j \\ M(i+1, j-1), & \text{ha } i < j \text{ és } S[i] = S[j] \\ 1 + \min(M(i+1, j), M(i, j-1)), & \text{ha } i < j \text{ és } S[i] \neq S[j] \end{cases}$$

Tehát az (i, j) részprobléma összetevői: $(i+1, j-1)$, $(i+1, j)$ és $(i, j-1)$.

A részproblémák kiszámítási sorrendje, táblázat-kitöltés.

Tároljuk a részproblémák megoldását táblázatban, az (i, j) megoldását az $M[i, j]$ táblázatelemében. Mivel az (i, j) részprobléma megoldása legfeljebb az $(i+1, j-1)$, $(i+1, j)$ és $(i, j-1)$ megoldásaitól függ, ezért a táblázat-kitöltés sorrendje lehet alulról felfelé, soronként pedig jobbról-balra haladó. A négyzetes táblázat tárigénye $5000 * 5000 * 2 = 50000000$ byte, ami túl sok. Látható azonban, hogy elegendő lenne csak két egymást követő sort tárolni. Sőt, egy sort is elég tárolni, ha megoldjuk, hogy ne írjuk felül azt a $T[i] = M(i, j)$ kitöltésekor az $M(i, j-1)$ értéket, amit szintén $T[i]$ tárol.

```
Function Megold(Const S:Sorozat; N:Word):Word;
{Megoldás lineáris táblázat-kitöltéssel}
Var
  T:Array[1..MaxN] of Word;
  i,j:Integer;
  Ment, Menti:Word;
Begin
  T[1]:=0;
  For j:=2 To N Do Begin
    T[j]:=0; Menti:=0;
    For i:=j-1 DownTo 1 Do Begin
      {M(i,j) szamitasa es tarolasa T[i]-ben}
```

n									0
								0	
							0		
j	?	x				0			
j-1	x	x			0				
				0					
			0						
		0							
	0								
1	0								
	1	i	i+1						n

7. ábra. Táblázat-kitöltési sorrend: soronként alulról felfelé, jobbról balra haladva.

```

Ment:=T[i];
If S[i]=S[j] Then
  T[i]:=Menti {Menti=M(i+1, j-1)}
Else {M(i, j)=1+Min (M(i, j-1)+M(i+1, j))}
  T[i]:=1+Min(T[i], T[i+1]);
Menti:=Ment;
End{for i};
End{for j};
Megold:=T[1]; {=M(1, N)}
End{Szamit};

```

Az algoritmus futási ideje $\Theta(n^2)$, tárigénye $\Theta(n)$.

Ha egy megoldást is elő kell állítani, akkor minden (i, j) részproblémára tarolni kell azt az információt, hogy melyik összetevőre kapjuk az optimális megoldást ha $S[i] \neq S[j]$. Vagy minden minden (i, j) részproblémára taroljuk $M(i, j)$ értékét, és ekkor $M(i+1, j) < M(i, j-1)$ összehasonlítással megadható, hogy az i -edik (első), avagy a j -edik (utolsó) betűt kell-e törölni, vagy egy külön L tömbben tároljuk, hogy a részsorozat melyik végéről kell törölni az optimális megoldáshoz. Ezt nevezzük az optimális megoldás visszafejtésének.

Ekkor algoritmus futási ideje is és tárigényre is $\Theta(n^2)$.

```

Procedure KiIr; {Global: T, S, N} Var
  Bal, Jobb:Word;
Begin
  Bal:=1; Jobb:=N;
  While Bal<Jobb Do Begin
    If S[Bal]=S[Jobb] Then Begin
      Inc(Bal); Dec(Jobb)
    End Else If T[Bal+1, Jobb] < T[Bal, Jobb-1] Then Begin
      Write(KiF, Bal, ' ');
      Inc(Bal);
    End Else Begin
      Write(KiF, Jobb, ' ');
      Dec(Jobb);
    End;
  End{while}
End{KiIr};

```

Feladat: Testvéries osztzkodás (CEOI'95)

Két testvér ajándékokon osztzkodik. Minden egyes ajándékot pontosan ez egyik testvérnek kell adni. Minden ajándéknak pozitív egész számmal kifejezett értéke van. Jelölje A az egyik, B pedig a másik testvér által kapott ajándékok összértékét. A cél az, hogy az osztzkodás testvéries legyen, tehát A és B különbségének abszolútértéke minimális legyen.

Írjunk programot, amely kiszámítja a testvéries osztzkodás eredményeként keletkező A és B értékeket.

Bemeneti specifikáció

A `testveri.be` szöveges állomány első sora az ajándékok n számát tartalmazza ($2 \leq n \leq 100$). A második sor pontosan n egész számot tartalmaz, egy ajándék értékét. Minden szám értéke legfeljebb 200. A `testveri.ki` szöveges állomány első és egyetlen sora két egész számot tartalmazzon, a testvéries osztzkodás eredményét.

Példa bemenet és kimenet

`testveri.be`

`testveri.ki`

```
7
28 7 11 8 9 7 27
```

```
48 49
```

Megoldás

Legyen $\{e_1, \dots, e_n\}$ az ajándékok értékeinek egy felsorolása és jelölje E az összegüket. Feltehetjük, hogy $A \leq B$. Mivel $A + B = E$, ezért A a legnagyobb olyan szám, amelyre $A \leq E/2$ és előállítható $\{e_1, \dots, e_n\}$ egy részhalmazának összegeként. **A megoldás szerkezetének elemzése.**

Jelölje Fel az $(A + B) \text{ div } 2$ értéket. Tegyük fel, hogy

$$A = e_{i_1} + \dots + e_{i_k}, \quad i_1 < \dots < i_k$$

. Ez akkor és csak akkor, ha

$$A - e_{i_k} = e_{i_1} + \dots + e_{i_{k-1}}$$

azaz $A - e_{i_k}$ előállítható legfeljebb a első $i_k - 1$ (e_1, \dots, e_{i_k-1}) szám v.m. részhalmazának összegeként.

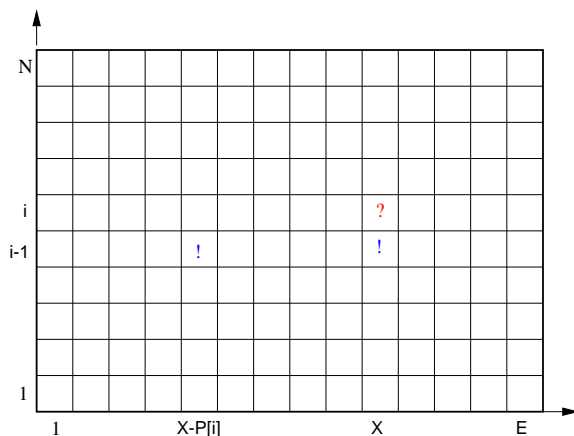
Részproblémákra bontás.

Bontsuk részproblémákra a kiindulási problémát: Minden $(X, i) (1 \leq X \leq Fel, 1 \leq i \leq n)$ számpárra vegyük azt a részproblémát, hogy az X érték előállítható-e legfeljebb az első e_1, \dots, e_i szám valamely részhalmazának összegeként. Jelölje $V(X, i)$ az (X, i) részprobléma megoldását, ami logikai érték; $V(X, i) = Igaz$, ha az X előállítható, egyébként $Hamis$.

Összefüggések a részproblémák és megoldásaik között.

Nyilvánvaló, hogy az alábbi összefüggések teljesülnek a részproblémák megoldásaira:

$$V(X, i) = \begin{cases} X = e_i, & \text{ha } i = 1 \\ V(X, i-1) \vee (X > e_i) \wedge V(X - e_i, i-1), & \text{ha } i > 1 \end{cases}$$



8. ábra. Táblázat-kitöltés

```

Program Testveri; Const
  MaxN=100; {az ajándékok max. száma }
  MaxFel=20000;{ Max. összeg }
Var
  N,i,Osszeg,A : Word;
  E:Array[1..MaxFel] Of Word; { az ajándékok értéki }
  BeF, KiF: Text;

Function F(M : Word) : Word;
{ F(M) a legnagyobb olyan szám, amely <=M és előáll az E[1..N] számok
összegeként }
Var
  V : Array[0..MaxFel] Of Boolean;
  x,i: Word;
Begin
  For x:=1 To M Do V[x]:= False; {inicializálás}
  V[0]:=True;
  For i:=1 To N Do Begin
    { V[x]=True <=> x előáll az első i szám összegeként }
    For x:=M Downto E[i] Do
      V[x]:= V[x] Or V[x-E[i]];
    End{i};
    For x:=M Downto 1 Do
      If V[x] Then Begin
        F:=x;
        Exit
      End;
    End (* F *);

Begin { Program }
  Assign(BeF,'testveri.be');
  Assign(KiF,'testveri.ki');
  Reset(BeF); Rewrite(KiF);
  ReadLn(BeF,N);
  Osszeg:=0;
  For i:=1 To N Do Begin
    Read(BeF, E[i]);
    Osszeg:=Osszeg+E[i];
  End;
  Close(BeF);

  A:= F(Osszeg Div 2);
  WriteLn(KiF, A, ' ',Osszeg-A);
  Close(KiF);
End.

```

Ha azt is meg kell adni, hogy az egyes testvérek mely ajándékokat kapják egy testvéries osztzkodást során, akkor nem elég lineáris tömb, minden (x,i) -re tárolni kell a $V(x,i)$ értékeket, hogy elő tudjunk állítani egy osztzkodást.

```

Procedure Osztzkodas(Const E:Ertekek; N: Word;
                    Var Db:Word; Var C :Megoldas);
Const
  MaxN=100; {az ajándékok max. száma }
  MaxFel=300;{ Max. összeg }
Var
  N,i,Osszeg,A, F: Word;
  E:Array[1..MaxN] Of Word; { az ajándékok értéki }

```

```

V:Array[0..MaxFel, 0..MaxN] Of Boolean;
i,X:Integer;
Begin{Osztozkodas}
  Fel:=0;
  For i:=1 To N Do Fel:=Fel+E[i];
  Fel:=Fel Div 2;
  For x:=1 To Fel Do
    V[X,1]:=False;      {az első sor, azaz V(X,1) számítása}
  V[E[1],1]:= E[1]<=Fel;
  For i:=2 To N Do      {az i-edik sor,azaz V(X,i) számítása}
    For X:=1 To E Do
      V[X,i]:=(E[i]=X) Or V[X,i-1] Or
        (X>E[i]) And V[X-E[i],i-1];

  For x:=Osszeg Div 2 Downto 1 Do
    If V[x,N] Then Begin
      F:=x;
      Break
    End;

  Db:=0; X:=F; i:=N      { egy megoldás előállítás}
  Repeat
    While (i>0) And V[X,i] Do Dec(i);
    Inc(Db); C[Db]:=i+1;   {i+1 bejegyzése a megoldásba}
    X:=X-E[i+1];          {X-E[i+1] felváltásával folytatjuk}
  Until X=0;
End{Osztozkodas};

```

Feladat: Vágás

Adott egy fémrúd, amelyet megadott számú darabra kell felvágni úgy, hogy a vágások pontos helyét is tudjuk. A vágások helyét a rúd egyik végétől mért, milliméterben kifejezett értékek adják meg. Olyan vágógéppel kell a feladatot megoldani, amely egyszerre csak egy vágást tud végezni. A vágások tetszőleges sorrendben elvégezhetőek. Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen (két darabra) vágunk. A célunk optimalizálni a művelet sor teljes költségét.

Írjunk olyan programot, amely kiszámítja a vágási művelet sor optimális összköltségét, és megad egy olyan vágási sorrendet, amely optimális költséget eredményez.

Bemeneti specifikáció

Avag.be szöveges állomány első sora egyetlen egész számot tartalmaz, a vágandó rúd h hosszát ($0 < h \leq 1000$). A második sorban az elvégzendő vágások n száma van ($1 \leq n \leq 1000$). A harmadik sor n darab egész számot tartalmaz egy-egy szóközzel elválasztva, az elvégzendő vágások helyeit. A számok szigorúan monoton növekvő sorozatot alkotnak, és mindegyik nagyobb, mint 0 és kisebb, mint h . Avag.ki szöveges állomány első sorába egyetlen számot, a vágási művelet sor optimális összköltségét kell írni! A második sor n darab egész számot tartalmazzon, ami a vágási helyek sorszámainak egy olyan felsorolása legyen, hogy ebben a sorrendben elvégezve a vágásokat, az összköltség optimális lesz.

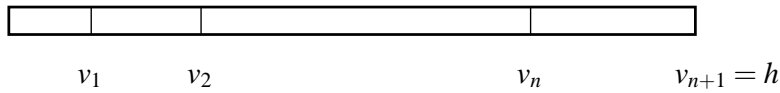
Példa bemenet és kimenet

vag.be	vag.ki
10	22
4 4 5 7 8	1 3 2 4

Megoldás. Az optimális megoldás szerkezetének vizsgálata.

Legyenek a vágások helyei, $n + 1$ -edig elemként kiegészítve h -val:

v_1, v_2, \dots, v_n, h



9. ábra. A vágási helyek, $v_{n+1} = h$

Ha az optimális vágás során először a v_k helyen történik a vágás, akkor az első darabon a $v_1, v_2, \dots, v_{k-1}, v_k$, a másodikon pedig v_{k+1}, \dots, v_n, h vágásoknak is optimálisnak kell lenni.

Az optimális megoldás rekurzív kifejezése.

Legyen $Opt(i, j)$ a v_i , vágási helytől a v_j , vágási hely által meghatározott rúddarab optimális vágásának költsége.

$$Opt(i, j) = \begin{cases} 0, & \text{ha } j = i + 1 \\ v_j - v_i + \min_{k=i+1}^{j-1} (Opt(i, k) + Opt(k, j)) & \text{ha } i < j + 1 \end{cases}$$

Legyen $S(i, j)$ az a k , amelyre a minimum adódik.

n							0	
j			?	x	x	0		
			x		0			
			x	0				
			0					
		0						
	0							
	0							
1								
	1		i				n	

10. ábra. Táblázat-kitöltési sorrend.

```

Program Vag; Const
  MaxN=100;
  Inf=200000000;
Var
  H:1..MaxM;
  N:Byte; {a vagasok szama}
  V:Array[0..MaxN+1] Of 0..MaxM;
  Kolts:Longint;
  S:Array[0..MaxN+1,0..Maxn+1] of 0..MaxN;

Procedure Beolvas; Var
  BeF:Text;
  i,x,y:Byte;
Begin{Beolvas}
  Assign(BeF, 'vag.be'); Reset(BeF);
  ReadLn(BeF, H);
  ReadLn(BeF, N);
  For i:=1 To N Do

```

```

    Read(BeF, V[i]);
    Close(BeF);
    V[0]:=0; V[N+1]:=H;
End{Beolvas};

Procedure KiIr; Var  KiF:Text;
    Sor:Array[1..MaxN] of 0..MaxN;
    Ind,i,k:Byte;
Procedure Bejar(i,j:Byte); Var k:Byte; Begin
    If j<=i+1 Then Exit;
    Inc(Ind);
    k:=S[i,j];
    Sor[Ind]:=k;
    Bejar(i,k);
    Bejar(k,j);
End{Bejar};

Begin{KiIr}
    Assign(KiF, 'vag.ki'); Rewrite(KiF);
    WriteLn(KiF,Kolts);
    Ind:=0;
    Bejar(0,N+1);
    For i:=1 To N Do Begin
        k:=Sor[i];
        Write(KiF, V[k]:1,' ');
    End;
    WriteLn(KiF);
    Close(KiF);
End{KiIr};

Procedure Szamit; {Global: N, V, S} Var
    Opt:Array[0..MaxN+1,0..MaxN+1] of Longint;
    i,j,k,u,G:Byte;
    Min,Uj:Longint;
Begin{Szamit}
    For i:=0 To N Do Begin{inicializalas}
        Opt[i,i+1]:=0; S[i,i+1]:=0;
    End{for i};
    For u:=2 To N+1 Do Begin{j-i=u}
        For i:=0 To N-u+1 Do Begin
            j:=i+u; Min:=Inf;
            For k:=i+1 To j-1 Do Begin
                Uj:=Opt[i,k]+Opt[k,j];
                If Uj <Min Then Begin
                    Min:=Uj;
                    G:=k;
                End;
            End{k};
            Opt[i,j]:=Min+V[j]-V[i];
            S[i,j]:=G;
        End{i};
    End{for u};
    Kolts:=Opt[0,N+1];
End{Szamit};

Begin{program}

```

```
Beolvas;  
Szamit;  
KiIr;  
End.
```

5. Gráf-algoritmusok

5.1. Feladat: Tej kimérés

Egy gazdának négy tejeskannája van, A, B, C és D. Az A, B és C kannák úrtartalmát ismeri, a D-ét nem, csak azt tudja, hogy ez a legnagyobb kannája. Kezdetben az A kanna tele van, a többi pedig üres. A gazda egyik kannából másik kannába áttöltéseket végezve azt szeretné elérni, hogy a D kannában adott mennyiségű tej legyen. Ezért csak olyan áttöltést végezhet, hogy mindig tudnia kell, hogy az egyes kannákban mennyi tej van. Tehát ha az X kannából az Y kannába tölt át, akkor vagy mindet át kell töltenie az Y-ba, ha belefér, vagy annyit kell áttöltenie Y-ba, hogy az Y kanna tele legyen. Tehát a kimérés során végrehajtott lépés egyértelműen meghatározott azzal, hogy melyik kannából melyik kannába tölt át. A lehetséges lépések: AB, AC, AD, BA, BC, BD, CA, CB, CD, DA, DB, DC. Az XY betűpárt azt jelenti, hogy az X jelű kannából az Y jelű kannába tölt át.

Írjunk olyan programot, amely kiszámít egy legkevesebb lépésből álló öntögetési sorozatot, amelynek eredményeként a D kannában a kívánt mennyiségű tej lesz!

Bemeneti specifikáció

A `kimer.be` szöveges állomány első sora három egész számot tartalmaz, a három ismert úrtartalmú kanna a , b és c úrtartalmát, ($100 \geq a > b > c \geq 1$). A második sor a D kannában előállítandó L mennyiséget tartalmazza ($1 \leq L \leq a$). A `kimer.ki` szöveges állomány első sora a megoldás lépéseinek k számát tartalmazza. Ha nincs megoldás, akkor ez a szám a -1 legyen. A további k sor tartalmazza azt az öntögetési lépéssort (soronként egy-egy lépés), amelyet végrehajtva a D kannában L liter tej lesz.

Példa bemenet és kimenet

kimer.be

```
13 11 5  
3
```

kimer.ki

```
4  
AC  
CB  
AC  
AD
```

Megoldás. A probléma gráf modellje: Tekintsük azt a $G = (V, E)$ irányított gráfot, amelynek pontjai az öntögetés során lehetséges kanna tartalmak: $V = \{(x, y, z, w) : 0 \leq x, y, z, w \leq a\}$. A negyedik komponens (w) felesleges, mert $w = a - (x + y + z)$, mivel az összes tejmennyiség nem változik.

A gráf élei: $(x, y, z) \rightarrow (\bar{x}, \bar{y}, \bar{z})$, ha egy öntés eredményeként (x, y, z) -ből $(\bar{x}, \bar{y}, \bar{z})$ keletkezik.

Tehát a feladat megoldása: ebben gráfban legrövidebb utat keresünk az $(a, 0, 0)$ pontból egy olyan (x, y, z) pontba, amelyre teljesül, hogy $L = a - (x + y + z)$.

A gráf E élhalmazát nem tároljuk (nincs is annyi memóriánk), mert minden (x, y, z) hármásra kiszámítható az a legfeljebb 12 $(\bar{x}, \bar{y}, \bar{z})$ hármás, amely egy-egy öntés eredményeként keletkezik (számított gráf).

5.2. Feladat: Bűvös négyzetek (IOI'96).

A bűvös kocka sikere után Rubik úr elkészítette a síkbeli változatot, a bűvös négyzeteket. Ez egy sík lap, amely nyolc azonos méretű négyzetből áll. (lásd az 1. ábrát). Ebben a feladatban azzal a változattal foglalkozunk, amelyben minden mező különböző színű. A színeket az első nyolc pozitív egész számmal jelöljük. A lap egy állapota a színek sorrendjének leírásával adható meg, a bal felső sarokból indulva és az óramutató járásával azonos irányba haladva. Például az 1. ábrán látható állapot leírása a $\langle 1, 2, 3, 4, 5, 6, 7, 8 \rangle$ sorozat. Ez a kezdőállapot. A lappal háromféle művelet végezhető, ezeket az 'A', 'B' és 'C' betűkkel jelöljük.

- 'A' Kicszeréli az alsó és a felső sort.
- 'B' A téglalap elemeinek körkörös jobbra léptetése (egy mezővel).
- 'C' A középső négy mező körkörös léptetése az óramutató járásával megegyező irányban (egy mezővel).

1	2	3	4
8	7	6	5

11. ábra. Kezdeti állás

A	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>8</td><td>7</td><td>6</td><td>5</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>8</td><td>7</td><td>6</td><td>5</td></tr></table>	1	2	3	4	8	7	6	5	1	2	3	4	8	7	6	5
1	2	3	4														
8	7	6	5														
1	2	3	4														
8	7	6	5														

B	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td>1</td><td>2</td><td>3</td></tr><tr><td>5</td><td>8</td><td>7</td><td>6</td></tr><tr><td>8</td><td>7</td><td>6</td><td>5</td></tr></table>	1	2	3	4	4	1	2	3	5	8	7	6	8	7	6	5
1	2	3	4														
4	1	2	3														
5	8	7	6														
8	7	6	5														

C	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>7</td><td>2</td><td>4</td></tr><tr><td>8</td><td>6</td><td>3</td><td>5</td></tr><tr><td>8</td><td>7</td><td>6</td><td>5</td></tr></table>	1	2	3	4	1	7	2	4	8	6	3	5	8	7	6	5
1	2	3	4														
1	7	2	4														
8	6	3	5														
8	7	6	5														

12. ábra. A három elemi lépés

A fenti három művelettel minden állapot elérhető.

Írj programot, amely előállítja a műveletek egy olyan sorozatát, amely az első ábrán látható kezdőállapotból előállít egy adott célállapotot. (A részfeladat - Subtask A). Egy tesztdatára további két pont jár, ha a program által adott megoldás 300 lépésnél nem hosszabb (B részfeladat - Subtask B).

Bemeneti specifikáció

Az `magic.in` fájl első sora 8 egész számot tartalmaz, a célállapot leírását. Az `magic.out` fájl első sorába programodnak a műveletsorozat L hosszát kell írnia. A következő L sornak a megoldásban szereplő műveletek azonosítóit kell tartalmaznia, minden sor első pozícióján egy-egy nagybetűt.

Példa bemenet és kimenet

`magic.be`

2 6 8 4 5 7 3 1

`magic.ki`

7
B
C
A
B
C
C
B

Megoldás

Minden lap egyértelműen leírható egy $\langle p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, \rangle$ sorozattal ($1 \leq p_i \leq 8$ és $p_i \neq p_j$ ha $i \neq j$), ami azt jelenti, hogy a lapon az i -edik helyen a p_i színű négyzet van. Jelölje \mathbb{P}_8 az $\{1, \dots, 8\}$ halmaz összes permutációjának halmazát. Mindegyik $L \in \{A, B, C\}$ elemi lépés egy $L : \mathbb{P}_8 \rightarrow \mathbb{P}_8$ függvényt jelent, tehát adott $p \in \mathbb{P}_8$ permutációra $L(p)$ az a lap, amelyet úgy kapunk, hogy a p lapra végrehajtjuk az L lépést.

A feladat megoldható gráf modellben, a kezdő állapotból a célállapotba vezető legrövidebb út keresésével. Tekintsük azt a gráfot, amelynek pontjai a lehetséges játékallok, azaz \mathbb{P}_8 . A gráfban akkor és csak akkor legyen él a p és q között, ha p -ből egy elemi lépés végrehajtásával kapható q . Ha $L(p) = q$, akkor legyen a $p \rightarrow q$ él címkéje L .

Minden elemi lépések megadható egy $\langle p_1, \dots, p_8 \rangle$ sorozattal, ami azt jelenti, hogy az i -edik helyen álló négyzet a lépés hatására a p_i -edik helyre kerül. Tehát a három elemi lépés:

$A = \langle 8, 7, 6, 5, 4, 3, 2, 1 \rangle$

$B = \langle 4, 1, 2, 3, 6, 7, 8, 5 \rangle$

$C = \langle 1, 7, 2, 4, 5, 3, 6, 8 \rangle$

Egy legrövidebb utat szélességi kereséssel oldjuk határozunk meg. Ehhez szükség van arra, hogy minden $p \in \mathbb{P}_8$ permutációra tároljuk, hogy a bejárás során jártunk-e már p -ben, és azt is tárolni kell, hogy melyik lépés végrehajtásával jutottunk p -be. Mivel \mathbb{P}_8 elemszáma $8! = 40320$, ezért egy

```
Lep : Array[0..40319] of Char
```

tömb elegendő a fent megkívánt információ tárolásához. A permutációkat azonosítsuk a lexikografikus (ábacé szenti) rendezés szerint sorszámukkal, jelölje adott $p \in \mathbb{P}_8$ -re $Rang(p)$ ezt a sorszámot. Pontosabban, $Rang(p)$ azon permutációk száma, amelyek megelőzik p -t a lexikografikus rendezésben. Azonban a szélességi kereséssel megtalált megoldás kiírásához azt is meg kell tudni határozni, hogy ha a p permutáció a legrövidebb úton úgy kapható, hogy utolsó lépés az L , akkor melyik a legrövidebb úton p -t megelőző q pont, tehát amelyre $L(q) = p$. Ezt külön szokás tárolni a mélységi bejárás során, azonban DOS környezetben erre már nincs elég memória. Vegyük észre, hogy erre nincs is szükség. Ugyanis minden elemi lépés, mint $L : \mathbb{P}_8 \rightarrow \mathbb{P}_8$ függvény kölcsönösen egyértelmű leképezés, tehát bármely $p, q \in \mathbb{P}_8$ -re, ha $p \neq q$, akkor $L(p) \neq L(q)$. Tehát minden L lépésnek van inverze, jelölje ezt \bar{L} , azaz $\bar{L}(L(p)) = p$ minden p -re. Meg is tudjuk adni az elemi lépések inverzét, ezek:

$$\bar{A} = \langle 8, 7, 6, 5, 4, 3, 2, 1 \rangle$$

$$\bar{B} = \langle 2, 3, 4, 1, 8, 5, 6, 7 \rangle$$

$$\bar{C} = \langle 1, 3, 6, 4, 5, 7, 2, 8 \rangle$$

```
Procedure Keres(Const T: Lap);
  { Legrövidebb út keresése szélességi bejárással }
Begin { Keres }
  For i:=0 To MaxRang Do Lep[i]:=' '; { inicializálás }
  Lep[0]:='.'; { a kezdő állapotból indulunk }
  InitSor; { a sor inicializálása }
  While True Do Begin
    Sorbol(R);
    For X:='A' To 'C' Do Begin { az elemi lépések alkalmazása R-re }
      Alkalmaz(R, X, S); { S:=X(R) }
      RangS:=Rang(S);
      If Lep[RangS]=' ' Then Begin { S új }
        Lep[RangS]:=X; { a lépés bejegyzése }
        If Egyenlo(T,S) Then Break; { megtaláltuk }
        Sorba(S);
      End;
    End { For X };
  End { While };
End { Keres };
```

```
Procedure LepsSor(Const T: Lap; Var S:String); { Global: Lep }
  Var
    RangQ:Word; X:Char;
    P,Q : Lap;
  Begin
    Q:=T;
    RangQ:=Rang(Q);
    S:='';
    While RangQ <> 0 Do Begin { amig Q<>Ini }
      X:=Lep[RangQ]; { az X lépéssel jutottunk Q-ba }
      S:=X+S; { az X karakter hozzáillesztése S elejéhez }
      Alkalmaz_1(Q,X,P); { X inverzének alkalmazása Q-ra }
      Q:=P; { visszalépés a legrövidebb úton }
      RangQ:=Rang(Q);
    End { While };
  End { LepsSor };
```

```

Begin { Program }
  Beolvas;
  Elokeszit;
  Keres(T);
  LepsSor(T, Megoldas);
  KiIr;
End.

```

Feladat: Iskolahálózat (IOI'96)

Néhány iskola számítógépes hálózatba van kötve. Az iskolák a szabadon terjeszthető programok elosztására megállapodást kötöttek: minden iskola egy listát vezet azokról az iskolákról ("címezett iskolákról"), amelyek számára a programokat továbbküldi. Megjegyzés: ha **B** iskola szerepel az **A** iskola terjesztési listáján, akkor **A** nem feltétlenül szerepel **B** terjesztési listáján.

Írj egy programot, amely meghatározza azt a legkisebb számot, ahány iskolához egy új programot el kell juttatni ahhoz, hogy - a megállapodás alapján, a hálózaton keresztül terjesztve - végül minden iskolába eljusson (A részfeladat - Subtask A).

További feladat annak biztosítása, hogy az új programot bármely iskolába eljuttatva az a terjesztési listákon keresztül az összes többi iskolába megérkezzen. Ehhez szükség lehet a terjesztési listák bővítésére. Számítsd ki, hogy minimálisan hány alkalommal kell bővítést végeznünk ahhoz, hogy utána bármely iskolát kiválasztva a program minden iskolába eljusson (B részfeladat - Subtask B)! Egy bővítés alatt azt értjük, hogy egy iskola terjesztési listájába felvesszünk egy ott nem szereplő iskolát.

Bemeneti specifikáció

Az INPUT.TXT állomány első sora egy egész számot, a hálózatba kapcsolt iskolák n számát tartalmazza ($2 \leq n \leq 100$). Az iskolákat az első n pozitív egész számmal azonosítjuk. A következő n sor mindegyike egy-egy terjesztési listát ír le. Az $i + 1$ -edik sor az i -edik iskola terjesztési listáján szereplő iskolák azonosítóit tartalmazza. Minden lista végén egy 0 szám áll. Az üres lista csak egy 0 számból áll. **A** program az eredményt, amely két sorból áll, az OUTPUT.TXT nevű fájlba kell írja. Az első sor egy pozitív egész számot: az A részfeladat eredményét, a második pedig a B részfeladatét tartalmazza.

Példa bemenet és kimenet

INPUT.TXT

```

5
2 4 3 0
4 5 0
0
0
1 0

```

OUTPUT.TXT

```

1
2

```

Az A részfeladat megoldása

Jelölje $G = (V, E)$; $V = \{1, \dots, n\}$ az iskolahálózat gráfját.

Legyen $D \subseteq V$ egy megoldáshalmaz, tehát

$$(\forall q \in V)(\exists p \in D)(p \rightsquigarrow q)$$

A D halmazt lépésenként építhetjük:

Adott $p \in V$ pontra jelölje $Eler(p)$ a p pontból elérhető pontok halmazát:

$$Eler(p) = \{q : p \rightsquigarrow q\}$$

Terjesszük ki ezt halmazokra:

$$Eler(D) = \bigcup_{p \in D} Eler(p)$$

begin

```

D := 0;
DbolEler := 0;
for p in V do

```

```

if  $p \notin \text{DbolElert}$  then begin
     $D := D - \text{Eler}(p) \cup \{p\}$ 
     $\text{DbolElert} := \text{DbolElert} \cup \text{Eler}(p)$ 
end

```

end

Az $\text{Eler}(p)$ halmaz elemei mélységi bejárással kiszámíthatók.

Az algoritmus futási ideje legrosszabb esetben $O(n^3)$.

Az algoritmus legrosszabb esete, ha a bemenetben p szomszédai: $\{1, 2, \dots, p-1\}$. Ha n 1000 is lehet, akkor ez az algoritmus biztosan nem elég gyors megoldás.

Gyorsítási ötlet: elsősor rakjuk a pontokat olyan sorrendbe, hogy ha $p \rightsquigarrow q$, de nincs $q \rightsquigarrow p$, akkor p előbb álljon a sorozatban, mint q .

Egy ilyen kívánt sorrendet egyetlen mélységi bejárással előállíthatunk:

a p pontra hívott (rekurzív) mélységi bejárásban, miután p -ből kiinduló összes élet bejártunk, rakjuk p -t a sorozat elejére. (Tehát a kívánt sorozatban a pontok elhagyási idő szerint csökkenő sorrendben lesznek.)

Az így módosított algoritmus két mélységi bejárást végez, tehát futási ideje $O(E)$

```

Procedure MelyBejar(u:Pont; Regi, Uj:Paletta);
    {Global: G, Szin, P0}
Var
    El>List;
    v:Pont;
Begin
    Szin[u]:=Uj;
    El:=G[u];
    While El<>Nil Do Begin{az u->v él vizsgálata}
        v:=El^.p;
        If Szin[v]=Regi Then Begin{v érintetlen pont }
            MelyBejar(v, Regi, Uj);
        End;
        El:=El^.csat;
    End{while u->v};
    Inc(ido);
    F[ido]:=u;
End{MelyBejar};

```

```

Procedure Szamit;
Var
    p,u:Pont;
Begin{Szamit}
    For u:=1 To N Do Begin
        Szin[u]:=Feher; {inicializálás}
    End;

    ido:=0; Mego:=0;
    For p:=1 To N Do Begin
        If Szin[p]=Feher Then Begin
            MelyBejar(P, Feher, Fekete);
        End;
    End{for};

    For u:=N DownTo 1 Do Begin
        p:=F[u];
        If Szin[p]=Fekete Then Begin
            Inc(Mego);

```

```

    D[Mego]:=p;
    MelyBejar(P, Fekete, Piros);
End;
End{for};

End{Szamit};

```

6. Geometriai feladatok

Számos olyan gyakorlati számítási probléma van, amely megoldható olyan geometriai modellben, amelyben csak egyszerű objektumok, pontok, egyenesek, szakaszok, szögek, szögtartományok, poligonok szerepelnek. Ilyen feladatokat vizsgálunk. Nem foglalkozunk olyan feladatokkal, amelyek lebegőpontos aritmetikát igényelnének.

Feladat: Pontok összekötése zárt, nem-metsző poligonná.

Adott a síkon n darab pont, amelyek nem esnek egy egyenesre. A pontok (x, y) koordinátaikkal adottak, amelyek egész számok. A pontokat a $1, \dots, n$ számokkal azonosítjuk. Kössünk össze pontpárokat egyenes szakaszokkal úgy, hogy olyan zárt poligont kapjunk, amelyben nincs metsző szakaszpár. Egy ilyen zárt, nem-metsző poligon megadható a pontok azonosítóinak egy felsorolásával: a felsorolásban egymást követő pontokat kötjük össze egyenes szakaszokkal, továbbá, az utolsót az elsővel is összekötjük.

Bemeneti specifikáció

A `poligon.be` szöveges állomány első sora a pontok n ($3 < n < 1000$) számát tartalmazza. A további n sor mindegyike két egész számot tartalmaz, egy pont x és y koordinátáit, ($-20000 \leq x, y \leq 20000$). A pontok nem esnek egy egyenesre. A `poligon.ki` szöveges állományba a bemenetre kiszámított zárt, nem-metsző poligont leíró sorozatot kell kiírni.

Példa bemenet és kimenet

<code>poligon.be</code>	<code>poligon.ki</code>
5	3 2 5 4 1
2 0	
1 4	
0 2	
3 2	
2 4	

Megoldás

Válasszuk ki a legkisebb x -koordinátájú pontot, ha több ilyen van, akkor ezek közül válasszuk a legkisebb y -koordinátájút. Ezt nevezzük (bal-alsó) sarokpontnak és jelöljük Q -val. Húzzunk (fél) egyenest a Q sarokpontból minden ponthoz.

Rendezzük az egyeneseket a Q ponton áthaladó, x -tengellyel párhuzamos egyenessel bezárt (előjeles) szög alapján.

Rendezzük a pontokat úgy, hogy a Q sarokpont legyen az első, és p_i előbb legyen mint p_j akkor és csak akkor, ha

A $\phi(Q, p_i) < \phi(Q, p_j)$, vagy

$\phi(Q, p_i) = \phi(Q, p_j)$ és p_i közelebb van Q -hoz, azaz $p_i.x < p_j.x$, vagy

$\phi(Q, p_i) = \phi(Q, p_j)$ és $p_i.x = p_j.x$ és $p_i.y < p_j.y$.

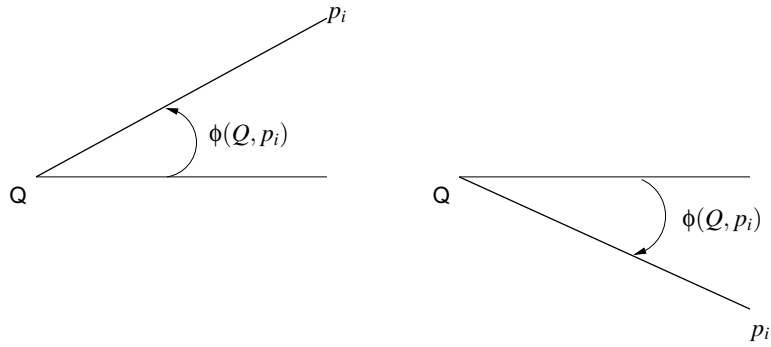
Ha ebben a sorrendben kötjük össze a pontokat, kivéve, hogy az utolsó egyenesen lévő pontokat fordított sorrendben vesszük, akkor egy zárt, nem metsző poligont kapunk.

Hogyan dönthető el, hogy $\phi(Q, p_i) < \phi(Q, p_j)$?

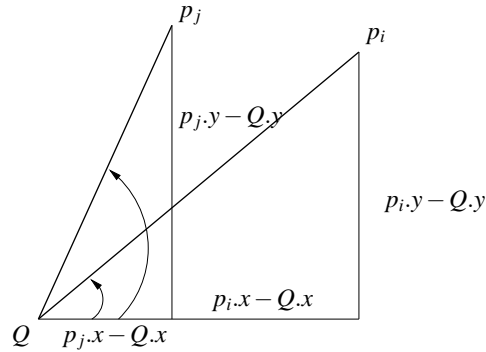
Minden $i > 1$ -re $-\frac{\pi}{2} < \phi(Q, p_i) \leq \frac{\pi}{2}$ és ebben a tartományban a tangens függvény szigorúan monoton növekvő, tehát

$\phi(Q, p_i) < \phi(Q, p_j)$ akkor és csak akkor, ha

$$\tan(\phi(Q, p_i)) = \frac{p_i.y - q.y}{p_i.x - Q.x} < \frac{p_j.y - Q.y}{p_j.x - Q.x} = \tan(\phi(Q, p_j))$$



13. ábra. A p_i ponthoz tartozó előjeles szög: $\phi(Q, p_i)$.



14. ábra. A $\phi(Q, p_i)$ és $\phi(Q, p_j)$ szögek viszonya.

Mivel Q sarokpont, így $p_i.x - Q.x \geq 0$ és $p_j.x - Q.x \geq 0$, tehát

$$\phi(Q, p_i) < \phi(Q, p_j)$$

akkor és csak akkor, ha

$$(p_i.y - q.y)(p_j.x - Q.x) < (p_j.y - Q.y)(p_i.x - Q.x)$$

Tehát a pontok rendezése: p_i megelőzi p_j -t akkor és csak akkor, ha

$$\begin{aligned} (p_i.y - q.y)(p_j.x - Q.x) &< (p_j.y - Q.y)(p_i.x - Q.x) \vee \\ (p_i.y - q.y)(p_j.x - Q.x) &= (p_j.y - Q.y)(p_i.x - Q.x) \wedge p_i.x < p_j.x \vee \\ (p_i.y - q.y)(p_j.x - Q.x) &= (p_j.y - Q.y)(p_i.x - Q.x) \wedge p_i.x = p_j.x \wedge p_i.y < p_j.y \end{aligned}$$

Program Poligon;

Const

MaxN=10000; {a pontok max. száma}

Type

Pont=Record x,y:Longint End;

PontHalmaz=Array[1..MaxN] of Pont;

Sorozat=Array[1..MaxN] Of Word;

Var

N:Word; {a pontok száma}

P:PontHalmaz; {az input ponthalmaz}

S:Sorozat; {az output poligon }

Procedure Beolvas; {Global: N, P} Var

```

Be:Text;
i:word;
Begin
  Assign(Be,'poligon.be'); Reset(Be);
  ReadLn(Be,N);

  For i:=1 To N Do Begin
    ReadLn(Be,P[i].x, P[i].y);
  End;
  Close(Be);
End {Beolvas};

Procedure KiIr(j:Word); {Global: N, S} Var
  Ki:Text;
  i:Word;
Begin
  Assign(Ki,'poligon.ki'); Rewrite(Ki);
  For i:=1 To j Do
    Write(Ki, S[i]:1,' ');
  For i:=N DownTo j+1 Do
    Write(Ki, S[i]:1,' ');

  WriteLn(Ki);
  Close(Ki);
End{KiIr};

Procedure PolarSzogRendez(Const P:PontHalmaz; Var S:Sorozat); {Global: S, N} Var
  Q:Pont; {a sarokpont}
  i0, {a sarokpont inexe}
  i:Word;

Function Megeloz(i,j:Word):Boolean;
{A (Q,P[i]) szög kisebb, mint a (Q,P[j]) szög?}
Begin{Megeloz}
  Megeloz:=
    ((P[i].y-Q.y)*(P[j].x-Q.x) < (P[j].y-Q.y)*(P[i].x-Q.x))
  Or (
    ((P[i].y-Q.y)*(P[j].x-Q.x)=(P[j].y-Q.y)*(P[i].x-Q.x)) And
    (P[i].x<P[j].x)
  )
  Or(
    ((P[i].y-Q.y)*(P[j].x-Q.x)=(P[j].y-Q.y)*(P[i].x-Q.x)) And
    (P[i].x=P[j].x) And (P[i].y<P[j].y)
  )
End{Megeloz};

Function Feloszt( Bal,Jobb : Word): Word ;
{Kimenet: S[bal..f-1] < S[f] < S[f+1..jobb]}
  Var
    Fe : Word;
    i,f : Word;
  Begin
    Fe := S[Bal]; {a felosztó pont indexe}
    f:=Bal;
    For i:=Bal+1 To Jobb Do
      If Megeloz(S[i], Fe) Then Begin

```

```

        S[f]:=S[i];
        Inc(f);
        S[i]:=S[f]
    End;
    S[f]:=Fe;
    Feloszt:= f;
End (* Feloszt *);

Procedure Rendez(Bal,Jobb : Integer);
Var
    f : Word;
Begin
    f:= Feloszt(Bal, Jobb);
    If Bal<f-1 Then
        Rendez(Bal, f-1);
    If f+1<Jobb Then
        Rendez(f+1, Jobb)
    End (* Rendez *);
Begin{PolarSzogRendez}
    i0:=1;
    S[1]:=1; {a sarokpont meghatározása}
    For i:=2 To N Do Begin
        S[i]:=i;
        If (P[i].x<P[i0].x)Or
            ((P[i].x=P[i0].x)And(P[i].y<P[i0].y)) Then
            i0:=i
    End{for i};
    Q.x:=P[i0].x; Q.y:=P[i0].y;
    S[i0]:=1; S[1]:=i0;

    Rendez(2, N)
End{PolarSzogRendez};

Procedure Szamol; {Global: N, S }
Var
    i:Word;
Begin{Szamol}

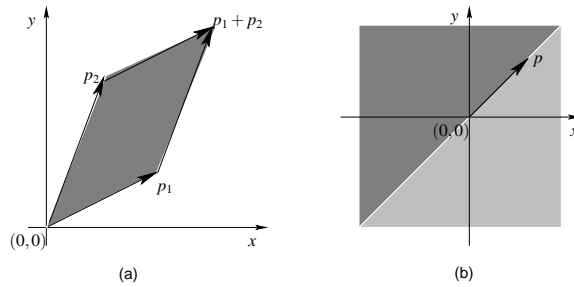
    PolarSzogRendez(P, S);
    {az utolsó előtti egyenesen a sarokponttól legtávolabbi pont meghatározása;}
    i:=N-1;
    While ((P[S[N]].y-P[S[1]].y)*(P[S[i]].x-P[S[1]].x)=
            (P[S[i]].y-P[S[1]].y)*(P[S[N]].x-P[S[1]].x)) Do Dec(i);

    KiIr(i);
    {Kimeneti sorozat: S[1..i],S[N..i+1]}
End{Szamol};

Begin{Program}
    Beolvas;
    Szamol;
End.

```

Gyakran előfordul, hogy a síkon adott p_1 és p_2 pontra el kell dönteni, hogy a p_1 ponthoz képest a p_2 pont milyen forgásirányba esik. Tekintsük a 3. ábrán látható p_1 és p_2 vektorokat. A $p_1 \times p_2$ **keresztsszorzat** a $(0,0)$, p_1, p_2 és $p_1 + p_2 = (p_1.x + p_2.x, p_1.y + p_2.y)$ pontok által alkotott paralelogramma előjeles területként értelmezhető.



15. ábra. (a) A p_1 és p_2 vektorok keresztszorzata a paralelogramma előjeles területe. (b) A világos tartomány azokat a pontokat tartalmazza, amelyek a p -től órajárással egyező irányba esnek, a sötétebb pedig azokat, amelyek órajárással ellentétes irányba esnek p -től.

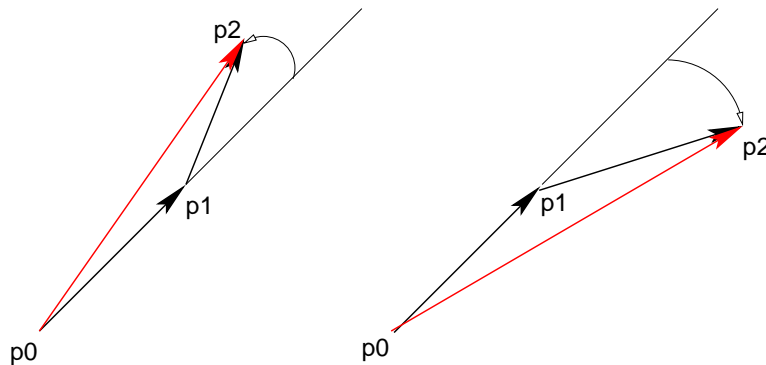
$$\begin{aligned}
 p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\
 &= x_1 y_2 - x_2 y_1 \\
 &= -p_2 \times p_1 .
 \end{aligned}$$

$p_1 \times p_2 < 0 \Leftrightarrow p_2$ az órajárással ellentétes irányban van p_1 -hez képest.

$p_1 \times p_2 = 0 \Leftrightarrow$ a $(0,0)$, p_1 és p_2 pontok egy egyenesre esnek (kollineárisak).

$p_1 \times p_2 > 0 \Leftrightarrow p_2$ az órajárással egyező irányban van p_1 -hez képest. Még általánosabban, adott két csatlakozó irányított szakasz, $\overrightarrow{p_0 p_1}$ és $\overrightarrow{p_1 p_2}$ milyen irányba fordul $\overrightarrow{p_1 p_2}$ a $\overrightarrow{p_0 p_1}$ -hez viszonyítva?

A válasz $(p_1 - p_0) \times (p_2 - p_0) = (p_1.x - p_0.x)(p_2.y - p_0.y) - (p_2.x - p_0.x)(p_1.y - p_0.y)$ keresztszorzat előjele alapján megad-



16. ábra. Csatlakozó szakaszok forgásiránya.

ható.

$(p_1 - p_0) \times (p_2 - p_0) < 0$: $\overrightarrow{p_1 p_2}$ balra fordul,

$(p_1 - p_0) \times (p_2 - p_0) = 0$: $\overrightarrow{p_0 p_1}$ és $\overrightarrow{p_1 p_2}$ kollineárisak,

$(p_1 - p_0) \times (p_2 - p_0) > 0$: $\overrightarrow{p_1 p_2}$ jobbra fordul.

```
Function ForgasIrany(P0,P1,P2:Pont):Integer;
```

```
{Kimenet: +1 ha P1-P2 balra fordul,
          0 ha P0,P1 és P2 kollineárisak,
          -1 ha P1-P2 jobbra fordul.}
```

```
Var
```

```
  KeresztSzorz:Longint;
```

```
Begin{ForgasIrany}
```

```
  KeresztSzorz:=(P1.y-P0.y)*(P2.x-P0.x)-(P2.y-P0.y)*(P1.x-P0.x);
```

```
  If KeresztSzorz < 0 Then
```

```
    ForgasIrany:=1
```

```

Else If KeresztSzorz>0 Then
  ForgasIrany:=-1
Else
  ForgasIrany:=0;
End{ForgasIrany};

```

6.1. Feladat: Fekete-fehér párosítás a síkon.

Adott a síkon n darab fehér és n darab fekete pont úgy, hogy bármely három pont nem esik egy egyenesre. Párosítani kell a fehér pontokat fekete pontokkal úgy, hogy a párokat összekötő szakaszok ne metszék egymást! A pontokat a $1, \dots, n$ számokkal azonosítjuk.

Bemeneti specifikáció

A `paros.be` szöveges állomány első sora a fehér (és fekete) pontok n ($2 < n < 10000$) számát tartalmazza. A további $2n$ sor mindegyike két egész számot tartalmaz, egy pont x és y koordinátáit, ($-20000 \leq x, y \leq 20000$). Az első n sor a fehér, a második n sor a fekete pontokat tartalmazza.

A `paros.ki` szöveges állományba pontosan n sort kell kiírni, minden sorban egy fehér és a hozzá párosított fekete pont sorszáma álljon.

Példa bemenet és kimenet

<code>poligon.be</code>	<code>poligon.ki</code>
5	1 2
6 17	2 4
0 2	3 2
14 1	4 1
-2 23	5 5
-7 19	
32 13	
26 14	
30 24	
21 22	
14 26	

Megoldás

Legyen $P = \{p_1, \dots, p_n, \dots, p_{2n}\}$ a pontok halmaza.

6.1. lemma. *Létezik olyan p_i és p_j különböző színű pontpár, hogy az $e(p_i, p_j)$ egyenes mindkét oldalán a fehér pontok száma megegyezik a fekete pontok számával.*

Bizonyítás. Rendezzük a pontokat a bal-alsó sarokponthoz viszonyított polárszög szerint. Tegyük fel, hogy a rendezésben az első pont fekete. Jelölje $d_i, i = 1, \dots, 2n$. az első i pont közül a feketék számából kivonva a fehérek számát. Tehát $d_1 = 1, d_{2n} = 0$ és $d_{i+1} = d_i + 1$ ha az $i + 1$ -edik pont fekete, egyébként $d_{i+1} = d_i - 1$. Ha a rendezésben utolsó, azaz $2n$ -edik pont fehér, akkor az 1 . és $2n$ -edik pontpár megoldás. Ha a $2n$ -edik pont fekete, akkor $d_{2n-1} = -1$, de mivel $d_1 = 1$ és $d_{i+1} = d_i \pm 1$, így van olyan $1 < i < 2n - 1$ index, hogy $d_i = 0$. Ha az i -edik pont nem fehér, akkor a keresést az $[1, i - 1]$ intervallumban kell folytatni, ami véges sok lépés után végetér. ■

PAROSITAS(P,N)

Procedure Parosit(bal, jobb);

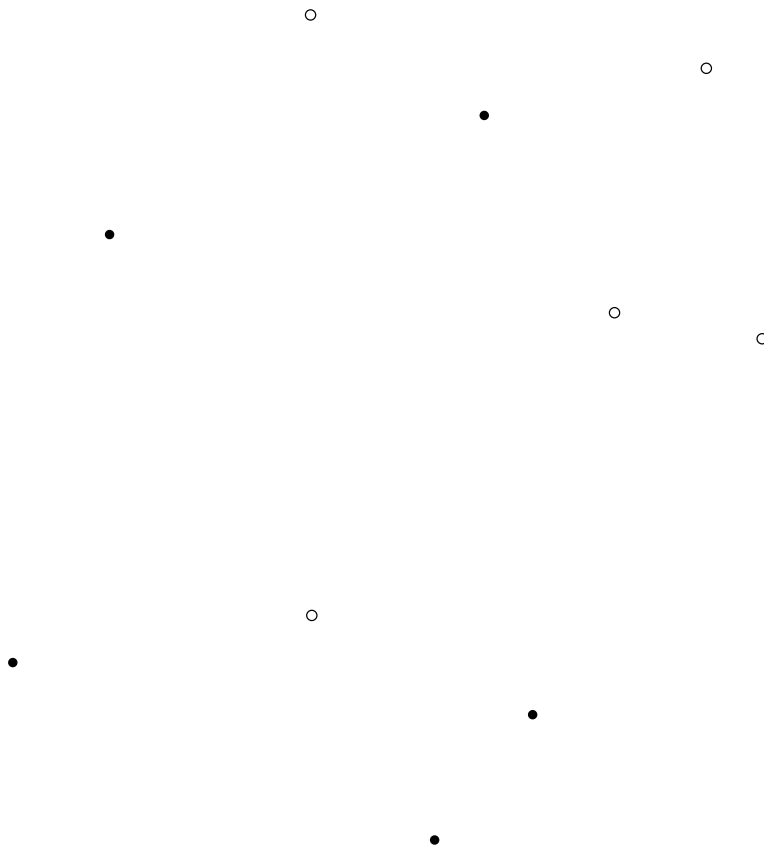
begin {Parosit}

if jobb=bal+1 **then begin**

 i:=bal; j:=jobb; exit;

end;

 PolarSzogRendez(P,S, bal jobb);



17. ábra. Párosítandó pontok

```

d:=1; i:=bal+1
while true do begin
  if (S[bal]>n) Xor (S[i]>n) then
    d:=d-1
  else
    d:=d+1;
  if (d=0)and (S[bal]>n) Xor (S[i]>n) then break;
  i:=i+1;
end {while}
Kilr(bal, i);
if bal+1 < i-1 then Parosit(bal+1, i-1);
if i+1 < jobb then Parosit(i+1,jobb);
end {Parosit}
begin {Parositas}
  Parosit(1, 2*n);
end {Parositas}

```

Az algoritmus futási ideje $O(n^2 \lg n)$.

Geometriai alapműveletek

```

FORGÁSI RANY(P0,P1,P2)
SZAKASZON(P1,P2,Q)
SZAKASZPÁRMETSZ(P1,P2,Q1,Q2)

```

7. Interaktív feladatok

1. Kétszemélyes játékok
2. Kitalálós feladatok

7.1. Feladat: Számjáték (IOI'96)

Adott az alábbi kétszemélyes játék. A játéktábla pozitív egész számok sorozata. A két játékos felváltva lép. Egy lépés azt jelenti, hogy a játékos sorozat bal vagy jobb végéről kiválaszt egy számot. A kiválasztott számot törlik a tábláról. A játék akkor ér véget, ha a számok elfogytak. Az első játékos nyer, ha az általa választott számok összege legalább annyi, mint a második játékos által választottak összege. A második játékos a lehető legjobban játszik. A játékot az első játékos kezdi. Ha kezdetben a táblán levő számok száma páros, akkor az első játékosnak van nyerő stratégiája.

Írjunk olyan programot, amely az első játékos szerepét játssza és megnyeri játékot! A második játékos lépéseit egy már adott számítógépes program szolgáltatja. A két játékos a rendelkezésedre bocsátott `Play` modul három eljárásán keresztül kommunikál egymással.

StartGame Az első játékos a játszmat a paraméter nélküli `StartGame` eljárás végrehajtásával indítja.

MyMove Ha az első játékos a bal oldalról választ számot, akkor a `MyMove('L')` eljárást hívja. Hasonlóképpen a `MyMove('R')` hívással közli a második játékosal, hogy a jobb oldalról választott.

YourMove A második játékos (tehát a gép) azonnal lép. Az első játékos a lépést a `YourMove(C)` utasítással tudhatja meg, ahol `C` egy karakter típusú változó. (C/C++ nyelven `YourMove(&C)`). A `C` változó értéke 'L' vagy 'R' lesz attól függően, hogy a második játékos a bal vagy a jobb oldalról választott.

Bemeneti specifikáció

Az `input.txt` fájl első sora a kezdőtábla n méretét (a számok darabszámát) tartalmazza. n páros és $2 \leq n \leq 100$. A második sor n számot tartalmaz, a játék kezdetén a táblán lévő számokat. A táblán 200-nál nagyobb szám nem szerepel. **Ha** a játék véget ért, akkor a programod írja ki a végeredményt az `OUTPUT.TXT` fájlba! A fájl első sorában két szám legyen! Az első szám az első

játékos által választott számok összegével, a második szám a második játékos által választott számok összegével egyezzen meg! A programodnak a játékot le kell játszania és az output a lejátszott játék eredményét kell tartalmazza.

Példa bemenet és kimenet

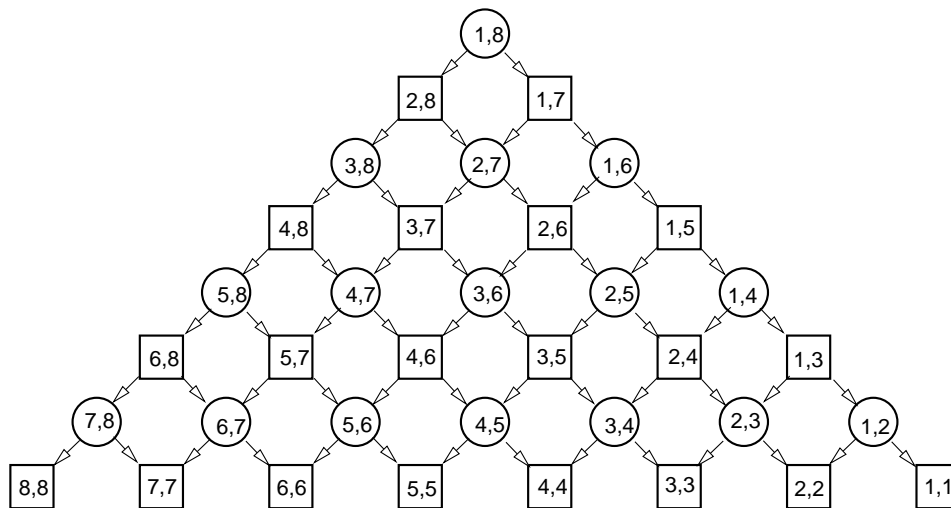
INPUT.TXT
6
4 7 2 9 5 2

OUTPUT.TXT
15 14

Megoldás

Jelölje $\langle a_1, \dots, a_n \rangle$ a kezdeti játékállást. Minden lehetséges játékállást egyértelműen meghatározza az, hogy mely számok vannak még a táblán. Tehát minden játékállás azonosítható (i, j) számpárral, ami azt jelenti, hogy a táblán az $\langle a_i, \dots, a_j \rangle$ számsorozat van. Mivel n páros szám, így minden esetben, amikor az első játékos lép, vagy i páros és j páratlan, vagy fordítva. Tehát az első játékos kényszerítheti a második játékost, hogy az mindig vagy csak páros, vagy csak páratlan indexű elemét válassza a számsorozatnak. Tehát ha a páros indexűek összege nagyobb, vagy egyenlő, mint a páratlanok összege, akkor az első játékos mindig páratlan indexűt választ, egyébként mindig párosat.

Érdekesebb a játék, ha az a cél, hogy az első játékos a lehető legtöbbet szerezze meg, feltéve, hogy erre törekszik a második játékos is. Ábrázoljuk a játékállásokat gráffal.

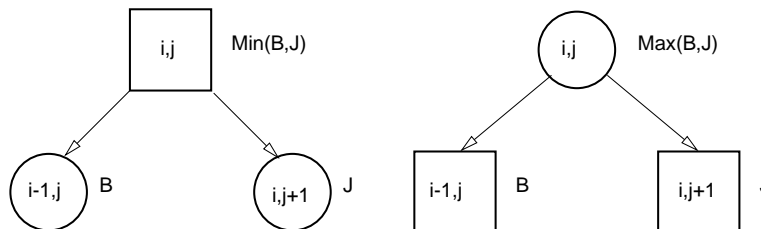


18. ábra. A játékállások gráfja $n = 8$ esetén. Körrel jelölt állásból $(i + j)$ páratlan) az első, négyzettel jelölt állásból $(i + j)$ páros) a második játékos lép.

Definiáljuk minden (i, j) játékállásra azt a maximális pontszámot, amit az első játékos elérhet ebből a játékállásból indulva. Jelölje ezt az értéket $Opt(i, j)$. $Opt(i, j)$ a következő rekurzív összefüggés számítható.

$$Opt(i, j) = \begin{cases} 0 & \text{ha } i = j \\ \max(a_i + Opt(i + 1, j), a_j + Opt(i, j - 1)) & \text{ha } i < j \text{ és } i + j \text{ páratlan} \\ \min(Opt(i + 1, j), Opt(i, j - 1)) & \text{ha } i < j \text{ és } i + j \text{ páros} \end{cases}$$

Tehát alkalmazható a dinamikus programozás módszere, vagyis az $Opt(i, j)$ értékeket a játék megkezdése előtt kiszámítjuk. Tároljuk minden



19. ábra. Mini-max szabály.

(i, j) játékhelyre a $Lep[i, j]$ tömbben az optimális lépést, tehát az 'L' karaktert, ha a képletben $a_i + Opt(i+1, j) > a_j + Opt(i, j-1)$, mert ekkor balról kell elvenni, egyébként pedig az 'R' karaktert, mert ekkor jobbról kell elvenni.

```
Program Jatek;
Uses Play; {a masodik játékost megvalósító modul}
Const
  MaxN=100;
Var
  InpF,OutF: Text;
  A:Array[1..MaxN] Of Word;{ a táblán lévő számok sorozata}
  N: Word;           { a tábla mérete}
  Opt:Array[1..MaxN,1..MaxN] of word;
  Lt:Array[1..MaxN,1..MaxN] of Char; {az 1. játékos optimális lépései}

Procedure Beolvas; Var i:Word; Begin
  Assign(InpF,'input.txt'); Reset(InpF);
  ReadLn(InpF,N);
  For i:=1 To N Do
    Read(InpF,A[i]);
  Close(InpF);
End;

Procedure Elofeldolgoz; Var i,j:Word;
  Pont,PontBal,PontJobb:Word;
Begin
  For j:=1 To N Do Begin
    Opt[j,j]:=0;
    For i:=j-1 DownTo 1 Do Begin
      If Odd(j-i+1) Then Begin{2. játékos lép}
        If Opt[i+1,j]<Opt[i,j-1] Then
          Opt[i,j]:=Opt[i+1,j]
        Else
          Opt[i,j]:=Opt[i,j-1]
      End Else Begin{1. játékos lép}
        PontBal:=A[i]+Opt[i+1,j];
        PontJobb:=A[j]+Opt[i,j-1];
        If PontBal>PontJobb Then Begin
          Opt[i,j]:=PontBal; Lt[i,j]:='L'
        End Else Begin
          Opt[i,j]:=PontJobb; Lt[i,j]:='R'
        End
      End;
    End{for i};
  End{for j};
End {Elofeldolgoz};

Procedure Jatszaz;
Var
  Bal,Jobb:Word;{az aktuális játékhely: A[Bal..Jobb]}
  L1,L2: Char; {a két játékos aktuális lépése}
Begin
  Bal:=1; Jobb:=N;           {a kezdő játékhely beállítása}
  While Bal<=Jobb Do Begin {amíg nem üres a tábla}
    MyMove(Lt[Bal, Jobb]); {az én lépésem}
    If Lt[Bal, Jobb]='L' Then {a játékhely aktualizálása}
      Inc(Bal)
    Else
      Dec(Jobb);
    L2:=YourMove;           {az ellenfél lépése}
    If L2='L' Then           {a játékhely aktualizálása}
      Inc(Bal)
    Else
```

```

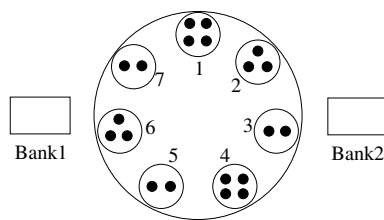
    Dec (Jobb);
End{while};
End{Jatszias};

Begin
    Beolvas;
    Elofeldolgoz;
    StartGame;
    Jatszias;
End.

```

7.2. Feladat: IOIWari játék (IOI'2001)

A Mancala családba tartozó játékok, amelyeket gyöngyökkel és üregekkel játszottak, a legősibbek közül valók. A wari játék IOI-s változatát két játékos játssza olyan kör alakú táblán, amelyben hét üreg van. A játék kezdetén az üregekbe véletlenszerűen beraknak húsz gyöngyöt úgy, hogy minden üregbe legalább két és legfeljebb négy gyöngy kerüljön. Mindkét játékosnak van egy-egy tálkája. A játékosok felváltva lépnek. A soron következő játékos kiválaszt egy nem üres üreget, a kezébe veszi belőle az összes gyöngyöt, és az óra járásával megegyező irányban haladva, a kiürített üreg szomszédjával kezdve, az alábbiakat teszi, amíg van gyöngy a kezében:



20. ábra. IOIWari játék egy kezdő állása.

- Mindaddig, amíg egynél több gyöngy van a kezében: ha a soron következő üregben pontosan öt gyöngy van, akkor kivesz egyet az üregből, és a tálkájába teszi, egyébként pedig a kezében lévő gyöngyökből egyet az üregbe rak.
- Ha már csak egy gyöngy van a kezében: ha a soron következő üregben legalább egy és legfeljebb négy gyöngy van, akkor az üregben lévő összes és a kezében lévő egyetlen gyöngyöt a tálkájába teszi, egyébként (azaz ha az üregben nincs gyöngy vagy pontosan öt gyöngy van) a kezében lévő gyöngyöt az ellenfél tálkájába teszi.

A játék akkor ér véget, ha minden üreg kiürült, és az a játékos győz, akinek a tálkájában több gyöngy van. A kezdőjátékosnak mindig van nyerő stratégiája.

Írjunk olyan programot, amely kezdőjátékosként loiwari-t játszik, és győz. Az értékeléskor az ellenfelet megvalósító program optimálisan játszik, azaz biztosan győz, ha lehetőséget adsz rá.

Bemenet és kimenet:

Az adatokat a standard inputról kell beolvasni, és a standard outputra kell kiírni. A programod az 1-es, az ellenfél programja a 2-es játékos. A játszma kezdetén a programodnak egy sort kell beolvasnia, amelyben hét egész szám, az 1-től 7-ig számozott üregekben lévő gyöngyök száma van (az üregeket az óramutató járása szerinti sorrendben számozzuk). A tálkák kezdetben üresek. A programodnak az alábbiak szerint kell működnie: Ha te lépsz, a programodnak ki kell írnia a standard outputra a kiválasztott üreg sorszámát. Ha az ellenfél lép, a programodnak be kell olvasnia a standard inputról az ellenfél által választott üreg sorszámát.

Segédeszközök:

Kapsz egy programot (Linux: ioiwari2, Windows: ioiwari2.exe), amely optimális ellenfélként játszik az alábbi kezdőállásból. Először ezt írja ki a standard outputra, amit a programodnak kezdéskor be kell olvasnia: 4 3 2 4 2 3 2.

A programodat és az ioiwari2-t külön ablakban futtasd, és az egyik program által kiírtakat másold át a másik programnak! Az ioiwari2 a párbeszédet az ioiwari.out állományba is beírja.

Programozási tanácsok:

Az alábbi példákban a last egész változó a legutoljára beolvasott sorszámot tárolja, a mymove egész változóban pedig a kiírandó sorszámnak kell lennie. Ha C++ nyelven az iostreams modul használod, a következőképpen olvasd, illetve írd:

```

cout<<mymove<<endl<<flush;
cin>>last;

```

Ha C vagy C++ nyelven a scanf és printf eljárásokat használod, a következőképpen olvasd, illetve írd:

```

printf("%d\n",mymove); fflush (stdout);
scanf ("%d", &last);

```

Pascal nyelven a következőképpen olvass, illetve írd:

```
Writeln(mymove);
Readln(last);
```

Példa:

lépés/üreg szám	1	2	3	4	5	6	7	bank1	bank2
kezdőállapot	4	3	2	4	2	3	2	0	0
1. játékos lépése:2	4	0	3	5	0	3	2	3	0
2. játékos lépése:3	4	0	0	4	1	4	0	3	4
1. játékos lépése:5	4	0	0	4	0	0	0	8	4
2. játékos lépése:4	0	0	0	0	1	1	1	8	9
1. játékos lépése:5	0	0	0	0	0	0	1	10	9
2. játékos lépése:7	0	0	0	0	0	0	0	11	9

Pontozás:

Ha a programod nyer, 4 pontot, ha döntetlent ér el, 2 pontot, egyébként 0 pontot kapsz tesztesenként.

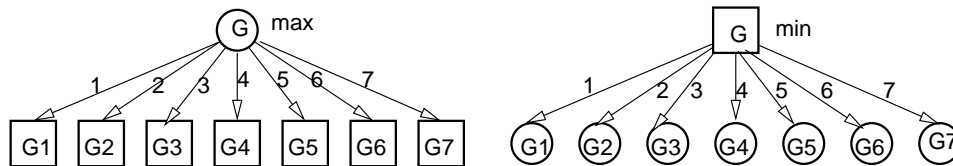
Megoldás

1. A játékszabályból következik, hogy minden játéállás megadható egy $G = \langle g_1, g_2, g_3, g_4, g_5, g_6, g_7 \rangle$ sorozat, ahol g_i ($0 \leq g_i \leq 5$) az i -edik üregben lévő gyöngyök száma.
2. Minden lépést egyértelműen meghatároz az, hogy melyik üreget választotta az éppen lépő játékos.
3. Minden lépésben legalább eggyel csökken az üregekben lévő gyöngyök összegének száma.
4. A játéállás önmagában nem határozza meg, hogy melyik játékos lép.

Következésképpen, a játéállások gráfja körmentes (nem következhet be ismét egy korábbi játéállás).

Minden G játéállásra és $w = 1, 2$ -re legyen $Opt(w, G)$ a G játéállásból az első játékos által elérhető legjobb különbség, feltéve, hogy w lép először.

$Opt(w, G)$ kiszámítását rekurzió-memorizálás módszerével célszerű végezni. Tehát ha kiszámítottuk adott (w, G) -re, akkor tároljuk egy T tömb-



21. ábra. A minimax játékfa pontjai.

ben az értéket.

Minden játéállás tekinthető egy 6-os számrendszerbeli számnak, tehát a játéállások azonosíthatók $0..279936$ számokkal. (Nem minden számhoz tartozik valódi játéállás.)

```
Program Owari;
Uses Game;
Const
  TSize=7;                {táblaméret}
  Total=20;               {a gyöngyök száma kezdetben}
  Inf =2*Total+1;         {a végtelen reprezentánsa}
  MaxN =6*6*6*6*6*6-1;   {max. 6-os szám=279936}

Type
  Board=Record
    Who:Boolean;         {=True: az 1. játékos lép, False: 2. játékos lép}
    Pit:Array[1..TSize] of Byte; {üregtartalom }
    Bank:Array[Boolean] of Integer; {Bankok}
  End;

Var
  T:Array[Boolean,0..MaxN] of Byte;
  {T[w,b] a legjobb különbség, amelyet a w játékos elérhet a b játéállásból
  folytatva a játékot. }
  Om1: Array[0..MaxN] of Byte;    {az első játékos nyerő lépései}

  B:Board;                       {az aktuális játéállás}
  i:Longint;
```

```

Function B6N(Var B:Board):Longint;
  {A B játékalállítás 6-os számrendszerbeli azonosítóját számítja ki}
Const
  P6:Array[1..TSize] of Longint=(6*6*6*6*6*6,6*6*6*6*6*6,6*6*6*6*6*6,6*6*6*6*6*6,6*6*6*6*6*6,6*6*6*6*6*6);
Var i:Integer;
  a:Longint;
{Megjegyzés: Ha előbb kiszámítanánk B rotációs ekvivalensét, akkor elés lenne 121305 elemű
tömb } Begin{B6N};
  a:=0;
  For i:=1 To TSize Do Begin
    a:=a+B.Pit[i]*P6[i];
  End;
  B6N:=a;
End{B6N};

Procedure Move(Var B:Board; i:Byte; Var BB:Board);
  {Végrehajtja az i. lépést a B játékalállításra, az eredmény BB-ben keletkezik.}
Var S:Integer;
  j:Byte; W0,W:Boolean;
Begin
  S:=B.Pit[i];
  W0:=B.Who; W:=Not W0; BB:=B;
  BB.Who:=W; BB.Pit[i]:=0; j:=i;
  While S>1 Do Begin
    Inc(j); If j>Tsize Then j:=1;
    If BB.Pit[j]=5 Then Begin
      Dec(BB.Pit[j]);
    Inc(BB.Bank[W0]);
    End Else Begin
      Inc(BB.Pit[j]);
      Dec(S);
    End;
  End{while};
  Inc(j); If j>Tsize Then j:=1;
  If (BB.Pit[j]>=1)And(BB.Pit[j]<=4) Then Begin
    Inc(BB.Bank[W0],BB.Pit[j]+1);
    BB.Pit[j]:=0;
  End Else Begin
    Inc(BB.Bank[W]);
  End;
End{Move};

Function MinMax(Var B:Board):Integer;
  {Az 1. játékos által elérhető legjobb különbséget számítja ki
és az optimális lépést beírja az Om1 tömbbe. }
Var
  i:Byte;
  BB:Board;
  Diffn,Diffs:Integer;
  a:Longint;
Begin{MinMax}
  a:=B6N(B); {B 6-os szárendszerbeli száma}

  If (T[B.Who,a]<>Inf) Then Begin {már kiszámítottuk, }
    MinMax:=T[B.Who, a]; {vesszük az értéket T-ből}
    Exit;
  End;

  If B.Who Then Begin {az 1. játékos lép}
    Diffn:=-Inf;
    For i:=1 To TSize Do {mind a 7 lépési lehetőségre }
      If (B.Pit[i]>0) Then Begin {ha nem üres az i. üreg}

```

```

    Move(B,i, BB);
    Diffs:=MinMax(BB)+(BB.Bank[True]-BB.Bank[False])-(
        (B.Bank[True]-B.Bank[False]));
    If Diffs>Diffn Then Begin {maximumot vesszük}
        Diffn:=Diffs;
        Oml[a]:=i;
    End;
End;
;{for i}

End Else Begin
    (a 2. játékos lép)
    Diffn:=Inf;
    For i:=1 To TSize Do
        {mind a 7 lépési lehetőségre }
        If (B.Pit[i]>0) Then Begin
            {ha nem üres az i. üreg}
            Move(B,i, BB);
            Diffs:=MinMax(BB)+(BB.Bank[True]-BB.Bank[False])-(
                (B.Bank[True]-B.Bank[False]));
            If Diffs<Diffn Then Begin {minimumot vesszük}
                Diffn:=Diffs;
            End;
        End;
    End;{for i}
End;

T[B.Who, a]:=Diffn;
MinMax:=Diffn;
End{MinMax};

Procedure Play(Var B:Board);
    {A játék lejátsszása }
    Var i,ii:Integer;
    BB:Board;
    a:Longint;
Begin
    BB:=B;

    While True Do Begin
        a:=B6N(BB);
        i:=Oml[a];
        {az optimális lépés a B játékaállásból}
        Move(BB,i, BB); {a lépés végrehajtása}
        MyMove(i);

        ii:=YourMove;
        {az ellenfél lépésének lekérdezése}
        Move(BB,ii, BB);{a lépés végrehajtása}
    End{while};

End{Play};

Begin{Prog}
    For i:=1 To TSize Do
        {beolvassuk a kezdeti játékaállást}
        B.Pit[i]:=Pit(i);
    B.Bank[True]:=0;B.Bank[False]:=0;{a bank üresítése}
    B.Who:=True;
        {az 1. játékos lép először}

    For i:=1 To MaxN Do Begin
        {inicializálás a memorizáláshoz}
        T[True, i]:=Inf;
        T[False,i]:=Inf;
    End;
    T[True, 0]:=Total;
    T[False,0]:=Total;
    Oml[0]:=0;

    MinMax(B);
        {előfeldolgozás}

```

```
Play(B);  
End.
```

```
{a játék végrehajtása}
```

Kitalálós feladatok

7.3. Feladat: Többségi elem kiválasztása (CEOI'2001)

Iskolád tanulói két csoportba tartoznak. Tudjuk, hogy az egyik csoportban többen vannak, mint a másikban, ezt nevezzük többségi csoportnak. Ki kell választani egy tanulót, aki a többségi csoporthoz tartozik. Ehhez egyetlen műveletet használhatunk, nevezetesen két tanulótól megkérdezhetjük, hogy ugyanabba a csoportba tartoznak-e.

Olyan programot kell írni, amelyik a lehető legkevesebb kérdéssel meghatároz egy többségi csoporthoz tartozó tanulót. A tanulókat sorszámmal azonosítjuk.

KÖNYVTÁRI MŰVELETEK

A feladat megoldásához három könyvtári művelet van.

Size A tanulók n számát adja. Ezt kell először hívni.

Member Két tanuló sorszámát kell argumentumként megadni, és a függvényeljárás 1 értéket ad, ha a két tanuló ugyanazon csoport eleme, egyébként 0-át.

Answer Ezzel a művelettel kell közölni a kiválasztott, többségi csoportba tartozó tanuló sorszámát.

A programod és a könyvtári modul közötti párbeszédet a `select.out` szöveges állományban rögzítik. Ez a nap állomány a programod által közölt megoldást is tartalmazza, továbbá azt is, hogy az helyes-e. A megoldást csak akkor fogadják el, ha a tanulók bármely olyan diszjunkt A és B részhalmazára, amely kompatibilis az általad feltett kérdésekkel, a közölt megoldás a nagyobb elemszámú részhalmazban van.

A válaszadó arra kényszerít, hogy szükséges számú kérdést tegyél fel.

Pascal program esetén

uses query; import direktívát kell használni.

C/C++ program esetén

#include "query.h" direktívát kell használni.

GYAKORLÁS

A könyvtári modul úgy használható, hogy a `select.in` szöveges állomány első és egyetlen sorába a tanulók n számát kell írni, ami páratlan szám kell legyen!

Példa bemenet és kimenet

select.in	select.out
7	Size=7
	Member(1,2)=0
	Member(3,4)=1
	Member(5,6)=1
	Member(4,6)=0
	Your Answer: 7, Correct
	Majority Group:
	2 5..7
	Non-majority Group:
	1 3 4
	Number of Queries: 4
	Full Possible Score: 3
	Your Score: 3

Például, az 1 válasz nem elfogadható, mert minden feltett kérdésre a $\{2,5,6,7\}$ és $\{1,3,4\}$ halmazok esetén a MEMBER függvény ugyanazt eredményezné, de 1 nem eleme a $\{2,5,6,7\}$ többségi csoportnak. Az $a..b$ jelölés az a és b közötti egész számok halmazát jelenti. **FELTÉTELEK**

- A tanulók n számára $5 \leq n \leq 30000$, n páratlan teljesül.
- A programod nem olvashat és nem írhat semmilyen fílet.
- A tanulók azonosítói: $1 \leq i \leq n$
- FreePascal könyvtári modulok filenevei: `query.ppw`, `query.ow`.
- A könyvtári műveletek Pascal deklarációi:

```
function Size: integer;  
function Member(x, y: integer): integer;  
procedure Answer(x: integer);
```

- C/C++ könyvtári modulok fájlnevei: `query.h`, `query.o`
- C/C++ deklarációk:

```
int Size(void);
int Member(int x, int y);
void Answer(int x);
```

PONTOZÁS

Helyes válasz esetén a kapott pontszám: $\max(0, n - k)$, ha a programod k MEMBER műveletet hajtott végre.

Megoldás

Jelölje $H = \{1, \dots, n\}$ a tanulók halmazát. Azt mondjuk, hogy egy $A \subseteq H$ **homogén** részhalmaz, ha A minden eleme ugyanabba a csoportba tartozik, azaz ha $(\forall x, y \in A)(member(x, y) = 1)$. Azt mondjuk, hogy az $U, V \subseteq H$ homogén részhalmazok **ellentétes** részhalmazok, ha U minden eleme az egyik, V minden eleme a másik csoportba tartozik, azaz ha $(\forall x \in U)(\forall y \in V)(member(x, y) = 0)$.

Vegyük észre, hogy ha $member(x, y) = 0$, akkor x és y elhagyható a halmazból, mert biztosan marad még többségi csoportba tartozó elem. Általánosan, ha $U, V \subseteq H$ homogén részhalmazok és elemszámuk megegyezik, továbbá valamely $x \in U$ és $y \in V$ elemekre $member(x, y) = 1$ -et kapunk, akkor U és V törölhető H -ből.

Bármely kérdéssorozat által nyert ismeret ábrázolható egy olyan halmazzal, amelynek az elemei diszjunkt halmazpárok. Pontosabban, az alábbi feltételeket kielégítő halmazzal.

$$I = \{(U_1, V_1), \dots, (U_m, V_m)\} \quad (6)$$

$$\text{az } U_i, V_j \text{ halmazok páronként diszjunktak } 1 \leq i, j \leq m \quad (7)$$

$$U_i \text{ és } V_i \text{ homogén részhalmaz, } 1 \leq i \leq m \quad (8)$$

$$U_i \text{ az } V_i \text{ ellentétesek } 1 \leq i \leq m \quad (9)$$

$$\bigcup_{i=1}^m (U_i \cup V_i) = \{1, \dots, n\} \quad (10)$$

$$(11)$$

A kezdeti helyzetet, amikor nincs semmi ismeretünk, az

$$I = \{(\{1\}, \emptyset), \dots, (\{n\}, \emptyset)\} \quad (12)$$

halmaz ábrázolja. Tegyük fel, hogy az eddig elvégzett kérdésekkel nyert információt az (6) halmaz ábrázolja, és a $member(x, y)$ kérdést tesszük fel. Mivel az I -beli részhalmazok páronként diszjunktak, pontosan egy olyan i és pontosan egy olyan j index van, hogy

$$x \in U_i \cup V_i \text{ és } y \in U_j \cup V_j$$

Ha $i = j$, akkor a kérdés redundáns, azaz a kérdésre a válasz megadható a korábbi kérdésekre kapott válaszok alapján, nevezetesen, a válasz *Igaz*, ha $x \in U_i$ és $y \in U_i$ vagy $x \in V_i$ és $y \in V_i$, egyébként a válasz 0. Ekkor nem jutunk új ismerethez.

Ha $i \neq j$, akkor új ismerethez jutunk, amit az

$$I' = I - \{(U_i, V_i), (U_j, V_j)\} \cup \{(U, V)\} \quad (13)$$

halmazzal ábrázolhatunk, ahol az (U, V) párt az alábbiak szerint kapjuk.

Ha $member(x, y) = 1$, akkor

$$(U, V) = \begin{cases} (U_i \cup U_j, V_i \cup V_j) & \text{ha } (x, y \in U_i \cup U_j) \vee (x, y \in V_i \cup V_j) \\ (U_i \cup V_j, U_j \cup V_i) & \text{ha } (x, y \in U_i \cup V_j) \vee (x, y \in U_j \cup V_i) \end{cases}$$

Ha $member(x, y) = 0$, akkor

$$(U, V) = \begin{cases} (U_i \cup U_j, V_i \cup V_j) & \text{ha } (x, y \in U_i \cup V_j) \vee (x, y \in U_j \cup V_i) \\ (U_i \cup V_j, U_j \cup V_i) & \text{ha } (x, y \in U_i \cup U_j) \vee (x, y \in V_i \cup V_j) \end{cases}$$

Nyilvánvaló, hogy bármely x akkor és csak akkor fogadható el a feladat helyes megoldásának, ha az elvégzett kérdésekhez tartozó I ismeretre teljesül, hogy $x \in U_i \cup V_i$ esetén, ha $x \in U_i$ akkor $|U_i| > |V_i|$, illetve ha $x \in V_i$ akkor $|V_i| > |U_i|$, és

$$\max(|U_i|, |V_i|) + \sum_{\substack{j=1 \\ j \neq i}}^n \min(|U_j|, |V_j|) > \min(|U_i|, |V_i|) + \sum_{\substack{j=1 \\ j \neq i}}^n \max(|U_j|, |V_j|) \quad (14)$$

Ha az I ismeretre teljesül a (13) egyenlőtlenség valamely i indexre, akkor azt mondjuk, hogy I **biztos** ismeret. Ekkor az U_i és V_i halmazok közül a nagyobbik elemszámú halmaz mindegyik eleme biztosan a többségi csoportba tartozik.

A (14) feltétel ekvivalens a (15) feltétellel.

$$||U_i| - |V_i|| > \sum_{j=1, j \neq i}^n ||U_j| - |V_j|| \quad (15)$$

```

I := {{1}, ..., {n}};
x:=0;
while |Max(I)| ≤ n/2 do begin
  x := x + 1;
  y := x + 1;
  if member(x,y) = 1 then begin
    U := {x,y};
    while van olyan V ∈ I, hogy |U| = |V| do begin
      y := V egy tetszőleges eleme;
      I := I - {V};
      if member(x,y)=1 then
        U := U ∪ V;
      else begin
        n := n - 2|U|;
        U := ∅;
        Break;
      end
    end;
    if U ≠ ∅ then
      I := I ∪ {U}
    end else
      n := n - 2;
end;
t := Max(I) egy eleme;

```

Az algoritmus legfeljebb

$$n - b(n) \quad (16)$$

kérdést tesz fel, ahol $b(n)$ az n szám kettes számrendszerbeli felírásában az egyes bitek száma, azaz

$$b(n) = \sum_{i=1}^k b_i \quad (17)$$

ha

$$n = \sum_{i=1}^k b_i 2^i (b_k \neq 0) \quad (18)$$

Megvalósítás

```

Program Select; Uses Query; Const
  MaxN=30000;           {max. tanulószám}
  MaxK=20;             {MaxN<=2^MaxK}
Var
  N:1..MaxN;          {atanulók száma}
  M:0..MaxN;          {a redukált halmaz elemszáma}
  Fel:Longint;        {a többségi csoport elemszáma}
  B:Array[0..MaxK] Of Boolean; {2^k elemű részcsoportok}
  R:Array[0..MaxK] Of 0..MaxN; {a részcsoportok egy reprezentánsa}
  Pow2:Array[0..MaxK] Of Longint; {Pow2[k]=2^k}
  L:Word;             {a legnagyobb részcsoport elemszáma 2^L}
  i,k:Integer;
Begin
  Pow2[0]:=1;
  For k:=1 To MaxK Do           {2 hatványok kiszámítása}
    Pow2[k]:=Pow2[k-1] Shl 1;
  N:=Size;                      {a tanulók számának lekérdezése}
  M:=N;
  Fel:=M div 2 + 1;
  L:=0; i:=0;

```

```

While i<N Do Begin                               {M elemű halmaz többségi elemének keresése}
  k:=0; B[0]:=True;
  Inc(i); R[0]:=i;
  Inc(i);                                       {a két következő elem: i és i+1}
  If i>N Then Break;                           {nincs több}
  While B[k] Do Begin                          {van két 2^k elemszámú részcsoport}
    If member(R[k],i)=1 Then Begin{i-t és R[k]-t tartalmazó két 2^k elememű}
      B[k]:=False;                             {részcsoport egyesítése}
      Inc(k);                                  {2^k+1 elemű lesz az új részcsoport}
      If k>L Then L:=k;                       {új legnagyobb részcsoprt?}
    End Else Begin                             {nem azonos csoportba tartozó részcsoportok}
      Dec(M, Pow2[k+1]);                       {M:=M-2^(k+1)}
      Dec(Fel, Pow2[k]);                       {Fel:=Fel-2^k}
      B[k]:=False;                             {töröljük a részcsoportot}
      If k=L Then                              {L aktualizálása}
        While (L>0)And Not B[L] Do
          Dec(L);
        k:=-1;
        Break;
      End;
    End{while B[k]};

  If k>=0 Then Begin
    B[k]:=True;                               {új 2^k elemszámú részcsoportot kaptunk}
    R[k]:=i;                                  {i az új részcsoport reprezentánsa}
  End;

  If (L>0)And(Pow2[L]>=Fel) Then{a legnagyobb részcsoport a többségi?}
    Break;
  End{while i<N};

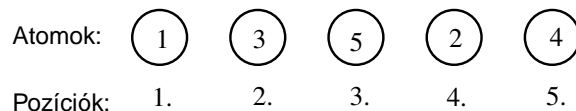
  Answer(R[L]);                               {a legnagyobb részcsoport reprezentánsa a megoldás}
End.

```

7.4. Feladat: Atomlánc (CEOI'2001)

Kutatók egy speciális molekulát vizsgálnak. Tudják, hogy a molekula n különböző atomot tartalmaz, amelyek egy lineáris láncot alkotnak. A kutatóknak van egy olyan mérőműszerük, amellyel meg tudják mérni a molekula két adott atomjának a távolságát. A műszer által mért távolság a két atom pozíció különbségének abszolút értéke. Az elvégezhető mérések azonban korlátozottak, egyetlen atom sem szerepelhet négynél több mérésben, mert ez tönkretenné a molekulát.

Olyan programot kell írni, amely meghatározza a molekula szerkezetét, azaz minden atom pozícióját a molekulában!



22. ábra. Atomlánc példa

KÖNYVTÁR

A mérőműszer használatát a Meter könyvtár három művelete biztosítja:

Size Egyszer kell hívni a program elején, az atomok n számát adja.

Span Két atom sorszámát kell argumentumként megadni, a visszaadott érték a két atom távolsága.

Answer A program végén kell hívni, a kiszámított eredmény közzléséhez. Két argumentuma van, i és x , ami azt jelenti, hogy a molekulában az i -edik pozíción a x sorszámú atom van. Minden i -re ($1 \leq i \leq n$) pontosan egyszer kell hívni, tetszőleges sorrendben. A megoldás tükörkép erejéig egyértelmű, a két megoldás közül bármelyiket meg lehet adni.

A Meter könyvtár két szöveges állományt készít: `chain.out` és `chain.log`. A `chain.out` két sort tartalmaz, az első sor a program által egy atomra végrehajtott legtöbb `Span` művelet számát tartalmazza. Ez a szám legfeljebb 5 lehet. A második sor az `Answer` műveletekkel közölt atom sorszámok sorozatát tartalmazza. A program és könyvtár közötti dialógust `chain.log` tartalmazza.

Utasítások Pascal programozóknak:

A `uses meter;` import utasítás szerepeljen a program első sorában.

Utasítások C/C++ programozóknak:

A `#include "meter.h"` import utasítás szerepeljen a program első sorában. Készítsen egy `chain.gpr` projekt állományt a feladatkönyvtárban és adja hozzá a projekthez a `chain.c` (`chain.cpp`) és `meter.o` állományokat és `compile/make` paranccsal végezze a fordítást.

HASZNÁLAT Készíteni kell egy `meter.in` szöveges állományt, amely két sort tartalmazzon. Az első sorban legyen az atomok n száma. A második sor pontosan n különböző számot tartalmazzon egy-egy szóközzel elválasztva, az atomok sorszámait. Minden szám 1 és n közötti egész szám legyen.

Példa bemenet és kimenet

```
meter.in
```

```
5
```

```
1 3 5 2 4
```

Eljárás hívások, amelyek egy megoldását adják a példa bemenetnek:

1. `Size` (Pascal-ban) vagy `Size()` (C/C++-ban) 5 értéket ad
2. `Span(1,3)` 1-et ad
3. `Span(2,5)` 1-et ad
4. `Span(3,5)` 1-et ad
5. `Span(1,4)` 4-et ad
6. `Answer(1,1)` `Answer(5,4)` `Answer(3,5)` `Answer(4,2)` `Answer(2,3)`

FELTÉTELEK

- Az atomok n számára: $5 \leq n \leq 10000$
- Ha bármely atom négynél többször szerepel `Span` műveletben, akkor a program megszakítását eredményezi.
- A megoldás program nem olvashat és nem írhat semmilyen állományt.
- Az atomok pozícióira: $1 \leq i \leq n$.
- FreePascal könyvtárnevek: `meter.ppw` and `meter.ow`

- Pascal deklarációk:

```
function Size: integer;  
function Span(x, y: integer):integer;  
procedure Answer(i, x integer);
```

- C/C++ könyvtárnevek: `meter.h`, `meter.o`

```
int Size(void);  
int Span(int x, int y);  
void Answer(int i, int x);
```

PONTOZÁS Ha a közölt megoldás helyes és egyetlen atom sem szerepelt háromnál többször `Span` műveletben, akkor teljes pontszám (100%) jár. Ha a közölt megoldás helyes és egyetlen atom sem szerepelt négynél többször `Span` műveletben, de volt olyan atom amely négy `Span`-ben szerepelt, akkor fél pontszám (50%) jár. Minden más esetben 0 pont jár. **Megoldás**

Biztosítható, hogy legyen három olyan atomunk; a, b, c , amelyek relatív pozícióját ismertjük, és mindegyik legfeljebb 2 kérdésben szerepelt eddig. De ekkor

$$Span(x,y) \neq Span(a,b) \text{ vagy } Span(x,y) \neq Span(a,c) \text{ vagy } Span(x,y) \neq Span(b,c)$$

```
Program Chain;
```

```
Uses Meter;
```

```
Const
```

```
MaxN=10000; {az atomok max. száma}
```

```
Var
```

```
N:Integer; {az atomok száma}
```

```
S:Array[0..MaxN] of Integer; {a megoldás}
```

```
i:Integer;
```

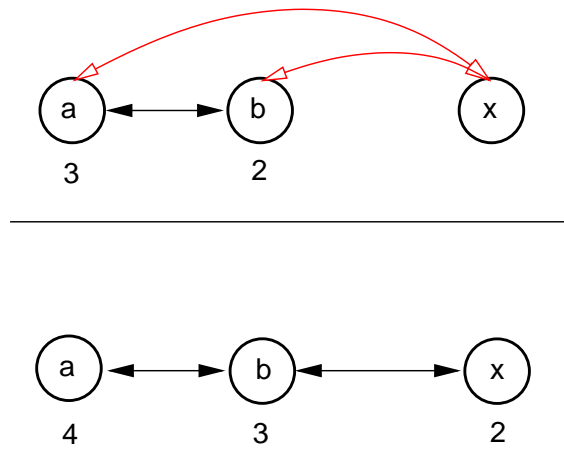
```
Procedure Compute; {Global: N, S }
```

```
Var
```

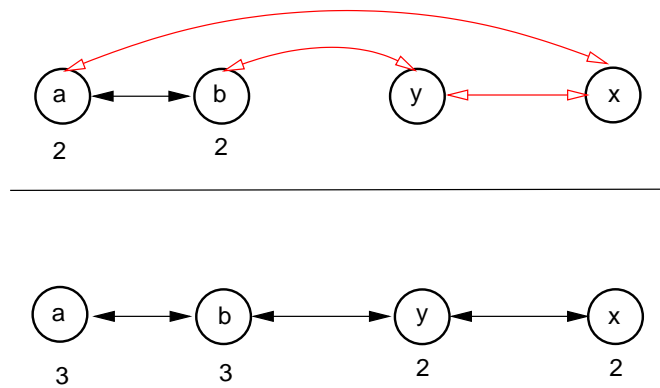
```
Pos:Array[0..MaxN] Of -MaxN..MaxN;{relatív atom pozíciók 1-hez képest}
```

```
a,b,c,x,y:Integer; {atom címkék}
```

```
Dab,Dxy,Dax,Dbx,Dby:Integer; {távolságok}
```



23. ábra. Ha ismerjük a és b atomok relatív pozícióját, és a eddig legfeljebb 3, b pedig legfeljebb 2 kérdésben szerepelt, akkor tetszőleges új x atom relatív pozícióját ki tudjuk számítani $Span(a,x)$ és $Span(b,x)$ alapján. Tehát ismét lesz két ismert atomunk; b és x amelyek 3, illetve 2 kérdésben szerepeltek.



24. ábra. Ha ismerjük a és b atomok relatív pozícióját, és mind a , mind b eddig pedig legfeljebb 2 kérdésben szerepelt, akkor két tetszőleges új x és y atom relatív pozícióját ki tudjuk számítani $Span(x,y)$, $Span(a,x)$ és $Span(b,x)$ alapján. Feltéve, hogy $Span(x,y) \neq Span(a,b)$. Ekkor ismét lesz két ismert atomunk; x és y amelyek 2 kérdésben szerepeltek.

```

i,k,min,z:Integer;

Begin{Compute};
a:=1; b:=2; c:=3; {3 atom választása}
Dab:=Span(a,b); Dax:=Span(a,c); Dbx:=Span(b,c);
Pos[1]:=0; Pos[2]:=Dab;

If (Dax=Dab+Dbx) Or (Dab=Dax+Dbx) Then {c a->b relatív pozíciójának kiszámítása}
  Pos[c]:=Pos[a]+Dax
Else If (Dbx=Dax+Dab) Then
  Pos[c]:=Pos[a]-Dax;
i:=3;
While i+2<=N Do Begin
{a,b és c 3-szor szerepelt kérdésben és rel. pozíciójuk ismert}
x:=i+1; y:=i+2; {a következő 2 atom: x, y}
Inc(i,2);
Dxy:=Span(x,y); { x és y távolságának lekérdezése}
{válasszunk 2 atomot [a,b,c] közül, hogy távolságuk különböző
legyen Dxy-től.}
If Dxy<>Abs(Pos[b]-Pos[c]) Then Begin {b-<>x--y}
  z:=a; a:=c; c:=z;
End Else If Dxy<>Abs(Pos[a]-Pos[c]) Then Begin{a-<>x--y}
  z:=b; b:=c; c:=z;
End; {Else a--b<>x--y }
Dab:=Abs(Pos[a]-Pos[b]);
If Pos[b]<Pos[a] Then Begin{ Pos[a]<Pos[b] biztosítása}
  z:=a; a:=b; b:=z;
End;

{Pos[a]<Pos[b], Dab<>Dxy}
Dax:=Span(a,x);
Dby:=Span(b,y);
{ x és y relatív pozíciójának meghatározása}
If (Dax=Dab+Dby+Dxy) Or {a--b--y--x}
  (Dax+Dxy=Dab+Dby) Then Begin {a--b--x--y}
  Pos[x]:=Pos[a]+Dax; {a--x--b--y}
  Pos[y]:=Pos[b]+Dby;
End Else If (Dab=Dax+Dxy+Dby) Or {a--x--y--b}
  (Dab+Dxy=Dax+Dby) Then Begin {a--y--b--x}
  Pos[x]:=Pos[a]+Dax; {a--y--x--b}
  Pos[y]:=Pos[b]-Dby; {y--a--b--x}
  {y--a--x--b}
End Else If (Dxy=Dab+Dax+Dby) Then Begin{x--a--b--y}
  Pos[x]:=Pos[a]-Dax;
  Pos[y]:=Pos[b]+Dby;
End Else If (Dby=Dxy+Dax+Dab) Or {y--x--a--b}
  (Dax+Dab=Dby+Dxy) Then Begin{x--y--a--b}
  Pos[x]:=Pos[a]-Dax; {x--a--y--b}
  Pos[y]:=Pos[b]-Dby;
End{If};
a:=x; b:=y; { a, b helyett x és y }
End{while};

If Not Odd(N) Then Begin{az utolsó elem feldolgozása, ha N páros}
  If Pos[b]<Pos[a] Then Begin
    z:=a; a:=b; b:=z;
  End;
  Dab:=Abs(Pos[a]-Pos[b]);
  Dax:=Span(a,N);
  Dbx:=Span(b,N);
  If Dax=Dab+Dbx Then
    Pos[N]:=Pos[b]+Dbx

```

```

Else If Dab=Dax+Dbx Then
  Pos[N]:=Pos[a]+Dax
Else
  Pos[N]:=Pos[a]-Dax
End;
min:=Pos[1];
For i:=2 To N Do
If Pos[i]<min Then min:=Pos[i];
min:=-min+1;
For i:=1 To N Do      {a relatív pozíciók 1..N tartományba léptetése}
  S[Pos[i]+min-1]:=i;
End{Compute};

Begin{program}
  N:=Size;
  Compute;
  For i:=0 To N-1 Do
    Answer(i+1,S[i]);
End.

```

7.5. Feladat: Median (IOI'2000)

Egy úrkísérletben n tárgyat használunk, melyeket 1-től n -ig számozunk, ahol n páratlan. Minden tárgy különböző súlyú (természetes számok), de magukat a súlyokat nem ismerjük. Minden y súlyra igaz, hogy $1 \leq y \leq n$. Mediánnak nevezzük azt a tárgyat, amelyiknél ugyanannyi könnyebb, mint nehezebb tárgy van. Írj programot, amely meghatározza a mediánt! A tárgyak súlyát olyan eszközzel hasonlíthatjuk össze, amely három tárgy közül megadja a mediánt.

Könyvtár

A device nevű könyvtárból az alábbi három művelet használható:

GetN egyszer kell meghívni, a programod legelején; az argumentum nélküli függvényhívás eredménye az n értéke.

Med3 három különböző tárgy sorszámaival kell hívni, függvényértéke e három sorszám közül a mediánjuk sorszáma.

Answer egyszer kell meghívni, a programod végén; argumentumként az N tárgy mediánjának a sorszámát kell megadni. Ez a hívás le is állítja a programodat.

A device könyvtár függvényei két szöveges állományt hoznak létre `MEDIAN.OUT` és `MEDIAN.LOG` néven. A `MEDIAN.OUT` első sorában egy egész szám lesz, az, amit az `ANSWER` eljárásnak adtál át. A második sorban a `MED3` hívások száma lesz. A programod és a könyvtár közötti párbeszédet a `MEDIAN.LOG` tartalmazza.

Pascal programozóknak:

programodba írd be a következő sort: uses device;

Kipróbálás

Programod kipróbálásához készíts `DEVICE.IN` néven olyan állományt, amely két sorból áll. Az elsőbe a tárgyak számát (n) kell írni. A második sor a tárgyak súlyát (1 és n közötti különböző egész számok) tartalmazza, ahol az i -edik érték az i -edik tárgy súlya.

Kikötések

- $5 \leq n \leq 1499$ és n páratlan.
- Minden i sorszáma igaz: $1 \leq i \leq n$.
- Minden y súlyra igaz: $1 \leq y \leq n$ és minden súly különböző.
- A Pascal könyvtár neve: device.tpu
- A Pascal függvények és eljárás deklarációja:


```
function GetN: integer;
function Med3(x,y,z:integer):integer;
procedure Answer(m:integer);
```
- Futtatásonként a `MED3` legfeljebb 7777-szer hívható.
- Programod nem olvashat és nem írhat egyetlen állományt sem.

Megoldások

1. Hagymahámzó algoritmus.

Alapelv: ismételten határozzuk meg és távolítsuk el a két szélső elemet, amíg egy elem marad, ez lesz megoldás.

A két szélső elem meghatározása:

begin

bal := 1; *jobb* := 2

for *x* := 3 **to** *n* **do**

if *Med3*(*bal*, *jobb*, *x*) = *bal* **then**

bal := *x*

else if *Med3*(*bal*, *jobb*, *x*) = *jobb* **then**

jobb := *x*

end

Program Median1; {Hagymahámzó algoritmus}

Uses Device;

Const MaxN=3000;

Type

 Node=1..MaxN;

Var

 N:Node; M:Integer;

Function Szamol:Integer; Var

 S:Array[1..MaxN] Of 0..MaxN;

 L,R,x,y,mi:Integer;

Begin{Compute};

 For x:=1 To N Do S[x]:=x;

 L:=1; R:=N;

 While L<R Do Begin

 For x:=L+1 To R-1 Do Begin

 mi:=Med3(S[L],S[x],S[R]);

 If mi=S[L] Then Begin

 y:=S[L]; S[L]:=S[x]; S[x]:=y;

 End Else If mi=S[R] Then Begin

 y:=S[R]; S[R]:=S[x]; S[x]:=y;

 End;

 End{for x};

 Inc(L); Dec(R);

 End{while};

 Compute:=S[L];

End{Szamol};

A két szélső elem $n - 2$ kérdéssel határozható meg, tehát a kérdések száma:

$$(n-2) + (n-4) + \dots + 3 + 1 = \left(\frac{n-1}{2}\right)^2$$

Ha $n \leq 177$, akkor legfeljebb 7744 hívás kell, de ha $n \geq 179$, akkor legalább 7921.

2. Teljes rendezés algoritmus.

3. Felét rendező algoritmus.

4. Szűkítő rendezés algoritmus.

begin

 Rendezzük sorba az $1, \dots, [n]$ elemeket;

for $[n] + 1$ **to** *n* **do begin**

 szűrjük be *x*-et a rendezett sorozatba harmadoló kereséssel;

 töröljük a sorozat első és utolsó elemét;

end

end

A sorozatnak ekkor egyetlen eleme lesz, ami a megoldás.

```

Program Median4; {Szűkítő rendezés }
Uses Device; Const
    MaxN=3000;           {az elemek max. száma}
Var
    N:Integer;           {az elemek száma}
    M:Integer;           {a megoldás}
    S:Array[1..MaxN] Of Integer;{a rendezett sorozat}

Function FindPos(L,R,X:Word):Word;
    { harmadoló kereséssel megkeresi x helyét az S[L..R] részsorozatban}
Var
    L0,R0,Lm,Rm,mi,d,Xm:Word;
Begin{FindPos}
    L0:=L; R0:=R;
    L:=L0-1; R:=R0+1;

    While (L+2<R) Do Begin
        d:=(R-L) Div 3;
        If (R-L) Mod 3=2 Then Begin {(d+1), d, (d+1) hosszú részek}
            Lm:=L+d+1; Rm:=R-(d+1);
        End Else Begin           {d, (d+1), d hosszú részek}
            Lm:=L+d; Rm:=R-d;
        End;
        Xm:=Med3(S[Lm], S[Rm], X);
        If Xm=S[Lm] Then
            R:=Lm
        Else If Xm=S[Rm] Then
            L:=Rm
        Else Begin
            L:=Lm; R:=Rm;
        End;
    End{while};

    If (R=L+2) Then Begin {extremális esetek}
        If L=0 Then L:=1;
        R:=L+1;
        Xm:=Med3(S[L], S[R], X);
        If Xm=S[L] Then
            L:=L-1
        Else If (Xm=S[R]) Then
            L:=R
    End;

    FindPos:=L;
End{FindPos};

Function Compute:Integer;
Var
    L,R,x,xp,i,Mi:Integer;
Begin{Compute};
    Mi:=N div 2+1;
    L:=1; R:=2;
    S[1]:=1; S[2]:=2;
    For x:=3 To Mi Do Begin {az 1..Mi elemek rendezése}
        xp:=FindPos(L,R,x);
        For i:=R Downto xp+1 Do {x beszúrása}
            S[i+1]:=S[i];
            S[xp+1]:=x;
        Inc(R);
    End{for};
    {S[1..Mi] rendezett }

```

```

For x:=Mi+1 To N Do Begin
{Invariáns: van N/2 elem S[L]-nél nagyobb és
      van N/2 elem S[R]-nél kisebb}
xp:=FindPos(L,R,x);
If (xp<L) Then           {a két szélső elem törlése}
  Dec(R)                 {bal oldalon lecsordult egy, a jobboldalit tör}
Else If (xp=R) Then
  Inc(L)                 {jobb oldalon lecsordult egy, a bal oldalt töro}
Else Begin               {x beszúrása [L..R]-be}
  For i:=R Downto xp+1 Do {lecsordulás }
    S[i+1]:=S[i];
  S[xp+1]:=x;
  Inc(L);                {bal oldali törlése}
End;
End{for};

Compute:=S[R];          {L=R}
End{Compute};

Begin{program}
N:=GetN;
M:=Compute;
Answer(M);
End.

```

4. Iterált harmadoló algoritmus.

Elvi algoritmus:

```

Function Keres(H:Halmaz; k:Integer):Integer;
{A H halmaz k-adik elemének keresése harmadoló felosztással }
Var H1,H2,H3:Halmaz;
    a,b:Integer;

Procedure Feloszt(H:Halmaz;
                  Var H1,H2,H3:Halmaz);

Var
    a,b:Integer;
Begin{Feloszt};
  Valaszt(H,a); Valaszt(H,b);
  Rendez(a,b); {a<b}
  H1:=[]; H2:=[];H3:=[];
  For x In H Do
    Case Med3(a,x,b) of
      a: H1:=H1+[x];
      x: H2:=H2+[x];
      b: H3:=H3+[x];
    End{Case};
  End{Feloszt};

Begin{Keres};
If |H|=1 Then Begin
  Keres:=H eleme
End Else If |H|=2 Then Begin
  Valaszt(H,a); Valaszt(H,b);
  Rendez(a,b); {a<b}
  If k=1 Then Keres:=a Else Keres:=b
End Else Begin
  Feloszt(H,H1,H2,H3,a,b);
  If k<=|H1| Then
    Keres:=Keres(H1,k)
  Else If k=|H1|+1 Then

```

```

    Keres:=a
Else If (k>|H1|+1) And (k<=|H1|+|H2|+1) Then
    Keres:=Keres(H2,k-|H1|-1)
Else If k=|H1|+|H2|+1 Then
    Keres:=b
Else
    Keres:=Keres(H3,k-(|H1|+|H2|+2))
End;
End{Keres};

Program Median5;
{Iterált harmadoló felosztás algoritmus}
Uses Device;
Const
    MaxN=3000;
Var
    N:Integer;
    M:Integer;{a megoldás}
    S:Array[1..MaxN] Of 0..MaxN;

Function Keres(K:Integer):Integer; Var Veg1,Veg2,Veg3:Integer;
    Bal,Jobb,a,b,i:Integer;
    H1,H2,H3:Integer;
    m1a,m1ab:Integer;

Procedure Feloszt(Bal,Jobb:Integer; Var V1,V2,V3,a,b:Integer);
Var i,X,m:Integer;
Begin{Feloszt};
    a:=S[Bal]; b:=S[Bal+1];
    If Bal>1 Then Begin{a~b rendezése 1->2 höz képest}
        m1a:=Med3(1,2,a);
        m1ab:=Med3(1,a,b);
        If (m1a=1) And (m1ab=a) Or (m1a<>1) And (m1ab<>a)
            Then Begin
                a:=S[Jobb]; b:=S[Bal]; S[Bal]:=a; S[Jobb]:=b
            End;
    End;
V1:=Bal+1; V2:=V1; V3:=V1;
For i:=Bal+2 To Jobb Do Begin
    X:=S[i];
    m:= Med3(a,X,b);
    If m=a Then Begin{x a H1-be megy}
        Inc(V1); Inc(V2); Inc(V3);
        S[V3]:=S[V2];
        S[V2]:=S[V1];
        S[V1]:=X
    End Else If m=X Then Begin{x a H2-be megy}
        Inc(V2); Inc(V3);
        S[V3]:=S[V2];
        S[V2]:=X
    End Else Begin{x a H3-be megy}
        Inc(V3);
    End;
End{for i};

End{Feloszt};

Begin{Keres};
For i:=1 To N Do S[i]:=i;
Bal:=1; Jobb:=N;
While Bal<Jobb Do Begin
    Feloszt(Bal, Jobb, Veg1,Veg2,Veg3, a,b);

```

```

H1:=Veg1-Bal-1; H2:=Veg3-Veg2; H3:=Veg3-Veg2;
If k<=H1 Then Begin
  Bal:=Bal+2; Jobb:=Veg1;
End Else If K=H1+1 Then Begin
  Keres:=a; Exit
End Else If (K>H1+1) And (K<=H1+H2+1) Then Begin
  K:=K-H1-1;
  Bal:=Veg1+1; Jobb:=Veg2;
End Else If k=H1+H2+1 Then Begin
  Keres:=b; Exit
End Else Begin
  K:=K-(H1+H2+2);
  Bal:=Veg2+1; {Jobb:=Veg3;}
End;
End{while};

Keres:=S[Bal];
End{Keres};

Begin{program}
  N:=GetN;
  M:=Keres(N Div 2+1);
  Answer(M);
End.

```