ÚJ SZÉCHENYI TERV

Ég és Föld vonzásában – a természet titkai

Informatikai tehetséggondozás:

Elemi programozási tételek 2

TÁMOP-4.2.3.-12/1/KONV

MAGYARORSZÁG MEGÚJUL

A projektek az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósulnak meg.

Feladataink egy jelentős csoportjában egyetlen bemenő sorozat alapján kell meghatároznunk egy értéket eredményként.

1. Eldöntés

Az előző programozási tétel két speciális esetében az ott közölt megoldás sok felesleges lépést tartalmazhat. Ez a tétel annak a hiányosságait pótolja e speciális esetekben. Az alábbi példákban keressük meg a közös vonást (ami speciális esetét jelenti az előzőnek)!

F1. Döntsük el egy számról, hogy prímszám-e!

F2. Döntsük el egy szóról a hónapnevek sorozata alapján, hogy egy hónap neve-e!

F3. Döntsük el egy tanuló év végi osztályzatai alapján, hogy kitűnő tanuló-e!

F4. Júniusban minden nap délben megmértük, hogy a Balaton Siófoknál hány fokos. Döntsük el a mérések alapján, hogy a víz hőfoka folyamatosan emelkedett-e!

E feladatok közös jellemzője, hogy a bemenetként megadott sorozathoz egy logikai értéket kell rendelni: a feltett kérdésre **igen**nel vagy **nem**mel kell válaszolni. A vizsgált sorozat elemei tetszőlegesek lehetnek, egyetlen jellemzőt kell feltételeznünk róluk: egy tetszőleges elemről el lehet dönteni, hogy rendelkezik-e egy bizonyos tulajdonsággal.

A feladatok így két csoportba sorolhatók, az egyikben azt kell eldönteni, hogy egy sorozatban létezik-e adott tulajdonságú elem, a másikban pedig azt, hogy mindegyik elem rendelkezik-e ezzel a tulajdonsággal. Vizsgáljuk először az első fajtájúakat!

Az algoritmus:

```
Függvény:
 T: Elemtípus → Logikai
Változók:
 N  : Egész
 [a feldolgozandó sorozat elemei száma]
 X  : Tömb(1..N:Elemtípus)       [a feldolgozandó sorozat]
 VAN: Logikai                                [az eredmény]
Eldöntés(N,X,S):
 S:=hamis
 Ciklus I=1-től N-ig
   S:=S ∨ X(I)
 Ciklus vége
Eljárás vége.
```

Észrevehetünk azonban egy fontos tulajdonságot. Ha a megoldásban az S változó értéke egyszer **igaz**ra változik, akkor a megoldás végéig biztosan az is marad. Tehát az ezutáni mű-

veletek elvégzése teljesen felesleges. Ezt a tulajdonságot kihasználva készíthetjük el az igazi megoldást.

Ebben az esetben egy olyan ciklus a megoldás, amely vagy akkor áll le, ha találtunk egy, a keresett tulajdonsággal rendelkező elemet, vagy pedig akkor, ha ilyen elem a sorozatban már nem létezhet, azaz elfogytak a megvizsgálandó elemek.

```
Eldöntés(N,X,VAN):
 I:=1
 Ciklus amíg I≤N és nem T(X(I))
   I:=I+1
 Ciklus vége
 VAN:=(I≤N)
Eljárás vége.
```

Fordítsuk most figyelmünket a másik csoportra! Azt, hogy mindegyik elem rendelkezik egy adott tulajdonsággal, átfogalmazhatjuk arra, hogy nem létezik az adott tulajdonsággal nem rendelkező elem. Ezek alapján a fenti megoldásban 2 helyen tagadást alkalmazva megkapjuk ennek a csoportnak a megoldástípusát is.

```
Eldöntés(N,X,MIND):
 I:=1
 Ciklus amíg I≤N és T(X(I))
   I:=I+1
 Ciklus vége
 MIND:=(I>N)
Eljárás vége.
```

Az eldöntés tipikusan olyan feladat, amelynél kódolási problémák merülhetnek fel. A programozási nyelvek egy jelentős részében a logikai kifejezések kiértékelésekor az **és,** illetve a **vagy** műveletek mindkét operandusát kiértékelik futáskor, még akkor is, ha az egyik operandus értéke alapján a végeredmény egyértelműen eldönthető lenne. E programozási tételnél így az I≤N feltétel nem teljesülése esetén is meg kell vizsgálni a T(X(I)) értékét. Ha a felhasznált tömb pontosan N elemű, akkor ez tömbindexelési hibához vezet. Többféle megoldással foglalkoztunk a kódolási kérdések kapcsán:

- vegyünk fel egy fiktív (N+1.) elemet a tömb végére,
- a ciklus egy lépéssel hamarabb álljon le, s vizsgáljuk a leállás okát,
- válasszuk szét a két részfeltétel kiértékelését, s a T(X(I))-t csak akkor értékeljük ki, amikor szükséges.

Néhány feladat megoldása következik.

F1. Létezik-e a számnak 1-től és önmagától különböző osztója?

```
elemek száma: N-2
osztók      : 2,3, ..., N-1
T(e): e|N
```

```
Eldöntés(N,PRIM):
  I:=2
  Ciklus amíg I≤N-1 és
                nem I|N
    I:=I+1
  Ciklus vége
  PRIM:=(I>N-1)
Eljárás vége.
```

F4. A hőmérsékletek sorozata monoton növekedő-e?

```
elemek száma : N-1
hőmérsékletek: X (N db elem)
T(e): X(e)≤X(e+1)
```

```
Eldöntés(N,X,MONOTON):
  I:=1
  Ciklus amíg I≤N-1 és
           X(I)≤X(I+1)
    I:=I+1
  Ciklus vége
  MONOTON:=(I>N-1)
Eljárás vége.
```

2. Kiválasztás

Ha kicsit belegondolunk, az előző programozási tétel megoldása a tőle vártnál több eredményt is adhat. Ennél a tételnél ezt a *több*et vizsgáljuk.

F5. Ismerjük egy hónap nevét, a hónapnevek sorozata alapján mondjuk meg a sorszámát!

F6. Adjuk meg egy természetes szám legkisebb, 1-től különböző osztóját!

F7. A naptárban található névnapok alapján adjuk meg legjobb barátunk (barátnőnk) névnapját! (Itt nem a „legjobbság” megfogalmazása okozza az algoritmikai problémát.)

Keressük itt is a közös jellemzőket! A feladattípus első ránézésre nagyon hasonlít az előzőre, csupán itt nem egy eldöntendő kérdésre kell válaszolni, hanem meg kell adni a sorozat egyik, adott tulajdonsággal rendelkező elemét. Kicsit részletesebb, pontosabb vizsgálódás után még az is kiderül, hogy ha e feladatokra az eldöntendő kérdést tennénk fel, akkor biztosan **igen** választ kapnánk.

Azt kell még eldöntenünk, hogy az elemet hogyan adhatjuk meg. Erre két lehetőségünk van: vagy a sorszámát, vagy pedig az értékét adjuk meg. Felhívjuk a figyelmet arra, hogy az előbbi változatból az utóbbi megoldását nagyon könnyen megkaphatjuk, fordítva viszont korántsem ez a helyzet. Emiatt mi az első változatot részesítjük előnyben, ez az általánosabb eset.

Az algoritmus:

Függvény:

T: Elemtípus → Logikai

```
Változók:
 N: Egész            [a feldolgozandó sorozat elemei száma]
 X: Tömb(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
 SORSZ: Egész                                  [az eredmény]
```

A megoldás sokban fog hasonlítani az előző feladattípus megoldásához, annak az esetnek a vizsgálatát kell kihagyni belőle, amely a keresett tulajdonságú elem nem létezése esetén állította le a megoldás keresését.

```
Kiválasztás(N,X,SORSZ):
 I:=1
 Ciklus amíg nem T(X(I))
   I:=I+1
 Ciklus vége
 SORSZ:=I
Eljárás vége.
```

A megoldásról könnyen megállapíthatjuk, hogy ha több, a kívánt tulajdonsággal rendelkező elem is előfordul a sorozatban, akkor azok közül az elsőt (helyesebben annak sorszámát) adja.

Könnyen átalakíthatjuk olyanra is, amely ilyenkor az utolsót adja közülük. Ekkor csupán annyi a teendő, hogy a sorozat elemeit hátulról visszafelé dolgozzuk fel.

Nézzük meg a bevezető feladatok egyikének megoldását!

F7. A legjobb barát (barátnő) névnapja.

```
elemek száma: N
névnapok: X (N db elem:
    Rekord(név,névnap) )
T(e): e.név=barát
```

```
Kiválasztás(N,X,BARÁT,NAP):
  I:=1
  Ciklus amíg X(I).név≠BARÁT
     I:=I+1
  Ciklus vége
  NAP:=X(I).névnap
Eljárás vége.
```

3. (Lineáris) keresés

Foglaljuk össze az előző két programozási tétel feladatát: az új feladattípusban a feltett kérdés az előző kettő mindegyikét tartalmazza!

F8. Ismerjük egy üzlet egy havi forgalmát: minden napra megadjuk, hogy mennyi volt a bevétel és mennyi a kiadás. Adjunk meg egy olyan napot – ha van –, amelyik nem volt nyereséges!

F9. A Budapest-Nagykanizsa vasúti menetrend alapján két adott állomáshoz adjunk meg egy olyan vonatot, amellyel el lehet jutni átszállás nélkül az egyikről a másikra!

F10. Egy tetszőleges (nem 1) természetes számnak adjuk meg egy osztóját, ami nem az 1 és nem is önmaga.

Tehát a közös jellemző, hogy egy adott tulajdonságú elemet kell megadnunk, ha egyáltalán van ilyen. Ha nincs, akkor a válasznak ezt a tényt kell tartalmaznia.

Az algoritmus:

```
Függvény:
 T: Elemtípus → Logikai
Változók:
 N: Egész              [a feldolgozandó sorozat elemei száma]
 X: Tömb(1..N:Elemtípus)  [a feldolgozandó sorozat elemei]
 VAN: Logikai          [az eredmény - van-e megfelelő elem]
 SORSZ: Egész     [az eredmény - a megfelelő elem sorszáma]
```

Vegyük az *eldöntés* algoritmusát, s egészítsük ki a *kiválasztás*nak megfelelő eredménnyel!

```
Keresés(N,X,VAN,SORSZ):
 I:=1
 Ciklus amíg I≤N és nem T(X(I))
   I:=I+1
 Ciklus vége
 VAN:=(I≤N)
 Ha VAN akkor SORSZ:=I
Eljárás vége.
```

Az egyik kitűzött feladat megoldása a következő lehet:

F8. Nem nyereséges nap megadása.

```
elemek száma: N
bevételek: B (N db elem)
kiadások: K (N db elem)
T(k-b): k-b≥0
```

```
Keresés(N,X,VAN,NAP):
  I:=1
  Ciklus amíg I≤N és
                K(I)−B(I)<0
     I:=I+1
  Ciklus vége
  VAN:=(I≤N)
  Ha VAN akkor NAP:=I
Eljárás vége.
```

Feladatatok programozási tételekre a Nemes Tihamér OITV-ről és az Informatika OKTV-ről

1. feladat

Egy vizsgabizottságban egy nap feljegyezték, hogy az egyes vizsgázók mikor vizsgáztak (egyszerre egy vizsgázó lehet), mindenkiről 4 adatot tudunk: `kezdőóra, kezdőperc, végóra, végperc.` A vizsgázók adatait idő szerinti sorrendben kapjuk.

Készíts programot (`vizsga.pas`,…), amely beolvassa a vizsgázók számát ($1 \le N \le 100$), az egyes vizsgázók vizsgájának kezdetét (v[i,1]) és végét (v[i,2]), majd megadja

A. annak a 60 perces időszaknak a kezdetét, amikor a legtöbb vizsgázó végzett a vizsgájával (közülük az első pontosan ebben a percben végezzen);
B. a leghosszabb vizsgaszünet hosszát – óra, perc (amikor 2 vizsgázó között senki sem volt a vizsgabizottságnál);
C. a leghosszabb időtartamot, amikor a vizsgabizottság nem tarthatott szünetet!

Példa

Bemenet:	Kimenet:	
N=6	10 10	(10 óra 10 perctől)
8 20 8 30	1 0	(1 óra, 0 perc)
8 50 9 0	1 5	(1 óra, 5 perc)
9 50 10 10		
10 10 10 30		
10 30 10 55		
11 55 12 10		

Az első fontos gondolat: beolvasáskor azonnal mindent számoljunk át percekbe, azzal könnyebb lesz dolgozni! Eredmény kiírásnál pedig számoljuk vissza órákra és percekre!

A. Keresünk egy olyan vizsgázót és az után 60 percen belül végzőt, amelyek sorszáma különbsége maximális;
B. NEM ILYEN TÉTEL;
C. NEM ILYEN TÉTEL!

```
A:
  i:=1; k:=végzett(i)
  Ciklus j=2-től n-ig
    Ha végzett(j)>k akkor i:=j; k:=végzett(j)
  Ciklus vége
Eljárás vége.
```

```
Végzett(j):
  v:=j+1
  Ciklus amíg v≤N és vég(v)-vég(j)≤60
    v:=v+1
  Ciklus vége
  végzett:=v-j
Függvény vége.
```

2. feladat

Egy titkosírás elkészítéséhez a következő táblázatot használjuk:

	a	b	c	d	e	f	g	h	i
a	i	o	q	h	b	f	y	l	w
b	n	r	a	g	s	k	t	e	z
c	d	u	p	x	c	j	v	m	

Egy szót ezzel a táblázattal úgy titkosítunk, hogy a betűit egyesével megkeressük a táblázat belsejében és a titkosított szövegbe a helyére a betű sorában, illetve oszlopában levő betűpárt tesszük. Feltehetjük, hogy csak az angol ábécé kisbetűit használjuk.

Példa:

```
A titkosítandó szó: balaton
A titkosított szó: aebcahbcbgabba
```

Készíts programot (`titkos.pas`, `titkos.c`, …), amely két funkcióra képes:

A. beolvas egy szót és kiírja titkosítva;

B. beolvas egy titkosított szót és kiírja a visszafejtését!

A titkosítás egy kiválasztás programozási tétel, mátrixra. A visszafejtést csak a teljesség kedvéért tesszük ide, egyetlen tömbelem hivatkozással megoldható.

```
Titkosítás(a,b):
  b:=''
  Ciklus i=1-től hossz(a)-ig
    j:='a'; k:='a'
    Ciklus amíg t(j,k)≠a(i)
      Ha k='i' akkor j:=j+1; k:='a'
            különben k:=k+1
    Ciklus vége
    b:=b+j+k;
  Ciklus vége
Eljárás vége.
```

```
Visszafejtés(b,a):
  a:=''; i:=1
  Ciklus amíg i<hossz(b)
    a:=a+t(b(i),b(i+1)); i:=i+2;
  Ciklus vége
Eljárás vége.
```

3. feladat

Az időjárás előrejelzésben ismerjük előre N (2≤N≤100) nap várható minimális és maximális hőmérsékletét. Készíts programot (`idojaras.pas`, `idojaras.c`, …), amely beolvassa N értékét és a 2*N db hőmérsékletet, majd megadja:

A. azt a K napos időtartamot (ha van), amelyben az előrejelzés szerint folyamatosan fagy lesz;

B. azt a két szomszédos napot, ahol a legnagyobbat változik a hőmérséklet;

C. azokat a napokat, ahol a napi minimális hőmérséklet a napi átlaghőmérsékletek átlaga fölötti!

Példa:

```
Bemenet:
Napok száma?5
Fagy hány napon keresztül?2
1. nap minimuma, maximuma: -9 -2
2. nap minimuma, maximuma: -1 4
3. nap minimuma, maximuma: -5 -4
4. nap minimuma, maximuma: -6,-1
5. nap minimuma, maximuma: 5 8
```

```
Kimenet:
Folyamatos fagy: 3 4
Legnagyobb változás: 4 5
Átlag fölöttiek: 2 5
```

A. olyan fagyott napot kell keresni, amitől kezdődően legalább K napig fagy van;

B. NEM ILYEN TÉTEL

C. NEM ILYEN TÉTEL

```
A(van,i):
  i:=1
  Ciklus amíg i≤n-k+1 és nem(h(i).max<0 és jó(i,k))
    i:=i+1
  Ciklus vége
  van:=(i≤n-k+1)
Eljárás vége.

jó(i,k):
  j:=i+1
  Ciklus amíg j-i<k és h(j).max<0
    j:=j+1
  Ciklus vége
  Ha j-i=k akkor jó:=igaz különben jó:=hamis; i:=j
Eljárás vége.
```

Megjegyzés: a `jó` eljárás gyorsíthat, mert az `i` értékét növelheti.

4. feladat

Ugyanarról a területről két időpontban készítettünk fényképet. A fényképek négy széléről le szeretnénk vágni azt a részt, amelyek egyformák.

Készíts programot (`kep.pas`, …), amely megadja hogy a kép 4 széléről maximum mekkora téglalapok vághatók le!

A `kep.be` szöveges állomány első sorában a fényképek sorai és oszlopai száma van ($1 \le N,M \le 1000$), egy szóközzel elválasztva. A következő N sorban az első kép, az azt követő N sorban a második kép képpontjai vannak. Minden sor M képpont leírását tartalmazza, egymástól egy-egy szóközzel elválasztva. A képpontokat egy 0 és 255 közötti fényességértékkel adjuk meg.

A `kep.ki` szöveges állomány első sorába a legnagyobb balról, alulról, jobbról, illetve felülről levágható téglalap szélességét kell írni!

Példa:

`kep.be`

```
8 10
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 3 3 3 3 3
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 5 5 5 5
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 3 3 3 3 3
2 2 9 9 2 2 2 2 2 2
2 2 2 2 2 2 5 5 5 5
1 1 1 1 1 1 1 1 1 1
1 3 1 1 3 1 1 1 1 1
1 1 1 1 1 1 5 1 1 1
0 0 0 0 0 0 0 0 0 0
```

`kep.ki`

```
1 1 3 2
```

Meg kell keresni felülről lefelé, azon belül pedig balról jobbra az első eltérést! Ha ilyen van, akkor ki kell választani az első eltérést:

- balról jobbra, azon belül felülről lefelé;
- alulról felfelé, azon belül jobbról balra;
- jobbról balra, azon belül alulról felfelé.

```
Eltérés(van,bfs,bfo,jas,jao):
  i:=1; j:=1
  Ciklus amíg i≤n és t[i,j]=u[i,j]
    Ha j<m akkor j:=j+1 különben i:=i+1; j:=1
  Ciklus vége
  van:=i≤n
  Ha van akkor
                bfs:=i
                i:=1; j:=1
                Ciklus amíg t[i,j]=u[i,j]
                  Ha i<n akkor i:=i+1 különben j:=j+1; i:=1
                Ciklus vége
                bfo:=j
                i:=n; j:=m
                Ciklus amíg t[i,j]=u[i,j]
                  Ha j>1 akkor j:=j-1 különben i:=i-1; j:=m
                Ciklus vége
                jas:=i
                i:=n; j:=m
                Ciklus amíg t[i,j]=u[i,j]
                  Ha i>1 akkor i:=i-1 különben j:=j-1; i:=n
                Ciklus vége
                jao:=j
  Elágazás vége
Eljárás vége.
```

5. feladat

Egy játéktáblán a 0. időegységben L bábu van. Mindegyiket elindítjuk valamerre. Egy időegység alatt mindegyik a neki megfelelő távolságra mozdul el, a tábla széléről visszafordulnak. Lehetséges, hogy előbb-utóbb két bábu összeütközik: ugyanarra a helyre lépnének vagy átlépnének egymáson.

Írj programot (`babu.pas`, …), amely megadja, hogy K időegységen belül mikor ütközik legelőször két bábu!

A program olvassa be a tábla szélességét ($1 \le N \le 100$), a bábuk számát ($1 \le L \le 10$) és az időtartamot ($1 \le K \le 100\,000$)! Ezután olvassa be a bábuk kezdő helyét ($1 \le S_i \le N$) és mozgás irányát ($X_i, \in \{J, B\}$ – jobbra, balra)!

A program írja ki az első ütközés időpontját! Ha K időegységen belül nincs ütközés, akkor -1-et kell kiírni!

Példa:

```
A tábla hossza: 10
A bábuk száma: 2
```

```
Az időtartam: 10
1 bábu helye: 3, iránya: B
2. bábu helye: 8, iránya: B
Ütközés időpont: 5
```

A megoldás egy keresési tétel alkalmazása: keresünk K időegységen belül olyan esetet, amikor két bábu ütközik. A keresést azonban könnyű megfogalmazni, ha időegységenként szimuláljuk a két bábu mozgását. A szimulációt úgy egyszerű végrehajtani, ha külön tároljuk minden időegység előtti (t) és utáni (u) játéktábla állapotot is.

```
Bábuk(ütközés,idő):
  idő:=0; ütközés:=hamis; u:=t;
  Ciklus amíg idő≤k és nem ütközés
    idő:=idő+1
    Ciklus i=1-től L-ig
      o:=b(i).hely
      Ha o=1 és b(i).irány='B' akkor b(i).irány:='J'
      Ha o=n és b(i).irány='J' akkor b(i).irány:='B'
      Ha B(i).irány='J'
         akkor Ha u(o+1)>0) vagy
                  t(o+1)<t(o) és b(t(o+1)).irány='B'
                         akkor ütközés:=igaz
               u(o):=0; u(o+1):=i; b(i).hely:=b(i).hely+1;
      különben ha B(i).irány='B'
         akkor Ha u(o-1)>0) vagy
                  t(o-1)<t(o) és b(t(o-1)).irány='J'
                         akkor ütközés:=igaz
               u(o):=0; u(o-1):=i; b(i).hely:=b(i).hely-1;
      Elágazás vége
    Ciklus vége
    t:=u;
  Ciklus vége
Eljárás vége.
```

6. feladat

Egy magyar szó magas hangrendű, ha csak magas magánhangzók (e, é, i, í, ö, ő, ü, ű) vannak benne. Mély hangrendű, ha csak mély magánhangzókat (a, á, o, ó, u, ú) tartalmaz. Vegyes hangrendű pedig akkor, ha van benne magas és mély hangrendű magánhangzó is.

Írj programot (hangrend.pas, …), amely beolvas egy magyar szót, majd kiírja, hogy milyen hangrendű!

Példa:

```
Bemenet: almafa     Kimenet: mély hangrendű
```

A kapott szó betűit kell vizsgálni! El kell dönteni a szóról, hogy van-e benne magas hangrendű magánhangzó, illetve van-e benne mély hangrendű magánhangzó! Ezután a kérdés egyszerűen megválaszolható.

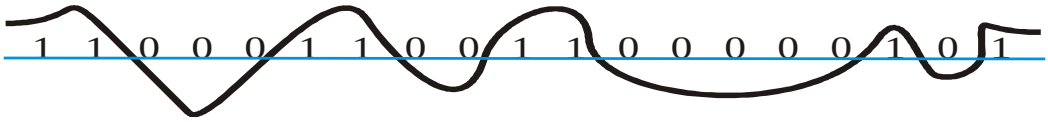
```
Hangrend(szó,h):
  magas:=hamis; mély:=hamis
  Ciklus i=1-től hossz(szó)-ig
    Ha eleme(szó(i),'aáoóuú') akkor mély:=igaz
    különben ha eleme(szó(i),'eéiíöőüű') akkor magas:=igaz
  Ciklus vége
  Ha magas és mély akkor h:="Vegyes"
  különben ha magas akkor h:="Magas"
  különben ha mély akkor h:="Mély"
Eljárás vége.

eleme(betű,szöveg):
  i:=1
  Ciklus amíg i≤hossz(szöveg) és betű≠szöveg(i)
    i:=i+1
  Ciklus vége
  eleme:=(i≤hossz(szöveg))
Függvény vége.
```

7. feladat

Régen a polinéz szigetvilágot az őslakók szigetről szigetre hajózva népesítették be. Egyes hajókkal csak a közelebbi szigetekre tudtak eljutni, másokkal távolabbra is.

A feladatban a szigetek egy vonalban, egymás mellett helyezkednek el az ábrának megfelelően (1-esek jelzik a szárazföldet, 0-k pedig a tengert):



Bal, illetve jobb oldalról indulhatnak hajók, amely a szigetek között legfeljebb T darab 0-val leírt távolságot tudnak megtenni.

Készíts programot (*mozaik.pas*, *mozaik.c*, …), amely az N adat (1≤N≤1000), a T távolság (1≤T≤100) és a szigetek elhelyezkedése ismeretében megadja, hogy az őslakók hány szigetet tudnak benépesíteni (balról, illetve jobbról indulva)!

Példa:

```
N=19, T=3, szigetek= 1 1 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 1
Eredmény: 3 sziget népesíthető be balról, 2 jobbról.
```

Első lépésként határozzuk meg a szigetek közötti távolságokat, majd elölről és hátulról indulva is addig menjünk, amíg a távolságok nem nagyobbak T-nél!

```
Sziget(N,érték,T,dbb,dbj)
  j:=1; föld:=igaz; s(j):=0
  Ciklus i=1-től N-ig
    Ha érték(i)=0 akkor s(j):=s(j)+1; föld:=hamis
    különben ha érték(i)=1 akkor
                Ha nem föld akkor j:=j+1; s(j):=0
                                  föld:=igaz
  Ciklus vége
  j:=j-1; dbb:=1
  Ciklus amíg dbb≤j és s(dbb)≤T
    dbb:=dbb+1
  Ciklus vége
  dbj:=j
  Ciklus amíg dbj≥1 és s(dbj)≤T
    dbj:=dbj-1
  Ciklus vége
  dbj:=j-dbj+1
Eljárás vége.
```

8. feladat

A bűvös négyzetek N*N számot tartalmaznak, négyzetes elrendezésben. Minden bűvös négyzetre igaz, hogy minden sor és minden oszlop összege is ugyanaz a szám. Kaptunk egy bűvös négyzetet, amiben lehet, hogy egyetlen számot elrontottak.

Írj programot (BUVOS.PAS, BUVOS.C, …), amely beolvassa a bűvös négyzet méretét (3≤N≤5) majd a bűvös négyzet számait, majd megadja, hogy a bűvös négyzet helyesen van-e kitöltve! Ha nem helyes, akkor azt is meg kell adnia, hogy hol van a hiba és milyen számot kellene oda írni!

Példa:

```
N=3            ⇒     Hibás a bűvös négyzet
2 3 2                Hibás hely: 2. sor 1. oszlop
3 1 4                Helyes értéke: 2
3 3 1
```

Ha minden sor és oszlop összege egyforma, akkor a bűvös négyzet jó. Ha az i-edik sor és a j-edik oszlop összege nem azonos a többiekével, akkor az (i,j) indexű hely a hibás.

```
Bűvös(N,b,jó,i,j,érték):
  Ciklus i=1-től N-ig
    s(i):=0; o(i):=0
    Ciklus j=1-től N-ig
      s(i):=s(i)+b(i,j); o(i):=s(i)+b(j,i)
    Ciklus vége
  Ciklus vége
  Ha s(1)=s(2) akkor k:=s(1) különben k:=s(3)
  i:=1
  Ciklus amíg i≤N és s(i)=k
    i:=i+1
  Ciklus vége
  jó:=i>N
  Ha i≤N akkor j:=1
               Ciklus amíg o(j)=k
                 j:=j+1
               Ciklus vége
               érték:=k-s(i]+b(i,j]
Eljárás vége.
```