NEMES TIHAMÉR
PROGRAMOZÓ VERSENY
(OKTV)
II. FORDULÓ

KÉP 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 sorainak és oszlopainak 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 kiírni!

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
```

Megjegyzés a feladat megoldása előtt:

Ezen a szép versenyfeladaton első pillantásra talán nem is látszik, milyen fontos szegmensét fedi le életünknek. A feladatban meghatározott elv ugyanis lehetővé teszi műholdak által készített nagyfelbontású képek gépi elemzését, a képek azon parányi részterületeinek automatikus kiválasztását, melyeken az esetleg néhány perccel korábban készült felvételekhez képest megváltozott a Földfelszín, elmozdultak a tereptárgyak (pl. földcsuszamlás).

A feladat megoldása:

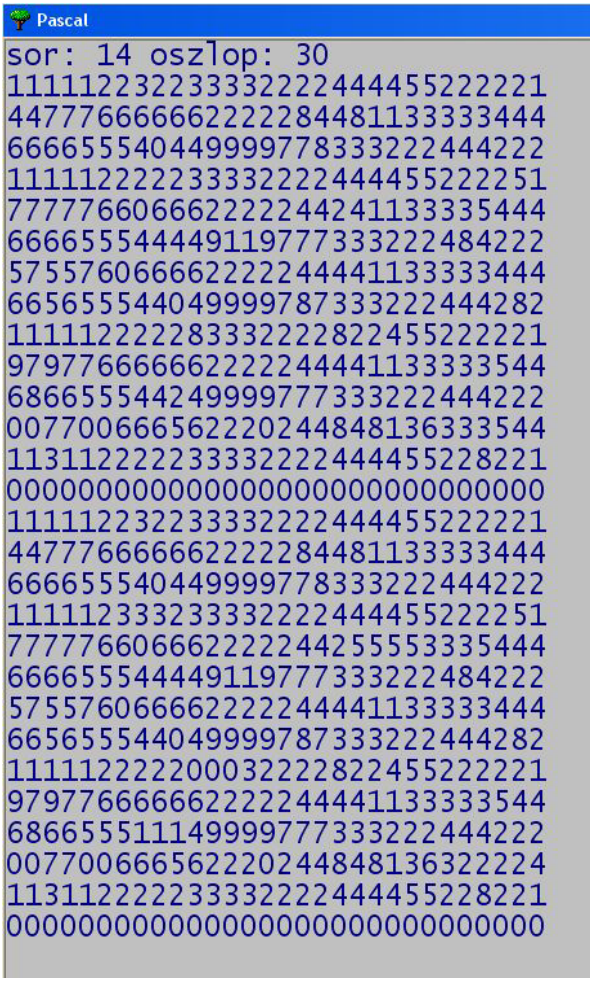
A megoldást szolgáltató kódot fokozatosan építjük fel.

Programunk első változatában mindössze arra törekszünk, hogy hibátlanul be tudjuk olvasni az adatokat a *kepbe.txt* külső szöveges minta állományból. Egyelőre megelégszünk 100 x 100 pixeles képek feldolgozásával. A zöld színnel jelölt sorokban lévő kód ellenőrzési célokat szolgál: visszaíratjuk a beolvasott állományt a képernyőre.

```
program kep_1;
uses crt;

var
i,j,sor,oszlop:byte;
f              :text;
adat           :array [1..100,1..100] of byte;

begin
textbackground(lightgray);
textcolor(blue);
clrscr;
assign(f,'kepbe.txt');
reset(f);
read(f,sor,oszlop);
for i:=1 to (2*sor) do
for j:=1 to oszlop do read(f,adat[i,j] );
close(f);
writeln('sor: ',sor,' oszlop: ',oszlop);
for i:=1 to (2*sor) do
for j:=1 to oszlop do
   begin
   gotoxy(j,i+1);
   write(adat[i,j],' ');
   end;
readln;
end.
```

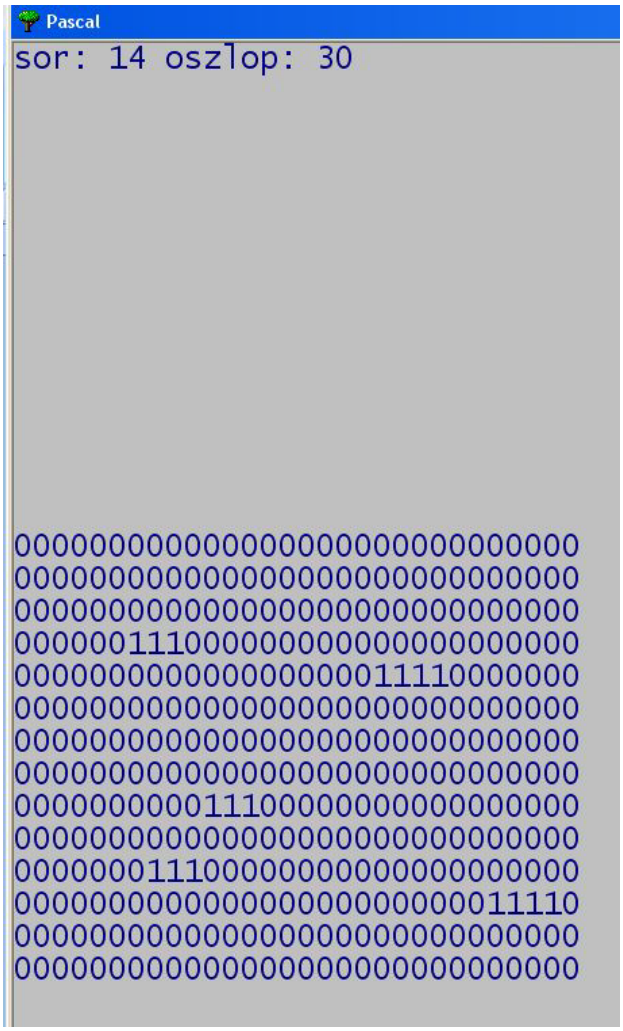

Pascal
sor: 14 oszlop: 30
11111223223333222244445522222 1
44777666666222228448113333344 4
66665554044999977833322244422 2
11111222223333222244445522225 1
77777660666222224424113333544 4
66665554444911977733322248422 2
57557606666222224444113333344 4
66565554404999978733322244428 2
11111222228333222282245522222 1
97977666666222224444113333544 4
68665554424999977733322244422 2
00770066656222024484813633354 4
11311222223333222244445522822 1
00000000000000000000000000000 0
11111223223333222244445522222 1
44777666666222228448113333344 4
66665554044999977833322244422 2
11111233323333222244445522225 1
77777660666222224425555333544 4
66665554444911977733322248422 2
57557606666222224444113333344 4
66565554404999978733322244428 2
11111222220003222282245522222 1
97977666666222224444113333544 4
68665551114999977733322244422 2
00770066656222024484813632222 4
11311222223333222244445522822 1
00000000000000000000000000000 0
```
```

A második változatban a beolvasási procedúrát két részre bontjuk. A második kép pontértékeinek beolvasásakor mindjárt össze is hasonlítjuk az értékeket a megfelelő első képbeli értékekkel. Ha egyezést találunk 0 értéket, egyébként 1 értéket írunk az adatmátrix második részébe, a megadott értékek helyett. Így tiszta, áttekinthető képet kapunk a nem-egyezési helyekről. Egyúttal a megoldást is le tudjuk olvasni (6 2 1 3), legalábbis ebben a kisméretű mintamátrixban.

```
program kep_2;
uses crt;

var
i,j,sor,oszlop:byte;
f              :text;
adat           :array [1..100,1..100] of byte;

begin
textbackground(lightgray);
textcolor(blue);
clrscr;
assign(f,'kepbe.txt');
reset(f);
read(f,sor,oszlop);
for i:=1 to sor do
for j:=1 to oszlop do read(f,adat[i,j] );
for i:=sor+1 to (2*sor) do
for j:=1 to oszlop do
   begin
   read(f,adat[i,j] );
   if adat[i,j]=adat[i-sor,j]
      then adat[i,j]:=0
      else adat[i,j]:=1;
   end;
close(f);
writeln('sor: ',sor,' oszlop: ',oszlop);
for i:=sor+1 to (2*sor) do
for j:=1 to oszlop do
   begin
   gotoxy(j,i+1);
   write(adat[i,j],' ');
   end;
readln;
end.
```

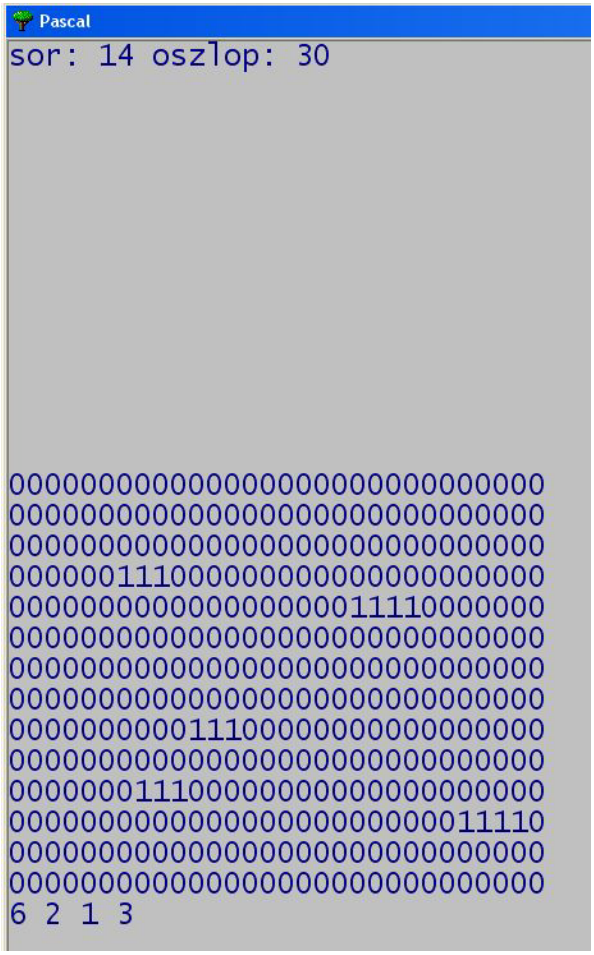


Harmadik fejlesztési lépésünk, hogy az összehasonlítások során feljegyezzük az adatmátrix azon elemeinek indexeit, ahol a képpont értéke megváltozott az eredetihez képest. Pontosabban alkalmazzuk a minimum és maximum kiválasztási tételt, külön a sorok és külön az oszlopok indexeire. Végül is kinyerjük a változási helyek indexeinek szélsőértékeit.

```
program kep_3;
uses crt;

var
i,j,sor,oszlop          :byte;
f                       :text;
adat                    :array [1..100,1..100] of byte;
smin,smax,omin,omax:integer;

begin
textbackground(lightgray);
textcolor(blue);
clrscr;
smin:=101;omin:=101;smax:=0;omax:=0;
assign(f,'kepbe.txt');
reset(f);
read(f,sor,oszlop);
for i:=1 to sor do
for j:=1 to oszlop do read(f,adat[i,j] );
for i:=sor+1 to (2*sor) do
for j:=1 to oszlop do
   begin
   read(f,adat[i,j] );
   if adat[i,j]=adat[i-sor,j]
      then adat[i,j]:=0
      else  begin
            adat[i,j]:=1;
            if i<smin then smin:=i;
            if i>smax then smax:=i;
            if j<omin then omin:=j;
            if j>omax then omax:=j;
            end;
   end;
close(f);
writeln('sor: ',sor,' oszlop: ',oszlop);
for i:=sor+1 to (2*sor) do
for j:=1 to oszlop do
   begin
   gotoxy(j,i);
   write(adat[i,j],' ');
   end;
write(omin-1,' ',2*sor-smax,' ',oszlop-omax,' ',smin-sor-1);
readln;
end.
```



Végezetül beillesztjük programunk végére a szöveges kimeneti állomány elkészítéséért felelős kódrészletet.

```
program kep_4;
uses crt;

var
i,j,sor,oszlop          :byte;
f,g                     :text;
adat                    :array [1..100,1..100] of byte;
smin,smax,omin,omax:integer;

begin
textbackground(lightgray);

        ...

close(f);
writeln('sor: ',sor,' oszlop: ',oszlop);

        ...

readln;
rewrite(g);
write(g,omin-1,' ',2*sor-smax,' ',oszlop-omax,' ',smin-sor-1);
close(g);
end.
```

A megoldás folytatása:

A feladat eredeti megfogalmazása szerint nagyméretű, akár 1000 x 1000 pixel méretű képállományok feldolgozását is meg kell tudnunk oldani! (Gondoljunk a NASA által a Világhálón nyilvánosságra hozott 4 000 x 10 000 pixeles nagyfelbontású képekre, melyeket űrszondájuk a Mars felszínéről közvetlen közelről készített.) Az eddigi megoldási változatokban a kétdimenziós tömbként definiált adatmátrix változónkra (*adat[1..100,1..100] of byte*) legfeljebb 100 x 100 pixeles képek feldolgozását bízhattuk, pedig változónk mérete már ekkor is 10 000 byte volt. Ha a feladat feltételeinek megfelelő 1 000 x 1 000-es (vagy még ennél is nagyobb) mátrix változót szeretnénk létrehozni, azt a fordító 16 bites címsínnel rendelkező processzort használva egyszerűen nem engedi. Ekkor ugyanis a maximális adatméret csak 64 kbyte lehet. (hibaüzenet: *Structure too large!*) Ha 32 bites processzorral dolgozunk és emuláljuk a Pascalt, az operációs rendszer a futtatókörnyezethez „atyai” módon, tudtunk nélkül további memóriaterületeket rendel, így az előbb említett hibaüzenetet – legalábbis egy viszonylag tág határig – nem tudjuk kikényszeríteni. Mindazonáltal továbbra is elvi kérdés marad, mit kezdhetünk óriás méretű állományokkal, hiszen nincs az a memória méret, melyhez ne lehetne olyan feladatot találni, amellyel ne tudnánk túlcsordulást előidézni. (Szimuláljuk pl. a több ezer milliárd csillagot tartalmazó galaxis halmazok klasztereinek mozgását a sötét anyag, az úgynevezett „Nagy Mozgató” hatása alatt – lásd *Lainakea szuperklaszter* szimuláció - 2014)

Négyzetes mátrixszal számolva, és feltételezve, hogy a megengedett foglalási határ a maximális igény n-ed része, eddig megírt programunk a maximálisan lehetséges képméretnek csak $1/n^2$ részét tudja ”szkennelni”. Egy ilyen részletekben történő vizsgálat a megfelelő indexek kinyerését jelentősen bonyolítaná. Megoldásunk egy további problémája, hogy a második kép mátrixában, amelyben a 0-1 konverziót megvalósítottuk, a sorindexeken y-irányú transzformációt kellett végrehajtani a helyes konvex burok kinyeréséhez. (A matematikában a mátrix elemek első indexe a sorindex, a második az oszlopindex. A számítógépes grafikai alkalmazásokban azonban a karakterek illetve pixelek első koordinátája az x koordináta ugyan, de ez a képernyőelem oszlop koordinátáját, míg a második y koordináta a sor koordinátáját adja meg, ráadásul felülről lefelé növekedve. Ezért kellett a kiíratásnál megcserélni az i és j koordinátaváltozókat.)

Óriás képek:

Ha figyelembe akarjuk venni az adatmátrix méretére vonatkozó, esetleg a fizikai valóságban is létező korlátozást (64 kbyte), kísérletezhetünk négyzetes mátrix helyett (250x250) téglalap mátrixszal (1000x50). Ekkor ugyan mentesülünk az index tartományok változásának x irányú nyomon követésétől. azonban előre olvasási problémába ütközünk. Az első sor képpontjainak esetleges változását csak akkor tudjuk megállapítani, ha ismerjük az 1001-edik sor adatait, ezek azonban még nincsenek beolvasva. Eljutva az 1001-edik sorig, viszont már rég el kellett dobnunk az első sor adatait, hiszen nincs memóriakapacitásunk annak őrzésére. Ekkor tehát csak egyet tehetünk: mivel a szöveges állományokban nem lehet sorokat indexelni, azaz nem tudjuk közvetlenül elérni az 1001-edik sort, az első sor olvasása után végig kell olvasnunk 999 sort feleslegesen, „üres” beolvasással, eldobva az olvasott értékeket, hogy a fájlmutató a megfelelő sorra érkezzen. Az összehasonlítások elvégzése után a *reset(f)* paranccsal a fájl elejére állunk és üres olvasással a fel nem dolgozott második sorra pozícionálunk, majd újabb 999 sor üres olvasásával eljutunk az 1002-edik sorhoz, és így tovább. Ezt ciklikusan további 998-szor ismételve célhoz érünk. Ez azonban, mondhatni, kíméletlen megoldás. Nem kíméljük ugyanis winchesterünk író-olvasó fejét. Próbálkozzunk tehát a következő megoldással: vágjuk ketté állományunkat fizikailag. Olvassuk be az első ezer sort egyszer és dobjuk el az olvasott értékeket, majd folytatva az állomány olvasását az 1001-edik sortól, soronként írjuk új állományba (pl.: kepbe2.txt) a második kép adatait. Ezután egyszerre megnyitva és egyetlen ciklusban olvasva a két képállomány azonos sorait, alkalmazzuk a konvex burok kinyerésére említett megoldási ötletünket. Elég egy egysoros ezerelemű tömböt foglalni és az y irányú transzformációtól is mentesülni fogunk. Sőt, ha jobban belegondolunk, elegendő mindössze két darab byte típusú változót foglalnunk (adat1, adat2), hiszen a két kép egy-egy azonos indexű elemét kell csak az összehasonlítás idejére a memóriában tartanunk. Egymillió byte helyett tehát csak két byte-ot foglalunk. Eltekinthetünk a 0-1 konverziótól is. Az óriás állomány adatait sem írjuk vissza a képernyőre:

```
program kep_5;
uses crt;

var
i,j,sor,oszlop              :byte;
f,g                         :text;
adat1,adat2                 :byte;
smin,smax,omin,omax:integer;

begin
assign(g,'kepbe2.txt');
rewrite(g);
assign(f,'kepbe.txt');
reset(f);
read(f,sor,oszlop);
for i:=1 to sor do
for j:=1 to oszlop do read(f,adat1);
for i:=sor+1 to (2*sor) do
for j:=1 to oszlop do
   begin
   read(f,adat1);
   if j=oszlop then write(g,adat1,chr(13),chr(10))  // ¶ elhelyezése a sor végén
               else write(g,adat1,' ');
   end;
close(f);
close(g);
textbackground(lightgray);
   ...
close(f);
{ }
rewrite(g);
write(g,omin-1,' ',2*sor-smax,' ',oszlop-omax,' ',smin-sor-1);
close(g);
end.
```

Tesztelés

Teszteljük programunkat különböző állományokkal. Létrehozhatunk ilyen teszt állományokat például wordpad szövegszerkesztővel is. Ügyeljünk a specifikáció betartására: az első sorban két szám állhat, szóközzel elválasztva, majd ezt kövessék a pixelek fényesség adatai. A teszt állományok akkor jók, ha lefedik a feladattér szingularitásait. Egy absztrakt tér egy elemét szinguláris nak nevezzük, ha ott a tér elemein értelmezett operátor viselkedése lényegesen eltér a közeli elemeken végzett operációs viselkedéshez képest.(Az 1/x függvénynek például szinguláris pontja az x = 0 érték.) Ezért a következő tesztállományokat célszerű elkészíteni: egyetlen pontból álló képállomány (1 1¶szám1¶szám2), egyetlen sorból illetve egyetlen oszlopból álló képállomány. A képállományok szövegfájlokban tárolt pixel-adatai egyszer legyenek teljesen azonosak egyszer különbözőek, a különböző képek egyszer egy különbséget, egyszer több különbséget tartalmazzanak. Az ”egy-különbség” eseteknél a különböző értékek helye legyen egyszer belső egyszer szélső pont.