Pásztor Attila

Algoritmizálás és programozás tankönyv az emeltszintű érettségihez

8. Elemi algoritmusok II

A 6. fejezetben megismertünk 6 olyan elemi algoritmust, amelyik sorozathoz egyetlen értéket rendel. Most olyan algoritmusokkal ismerkedünk meg, amelyek sorozathoz sorozatot, vagy sorozatokat rendelnek. Most is példákból fogunk kiindulni.

8.1. Másolás

8.1.1. feladat: Egy sorozatban megadott számok négyzetét állítsuk elő (egyenként)!
8.1.2. feladat: Egy szövegben cseréljük ki az összes ’é’ betűt ’e’-re!

A feladatokban közös, hogy az eredmény ugyanannyi elemet tartalmaz, mint a bemenő adat. Az eredmény sorozat egy elemének meghatározásához a bemenő sorozat egy eleme szükséges, és ismerjük azt a függvényt, amelyik a bemenő értékből eredményt készít. Első megoldásként az eredményt egy másik sorozatban várjuk:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
      B:Tömb(1..N:ElemTípus)     [N elemű kimenő sorozat]
Függvény:
      f: ElemTípus -> ElemTípus2  [a kimenő adat típusa lehet más is]

Eljárás Masolas(Konstans N,A, Változó: B):
  Ciklus i:=1-től N-ig
    B(i):=f(A(i)) [f számolja ki a bemenő elemből a kimenő elemet]
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Masolas(Const N:Integer; Const A:Sorozat;
                          Var B:Sorozat);
Var i:Integer;
Begin
  For i:=1 to N do B[i] := Fuggv(A[i]);
End;
```

A leképző függvény a 8.1.1. feladatra:

Alg.

```
Függvény f(Konstans E:egész): egész;
           [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  f:=E*E   [a négyzetének is bele kell férnie az ábrázolásba]
Eljárás vége.
```

TP kód

```
Function Fuggv(Const E:Integer): Boolean;
      [megjegyzés az algoritmusnál]
  Fuggv:=E*E;
End;
```

A leképző függvény a 8.1.2. feladatra:

Alg.

```
Függvény f(Konstans E:karakter): karakter;
           [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Ha E=’é’ akkor f:=’e’ különben f:=E
Eljárás vége.
```

TP kód

```
Function Fuggv(Const E:Integer): Boolean;
      [megjegyzés az algoritmusnál]
  If E=’é’ then Fuggv:=’e’ else fuggv:=E;
End;
```

Ha az eredményt ugyanabban a sorozatban várjuk, akkor az eljárás ciklusában az értékadást kell kicserélni:

```
B(i):=f(A(i)) helyett A(i):=f(A(i))
```

8.2. Kiválogatás

8.2.1. feladat: Egy egész számokat tartalmazó sorozatból adjuk meg a páros elemeket!
8.2.2. feladat: Egy szövegből adjuk meg a magánhangzók sorszámát (hányadik pozícióban van magánhangzó)!
8.2.3. feladat: Adjuk meg egy természetes szám összes osztóját!

Ebben az esetben nem feltétlenül kell lemásolni a teljes sorozatot. A feladat emlékeztet a keresésre és a megszámolásra is: adott tulajdonságú elemet kell megadni, de nem egyet, hanem az összest, illetve nem megszámolni kell, hanem megadni (értékét, vagy sorszámát. Az első általános algoritmus értékeket fog megadni egy másik sorozatban (B), megadva az eredmény sorozat darabszámát (DB) is.

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
      DB:egész                   [az eredmény sorozat elemszáma]
      B:Tömb(1..DB:ElemTípus)    [N elemű kimenő sorozat]
Függvény:
      Jó: ElemTípus -> logikai  [eldönti egy elemről, hogy jó-e]

Eljárás Kivalogatas(Konstans N,A, Változó: DB,B):
  DB:=0
  Ciklus i:=1-től N-ig
    Ha Jó(A(i)) akkor     DB:=DB+1     [találtunk egy jó elemet]
                          B[DB]:=A[i]  [átmásolom az értéket B sorozatba]
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Kivalogatas(Const N:Integer; Const A:Sorozat;
                      Var DB:Integer; Var B:Sorozat);
Var i:Integer;
Begin
  DB:=0;
  For i:=1 to N do
    If Jo(A[i]) then
      Begin
        DB:=DB+1; B[DB]:=A[i];
      End;
End;
```

Ha nem az elemeket, hanem a sorszámokat kell kiválogatni, akkor így módosul:

```
Algoritmusban:  B(DB):=A(i) helyett B(DB):=i
TP kódban:      B[DB]:=A[i]; helyett B[DB]:=i;
```

A „jó” függvény a 8.2.1. feladatra:

Alg.

```
Függvény Jó(Konstans E:egész): logikai;
           [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Jó:=(Maradék(E,2)=0)
Függvény vége.
```

TP kód

```
Function Jo(Const E:Integer): Boolean;
      [megjegyzés az algoritmusnál]
  Jo:=(E DIV 2 = 0);
End;
```

A „jó” függvény a 8.2.2. feladatra:

Alg.
```
Függvény Jó(Konstans E:karakter): logikai;
Konstans magánhangzók='aAeEuUiIoOíÍöÖüÜóÓőŐúÚéÉáÁűŰ'
  Jó:=(BenneVan(E,magánhangzók)>0)
Függvény vége.
```

TP kód
```
Function Jo(Const E:Char): Boolean;
Const maganhangzok='aAeEuUiIoOíÍöÖüÜóÓőŐúÚéÉáÁűŰ'
  Jo:=(Pos(E,maganhangzok)>0);
End;
```

A 8.2.3. feladatban az A sorozatot töltsük fel a természetes számokkal 0-tól N-ig, majd a kiválogatás jó függvénye:

Alg.
```
Függvény Jó(Konstans E:egész): logikai;
            [most is szerepelhetne ez a rész magában az eljárásban]
  Jó:=(Maradék(N,E)=0)
Függvény vége.
```

TP kód
```
Function Jo(Const E:Integer): Boolean;
            [megjegyzés az algoritmusnál]
  Jo:=(N DIV E = 0);
End;
```

Lehetséges, hogy az eredeti sorozatra már nincs szükség, ezért a kiválogatás eredményét az eredeti sorozatban várjuk. Ezt nevezzük **helyben kiválogatás**nak. Az algoritmus alig változik:

Nem kell a B sorozat, valamint:
```
Algoritmusban: B(DB):=A(i) helyett A(DB):=A(i)
TP kódban:     B[DB]:=A[i]; helyett A[DB]:=A[i];
```

Elképzelhető, hogy a kiválogatást úgy végezzük, hogy a nem mozgatjuk az elemeket, csupán a nem megfelelő elemek helyére egy speciális elemet teszünk (SPEC). Ez a **kiválogatás kihúzással**. Az algoritmus változó része.

Nem kell B sorozat, nem kell DB sem, valamint:
```
Algoritmusban: Ha Jó(A(i)) teljes elágazás helyett:
                          Ha nem Jó(A(i)) akkor A(i):=SPEC
TP kódban:     If Jo(A[i]) elágazás egésze helyett:
                          If not Jo(A[i]) then A[i]:=SPEC;
```

Előfordulhat, hogy a kiválogatás eredményét nem kell eltárolni, csak kiírni (például a képernyőre). Ez a **kiválogatás kiírással**. Az algoritmus az eredetitől az alábbiakban tér el:

Nem kell a B sorozat, nem kell DB, valamint:
```
Algoritmusban: DB:=DB+1
               B(DB):=A(i) helyett  Ki: A(i) [vagy Ki: i]
TP kódban:     DB:=DB+1;
               B[DB]:=A[i]; helyett  WriteLn(A[i];(*vagy csak i*)
```

8.3. Szétválogatás

8.3.1. Válogassuk szét egy egész számokat tartalmazó sorozat elemeit párosakra és páratlanokra!
8.3.2. Neveket tartalmazó sorozatot válogassunk szét magánhangzóval, vagy mássalhangzóval kezdődőkre!

A feladat nagyon hasonlít a kiválogatásra, meg lehetne oldani két kiválogatással de hatékonyabb, ha az elem vizsgálatánál a jó és a rossz ágban is kezeljük az elemet. Szükségünk lesz tehát két kimenő sorozatra két kimenő darabszámmal (J,JDB, R,RDB). Az algoritmus:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
      JDB:egész                  [a jó elemek darabszáma]
      J:Tömb(1..JDB:ElemTípus)   [JDB elemű kimenő sorozat jó elemeinek]
      RDB:egész                  [a rossz elemek darabszáma]
      R:Tömb(1..RDB:ElemTípus)   [RDB elemű kimenő sorozat rossz elemeinek]
Függvény:
      Jó: ElemTípus -> logikai  [eldönti egy elemről, hogy jó-e]

Eljárás Szétválogatás(Konstans N,A, Változó: DB,B):
  DB:=0
  Ciklus i:=1-től N-ig
    Ha Jó(A(i)) akkor     JDB:=JDB+1    [találtunk egy jó elemet]
                          J[JDB]:=A[i]  [átmásolom az értéket J sorozatba]
                különben  RDB:=RDB+1    [találtunk egy rossz elemet]
                          R[RDB]:=A[i]  [átmásolom az értéket R sorozatba]
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Szetvalogatas(Const N:Integer; Const A:Sorozat;
                        Var JDB:Integer; Var J:Sorozat;
                        Var RDB:Integer; Var R:Sorozat);
Var i:Integer;
Begin
  For i:=1 to N do
    If Jo(A[i]) then
      Begin
        JDB:=JDB+1; J[JDB]:=A[i];
      End
    else
      Begin
        RDB:=RDB+1; R[RDB]:=A[i];
      End
End;
```

Ha nem az elemeket, hanem a sorszámokat kell kiválogatni, akkor így módosul:

```
Algoritmusban:  J(JDB):=A(i) helyett J(JDB):=i
                R(RDB):=A(i) helyett R(RDB):=i
TP kódban:      J[JDB]:=A[i]; helyett J[JDB]:=i;
                R[RDB]:=A[i]; helyett R[RDB]:=i;
```

A „jó” függvény a 8.3.1. feladatra (azonos a 8.2.1. feladatéval):

Alg.

```
Függvény Jó(Konstans E:egész): logikai;
           [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Jó:=(Maradék(E,2)=0)
Függvény vége.
```

TP kód

```
Function Jo(Const E:Integer): Boolean;
      [megjegyzés az algoritmusnál]
  Jo:=(E DIV 2 = 0);
End;
```

A „jó” függvény a 8.3.2. feladatra (kihasználjuk, hogy a szöveg karakterek sorozata):

Alg.
```
Függvény Jó(Konstans E:String): logikai;
Konstans magánhangzók='aAeEuUiIoOíÍöÖüÜóÓőŐúÚéÉáÁűŰ'
  Jó:=(BenneVan(E(1),magánhangzók)>0)
Függvény vége.
```

TP kód
```
Function Jo(Const E:Char): Boolean;
Const maganhangzok='aAeEuUiIoOíÍöÖüÜóÓőŐúÚéÉáÁűŰ'
  Jo:=(Pos(E[1],maganhangzok)>0);
End;
```

Láthatjuk, hogy fölösleges a két sorozat, mivel a szétválogatott elemek összdarabszáma most is N marad. Egyetlen N elemű sorozatban elő lehet állítani a szétválogatás eredményét úgy, hogy a sorozatot az elejétől jó elemekkel, míg a végétől visszafelé a rossz elemekkel töltjük fel. Az **egy sorozatba szétválogatás** algoritmusa:

Algoritmus
```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
      B:Tömb(1..N:ElemTípus)     [N elemű kimenő sorozat az eredménynek]
      DB:egész                   [a jó elemek darabszáma]
Függvény:
      Jó: ElemTípus -> logikai  [eldönti egy elemről, hogy jó-e]

Eljárás Szétvalogatás(Konstans N,A, Változó: DB,B):
  RINDEX:=N+1 : DB:=0
  Ciklus i:=1-től N-ig
    Ha Jó(A(i)) akkor     DB:=DB+1       [találtunk egy jó elemet]
                          B[DB]:=A[i]    [másolom az értéket B sorozatba]
                különben  RINDEX:=RINDEX-1  [találtunk egy rossz elemet]
                          B[RINDEX]:=A[i]   [másolom  B sorozat végére]
  Ciklus vége
Eljárás vége
```

TP kód
```
Procedure Szetvalogatas(Const N:Integer; Const A:Sorozat;
                        Var DB:Integer; Var B:Sorozat);
Var i,RINDEX:Integer;
Begin
  RINDEX:=N+1; DB:=0;
  For i:=1 to N do
    If Jo(A[i]) then
      Begin
        DB:=DB+1; B[DB]:=A[i];
      End
    else
      Begin
        RINDEX:=RINDEX-1; B[RINDEX]:=A[i];
      End
End;
```

Megjött az étvágyunk: ne kelljen eredmény sorozat, válogassuk szét a sorozatot önmagában, ez lesz a **szétválogatás helyben**! Az eredeti sorozat elején lesznek a „jó” elemek.
Megcsinálhatnánk úgy, hogy mindkét irányban haladunk a sorozatban befelé, és ha megcserélendő elemeket találunk, akkor megcseréljük azokat, de létezik ennél hatékonyabb megoldás:
Vegyük ki a sorozat első elemét, majd az utolsó elemtől visszafelé haladva keressünk nem oda illő, azaz jó elemet. Ha találunk ilyet, akkor tegyük be a kivett elem helyére (felszabadul hátul egy hely. Most a sorozat elejétől keressünk nem jó tulajdonságú elemet, s ha találunk, akkor tegyük az a végén lévő üres helyre. Ezt ismételjük addig, amíg a sorozatban két irányban haladva össze nem érünk. Ha összeértünk, akkor az üres helyre tegyük be a kivett (első) elemet, és annak tulajdonsága alapján megmondhatjuk a jó elemek darabszámát.

Az algoritmus:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
      BDB:egész                  [a jó elemek darabszáma]
Függvény:
      Jó: ElemTípus -> logikai   [eldönti egy elemről, hogy jó-e]

Eljárás SzetválogatásHelyben(Konstans N,A, Változó: DB):
  U:=N : E:=1 : segéd:=A(1)
  Ciklus amíg E<U                [amíg össze nem érek]
    Ciklus amíg E<U és nem Jó(A(U))
      U:=U-1                     [jó elemet keresek a végén]
    Ciklus vége
    Ha E<U akkor A(E):=A(U) : E:=E+1
                Ciklus amíg E<U és Jó(A(E))
                   E:=E+1        [rossz elemet keresek az elején]
                Ciklus vége
                Ha E<U akkor A(U):=A(E) : U:=U-1
    Elágazás vége
  Ciklus vége
  A(E):=segéd
  Ha Jó(A(E)) akkor DB:=E különben DB:=E-1
Eljárás vége
```

TP kód

```
Procedure SzetvalogatasHelyben(Const N:Integer; Const A:Sorozat;
                               Var DB:Integer);
Var E,U:Integer;
    seged:ElemTipus;
Begin
  U:=N; E:=1; seged:=A[1];
  While E<U do
    Begin
      While (E<U) and not Jo(A[U]) do U:=U-1;
      If E<U then
        Begin
          A[E]:=A[U]; E:=E+1;
          While (E<U) and Jo(A[E]) do  E:=E+1;
          If E<U then
            Begin
              A[U]:=A[E]; U:=U-1;
            End;
        End;
    End; (* While E<U*)
    A[E]:=seged;
If Jo(A[E] then DB:=E else DB:=E-1;
End;
```

8.4. Metszet (közös rész)

8.4.1. feladat: Két halmaz elemeit tároljuk két sorozatban. Adjuk meg a két halmaz közös részét (metszetét) egy sorozatban!
8.4.2. feladat: Adott két verseny versenyzőinek névsora, adjuk meg azoknak a nevét, akik mindkét versenyen indultak!

Ezek a feladatok két (vagy több) sorozathoz rendelnek egy sorozatot. A tradicionális metszet halmaz művelet sorozatokon is értelmezhető feltéve, hogy a sorozat halmazfelsorolás, azaz nincs benne ismétlődő elem. Menjünk végig az első sorozat elemein, és csak akkor tegyük be ezt az elemet az eredmény sorozatba, ha a másik sorozatban is benne van. A feladat felfogható úgy, mint egy kiválogatás, aminek a belsejében egy eldöntés van. Nyilvánvaló, hogy az eredmény sorozat elemszáma nem lehet nagyobb a kisebb bemenő sorozat elemszámánál. Az algoritmus:

Algoritmus

```
Változó:
      AN:egész                  [A sorozat elemszáma]
      A:Tömb(1..AN:ElemTípus)   [AN elemű bemenő sorozat]
      BN:egész                  [B sorozat elemszáma]
      B:Tömb(1..BN:ElemTípus)   [BN elemű bemenő sorozat]
      CB:egész                  [eredmény sorozat (C) darabszáma]
      C:Tömb(1..CN:ElemTípus)   [CN elemű kimenő sorozat az eredménynek]

Eljárás Metszet(Konstans AN,A,BN,B Változó: CN,C):
  CN:=0
  Ciklus i:=1-től AN-ig             [A sorozaton megyek végig]
    j:=1                            [eldöntöm, hogy benne van-e B-ben]
    Ciklus amíg j<=BN és A(i)<>B(j)
         j:=j+1
    Ciklus vége
    Ha j<=BN akkor CDB:=CDB+1 : C(CDB):=A(i)  [mindkét sorozatban benne van]
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Metszet(Const AN:Integer; Const A:Sorozat;
                  Const BN:Integer; Const B:Sorozat;
                  Var CN:Integer; Var C:Sorozat);
Var i,j:Integer;
Begin
  CN:=0;
  For i:=1 to AN do
    Begin
      j:=1;
      While (j<=BN) and (A[i]<>B[j]) do j:=j+1;
      If j<=BN then
        Begin
          CDB:=CDB+1; C[CDB]:=A[i];
        End;
    End; (* For *)
End;
```

Ez a feladattípus további feladatok megoldásának alapja is lehet:

Van-e két sorozatnak közös eleme? (eldöntés)!
Adjuk meg két sorozat egyik közös elemét, ha biztosan van (kiválasztás)!
Ha van két sorozatnak közös eleme, akkor adjunk meg egyet (keresés)!
Hány közös eleme van két sorozatnak (megszámolás)!

Ezek megoldását most nem részletezzük, a fenti algoritmusból egyszerűen kinyerhető.

8.5. Unió (egyesítés)

8.5.1. feladat: Két halmaz elemeit tároljuk két sorozatban. Adjuk meg a két halmaz egyesítését (unióját) egy sorozatban!
8.4.2. feladat: Adott két verseny versenyzőinek névjegyzéke (nem feltétlenül névsorban), adjuk meg azoknak a nevét, akik legalább egy versenyen indultak!

Itt is feltételezzük, hogy a két sorozat halmazfelsorolás. Az első sorozat elemei biztosan benne lesznek az eredményben. Menjünk végig a másik sorozat elemein, és csak akkor tegyük be egy elemét az eredmény sorozatba, ha az az elem nem fordul elő az első sorozatban (vagy az eredményben). A feladat első része egy értékadás (átmásolás), a második része felfogható úgy, mint egy kiválogatás, aminek a belsejében egy eldöntés van. Nyil-

vánvaló, hogy az eredmény sorozat elemszáma nem lehet nagyobb a két bemenő sorozat elemszámának összege. Az algoritmus:

Algoritmus

```
Változó:
      AN:egész                    [A sorozat elemszáma]
      A:Tömb(1..AN:ElemTípus)     [AN elemű bemenő sorozat]
      BN:egész                    [B sorozat elemszáma]
      B:Tömb(1..BN:ElemTípus)     [BN elemű bemenő sorozat]
      CB:egész                    [eredmény sorozat (C) darabszáma]
      C:Tömb(1..CN:ElemTípus)     [CN elemű kimenő sorozat az eredménynek]

Eljárás Unió(Konstans AN,A,BN,B Változó: CN,C):
  CN:=AN : C:=A                        [A sorozat biztos benne van C-ben]
  Ciklus i:=1-től BN-ig                [B sorozaton megyek végig]
    j:=1                               [eldöntöm, hogy benne van-e A-ban]
    Ciklus amíg j<=AN és B(i)<>A(j)
         j:=j+1
    Ciklus vége
    Ha j>AN akkor CDB:=CDB+1 : C(CDB):=B(i)  [nem volt benne]
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Unio(Const AN:Integer; Const A:Sorozat;
               Const BN:Integer; Const B:Sorozat;
               Var CN:Integer; Var C:Sorozat);
Var i,j:Integer;
Begin
  CN:=AN; C:=A;  (* megtehető, ha a típusuk név szerint megegyezik *)
  For i:=1 to BN do
    Begin
      j:=1;
      While (j<=AN) and (B[i]<>A[j]) do j:=j+1;
      If j>AN then
        Begin
          CDB:=CDB+1; C[CDB]:=B[i];
        End;
    End; (* For *)
End;
```

Az előbbi algoritmus alkalmas arra, hogy sorozatból halmazfelsorolást készítsen (ne legyen benne ismétlődő elem). Ehhez az kell, hogy az eldöntés az eredmény sorozatban dolgozzon (AN helyett CN, A helyett C írandó, és kész a megoldás).

8.6. Összefuttatás

8.6.1. feladat: Két halmaz elemeit tároljuk két sorozatban. Az elemek növekvően rendezettek mindkét bemenő sorozatban. Adjuk meg a két halmaz egyesítését (unió) egy sorozatban!
8.6.2. feladat: Adott két verseny versenyzőinek névsora (sorba rendezve), adjuk meg azoknak a névsorát, akik legalább egy versenyen indultak!

Ezekből a feladatokból látható, hogy speciális esetben az uniót egyszerűbben is elvégezhetjük. Haladjunk párhuzamosan a két sorozatban! Ha a két sorozat első elemei nem egyezik meg, akkor a kisebb elemet másoljuk az eredmény sorozatba, és abban a bemenő sorozatban a következő elemre lépünk. Ha egyformák a vizsgált elemek, akkor betesszük az elemet az eredmény sorozatba és mindkét sorozatban lépünk egyet. Ha az egyiknek a végére értünk, akkor a másik sorozatot vizsgálat nélkül hozzámásoljuk az eredményhez. Készítsük el az összefuttatás, azaz a rendezett sorozatok uniójának algoritmusát:

Algoritmus

```
Változó:
      AN:egész                    [A sorozat elemszáma]
      A:Tömb(1..AN:ElemTípus)     [AN elemű bemenő sorozat]
      BN:egész                    [B sorozat elemszáma]
      B:Tömb(1..BN:ElemTípus)     [BN elemű bemenő sorozat]
      CB:egész                    [eredmény sorozat (C) darabszáma]
      C:Tömb(1..CN:ElemTípus)     [CN elemű kimenő sorozat az eredménynek]
Eljárás Összefuttatás(Konstans AN,A,BN,B Változó: CN,C):
  i:=1 : j:=1 : CN:=0             [bemenő sorozatok elejére állunk, C üres]
  Ciklus amíg i<=AN és j<=BN      [addig megyünk, amíg az egyik üres nem lesz]
    CN:=CN+1                      [egy elem biztosan belekerül C-be]
    Elágazás
      A(i)<B(j) esetén C(CN):=A(i) : i:=i+1 [A eleme volt kisebb]
      A(i)>B(j) esetén C(CN):=B(j) : j:=j+1 [B eleme volt kisebb]
      A(i)=B(j) esetén C(CN):=A(i) : i:=i+1 : j:=j+1 [egyformák]
    Elágazás vége                 [csak az egyik sorozatban maradhatott elem]
  Ciklus vége                     [valamelyik sorozat végére értünk]
  Ciklus amíg i<=AN               [A hátralévő elemeit bemásolom]
    CN:=CN+1 : C(CN):=A(i) : i:=i+1
  Ciklus vége
  Ciklus amíg j<=BN               [B hátralévő elemeit bemásolom]
    CN:=CN+1 : C(CN):=B(j) : j:=j+1
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Osszefuttatas(Const AN:Integer; Const A:Sorozat;
                        Const BN:Integer; Const B:Sorozat;
                        Var CN:Integer; Var C:Sorozat);
Var i,j:Integer;
Begin
  i:=1; j:=1; CN:=0;
  While (i<=AN) and ((j<=BN) do
    Begin
      CN:=CN+1;
      If A[i]<B[j] then
        Begin
          C[CN]:=A[i]; i:=i+1;
        End        (* nincs pontosvessző *)
      else
      If A[i]>B[j] then
        Begin
          C[CN]:=B[j]; j:=j+1;
        End  (* nincs pontosvessző *)
      else
        Begin
          C[CN]:=A[i]; i:=i+1; j:=j+1;
        End;
    End; (* While i,j*)
  While i<=AN then
    Begin
      CN:=CN+1; C[CN]:=A[i]; i:=i+1;
    End;
  While j<=BN then
    Begin
      CN:=CN+1; C[CN]:=B[j]; j:=j+1;
    End;
End;
```

Az algoritmus jó, de ne legyünk elégedettek. Ha elérnénk, hogy a két sorozatnak egyszerre érjünk a végére, akkor a két utolsó ciklus felesleges lenne. Sajnos ez az eset általában nem fordul elő, de előidézhetjük mi magunk: tegyünk mindkét bemenő sorozat végére egy nagyon nagy egyforma elemet (olyan nagy legyen, hogy a sorozatokban nála csak nem nagyobbak lehetnek). Ekkor biztosan egyszerre érünk a két sorozat végére, már csak arról kell gondoskodnunk, hogy ez az egy fiktív elem ne kerüljön be az eredménybe. Ezt is egyszerűen megoldhatjuk: most a bemenő sorozat elemszáma AN+1, illetve BN+1, írjuk át a ciklust úgy, hogy csak az előző elemig menjen. Íme a megoldás:

Algoritmus

```
Változó:
      AN:egész                    [A sorozat elemszáma]
      A:Tömb(1..AN+1:ElemTípus)   [AN+1 elemű bemenő sorozat]
      BN:egész                    [B sorozat elemszáma]
      B:Tömb(1..BN+1:ElemTípus)   [BN+1 elemű bemenő sorozat]
      CB:egész                    [eredmény sorozat (C) darabszáma]
      C:Tömb(1..CN:ElemTípus)     [CN elemű kimenő sorozat az eredménynek]
Eljárás Összefuttatás(Konstans AN,A,BN,B Változó: CN,C):
  i:=1 : j:=1 : CN:=0             [bemenő sorozatok elejére állunk, C üres]
  A[AN+1]:=MAXELEM : B[BN+1]:=MAXELEM [az ábrázolható legnagyobb érték]
  Ciklus amíg i<AN+1 és j<BN+1    [addig megyünk, amíg végére érünk]
    CN:=CN+1                      [egy elem biztosan belekerül C-be]
    Elágazás
      A(i)<B(j) esetén C(CN):=A(i) : i:=i+1 [A eleme volt kisebb]
      A(i)>B(j) esetén C(CN):=B(j) : j:=j+1 [B eleme volt kisebb]
      A(i)=B(j) esetén C(CN):=A(i) : i:=i+1 : j:=j+1 [egyformák]
    Elágazás vége                 [csak az egyik sorozatban maradhatott elem]
  Ciklus vége                     [valamelyik sorozat végére értünk]
Eljárás vége
```

TP kód

```
Procedure Osszefuttatas(Const AN:Integer; Const A:Sorozat;
                        Const BN:Integer; Const B:Sorozat;
                        Var CN:Integer; Var C:Sorozat);
Var i,j:Integer;
Begin
  i:=1; j:=1; CN:=0;
  A[AN+1]:=MAXELEM; B[BN+1]:=MAXELEM; [legnagyobb ábrázolható]
  While (i<AN+1) and ((j<BN+1) do
    Begin
      CN:=CN+1;
      If A[i]<B[j] then
        Begin
          C[CN]:=A[i]; i:=i+1;
        End        (* nincs pontosvessző *)
      else
      If A[i]>B[j] then
        Begin
          C[CN]:=B[j]; j:=j+1;
        End  (* nincs pontosvessző *)
      else
        Begin
          C[CN]:=A[i]; i:=i+1; j:=j+1;
        End;
    End; (* While i,j*)
End;
```

Ha a két sorozatnak biztosan nincs közös eleme, akkor a feladatot **összefésülésnek** nevezzük, ekkor kimaradhat az elágazások egyenlőségre vonatkozó ága. Csak az algoritmust írom le, akód abból egyszerűen származtatható:

Algoritmus

```
Változó:
      AN:egész                    [A sorozat elemszáma]
      A:Tömb(1..AN+1:ElemTípus)   [AN+1 elemű bemenő sorozat]
      BN:egész                    [B sorozat elemszáma]
      B:Tömb(1..BN+1:ElemTípus)   [BN+1 elemű bemenő sorozat]
      CB:egész                    [eredmény sorozat (C) darabszáma]
      C:Tömb(1..CN:ElemTípus)     [CN elemű kimenő sorozat az eredménynek]
Eljárás Összefésülés(Konstans AN,A,BN,B Változó: CN,C):
  i:=1 : j:=1 : CN:=0             [bemenő sorozatok elejére állunk, C üres]
  A[AN+1]:=MAXELEM : B[BN+1]:=MAXELEM [az ábrázolható legnagyobb érték]
  Ciklus amíg i<AN+1 és j<BN+1    [addig megyünk, amíg végére érünk]
    CN:=CN+1                      [egy elem biztosan belekerül C-be]
    Hs  A(i)<B(j) akkor C(CN):=A(i) : i:=i+1 [A eleme volt kisebb]
                különben C(CN):=B(j) : j:=j+1 [B eleme volt kisebb]
  Ciklus vége                     [valamelyik sorozat végére értünk]
Eljárás vége
```

8.7. Rendezés

A következő részben több algoritmust fogunk készíteni egy feladatra: N elemű sorozat elemeit rendezzük nagyság szerint növekvő sorrendbe, ráadásul helyben, azaz az eredeti sorrend elvész. A megvalósításhoz szükséges, hogy a sorozat elemi között értelmezett legyen a < és ≤ reláció.
Az egyes megoldásokat összehasonlítjuk majd tárigény és végrehajtási idő (hasonlítások, mozgatások száma) szerint. Természetesen a legtöbb esetben a hasonlítások és a mozgatások száma függ a rendezendő elemektől, ezért minimális és maximális számukról érdemes beszélni.

Mindegyik rendezésnek kelléke egy csere eljárás, amelyik a sorozat két elemét megcseréli. Ezt az eljárást most elkészítjük, hogy a későbbiekben ne kelljen vele foglalkozni.

Algoritmus

```
Változó:
      X,Y:ElemTípus
Eljárás Csere(Változó X,Y):
  Z:=X
  X:=Y
  Y:=Z
Eljárás vége
```

TP kód

```
Procedure Csere(Var X,Y: ElemTipus);
Var Z:ElemTipus;
Begin
  Z:=X
  X:=Y
  Y:=Z
End;
```

Az algoritmusokat konkrét példával fogom szemléltetni Az egyszerűség kedvéért egyetlen számpéldán nézzük meg a lehetséges algoritmusokat:

Az elemek száma: **N=5**
A sorozat: **A=8, 2, 15, 1, 9**

A szemléletesebb ábrázolás kedvéért az éppen vizsgált elemeket keretezéssel fogom jelölni, például:

2 8 15 1 9

Ha a vizsgálatot csere is követi, akkor az elemek cellájának háttere is színes:

2 8 15 1 9

Ha egy elem a helyére kerül, és többé nem mozgatjuk el, akkor az elem betűszíne is megváltozik:

1 8 15 2 9

8.7.1. Egyszerű cserés rendezés

Ötlet: Hasonlítsuk össze a sorozat első elemét az összes mögötte lévővel (egyesével). Ha valamelyik kisebb nála, akkor megcseréljük őket. Ezzel elérjük, hogy a sorozat elemein végighaladva az első helyére a legkisebb elem kerül. Alkalmazzuk ugyanezt a módszert a második elemre, és így tovább egészen az utolsó előttiig.
Az algoritmus a következő:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
Eljárás E_Cs_Rendezés(Változó N,A):
  Ciklus i:=1-től N-1-ig         [haladok végig a sorozat elemein]
    Ciklus j:=i+1-től N-ig
      Ha A(i)>A(j) akkor Csere(A(i),A(j)  [összehasonlítom a következőkkel]
    Ciklus vége
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure E_Cs_Rendezes(Var N:Integer; Var A:Sorozat);
Var i,j:Integer;
Begin
  For i:=1 to N-1 do
    For j:=i+1 to N do
      If A[i]>A[j] then Csere(A[i],A[j])
End;
```

Most nézzük meg, hogy mindez a konkrét példán hogyan történik!
Az épp összehasonlított elemeket keretezéssel jelzem, ha cserélni kell, akkor színes a cserélendő elemek háttere: A helyére került elem betűszíne kék:

i=1	**8**	**2**	**15**	**1**	**9**
	2	**8**	**15**	**1**	**9**
	2	**8**	**15**	**1**	**9**
	1	**8**	**15**	**2**	**9**
i=2	**1**	**8**	**15**	**2**	**9**
	1	**8**	**15**	**2**	**9**
	1	**2**	**15**	**8**	**9**
i=3	**1**	**2**	**15**	**8**	**9**
	1	**2**	**8**	**15**	**9**
i=4	**1**	**2**	**8**	**15**	**9**
végeredmény	**1**	**2**	**8**	**9**	**15**

Helyfoglalás:
N+1

Hasonlítások száma:
N*(N-1)/2

Mozgatások száma:
0 – 3*N*(N-1)/2

A csere 3 mozgatásnak felel meg!

8.7.2. Minimumkiválasztásos rendezés

Az előző módszer hibája, hogy rengeteg fölösleges cserét tartalmaz. Elegendő lenne az aktuális elemet a mögötte lévők legkisebbikével megcserélni, ehhez a belső ciklusban minimumkiválasztást kell alkalmazni:

Algoritmus

```
Változó:
      N:egész                        [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)         [N elemű bemenő sorozat]
Eljárás Min_Rendezés(Változó N,A):
  Ciklus i:=1-től N-1-ig             [haladok végig a sorozat elemein]
    MIN:=i
    Ciklus j:=i+1-től N-ig
      Ha A(MIN)>A(j) akkor MIN:=j    [összehasonlítom a következőkkel]
    Ciklus vége
    Csere(A(i),A(MIN))
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Min_Rendezes(Var N:Integer; Var A:Sorozat);
Var i,j:Integer;
    MIN:ElemTipus;
Begin
  For i:=1 to N-1 do
    Begin
      MIN:=i;
      For j:=i+1 to N do
          If A[MIN]>A[j] then MIN:=j;
      Csere(A[i],A[MIN])
    End;
End;
```

Láthatjuk, hogy lehetséges az az eset, hogy önmagával cseréli meg az elemet.

Nézzük meg a példát (a cserét CS betűvel jelzem a minimum elem indexe mellett:

i=1	**8**	**2**	15	1	9	MIN=2
	8	2	**15**	1	9	MIN=2
	8	2	15	**1**	9	MIN=4
	8	2	15	**1**	**9**	MIN=4 CS
i=2	**1**	**2**	**15**	8	9	MIN=2
	1	**2**	15	**8**	9	MIN=2
	1	**2**	15	8	**9**	MIN=2 CS
i=3	**1**	**2**	**15**	**8**	9	MIN=4
	1	**2**	**15**	**8**	**9**	MIN=4 CS
i=4	**1**	**2**	**8**	**15**	**9**	MIN=5 CS
végeredmény	**1**	**2**	**8**	**9**	**15**	

Helyfoglalás: N+1

Hasonlítások száma: N*(N-1)/2

Mozgatások száma: 3*(N-1)

A csere 3 mozgatásnak felel meg!

A mozgatások száma szerencsétlen esetben még rosszabb is lehet, mint az előző esetben, mert a külső ciklusban mindenképpen cserélünk. Meg lehetne vizsgálni, hogy kell-e cserélni, de a vizsgálat a legtöbb esetben növelné a végrehajtási időt.

8.7.3. Buborékos rendezés

Új megoldási ötlet: mindig két szomszédos elemet vizsgálunk, ha sorrendjük nem jó, akkor felcseréljük őket. Egyszer végighaladva a sorozaton a legnagyobb elem kerül biztosan a helyére, a nagyobb elemek hátrafelé, a kisebbek előrefelé mozognak (ezért nevezik buborékos módszernek). A következő lépésben már csak az utolsó előtti elemig kell végrehajtani a fenti módszert. Nézzük az algoritmust!

Algoritmus

```
Változó:
      N:egész                      [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)       [N elemű bemenő sorozat]
Eljárás Buborékos_Rendezés(Változó N,A):
  Ciklus i:=N-től 2-ig -1-esével [haladok végig a sorozat elemein]
    Ciklus j:=1-től i-1-ig
      Ha A(j)>A(j+1) akkor Csere(A(j),A(j+1)  [csere, ha kell]
    Ciklus vége
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Buborekos_Rendezes(Var N:Integer; Var A:Sorozat);
Var i,j:Integer;
Begin
  For i:=N downto 2 do
    For j:=1 to i-1 do
      If A[j]>A[j+1] then Csere(A[j],A[j+1]);
End;
```

Nézzük meg az algoritmus működését a konkrét példával (a színhasználat a szokásos):

i=4	**8**	**2**	15	1	9
	2	**8**	**15**	1	9
	2	8	**15**	**1**	9
	2	8	1	**15**	**9**
i=3	**2**	**8**	1	9	15
	2	**8**	**1**	9	15
	2	1	**8**	**9**	15
i=2	**2**	**1**	8	9	15
	1	**2**	**8**	9	15
i=1	**1**	**2**	8	9	15
végeredmény	1	2	8	9	15

Helyfoglalás:
N+1

Hasonlítások száma:
N*(N-1)/2

Mozgatások száma:
0 – 3*N*(N-1)/2

A csere 3 mozgatásnak felel meg!

Sajnos a hatékonysági jellemzők nem javultak.

8.7.4. Javított buborékos rendezés

Azért nem javultak a hatékonysági mutatók, mert nem használtuk ki, hogy az elemek a helyük felé elmozdulnak. Ha a belső ciklusban valahol volt csere, akkor biztosak lehetünk benne, hogy az azt követő rész már rendezett. Írjuk át az algoritmust eszerint!

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
Eljárás Mod_Buborékos_Rendezés(Változó N,A):
  i:=N
  Ciklus amíg i>=1        [haladok végig a sorozat elemein]
    CS:=0
    Ciklus j:=1-től i-1-ig
      Ha A(j)>A(j+1) akkor Csere(A(j),A(j+1)) : CS:=j  [csere, ha kell]
    Ciklus vége
    i:=CS
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Mod_Buborekos_Rendezes(Var N:Integer; Var A:Sorozat);
Var i,j,CS:Integer;
Begin
  i:=N;
  While i>=1 do
    Begin
      CS:=0;
      For j:=1 to i-1 do
           If A[j]>A[j+1] then Csere(A[j],A[j+1]); CS:=j;
      i:=CS;
    End;
End;
```

Tekintsük meg a konkrét példát (sorok maradnak ki belőle)! A CS változó értékét is feltüntettem a táblázatban.

i=4	**8**	**2**	**15**	**1**	**9**	CS=1
	2	**8**	**15**	**1**	**9**	
	2	**8**	**15**	**1**	**9**	CS=3
	2	**8**	**1**	**15**	**9**	CS=4
i=3	**2**	**8**	**1**	**9**	**15**	
	2	**8**	**1**	**9**	**15**	CS=2
	2	**1**	**8**	**9**	**15**	
i=1	**2**	**1**	**8**	**9**	**15**	CS=1
végeredmény	**1**	**2**	**8**	**9**	**15**	

Helyfoglalás: N+1

Hasonlítások száma: N-1 - N*(N-1)/2

Mozgatások száma: 0 – 3*N*(N-1)/2

A csere 3 mozgatásnak felel meg!

A hasonlítások számában javulást látunk, az igazi javulás azonban nem a minimális és a maximális, hanem az átlagos végrehajtási időben van.

8.7.5. Beillesztéses rendezés

Az eddigi módszerek mindegyike felosztotta a sorozatot már rendezett és még rendezetlen részre, valamint az elkezdéshez már az összes elemre szükség volt. Most megismerkedünk egy olyan módszerrel, amely leginkább a kártyalapok keveréséhez hasonlítható. Egy elem mindig rendezettnek tekinthető, a következő elemet a nagyság szerinti helyére illesztjük be.

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
Eljárás Beillesztéses_Rendezés(Változó N,A):
  Ciklus i:=2-től N-ig           [haladok végig a sorozat elemein]
    j:=i-1
    Ciklus amíg j>0 és A(j)>A(j+1)
      Csere(A(j),A(j+1)          [visszafelé cserélget]
      j:=j-1
    Ciklus vége
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Beilleszteses_Rendezes(Var N:Integer; Var A:Sorozat);
Var i,j:Integer;
Begin
  For i:=2 to N do
    Begin
      j:=i-1;
      While (j>0) and (A[j]>A[j+1]) do
        Begin
          Csere(A[j],A[j+1]);
          j:=j-1;
        End;
    End;
End;
```

Vizsgáljuk meg az algoritmus működését a konkrét példával:

i=2	**8**	**2**	**15**	**1**	**9**
i=3	**2**	**8**	**15**	**1**	**9**
i=4	**2**	**8**	**15**	**1**	**9**
	2	**8**	**1**	**15**	**9**
	2	**1**	**8**	**15**	**9**
i=5	**1**	**2**	**8**	**15**	**9**
	1	**2**	**8**	**9**	**15**
végeredmény	**1**	**2**	**8**	**9**	**15**

Helyfoglalás:
N+1

Hasonlítások száma:
N-1 - N*(N-1)/2

Mozgatások száma:
0 – 3*N*(N-1)/2

A csere 3 mozgatásnak felel meg!

8.7.6. Javított beillesztéses rendezés

A beillesztéses módszer érdekessége, hogy a legtöbb elem egyszer mozdul el, de a beillesztendő sokszor. Ezt a sok mozgást csökkenthetjük, ha a beillesztendőt nem tesszük be azonnal a sorozatba, hanem csak a többieket mozgatjuk hátrafelé, s csak a végén tesszük a helyére a beillesztendőt.

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű bemenő sorozat]
Eljárás Jav_Beillesztéses_Rendezés(Változó N,A):
  Ciklus i:=2-től N-ig           [haladok végig a sorozat elemein]
    j:=i-1
    segéd:=A(i)
    Ciklus amíg j>0 és A(j)>segéd
      A(j+1):=A(j)
      j:=j-1
    Ciklus vége
    A(j+1):=segéd
  Ciklus vége
Eljárás vége
```

TP kód

```
Procedure Jav_Beilleszteses_Rendezes(Var N:Integer; Var A:Sorozat);
Var i,j:Integer;
    segéd:ElemTipus;
Begin
  For i:=2 to N do
    Begin
      j:=i-1;
      seged:=A[i];
      While (j>0) and (A[j]>seged) do
        Begin
          A[j+1]):=A[j];
          j:=j-1;
        End;
        A[j+1]:=seged;
    End;
End;
```

Nézzük meg a konkrét példát:

i=2	**8**	**2**	**15**	**1**	**9**	seged=2
	2	**8**	**15**	**1**	**9**	
i=3	**2**	**8**	**15**	**1**	**9**	seged=15
i=4	**2**	**8**	**15**	**1**	**9**	seged=1
	2	**2**	**8**	**15**	**9**	
	2	**2**	**8**	**15**	**9**	
	1	**2**	**8**	**15**	**9**	
i=5	**1**	**2**	**8**	**15**	**9**	segéd=9
	1	**2**	**8**	**15**	**15**	
végeredmény	**1**	**2**	**8**	**9**	**15**	

Helyfoglalás:
N+1

Hasonlítások száma:
N-1 - N*(N-1)/2

Mozgatások száma:
2*(N-1) - 2*(N-1)+N*(N-1)/2

A csere 3 mozgatásnak felel meg!

8.7.7. Egyéb rendezések

Hat egyszerű rendező algoritmust ismertünk meg az előzőekben. Most felvázolok néhány újabbat, de csupán a gondolatot fogalmazom meg, hogy aki érdeklődik, az utána nézhessen.

Szétosztó rendezés

Feltételezzük, hogy az adatok rekordok, és a mezők között létezik kulcsmező, melyből nincs két egyforma értékű, és ráadásul a kulcs 1 és N közötti egész számérték (ezt a megfeleltetést megtehetjük). A kulcsmező itt egyértelműen megadja, hogy az elemnek hol a helye a rendezett sorozatban. Az eredmény egy újabb tömbben születhet meg a legegyszerűbben.

Számlálva szétosztó rendezés

A kulcsmező intervalluma lehet szélesebb, mint az előző, és nem követeljük meg a különböző kulcsokat. Megszámoljuk, hogy minden lehetséges kulcsértékből hány darab fordul elő. Ezt követően minden lehetséges kulcsértékhez meghatározzuk, hogy hány nála kisebb, vagy egyenlő kulcsú rekord van. Harmadik lépésként minden rekordot közvetlenül a helyére teszünk a fenti információ birtokában.

Számláló rendezés

Az egyszerű cserés rendezésnél ne cserélgessünk, hanem számoljuk össze, hogy a vizsgált elemnél hány kisebb van, és hogy az őt megelőzők közül hány vele egyenlő van (őt is beleértve). Ezzel az elemekhez a [0..N-1] egészekből álló intervallum egy-egy értékét rendeltük. Ez az adat lehet a kulcsmező a szétosztó rendezéshez.

Shell módszer

Először a nagy távolságú elemeket hasonlítjuk és cseréljük, mert így az egyes elemek nagyon gyorsan közel kerülhetnek végleges helyükhöz. Ezt a módszert lehet például a beillesztéses rendezésre alkalmazni.

A fenti algoritmusokat akár az internet segítségével is megkeresheted, és kipróbálhatod.

8.8. Keresések

A 6. fejezet (Elemi algoritmusok I) 5. leckéjében már megismerkedtünk egy kereséssel: a lineáris kereséssel. Nevét onnan kapta, hogy az átlagos összehasonlítások száma arányos az elemszámmal. Nézzük meg, hogyan módosítható az algoritmus, ha tudunk valamit a sorozatról!

8.8.1. Keresés rendezett sorozatban

Ha a sorozat rendezett, amelyben egy adott értékű elem sorszámát kell meghatározni (például az adott sorszámhoz tartozó többi adat kinyerése céljából), akkor elegendő addig vizsgálni a sorozat elemeit, amíg a keresettnél nagyobbat nem találunk,hiszen a rendezettség miatt a továbbiakban már biztosan nem találhatunk egyezőt. Az algoritmus tehát így módosul:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat, rendezett!!!]
      E: ElemTípus               [a keresett elem]
      VAN: logikai               [az eredmény]
      SORSZ: egész               [a megfelelő elem sorszáma, ha van ]
Eljárás Keresés_rendezettben(Konstans N,A,E Változó: VAN,SORSZ):
  I:=1
  Ciklus amíg i<=N és A(i)<E
    i:=i+1                [ha a vizsgált elem nem jó, továbbmegyek]
  Ciklus vége
  VAN:=(i<=N) és A(i)=E          [még benne vagyok a sorozatban és éppen E]
  Ha VAN akkor SORSZ:=i
  Elágazás vége
Eljárás vége.
```

TP kód

```
Procedure Kereses_rendezettben(Const N:Integer; Const A:Sorozat;
                    Const E:ElemTipus; Var VAN:Boolean; Var SORSZ:Integer);
Var i:Integer;
Begin
  i:=1;
  While (i<=N) and (A[i]<E) do i:=i+1;   (*precedencia miatt zárójelezés*)
  VAN:=(i<=N) and (A[i]=E);
  If VAN then SORSZ:=i;
End;
```

Észrevételek:

A **VAN:=(i<=N) and (A[i]=E)** utasítás nem mindig értelmes. Ha a sorozat mindegyik eleme kisebb, mint a keresett, akkor i értéke N+1 lesz, és nincs a sorozatnak N+1-edik eleme.

A TP balról jobbra értékeli ki a kifejezést, és az első feltétel hamis volta esetén az és logikai műveleti jobboldalát nem értékeli ki, ezért nem okoz hibát.

Milyen összehasonlítás szám jellemzi ezt az algoritmust?

- Minimum: 1
- Maximum: N
- Átlag: (N+1)/2

Az átlagos összehasonlítás-számban lévő lineáris összefüggés miatt ez is lineáris keresés, de hatékonyabb mint a korábbi fejezetben megismert változat.

8.8.2. Logaritmikus keresés

Most is rendezett sorozatban keressük egy adott értékű elem sorszámát. A rendezettséget másként is kihasználhatjuk, mint az előbb. Első lépésként ne a sorozat első elemét vizsgáljuk, hanem a középsőt. Ha ez a keresett elem, akkor készen is vagyunk. Ha a középső elem kisebb a keresettnél, akkor a sorozat következő részében kell tovább keresni. Ha a középső elem kisebb a keresettnél, akkor pedig a sorozat első részében kell keresni. Ezt az ötletet már használtuk a számkitalálós programban is. Készítsük el az algoritmust!

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat, rendezett!!!]
      X: ElemTípus               [a keresett elem]
      VAN: logikai               [az eredmény]
      SORSZ: egész               [a megfelelő elem sorszáma, ha van ]
Eljárás Logaritmikus_Keresés(Konstans N,A,X Változó: VAN,SORSZ):
  E:=1 : U:=N
  Ciklus
    K:=EgészRész((E+U)/2)        [index csak egész lehet]
    Elágazás
      X<A(K) esetén U:=K-1
      X>A(K) esetén E:=K+1
    Elágazás vége
  amíg E<=U és A(K)<>X
  ciklus vége
  VAN:=(E<=N)                    [még benne vagyok a sorozatban]
  Ha VAN akkor SORSZ:=i
Eljárás vége.
```

TP kód

```
Procedure Logaritmikus_Kereses(Const N:Integer; Const A:Sorozat;
                               Const X:ElemTipus; Var VAN:Boolean;
                               Var SORSZ:Integer);
Var E,U,K:Integer;
Begin
  E:=1; U:=N;
  Repeat
    K:= (E+U) div 2;
    If X<A[K] then U:=K-1
    else If X>A[K] then E:=K+1;
  until (E>U) or (A[K]=X)                (* not(E<=U és A(K)<>X) *)
  VAN:=(i<=U);
  If VAN then SORSZ:=K;
End;
```

Milyen összehasonlítás szám jellemzi ezt az algoritmust?

- Minimum: 1
- Maximum: $\log_2(N)+1$
- Átlag: $\log_2(N)$

Innen ered az algoritmus elnevezése: logaritmikus keresés. Szokás még felezéses, vagy bináris keresésnek is nevezni.

A logaritmikus keresésből megalkothatók az alábbi algoritmusok is (rendezett sorozatokra):

- eldöntés,
- kiválasztás,
- megszámolás,
- kiválogatás.

Ezek elkészítését az olvasóra bízom.

8.8.3. Visszalépéses keresés

8.8.3.1. feladat: Helyezzünk el a sakktáblán 8 vezért úgy, hogy egyik se üsse a másikat!
8.8.3.2. feladat: Legyen a mesebeli faluban a gyermekes családok száma N, a mikulásoké pedig M. Tudjuk, hogy egy család hány ajándékot szeretne, és hogy mely mikulásokkal áll kapcsolatban. Adjuk meg, hogy melyik család melyik mikulástól kérjen ajándékokat, ha egy család csak egy mikulástól rendelhet, és ismerjük, hogy az egyes mikulások puttonyaiba hány darab ajándék fér!

A feladatok közös jellemzője, hogy megoldásuk egy sorozat:
A 8.8.3.1. feladat esetén a sorozat i-edik eleme megadja, hogy a tábla i-edik oszlopának hányadik sorban van a vezér (nyilvánvaló, hogy egy oszlopban csak egy vezér lehet).
A 8.8.3.2. feladat esetén a sorozat a családokat reprezentálja, és egyes elemei az adott családhoz tartozó mikulás sorszámát tartalmazza.

Az algoritmus legfelső szintjén az i-edik oszlopban állva keresünk megfelelő elemet (egy sorszámot). Ha találunk abban az oszlopban megfelelő elemet, akkor azt eltároljuk az X megoldás sorozatban, az i-edik indexen. Ha nem találunk, akkor X sorozat jelenlegi indexű elemébe 0-t teszünk – azaz nincs kiválasztott elem), és visszalépünk X előző indexére, és ott próbálunk meg mást választani – ez a visszalépés lényege. Sikeres a keresés, ha X sorozat minden helyére találtunk értéket. Sikertelen a keresés, ha már az első helyről is visszalépést kellene csinálni. Az algoritmus:

```
Változó:
   N:egész                        [a sorozat elemszáma]
   M:Tömb(1..N:egész)      [melyik oszlopban meddig mehetünk]
   VAN:logikai             [van-e megoldása a feladatnak]
   X:Tömb(1..N:egész)           [megoldás: mindegyik eleme egy
sorszám]

Eljárás Visszalépéseskeresés(Konstan N,M, Változó VAN,X):
  I:=1
  X[1..N]:=(0,…,0)         [egyik helyre sem választottunk]
  Ciklus amíg i>=1 és i<=N [sikeres, v. sikertelen végkifejlet]
    JóesetKeresés(M,X,i,VAN,MELYIK):
    Ha VAN akkor X(i):=MELYIK : i:=i+1 [talált megoldás, tovább]
           Különben X(i):=0 : i:=i-1    [nincs mo., visszalép]
  Ciklus vége
  VAN:=(i>=1)
Eljárás vége.
```

A 8 vezéres feladatnál N értéke 8, és M vektor mind a nyolc eleme is 8, hiszen 8x8-as sakktáblán dolgozunk. X vektor i-edik elemébe az adott oszlopban lévő vezér sorának száma (1..N) kerül majd.
A mikulásos feladatban N a családok száma, M(i) az összes mikulás száma, tehát minden értéke M. Az X vektor pedig a mikulások sorszámát fogja tartalmazni 1 és M között.

Hogyan találjunk az i-edik helyre elemet? Ezt végzi a **JósetKeresés** eljárás. Ciklikusan kiválasztja az eredménytömb megfelelő indexű helyén álló értéket követőt, ha azt még szabad (nem nagyobb a lehetséges mélységnél: M(i)), önmagában ott kiválasztható (a tulajdonságfüggvény – ft adja meg), és nem **RosszEset** ez a választás.

```
JóesetKeresés(Konst. M,X,i, Vált. VAN,MELY):
  MELY:=X(i)+1              [a következő lehetségest választja]
  Ciklus amíg MELY<=M(i) és
              (RosszEset(i,X,MELY) vagy nem ft(i,MELY))
     MELY:=MELY+1
  Ciklus vége
  VAN:=(MELY<=M(i))
Eljárás vége.
```

A 8 vezéres feladatnál mindegyik M(i) érték 8. Önmagában minden helyre letehető a vezér, így nincs szükség tulajdonság függvényre (**ft**).
A mikulásos feladatnál már említettem, hogy mindegyik M(i) értéke M, azaz a mikulások száma. A tulajdonságfüggvény (**ft**) vizsgálhatja, hogy az adott család kapcsolatban áll-e az adott mikulással.

Az eddigieken túl rossz egy választás, ha a kölcsönösségi függvény (**fk**) szerint a korábbi értékek miatt nem választható az óhajtott érték. Itt feltételezzük, hogy a kölcsönösségi függés elem-páronként nézve eldönthető (8 vezéres példa ilyen).

```
Függvény RosszEset(I,X,MELY):
  J:=1
  Ciklus amíg j<i és fk(i,MELY,j.X(j))
    j:=j+1
  Ciklus cége
  Rosszeset:=(i<j)
Függvény vége.
```

A 8 vezéres feladatban a kölcsönösségi függvény vizsgálja, hogy valamelyik korábban elhelyezett vezérrel páronként ütik-e egymást.
A mikulásos feladatban a kölcsönösségi függvény segítségével határozhatjuk meg, hogy a kiválasztott mikulás a családnak megfelelő darabszámú ajándékot még ki tud-e vinni. Ebben az esetben nem lehet kettesével vizsgálni a kiválasztott elemeket, a kölcsönösségi függvény az összes korábbi ugyanarra a mikulásra vonatkozó választásból számolhat puttony-telítettséget összegzéssel.

Miért nem alkalmazzuk az ilyen feladatokra azt a megoldást, hogy előállítjuk az összes lehetséges állapotot, és keresünk közülük egy megfelelőt? A 8x8-as sakktáblán 8^8 féle elhelyezés képzelhető el. Ez pontosan 16 777 216 eset. Ha el is tudnánk tárolni ennyi lehetőséget, akkor sem lenne hatékony ez a módszer.

Csak egy megoldás megtalálására alkalmas a visszalépéses keresés? Természetesen nem.
Az összes megoldás megkereshető:
A VisszalépésesKeresés eljárásban i:=i+1 helyett írjuk ezt:

```
   Ha i=N akkor Ki:X különben i:=i+1
```

Így siker esetén is tovább próbálkozunk.

Optimális megoldást is lehet vele keresni (például legolcsóbbat,…):
Ha a VisszalépésesKeresés eljárásban i:=i+1 helyett írjuk ezt:

```
   Ha i=N és Költség(X)<MIN akkor MIN:=Költség(x) különben i:=i+1
```

Kiválasztás és megszámolás is megvalósítható visszalépéses kereséssel.

8.9. Összefoglalás

A fejezet első felében sorozatokhoz sorozatot (sorozatokat) hozzárendelő algoritmusokat ismertünk meg:

- másolás
- kiválogatás (másik sorozatba, kiírással, helyben, kihúzással),
- szétválogatás (két sorozatba, egy sorozatba, helyben),
- metszet
- unió
- összefuttatás.(rendezettek uniója), összefésülés.

A fejezet második felében a sorozatok eleminek rendezésére készítettünk algoritmusokat. Megállapítottuk, hogy „legjobb” nem létezik, más-más adatsorozatra más-más algoritmus ad hatékonyabb megoldást. A részletesen kidolgozott keresések:

- egyszerű cserés,
- minimumkiválasztásos,
- buborékos,
- javított buborékos,
- beillesztéses,
- javított beillesztéses.

Felvázoltuk még néhány rendezési algoritmus elvét (szétosztó, számlálva szétosztó, számláló, shell).

Ezt követően rendezett sorozatokban kerestük konkrét elem sorszámát a módosított lineáris kereséssel és a logaritmikus kereséssel. Megismertük a visszalépéses keresés algoritmusát is.

8.10. Gyakorló feladatok

1. Van több rekordokat tartalmazó sorozatunk, melyek az iskolai kosárlabda csapatok adatait tartalmazzák. Egy rekordban egy játékos adatai vannak: mezszáma, neve, magassága. Készítsünk programot, amely választ ad az alábbi kérdésekre:
2. Válogassuk ki egy csapat 190 cm-nél magasabb játékosait (külön sorozatba, és kiírással)!
3. Adjuk meg az összes kosárlabdázó játékos nevét!
4. Írjuk ki azon játékosok nevét, akik mindegyik csapatban játszanak (ha van ilyen)!
5. Írjuk ki a csapatok tagjainak nevét névsorban!
6. Van-e a játékosok között Kovács vezetéknevű?
7. Állíts össze egy olyan csapatot, amelyikben minden játékos 190 cm-nél magasabb, és a vezetéknevük nem kezdődhet azonos betűvel!
8. Kódold a részletesen tárgyalt 6 rendezést egy programban, és egy véletlenszerűen generált sorozatra vizsgáld meg, hogy melyik algoritmus ad a leggyorsabban eredményt. A **GetTime(ora,perc,masodper,szazadmasodperc)** eljárás a paraméterekbe tölti az aktuális óraállást, ennek segítségével mérhető a futásidő.

9. Összetett feladatok

Az élet ritkán olyan kegyes hozzánk, hogy a megoldandó feladat egyetlen elemi algoritmus alkalmazásával megoldható lenne. Az esetek többségében több elemi algoritmus egymás utáni alkalmazásával juthatunk el az eredményhez. Most nem általános megoldást keresünk, hanem megnézünk néhány példát

9.1. Elemi algoritmusok összeépítése

9.1. feladat: Határozzuk meg egy N elemű egészekből álló sorozat négyzetösszegét!
9.2. feladat: Ha van, adjuk meg egy N elemű egészekből álló sorozat 5. páros elemét!
9.3. feladat: Hány darab maximális eleme van egy N elemű egészekből álló sorozatnak?
9.4. feladat: Határozzuk meg egy N elemű egészekből álló sorozat páros elemeinek összegét!
Vegyük sorra a feladatokat! Az algoritmusok mellett, most a TP kódot is elkészítjük.

A 9.1. feladat megoldható úgy, hogy másolás tétellel előállítjuk az elemek négyzetét tartalmazó sorozatot, majd ezen az új sorozaton alkalmazzuk a sorozatszámítás tételt. A másolás tétellel a sorozatszámítás összeépíthető, így egy algoritmusban elvégezhető a feladat: Induljunk ki a sorozatszámítás tétel összegzésre elkészített változatából:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]

Eljárás Összegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    S:=S+A(i)
  Ciklus vége
Eljárás vége
```

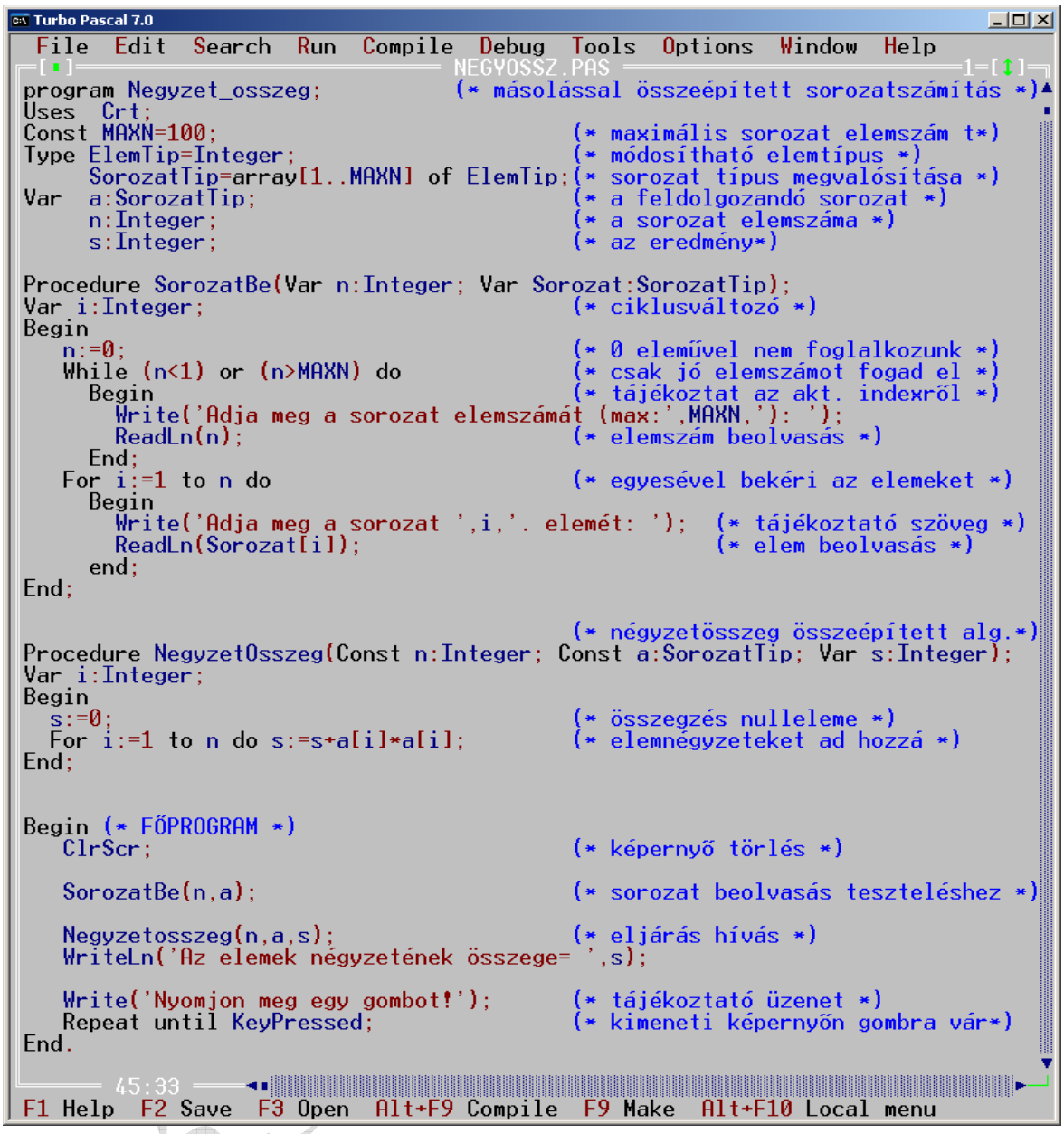
Egyetlen helyen kell módosítani: mivel nem az elemeket, hanem a négyzetüket kell összeadni, ezért az eddigi összeghez nem A(i)-t, hanem A(i)*A(i)-t kell írni. Az összeépített algoritmus:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]

Eljárás Összegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    S:=S+A(i)*A(i)               [az elem négyzetét adja hozzá]
  Ciklus vége
Eljárás vége
```

Vegyük elő a 6. fejezetben elkészített keretprogramot, és készítsünk egy eljárást benne **„Negyzetosszeg”** néven! A főprogramban olvastassunk be egy sorozatot (billentyűzetről), hívjuk meg a négyzetösszeget előállító eljárást, és írassuk ki az eredményt. A program kódja:



A 9.2. feladatban addig kell páros elemet keresni, amíg az ötödiket meg nem találjuk, vagy elértük a sorozat végét. Ez a feladat tehát a megszámolás és a keresés egymásra építésével oldható meg. Induljunk ki a keresés algoritmusából (a párosság vizsgálat is benne van):

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      VAN: logikai               [az eredmény]
      SORSZ: egész               [a megfelelő elem sorszáma, ha van ]
      ELEM: ElemTípus            [a megfelelő elem, ha van]

Eljárás Kereses(Konstans N,A, Változó: VAN,SORSZ,ELEM):
  I:=1
  Ciklus amíg i<=N és Maradék(A(i),2)<>0
    i:=i+1                [ha a vizsgált elem nem jó, továbbmegyek]
  Ciklus vége
  VAN:=(i<=N)             [az i<=N reláció logikai értéke lesz az eredmény]
  Ha VAN akkor SORSZ:=i : ELEM:=A(i)
  Elágazás vége
Eljárás vége.
```

Alakítsuk át az algoritmust úgy, hogy a jó elemeket számolja, és csak az ötödikig keressen:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      K: egész                   [az ennyiedik páros elemet keressük]
      VAN: logikai               [az eredmény]
      SORSZ: egész               [a megfelelő elem sorszáma, ha van ]
      ELEM: ElemTípus            [a megfelelő elem, ha van]

Eljárás Megszámolás_Keresés(Konstans N,A,K Változó: VAN,SORSZ,ELEM):
  I:=0
  Ciklus amíg i<=N és DB<K
    i:=i+1                [ha a vizsgált elem nem jó, továbbmegyek]
    Ha Maradék(A(i),2)=0 akkor DB:=DB+1
  Ciklus vége
  VAN:=(DB=K)                    [a K-adik párosnál állt meg, akkor igaz]
  Ha VAN akkor SORSZ:=i : ELEM:=A(i)
  Elágazás vége
Eljárás vége.
```

Készítsük el ismét a TP kódot is, a 6. fejezetben kidolgozott keretprogram alapján!

Turbo Pascal 7.0

File Edit Search Run Compile Debug Tools Options Window Help

MEGSZKER.PAS

```
program Megszamolas_Kereses;          (* megszámolással összeépített keresés *)
Uses  Crt;
Const MAXN=100;                       (* maximális sorozat elemszám t*)
Type ElemTip=Integer;                 (* módosítható elemtípus *)
     SorozatTip=array[1..MAXN] of ElemTip;(* sorozat típus megvalósítása *)
Var  a:SorozatTip;                    (* a feldolgozandó sorozat *)
     n:Integer;                       (* a sorozat elemszáma *)
     Van_e:Boolean;                   (* keresés által visszadott érték *)
     Index:Integer;                   (* keresett elem sorszáma *)
     Ertek:ElemTip;                   (* a keresett elem *)

Procedure SorozatBe(Var n:Integer; Var Sorozat:SorozatTip);
Var i:Integer;                        (* ciklusváltozó *)
Begin
   n:=0;                              (* 0 eleműve nem foglalkozunk *)
   While (n<1) or (n>MAXN) do         (* csak jó elemszámot fogad el *)
     Begin                            (* tájékoztat az akt. indexről *)
       Write('Adja meg a sorozat elemszámát (max:',MAXN,'): ');
       ReadLn(n);                     (* elemszám beolvasás *)
     End;
   For i:=1 to n do                   (* egyesével bekéri az elemeket *)
     Begin
       Write('Adja meg a sorozat ',i,'. elemét: ');  (* tájékoztató szöveg *)
       ReadLn(Sorozat[i]);                           (* elem beolvasás *)
     end;
End;

                                     (* megszámolás és keresés összeépítése *)
Procedure MegszamolasKereses(Const n:Integer; Const Sorozat:SorozatTip;
      Const K:Integer; Var VAN:Boolean; Var SORSZ:Integer; Var ELEM:ElemTip);
Var i,DB:Integer;                     (* ciklusváltozó, segédváltozó *)
Begin                                 (* kezdeetben 0 db páros elem van *)
   i:=0; DB:=0;                       (* i:=0, mert a ciklusban növelem *)
   While (i<n) and (db<K) do          (* a sorozat végéig, és DB=K-ig *)
     Begin
       i:=i+1;                        (* ezért indult 0-ról*)
       If (Sorozat[i] MOD 2 = 0) then DB:=DB+1;   (* ha páros, akkor db növ.*)
     End;
     VAN:=(DB=K);                     (* igaz, ha K db-ot találtunk *)
     If VAN then
       Begin
         SORSZ:=i;                    (* viasszadandó elemsorszám *)
         ELEM:=Sorozat[SORSZ];        (* visszaadandó elem *)
       End;

End;
```

```
Begin (* FŐPROGRAM *)
   ClrScr;                                  (* képernyő törlés *)

   SorozatBe(n,a);                          (* sorozat beolvasás teszteléshez *)

   MegszamolasKereses(n,a,5,Van_e,Index,Ertek);  (* eljárás hívás *)
                                            (* eredmény kijelzés *)
   If Van_e then WriteLn('5. páros elem sorszáma: ',Index,' értéke: ',Ertek)
            else WriteLn('Nincs 5. páros elem.');

   Write('Nyomjon meg egy gombot!');        (* tájékoztató üzenet *)
   Repeat until KeyPressed;                 (* kimeneti képernyőn gombra vár*)
End.
                29:35
F1 Help  F2 Save  F3 Open  Alt+F9 Compile  F9 Make  Alt+F10 Local menu
```

Ellenőrizzük a programot a feltételt kielégítő és ki nem elégítő sorozattal is.

A 9.3. feladat megoldható úgy, hogy megkeresem az elemek maximumát (maximumkiválasztás), majd megszámolom, hogy hány ilyen értékű elem van benne (megszámolás). A két algoritmus azonban összeépíthető: a maximumkiválasztás algoritmusa során az aktuális maximum beállításakor egy darabszám változót állítsunk 1-re. Ha az aktuális maximummal megegyező elemet találunk, akkor növeljük a darabszámot, ha nagyobb elemet találunk, akkor ez lesz az új maximum és a darabszám ismét 1. Nézzük meg először a maximumkiválasztás algoritmusát!

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      MAXSORSZ: egész            [az egyik eredmény: a sorszám]
      MAX: ElemTípus             [a másik eredmény: az érték]

Eljárás MaximumKiválasztás(Konstans N,A, Változó: MAXSORSZ:egész;
                           Változó MAX:ElemTípus):
  MAXSORSZ:=1 : MAX:=A(1)
  Ciklus i:=2-től N-ig
    Ha A(i)>MAX akkor MAXSORSZ:=i      [ha a vizsgált nagyobb, akkor]
                      MAX:=A(i)        [ezt tárolom]
  Ciklus vége
Eljárás vége.
```

Írjuk bele az azonos maximális elemek megszámolását (a sorszámnak nincs értelme)!

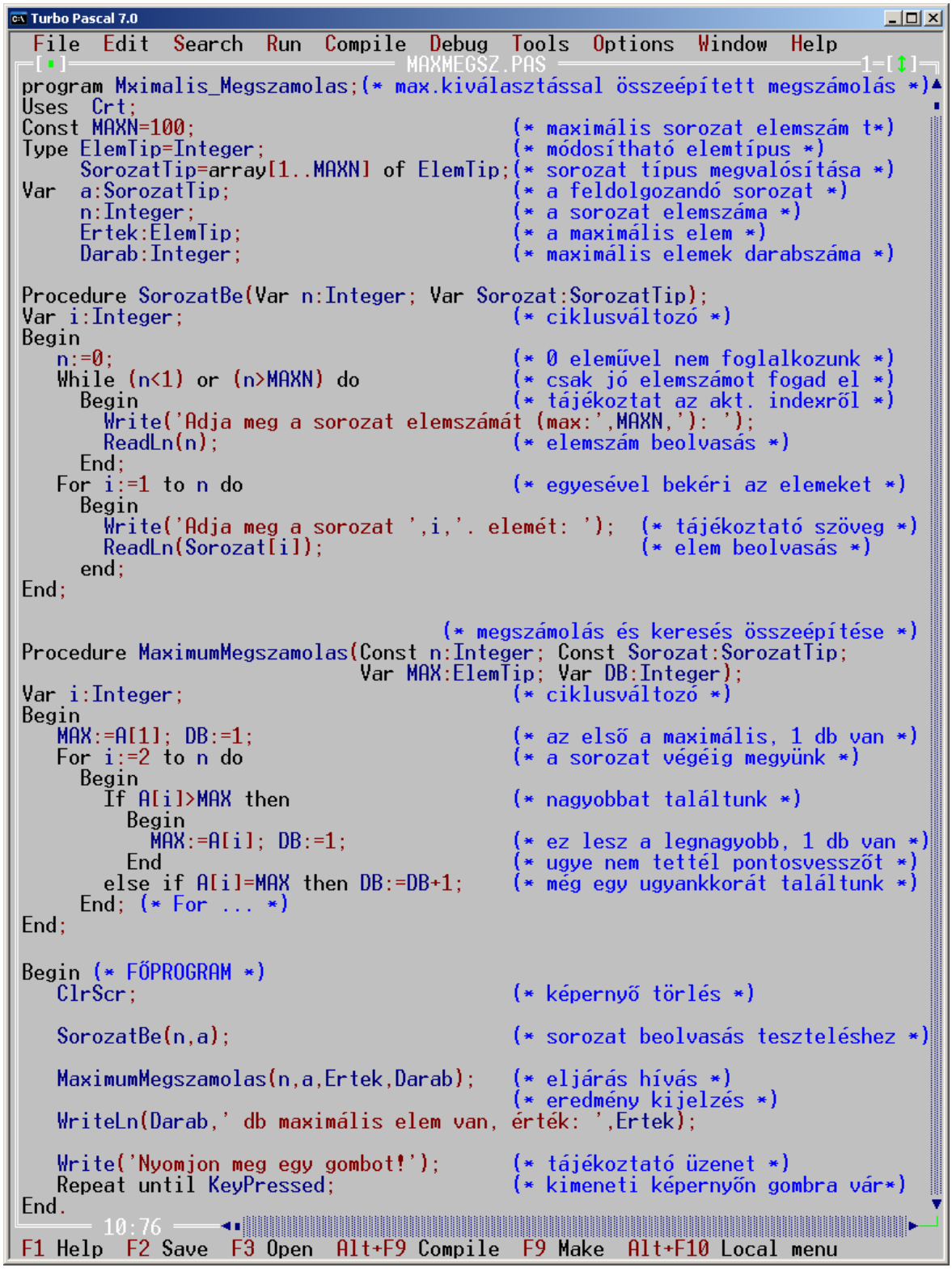
Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      MAX: ElemTípus             [a másik eredmény: az érték]
      DB: egész                  [a maximális elemek darabszáma]

Eljárás MaximumMegszámolás(Konstans N,A, Változó MAX:ElemTípus, DB:egész):
  MAX:=A(1) : DB:=1
  Ciklus i:=2-től N-ig
    Elágazás
      A(i)>MAX esetén: MAX:=A(i) : DB:=1     [nagyobbat találtunk]
      A(i)=MAX esetén: DB:=DB+1              [ugyanakkorát találtam]
    Elágazás vége
  Ciklus vége
Eljárás vége.
```

Készítsük el a TP kódot a keretprogram felhasználásával!

A Turbo Pascal kód:



Ellenőrizzük a program működését többféle bemenő sorozattal (1 db maximális elem, több maximális elem,…)!

A 9.4. feladatban az N elemű egészekből álló sorozat páros elemeinek összegét meghatározhatjuk úgy, hogy kiválogatjuk a páros elemeket (kiválogatás) és összeadjuk azokat (sorozatszámítás). Megoldhatjuk a feladatot úgy is, hogy a sorozatszámítás tétel belsejében csak a megfelelő elemeket összegezzük, azaz összeépítjük a kiválogatást a sorozatszámítással.

A sorozatszámítás algoritmusa összegzésre:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]
Eljárás Összegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    S:=S+A(i)
  Ciklus vége
Eljárás vége
```

Módosítsuk az algoritmus ciklusmagját úgy, hogy csak akkor adja hozzá az eddigi összeghez az új elemet, ha az pozitív.

Az összeépített algoritmus általánosan:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]
Függvény:
      Jó: ElemTípus -> logikai   [az elem megfelelő-e]
Eljárás KiválogatásÖsszegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    Ha Jó(A(i)) akkor S:=S+A(i)
  Ciklus vége
Eljárás vége
```

A „Jó” függvény a konkrét példára:

Alg.

```
Függvény Jó(Konstans E:egész): logikai;
           [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Jó:=(Maradék(E,2)=0)
Eljárás vége.
```

Az egyszerűbb algoritmus érdekében az eljárás ciklusmagjába írjuk be a „Jó” függvényt!

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]
Eljárás KiválogatásÖsszegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    Ha Maradék(A(i),2)=0 akkor S:=S+A(i)
  Ciklus vége
Eljárás vége
```

Készítsük el ezt a programot is! A kód a következő oldalon látható.

Ellenőrizd a programot csak páros, csak páratlan, illetve páros és páratlan elemeket tartalmazó sorozatra is!

9.2. Összefoglalás

Ebben a fejezetben elemi algoritmusok egymás utáni végrehajtásával megoldható feladatok algoritmusát és pascal kódját készítettük el az elemi algoritmusok összeépítésével.

Természetesen az érettségin, illetve versenyen nem feltétlenül kell hatékony algoritmust készíteni összeépítéssel, de néha az összeépítés jelentősen csökkentheti a kódolás idejét és a memória felhasználást.

9.3. Gyakorló feladatok

1. Egy N elemű sorozatban tároljuk az iskolai kosárlabda csapat játékosainak adatait. Egy-egy játékos adata egy-egy rekordban van (mezszám, név, magasság, életkor). Adjunk választ a következő kérdésekre:
2. Adjuk meg a 16 évnél fiatalabb játékosok átlagéletkorát!
3. Adjuk meg a mezszám szerint rendezett sorozatból a harmadik 180 cm-nél magasabb játékos nevét!
4. A legkisebb magasságú játékosból hány egyforma van?