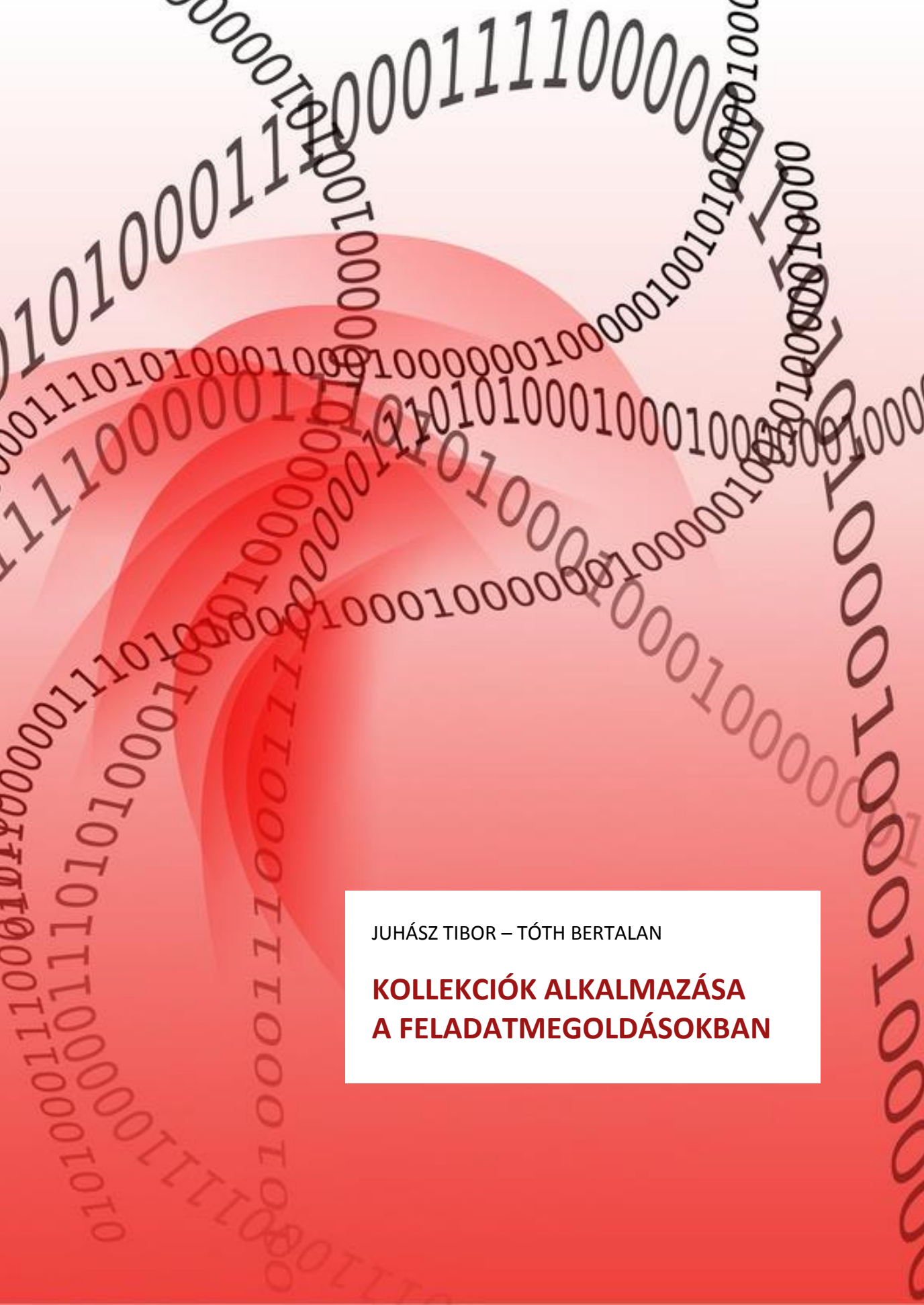


JUHÁSZ TIBOR – TÓTH BERTALAN

KOLLEKCIÓK ALKALMAZÁSA A FELADATMEGOLDÁSOKBAN

Juhász Tibor – Tóth Bertalan:

Kollekciók alkalmazása a feladatmegoldásokban

2., átdolgozott kiadás

– 2015 –

Jelen dokumentumra a Creative Commons Nevezd meg! – Ne add el! – Ne változtasd meg! 3.0 Unported licensz feltételei érvényesek: a művet a felhasználó másolhatja, többszörözheti, továbbadhatja, amennyiben feltünteti a szerzők nevét és a mű címét, de nem módosíthatja, és kereskedelmi forgalomba se hozhatja.

Lektorálta: Tóth Bertalan

© Juhász Tibor, Tóth Bertalan, 2013, 2015

Tartalom

1. Bevezetés

A középiskolában oktatott programozási nyelvek nem mindegyike ismeri a fejlett adatszerkezeteket. Ezért a *Programozási ismeretek haladóknak* (Műszaki Kiadó, 2012) című könyv *III. Algoritmusok* fejezetében bemutatott algoritmusok statikus tömbökkel oldják meg a feladatokat. Egy alkalmasabb adattípus kiválasztásával azonban megkönnyíthetjük a programírást. Általában is elmondhatjuk, hogy minél összetettebb adatszerkezetet használunk, annál egyszerűbb az algoritmus programozása és fordítva, minél egyszerűbb az adatszerkezet, annál bonyolultabb lesz egy-egy algoritmus kódolása. Szélsőséges esetben az adatbeolvasást és -tárolást követően máris rendelkezésünkre állnak a válaszok a feltett kérdésekre.

Példatárunkban – főleg versenyfeladatokon keresztül – bemutatjuk a kollekciók alkalmazását. Elsődleges célunk, hogy a diákok minél sikeresebben szerepeljenek a programozási versenyeken. Ennek rendeljük alá eszközeinket, módszereinket. Márpedig egy versenyen (vagy akár az érettségin) a hatékonyság helyett/mellett előnyben részesítjük a gyorsan megtervezhető és gyorsan begépelhető (!) programokat. ☺ Megjegyezzük, hogy a megoldásokat tovább egyszerűsíthetnénk a LINQ eszközeivel, de ezeket csak a .NET-ben támogatott nyelvek ismerik.

A megoldásoknál feltételezzük a *Programozási ismeretek* (Műszaki Kiadó, 2011), tankönyv anyagának, illetve a *Programozási ismeretek haladóknak* (Műszaki Kiadó, 2012) tankönyv II. fejezetének (*Összetett adattípusok*) ismeretét. Minden témakör elején felsoroljuk az adott témához kapcsolódó tankönyvi leckéket. Javasoljuk az Olvasónak, hogy ezek alapján először ismételje át a szükséges tudnivalókat.

Példatárunk leckéi egymásra épülnek, tehát például a halmazok bemutatásánál még nem alkalmazunk listát, struktúrát vagy algoritmusokat megvalósító metódusokat. Célszerű a későbbiekben visszatérni az előző leckék feladataihoz, és az újabb eszközökkel egyszerűsíteni a megoldást.

Az első korcsoport versenyzőinek főleg a halmazokkal és az asszociatív tömbökkel foglalkozó részeket ajánljuk. Ezért a halmazoknál elkülönítettük az 1. korcsoport feladatait, az asszociatív tömbökről szóló rész pedig teljes egészében ilyen versenyfeladatokat tartalmaz. De a további leckékben is gyakran 1. korcsoportos feladatokkal kezdjük az adott kollekció gyakorlását.

A programozási versenyfeladatok nem a kollekciók használatának bemutatását szemléltetik. Ezért előfordulhat, hogy egy-egy feladatot egyszerűbben oldhatunk meg másfajta kollekció, esetleg statikus tömb alkalmazásával, mint ahogy az a feladatgyűjteményben szerepel.

Miután elsajátítottuk a kollekciók biztonságos kezelését, az algoritmusok gyakorlásával folytathatjuk a programozási versenyekre való felkészülést. Ehhez nyújt segítséget a *Programozási ismeretek versenyzőknek* (Műszaki Kiadó, 2015) példatár.

A szerzők köszönetet mondanak Zsakó Lászlónak, amiért engedélyezte a versenyfeladatok szövegének közlését.

A feladatok megoldásairól

A bemutatott feladatoknál megadjuk a verseny évét, a korcsoportot, a forduló és a feladat sorszámát. A 3. korcsoport megfelel az országos középiskolai tanulmányi versenynek (OKTV).

A Nemes Tihamér verseny pontos megnevezése az elmúlt évek során megváltozott. Itt a jelenleg érvényes jelölést használjuk (OITV). A feladatok eredeti megfogalmazását csak ritkán és minimális mértékben módosítottuk (ha például nem volt megadva a program neve).

A megoldások mondatszerű leírásában általában nem szerepeltetjük a kollekciókat létrehozó utasításokat. Az újonnan létrejövő kollekciót üresnek tekintjük (azaz nem tartalmaz elemeket). Feltételezzük, hogy a kollekció elemeire indexükkel is hivatkozhatunk.[1] Az elemeket 0-val kezdve, a hozzáadás sorrendjében indexeljük. A rendezett kollekció pedig növekvő sorrendben indexeli az elemeit. A statikus tömbök deklarációjában feltüntetjük az indexhatárokat (például: 1…*N*).

A tankönyvekben alkalmazott jelölésmódnak megfelelően a kettős kereszt (#) a logikai rövidzárat jelenti (például *ÉS#*). Az értékadásnál egyenlőségjelet (=), az egyenlőség relációnál két egyenlőségjelet használunk (= =).

Az egyes leckék végén olyan további versenyfeladatokat is felsorolunk, melyek megoldásával elmélyíthető a bemutatott kollekció használata. A feladatokra a verseny archívumában megszokott módon hivatkozunk (évszám – a tanév második félévének éve, majd a forduló és a korcsoport megjelölése).

A leckékben szereplő példák, illetve feladatok megoldása Visual Basic és C++ nyelven egyaránt letölthető a programozás tankönyvek webhelyéről:

www.zmgzeg.sulinet.hu/programozas

A Visual Basic programok a Microsoft Visual Studio Express for Windows Desktop ingyenes fejlesztőrendszerrel készültek. Használatukhoz készítsünk egy új projektet, majd a megoldást tartalmazó *Module1.vb* forrásfájllal írjuk felül a projekt azonos nevű állományát!

A C++ programok a C++11 szabványon alapulnak. Javasoljuk a Code::Blocks fejlesztőrendszer 13.12-es vagy újabb változatának a használatát. A forrásfájlok (programok és adatok) ANSI-kódolásúak, így állítsuk be a Windows-1250 kódolást (*Settings*/*Editor*, *General settings*/*Other settings*, *Encoding*)!

A forráskódban meghagytuk a feladatok szövegében szereplő jelöléseket és változóneveket. Ezeket a változókat, továbbá a tároláshoz/megoldáshoz szükséges kollekciókat többnyire modulszinten deklaráltuk. A többi változó deklarációja ott szerepel, ahol először szükség van rá.

A bemenő adatokat általában valamilyen kollekcióban tároljuk, majd ezt követi az algoritmus megvalósítása. Sok esetben már a beolvasás során elvégezhetnénk a feldolgozást. Így ugyan rövidebb, de nehezebben áttekinthető programot kapnánk. A megoldást inkább önálló részekre (például adatbeolvasás, feldolgozás, kiírás) bontjuk fel.

A teszteléshez lehetőség szerint a versenybizottság tesztprogramjait használtuk. Ha a tesztprogram maximális pontszámot adott a programra, akkor helyesnek tekintettük a megoldást. A feladatot csak annyira általánosítottuk, illetve csak olyan speciális ese-

[1] .NET alatt a *Dictionary*, *SortedDictionary* és *SortedList* elemeit a kulccsal indexeljük.

teket vettünk figyelembe, amilyeneket a megoldási útmutató az értékelésben felsorolt (lásd a versenyfeladatok forrását az irodalomjegyzékben).

Javasoljuk az Olvasónak, hogy írja meg a programokat csupán statikus tömbök és keresések alkalmazásával. Így tudja igazán értékelni a fejlett adattípusokat alkalmazó megoldások szépségeit. ☺

Felhasznált irodalom

Benkő Tiborné–Poppe András: Objektumorientált C++
(Együtt könnyebb a programozás, ComputerBooks, 2010)
Horváth Gyula–Zsakó László: Programozási versenyfeladatok tára III–IV.
(Neumann János Számítógép-tudományi Társaság, 2006, 2010)
Juhász Tibor–Kiss Zsolt: Programozási ismeretek (Műszaki Könyvkiadó, 2011)
Juhász Tibor–Kiss Zsolt: Programozási ismeretek haladóknak
(Műszaki Könyvkiadó, 2012)
Juhász Tibor–Tóth Bertalan: Programozási ismeretek versenyzőknek
(Műszaki Könyvkiadó, 2015)
Lambda Expressions (MSDN Library)
http://msdn.microsoft.com/en-us/library/vstudio/bb531253(v=vs.100).aspx
Ng, Timothy: Lambda Expressions (MSDN Magazine, September 2007)
http://msdn.microsoft.com/en-us/magazine/cc163362.aspx
Tóth Bertalan: Programozzunk C++ nyelven! Az ANSI C++ tankönyve
(ComputerBooks, 2003)
Zsakó László: Programozási versenyfeladatok tára I–II.
(Neumann János Számítógép-tudományi Társaság, 2002, 2005)

A versenyfeladatok és megoldási útmutatók forrása:
http://nemes.inf.elte.hu/nemes_archivum.html

2. Halmaz, rendezett halmaz

Kapcsolódó lecke:
Programozási ismeretek (Műszaki Kiadó, 2011)
40. A halmaz adatszerkezet

A halmazok nagymértékben megkönnyítik minden olyan feladat megoldását, amelyben az ismétlődő elemeket csak egyszer vehetjük figyelembe. Statikus tömbök használata esetén ilyenkor rengeteg kereséssel tudjuk kiküszöbölni az ismétlődéseket.

Bár a régebbi programozási nyelvek is rendelkeznek ezzel az adatszerkezettel, halmazaik erősen korlátozott méretűek, erősen korlátozott típusú elemekkel. Ráadásul, csupán lineáris kereséssel tudjuk ellenőrizni, hogy a halmaz tartalmaz-e egy meghatározott elemet. A modern programozási nyelvek halmaz típusára mindezek a korlátozások nem érvényesek. A halmazok elemei tetszőleges, akár összetett típusúak is lehetnek. Ez utóbbi esetben azonban az összehasonlítás (például a *Tartalmazza* metódus hívásánál) gyakran objektumorientált módszereket igényel.

A C++ STL-ben a halmaz (*set*) rendezetten tárolja az elemeket. Mivel a rendezettség megtartása eléggé időigényes, a C++11 szabvány rendezettség nélküli változattal bővítette a sablontárat (*unordered_set*). Rendezettség nélkül az új halmaztípusokkal hibás eredményt adnak a sablontár halmazműveleteket megvalósító algoritmusai (*set_intersection*, *set_union*, *set_difference*, *set_symmetric_difference*). Amennyiben nem okoz gondot a futásidő, javasoljuk a hagyományos halmaz sablon alkalmazását.

Visual Basic-ben a halmaz/rendezett halmaz elemeire indexükkel is hivatkozhatunk. A *Halmaz*(*I*) értéke azonban nem létező index esetén az elemtípus alapértelmezett értéke lesz (numerikus értékek esetén például 0). Az *ElementAt* metódus nem létező indexre való hivatkozásnál futási hibát okoz. Helyette használhatjuk az *ElementAtOrDefault* metódust, amely ilyenkor szintén az elemtípus alapértelmezett értékével tér vissza.
Két, érték típusú[2] elemeket tartalmazó halmaz egyenlőségét (elemenként összehasonlítva) a *Halmaz1*.*SetEquals*(*Halmaz2*) metódushívással vizsgálhatjuk meg. A visszatérési érték *True*, ha a két halmaznak azonosak az elemei.

A multihalmaz adatszerkezetet az asszociatív tömböknél ismertetjük.

Feladatok az 1. korcsoport számára

Lakások

Nemes Tihamér OITV 2013. 1. forduló 1. korcsoport 4. feladat

Egy ingatlanforgalmazó cég tárolja az eladó lakások alapterületét és árát (1≤*lakások száma*≤100). Írj programot *lakas* néven, amely megadja

a) a legdrágább lakás sorszámát (ha több megoldás van, akkor közülük a legkisebb sorszámút);
b) a 100 négyzetméternél nagyobbak közül a 40 millió forintnál olcsóbbak számát;
c) hányféle alapterületű lakás van!

[2] Lásd: Programozási ismeretek, 61. oldal.

A bemenet első sorában a lakások száma (≥1), alatta soronként egy-egy lakás alapterülete (négyzetméterben) és ára van (millió forintban, egész számok), egyetlen szóközzel elválasztva. A kimenetre soronként ki kell írni a fenti kérdésekre, feladatokra adott válaszokat!

Példa:

Bemenet	*Kimenet*	*Magyarázat*
`6`	`4`	a 4. lakás a legdrágább
`42 15`	`2`	2 lakás nagyobb 100 négyzetméternél,
`110 20`		de olcsóbb 40 millió Ft-nál
`125 160`	`4`	négyféle alapterületű lakás van
`166 180`		
`42 10`		
`110 39`		

Megoldás

A maximumkiválasztást és a számlálást beolvasás közben végezzük el. Ugyancsak beolvasás közben, a *Területek* halmazban gyűjtjük az alapterületeket:

```
VÁLTOZÓ Max, Melyik, Db MINT EGÉSZ
VÁLTOZÓ Területek MINT Halmaz(Elemtípus: Egész)
Be: N
CIKLUS I=1-től N-ig
    Be: Terület, Ár
    HA Max < Ár AKKOR Max = Ár, Melyik = I
    HA Terület > 100 ÉS Ár < 40 AKKOR Db = Db + 1
    Területek.Add(Terület)
CIKLUS VÉGE
```

Az adatok további feldolgozást nem igényelnek. Ki kell írni a *Melyik* és a *Max* változó értékét, illetve a *Területek* halmaz elemszámát.

A teljes megoldást a *lakas* program tartalmazza.

Pénz

Nemes Tihamér OITV 2011. 2. forduló 1. korcsoport 2. feladat

Bergengóciában N-féle pénzérmét használnak: $P(1)$, $P(2)$, …, $P(N)$ forintost.

Írj programot *penz* néven, amely megadja, hogy mely 1 és M forint közötti összegek fizethetők ki egyetlen pénzérmével, melyek legfeljebb 2 pénzérmével és melyek legfeljebb 3 pénzérmével!

A program olvassa be a pénzérmék számát ($1 \le N \le 10$), és a kifizetendő összeget ($1 \le M \le 1000$)! Ezután olvassa be a pénzérmék értékét ($1 \le P_i \le M$)!

A program írja ki a legfeljebb 1, 2, majd a 3 pénzérmével kifizethető összegeket!

Példa:

Bemenet:

```
Érmék száma: 2
Maximális összeg: 100
1. érme: 1
2. érme: 5
```

Kimenet:

```
1 érmével: 1 5
2 érmével: 1 2 5 6 10
3 érmével: 1 2 3 5 6 7 10 11 15
```

Megoldás

Az érmék értékét az *Érmék* tömbben tároljuk:

```
VÁLTOZÓ Érmék(1…N) MINT Egész
VÁLTOZÓ M, N MINT Egész
Be: N, M, Érmék()
```

Az első feladat megoldását maguk az érmék szolgáltatják (feltételezzük, hogy nincsenek közöttük azonos értékűek). A továbbiakban azonban szükségünk lesz az értékeket tartalmazó halmazra, ezért ezt itt töltjük fel elemekkel.

```
VÁLTOZÓ Összegek MINT RendezettHalmaz(Elemtípus: Egész)
CIKLUS I=1-től N-ig
    Összegek.Add(Érmék(I))
CIKLUS VÉGE
CIKLUS MINDEN Elem-re az Összegek-ben
    Ki: Elem
CIKLUS VÉGE
```

Bár a feladat szövege nem kívánja meg, a rendezett halmaz alkalmazásával nagyság szerint növekvő sorrendben írjuk ki az összegeket.

A két érmével kifizethető összegekhez előállítjuk az összes lehetséges érmepár összegét:

```
CIKLUS I=1-től N-ig
    CIKLUS J=1-től N-ig
        Összegek.Add(Érmék(I) + Érmék(J))
    CIKLUS VÉGE
CIKLUS VÉGE
Ki: az Összegek halmaz M-nél nem nagyobb elemei
```

Ha a pontosan két érmével kifizethető összegekre lennénk kíváncsiak, akkor a ciklusok előtt törölni kellene az *Összegek* halmazt.

A három érmével kifizethető összegeket hasonló módon, három ciklus egymásba ágyazásával állíthatjuk elő. Ennek megírását az Olvasóra bízzuk. ☺

A teljes megoldást a *penz* program tartalmazza.

Általánosítsuk a feladatot! Határozzuk meg a legfeljebb *K* darab érmével kifizethető összegeket!

> Ha Visual Basic-ben átirányítjuk fájlba a standard bemenetet, illetve kimenetet, akkor az ékezetes karakterek helyes kezeléséhez használjuk a
> `Console.InputEncoding = Text.Encoding.Default`
> `Console.OutputEncoding = Text.Encoding.Default`
> utasításokat!

Lövészverseny

Nemes Tihamér OITV 2009. 2. forduló 1. korcsoport 3. feladat
Nemes Tihamér OITV 2010. 3. forduló 1. korcsoport 1. feladat

A 2009-es versenyen az alábbiak közül csak a b) és c) kérdés szerepelt.

Egy lövészversenyen a versenyzők egymás után lőnek. Ismerjük N ($1 \le N \le 1000$) versenyző eredményét. Készíts programot *lovesz* néven, amely beolvassa N értékét és az N db eredményt, majd megadja:

a) minden versenyzőre, hogy az addig szereplők közül hányan értek el nála jobb eredményt;
b) azokat a versenyzőket, akik a verseny valamelyik időszakában álltak az első helyen;
c) azokat a versenyzőket, akik a verseny valamelyik időszakában álltak az utolsó helyen;
d) a verseny győztesét!

Példa:[3]

Bemenet	*Kimenet*
`6` (N)	`0 0 2 0 3 3` (jobb eredmény)
`594` (1. versenyző)	`1 2 4` (állt az első helyen)
`596` (2. versenyző)	`1 3` (állt az utolsó helyen)
`582` (stb.)	`4` (győztes vagy győztesek)
`599`	
`590`	
`590`	

Megoldás

A versenyzők pontszámait a *Pontszámok* tömbben tároljuk:

```
VÁLTOZÓ Pontszámok(1...N) MINT Egész
VÁLTOZÓ N MINT Egész
Be: N, Pontszámok()
```

Az a) kérdés megválaszolásához minden versenyző eredményét összehasonlítjuk az előző eredményekkel, és megszámoljuk az övénél nagyobb értékeket.

```
VÁLTOZÓ Db MINT Egész
CIKLUS I=1-től N-ig
    Db = 0
    CIKLUS J=1-től I–1-ig
        HA Pontszámok(I) < Pontszámok(J) AKKOR
            Db = Db + 1
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
    Ki: Db
CIKLUS VÉGE
```

Gondoljuk meg, hogy miért a belső ciklusban szerepel a *Db* változó nullázása. A külső ciklust miért 1-től indítottuk?

[3] A formátumot némileg egyszerűsítettük.

A b) és c) kérdés megválaszolásánál halmazokba gyűjtjük mindazon versenyzőket, akik a verseny valamelyik időszakában az első, illetve az utolsó helyen álltak.

```
VÁLTOZÓ Elsők, Utolsók MINT Halmaz(Elemtípus: Egész)
```

Egy ciklusban végignézzük a versenyzőket. Amelyiknek a pontszáma nagyobb az addigi maximumnál (illetve kisebb az addigi minimumnál), azt feljegyezzük az *Utolsók* (illetve az *Elsők*) halmazba. Módosítjuk a maximum/minimum értékét is.

```
VÁLTOTÓ Min, Max MINT Egész
Min = +∞
Max = –∞
CIKLUS I=1-től N-ig
    HA Pontszámok(I) ≥ Max AKKOR
        Elsők.Add(I)
        Max = Pontszámok(I)
    ELÁGAZÁS VÉGE
    HA Pontszámok(I) ≤ Min AKKOR
        Utolsók.Add(I)
        Min = Pontszámok(I)
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
Ki: Elsők, Utolsók elemei
```

Miért engedtük meg az egyenlőséget a feltételes elágazások vizsgálataiban? Miért alkalmaztunk két, egymástól független feltételes elágazást a többágú szelekció (*HA ... EGYÉBKÉNT HA*) helyett?

Vegyük észre, hogy halmazokban való tárolás nélkül, egyből kiírhattuk volna a versenyzők sorszámát! A fenti módszerrel példát mutattunk a kiválogatás algoritmusának egyszerűsítésére.[4]

A d) kérdés megválaszolásához ki kell válogatni a maximális pontszámot elért versenyzők sorszámait:

```
CIKLUS I=1-től N-ig
    HA Pontszámok(I) == Max AKKOR Ki: I
CIKLUS VÉGE
```

A teljes megoldást a *lovesz* program tartalmazza.

Oldjuk meg a feladatot halmazok helyett statikus tömbök alkalmazásával!

[4] Ha ismétlődhetnek az értékek, akkor a kiválogatáskor halmaz helyett használjunk dinamikus tömböt (listát)!

Hangrend

Nemes Tihamér OITV 2009. 2. forduló 1. korcsoport 2. feladat

Egy magyar szó magas hangrendű, ha csak magas hangrendű 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* néven, amely beolvas egy magyar szót, majd kiírja, hogy milyen hangrendű!

Példa

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

Megoldás

Halmazok felhasználásával nincs szükségünk sem lineáris keresésre, sem pedig ciklusra a megoldáshoz. Meghatározzuk a beolvasott szó karaktereiből álló halmaz metszetét mind a magas, mind pedig a mély magánhangzók halmazával.

```
VÁLTOZÓ Magas, Mély MINT Logikai
Magas =
  ({magas magánhangzók halmaza} ∩ {a szó karaktereinek halmaza}).Elemszám > 0
Mély =
  ({mély magánhangzók halmaza} ∩ {a szó karaktereinek halmaza}).Elemszám > 0
```

Ezek után a válasz a kérdésre:

```
HA Magas ÉS Mély AKKOR Ki: Vegyes hangrendű ' fontos a sorrend!
EGYÉBKÉNT HA Magas AKKOR Ki: Magas hangrendű
EGYÉBKÉNT Ki: Mély hangrendű
ELÁGAZÁS VÉGE
```

A teljes megoldást a *hangrend* program tartalmazza. A megoldással a halmazok metszetének alkalmazása mellett azt is szemléltetjük, hogyan lehet egy sztring karaktereiből gyorsan halmazt készíteni.

> C++-ban a halmazműveletek eredményét egy harmadik konténerbe kell beszúrnunk. A műveletek során az eredeti halmazok tartalma nem módosul:
> ```
> set_union(begin(halmaz1), end(halmaz1),
>           begin(halmaz2), end(halmaz2),
>           inserter(halmaz3, begin(halmaz3)));
> ```
> Az eredményhalmaz tartalmát a *swap* (csere) algoritmus hívásával gyorsan és kényelmesen áthelyezhetjük például a *halmaz1*-be: `swap(halmaz1, halmaz3);`

> Visual Basic-ben kétféleképpen végezhetünk halmazműveleteket.
> 1. A *UnionWith, IntersectWith*, … metódusok hívásával. Ezek olyan eljárások, melyek megváltoztatják a metódust minősítő halmazt. A
> `Halmaz1.UnionWith(Halmaz2)`
> például megfelel a *Halmaz1 = Halmaz1 ∪ Halmaz2* értékadásnak.
> 2. A *Union, Instersect*, … metódusok hívásával. Ezek függvények, melyek visszatérési értéke egy felsoroló objektum, amit például a *New* operátor alkalmazásával alakíthatunk át halmazzá. A
> `Halmaz3 = New HashSet(of Elemtípus) (Halmaz1.Union(Halmaz2))`
> például megfelel a *Halmaz3 = Halmaz1 ∪ Halmaz2* értékadásnak.

Gráfok feldolgozása

Nagyon sok versenyfeladat adatszerkezete szemléltethető olyan ábrával, amelyben az egyes elemeket a köztük fennálló kapcsolatot jelző vonalakkal kötjük össze. Ezt az adatszerkezetet hálós adattípusnak vagy gráfnak nevezzük. A gráfok feldolgozását, a gyakoribb gráfalgoritmusok alkalmazását a *Programozási ismeretek versenyzőknek* példatár mutatja be. Itt néhány olyan, az 1. korcsoportban szereplő feladat következik, amelynek megoldását megkönnyíti a gráfok ismerete.

A továbbiakban felhasználjuk a *Programozási ismeretek haladóknak* tankönyv 36. leckéjében (A hálós adattípus) szereplő fogalmakat.

Ha a csúcsokat sorszámokkal azonosítjuk, akkor a gráfot statikus tömbben tárolhatjuk. Az *I* indexű tömbelem az *I* azonosítójú csúcs szomszédjait tartalmazza. Ha két csúcsot legfeljebb egy él köt össze (egyszerű gráf), akkor a csúcs szomszédjainak azonosítóit halmazban gyűjtjük. Ellenkező esetben listát használunk (lásd a továbbiakban a *4. Dinamikus tömbök* leckét).

Állat

Nemes Tihamér OITV 2012. 3. forduló 1. korcsoport 1. feladat

Egy állatkertben ismerjük a bejárható útvonalakat. A bejárat a 0 sorszámú pont. Az egyes állatokat az 1 és *N* közötti sorszámukkal azonosítjuk ($1 \le N \le 100$), az utakat pedig két olyan állat sorszámával, amelyek ketrece között vezetnek.

Készíts programot *allat* néven, amely az állatkerti utak ismeretében megadja, hogy hány olyan állat van, amelyik zsákutca végén található, valamint azt, hogy melyik állathoz vezet a legtöbb út (ha több is van, bármelyik megadható)!

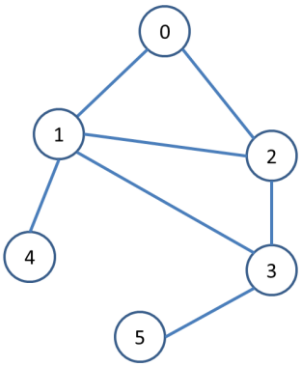
Példa:

Bemenet

```
Állatok száma: 5
Utak száma: 7
1. út: 0 1
2. út: 1 4
3. út: 3 1
4. út: 3 5
5. út: 2 0
6. út: 2 3
7. út: 1 2
```

Kimenet

```
Állatok száma zsákutca végén: 2
Legtöbb út: 1
```



Megoldás

Az adatokat szemléltethetjük körökből és vonalakból álló ábrával (gráffal). A körök jelentik az egyes állatokat, illetve a bejáratot (a gráf csúcsait), a vonalak pedig a közöttük vezető utakat (a gráf éleit).

Tároljuk a gráfot az *Állatok* tömbben! Az *Állatok*(*I*) tömbelem az *I*. állattól kiinduló utak másik végpontját tartalmazó halmaz (szomszédsági „lista”).

```
VÁLTOZÓ Állatok(0…N) MINT Halmaz(Elemtípus: Egész)
VÁLTOZÓ N, M MINT Egész
Be: N, M
```

Az utak beolvasása előtt létrehozzuk a halmazokat:

```
CIKLUS I=0-tól N-ig
   Állatok(I) = Új Halmaz(Elemtípus: Egész)
CIKLUS VÉGE
```

A beolvasásnál ügyelünk arra, hogy az utakat mindkét végpontjuknál felvegyük a halmazokba (irányítatlan gráf)!

```
CIKLUS I=1-től M-ig
   Be: Állat1, Állat2
   Állatok(Állat1).Add(Állat2)
   Állatok(Állat2).Add(Állat1)
CIKLUS VÉGE
```

A jól megválasztott adatszerkezet miatt könnyen meghatározhatjuk a kérdezett mennyiségeket. Azok az állatok találhatók zsákutca végén, melyek halmaza csak egy elemet tartalmaz:

```
VÁLTOZÓ Db MINT Egész
Db = 0
CIKLUS I=1-től N-ig
   HA Állatok(I).Elemszám == 1 AKKOR Db = Db + 1
CIKLUS VÉGE
Ki: Db
```

A legtöbb úthoz illeszkedő állatot maximumkiválasztással határozzuk meg:

```
VÁLTOZÓ Max, Melyik MINT Egész
Max = –∞
CIKLUS I=1-től N-ig
   HA Állatok(I).Elemszám > Max AKKOR
      Max = Állatok(I).Elemszám
      Melyik = I
   ELÁGAZÁS VÉGE
CIKLUS VÉGE
Ki: Melyik
```

A teljes megoldást az *allat* program tartalmazza.

Megjegyzés: ha két állatot közvetlenül több út is összeköt, akkor halmaz helyett használjunk listát az utak tárolására! Ha az állatokat sorszámok helyett sztringekkel azonosítjuk (név, faj), akkor statikus tömb helyett alkalmazzunk asszociatív tömböt (lásd ott)!

Elszigetelt falu

Nemes Tihamér OITV 2011. 3. forduló 1. korcsoport 3. feladat

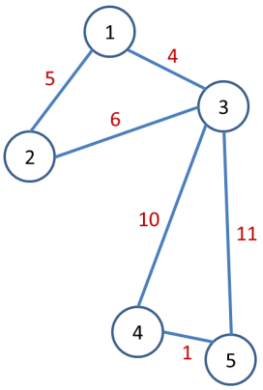
Egy megyében *N* (1≤*N*≤100) falu van. A falvakat *M* (3≤*M*≤1000) út köti össze, ismerjük minden út hosszát. A legelszigeteltebb falunak azt nevezzük, amelytől a legközelebbi szomszédja a lehető legtávolabb van.

Készíts programot *falu* néven, amely megadja a legelszigeteltebb falut! Ha több megoldás van, akkor bármelyik kiírható.

Példa

Bemenet

```
Falvak száma? 5
Utak száma? 6
1. út kezdete, vége, hossza? 1 2 5
2. út kezdete, vége, hossza? 2 3 6
3. út kezdete, vége, hossza? 3 1 4
4. út kezdete, vége, hossza? 3 4 10
5. út kezdete, vége, hossza? 4 5 1
6. út kezdete, vége, hossza? 3 5 11
```



Kimenet

```
A legelszigeteltebb: 2
```

Megjegyzés: a feladat megoldási útmutatója szerint lehetséges olyan falu is ahová egyáltalán nem vezet út. Ebben az esetben egy ilyen falu számít a legelszigeteltebbnek.

Megoldás

A távolságokat a rendezett halmazokat tartalmazó *Szomszédok* tömbben tároljuk. Ebben a feladatban nem a szomszédok sorszámát kell figyelnünk, hanem a távolságokat (súlyozott gráf). Ezért a tömb *I*. elemének halmazában az *I*. falu szomszédjainak távolságát jegyezzük fel. Bár két szomszéd lehet ugyanolyan messze a falutól, a legelszigeteltebb település meghatározásához a távolságot elegendő egyszer tárolni.[5]

```
VÁLTOZÓ Szomszédok(1…N) MINT RendezettHalmaz(Elemtípus: Egész)
```

A távolságok beolvasása előtt létrehozzuk a halmazokat:

```
Be: N, M
CIKLUS I=1-től N-ig
    Szomszédok(I) = Új RendezettHalmaz(Elemtípus: Egész)
CIKLUS VÉGE
```

A távolságok beolvasásánál ügyeljük arra, hogy egy távolság két faluhoz tartozik (irányítatlan gráf):

```
CIKLUS I=1-től M-ig
    Be. Kezd, Vég, Hossz
    Szomszédok(Kezd).Add(Hossz)
    Szomszédok(Vég).Add(Hossz)
CIKLUS VÉGE
```

[5] Az összes távolság tárolásához halmaz helyett használjunk dinamikus tömböt!

A legelszigeteltebb falu kereséséhez először megvizsgáljuk, hogy van-e olyan falu, ahová nem vezet út. Ennél a falunál a szomszédok halmaza üres lesz:

```
VÁLTOZÓ Melyik MINT Egész
Melyik = 1
CIKLUS AMÍG Melyik ≤ N ÉS# Szomszédok(Melyik).Elemszám > 0
    Melyik = Melyik + 1
CIKLUS VÉGE
```

Ha volt ilyen falu, akkor az lesz a megoldás. Egyébként pedig megkeressük a rendezett halmazok első elemei, azaz a legközelebbi szomszédok távolságai közül a legnagyobbat:

```
VÁLTOZÓ Max MINT Egész
HA Melyik > N AKKOR ' nincs olyan falu, ahová nem vezet út
    Max = –∞
    CIKLUS I=1-től N-ig
        HA Szomszédok(I).Első > Max AKKOR
            Max = Szomszédok(I).Első
            Melyik = I
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
ELÁGAZÁS VÉGE
```

A legelszigeteltebb falut a *Melyik* változó értéke adja meg.

A teljes megoldást a *falu* program tartalmazza. Javasoljuk az Olvasónak, hogy oldja meg a feladatot halmazok helyett statikus tömb alkalmazásával! A legközelebbi szomszéd távolságát egy-egy falura már beolvasás közben határozza meg!

Ismerősök

Nemes Tihamér OITV 2008. 2. forduló 2. korcsoport 2. feladat

Egy közösségi portálra N ember jelentkezett be. Mindenki megadta, hogy kiket ismer, amit az ismerősük vissza is igazolt.

Készíts programot *ismer* néven, amely megadja azon párokat,

a) akiknek minden ismerőse közös (de van legalább 1);

b) akiknek van közös ismerősük!

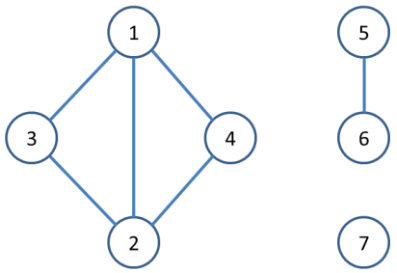
Az *ismer.be* szöveges állomány első sorában az emberek N száma ($1 \le N \le 100$) és az ismeretségek M száma ($0 \le M \le 5000$) van, egy szóközzel elválasztva. A következő M sor mindegyike két egymást ismerő ember sorszámát tartalmazza ($1 \le i \ne j \le N$), egy szóközzel elválasztva.

Az *ismer.ki* szöveges állomány első sorába azon párok K számát kell írni, akiknek minden ismerősük közös, a következő K sorba pedig egy-egy ilyen pár sorszámát, a sorszámokat egy szóközzel elválasztva! A következő sorba azon párok L számát kell írni, akiknek van közös ismerősük, s az ezt követő L sorba pedig egy-egy ilyen pár sorszámát, a sorszámokat egy szóközzel elválasztva!

Példa:

```
ismer.be
7 6
1 2
1 3
1 4
3 2
4 2
5 6
```

```
ismer.ki
2
1 2
3 4
6
1 2
1 3
1 4
2 3
2 4
3 4
```



Megoldás

Az egyes emberek ismerőseit rendezett halmazokba gyűjtjük. A rendezett halmazokat tömbben tároljuk:

```
VÁTOZÓ Ismerősök(1…N) MINT RendezettHalmaz(Elemtípus: Egész)
```

Az adatok beolvasásánál ne feledkezzünk meg arról, hogy az ismeretség szimmetrikus kapcsolat (irányítatlan gráf)! A beolvasás előtt létrehozzuk a halmazokat.

```
VÁLTOZÓ Egyik, Másik MINT Egész ' segédváltozók
Be: N, M
CIKLUS I=1-től N-ig
    Ismerősök(I) = Új RendezettHalmaz(Elemtípus: Egész)
CIKLUS VÉGE
CIKLUS I=1-től M-ig
    Be: Egyik, Másik
    Ismerősök(Egyik).Add(Másik)
    Ismerősök(Másik).Add(Egyik)
CIKLIUS VÉGE
```

Az *A* és a *B* minden ismerőse közös, ha az *A* ismerőseinek a halmazából kivéve a *B*-t ugyanazt a halmazt kapjuk, mint amikor a *B* ismerőseinek a halmazából vesszük ki az *A*-t (gondoljuk meg, miért!). Ezt a vizsgálatot elvégezzük minden párra:

```
Db = 0
CIKLUS I=1-től N–1-ig
    CIKLUS J=I+1-től N-ig
        HA Ismerősök(I).Kivesz(J) == Ismerősök(J).Kivesz(I) ÉS
                                                Ismerősök(I).Elmeszám > 1 AKKOR
            Db = Db + 1
            Ki: I, J
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
CIKLUS VÉGE
```

A program megírása során ügyeljünk arra, hogy a feltételes elágazásban szereplő *Kivesz* metódus ténylegesen megváltoztatja-e az eredeti halmazt (eljárás-e vagy pedig függvény)! Ha igen, akkor ezt segédváltozók felhasználásával kerülhetjük el.

A feladat szövege szerint a darabszámot az *I, J* párok előtt kell kiírni, amit szintén egy segédváltozóval érhetünk el.

A közös ismerőssel rendelkező személyeket a halmazok metszetével tudjuk megállapítani. Két személynek akkor vannak közös ismerősei, ha az ismerőseik halmazainak metszete nem üres. Ezt ismét az összes lehetséges párra ellenőrizzük:

```
Db = 0
CIKLUS I=1-től N–1-ig
    CIKLUS J=I+1-től N-ig
        HA Ismerősök(I) ∩ Ismerősök(J) ≠ ∅ AKKOR
            Db +=1
            Ki: I, J
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
CIKLUS VÉGE
```

A teljes megoldást az *ismer* program tartalmazza.

Javasoljuk az Olvasónak, hogy oldja meg a 2006-os verseny 3. fordulójának 2. korcsoportjából az ehhez nagyon hasonló 1. feladatot (*Ismerősök*).

Verseny

Nemes Tihamér OITV 2012. 2. forduló 2. korcsoport 2. feladat

Egy kieséses versenyben ismerjük a csapatok mérkőzéseit: ki kit győzött le.

Írj programot *kieses* néven, amely megadja

a) a még versenyben lévő csapatokat;
b) azokat a csapatokat, amelyek legalább egyszer győztek, de már kiestek;
c) a legtöbb csapatot közvetlenül vagy közvetve legyőző csapatot!

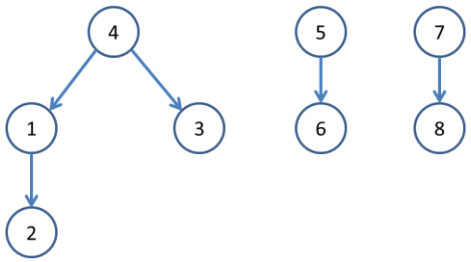
A *kieses.be* szöveges állomány első sorában a csapatok száma ($2 \le N \le 1000$) és a mérkőzések száma van ($1 \le M < N$), egy szóközzel elválasztva. A következő M sor - mindegyikében két csapat I és J sorszáma található ($1 \le I \ne J \le N$), ami azt jelenti, hogy az I-edik csapat legyőzte a J-edik csapatot.

A *kieses.ki* szöveges állomány első sorába a még versenyben levő csapatok darabszámát, majd a sorszámát kell írni (egy-egy szóközzel elválasztva, növekvő sorrendben), a második sorba azok darabszámát, majd a sorszámát, amelyek úgy estek ki, hogy legalább egyszer győztek (egy-egy szóközzel elválasztva, növekvő sorrendben), a harmadik sorba pedig azt a csapatot, amely a legtöbb más csapatot győzte le közvetve vagy közvetlenül! Ha több megoldás van, bármelyik kiírható.

Példa:

```
kieses.be
8 5
1 2
4 3
4 1
7 8
5 6
```

```
kieses.ki
3 4 5 7
1 1
4
```



Megoldás

Az egymást legyőző csapatok kapcsolatát irányított gráffal szemléltethetjük. A gráf csúcsai jelentik a csapatokat. Az A csúcsból akkor fut él a B csúcsba, ha az A csapat legyőzte a B csapatot. A kieséses verseny miatt minden csúcsba legfeljebb egy él fut be (legfeljebb egyszer kaphat ki a csapat, ekkor ugyanis kiesik a versenyből).

A gráfot a *Legyőzte* tömbben tároljuk. Legyen *Legyőzte*(*I*) azoknak a csapatoknak a halmaza, amelyeket az *I*. csapat legyőzött (a kifutó élek végpontjainak a halmaza, tulajdonképpen szomszédsági „lista”).

```
VÁLTOZÓ Legyőzte(1…N) MINT Halmaz(Elemtípus: Egész)
Változó N, M MINT Egész
Be: N, M
```

Létrehozzuk a halmazokat (!), majd beolvassuk a mérkőzések eredményeit:

```
VÁLTOZÓ Ki, Kit MINT Egész
CIKLUS I=1-től N-ig
    Legyőzte(I) = Új Halmaz(Elemtípus: Egész)
CIKLUS VÉGE
CIKLUS I=1-től M-ig
    Be. Ki, Kit
    Legyőzte(Ki).Add(Kit)
CIKLUS VÉGE
```

A továbbiakban többször is szükségünk lesz az összes csapat halmazára:

```
VÁLTOZÓ Összes MINT Halmaz(Elemtípus: Egész)
CIKLUS I=1-től N-ig
    Összes.Add(I)
CIKLUS VÉGE
```

Az a) feladat megoldását azok a csapatok képezik, akik nem szerepelnek a legyőzöttek között. Az összes csapat halmazából kivonjuk az egyes csapatok által legyőzött csapatok halmazait:

```
VÁLTOZÓ Versenyben MINT RendezettHalmaz(Elemtípus: Egész)
Versenyben ← Összes ' átmásolja az elemeket
CIKLUS I=1-től N-ig
    Versenyben = Versenyben – Legyőzte(I) ' halmazok különbsége
CIKLUS VÉGE
Ki: Versenyben.Elemszám, a Versenyben halmaz elemei
```

A b) kérdéshez először vegyük azoknak a csapatoknak a *Győzött* halmazát, amelyek győzelmet értek el valamelyik mérkőzésen! Sorszámaikat a *Legyőzte* tömb nem üres elemeinek indexei alkotják.

```
VÁLTOZÓ Győzött MINT RendezettHalmaz(Elemtípus: Egész)
CIKLUS I=1-től N-ig
    HA Legyőzte(I).Elemszám > 0 AKKOR Győzött.Add(I)
CIKLUS VÉGE
```

Szükségünk van még a kiesett csapatok halmazára. Ezt úgy kapjuk meg, hogy az összes csapat halmazából kivonjuk a még versenyben lévő csapatok halmazát:

```
VÁLTOZÓ Kiesett MINT RendezettHalmaz(Elemtípus: Egész)
Kiesett = Összes – Versenyben ' halmazok különbsége
```

A legalább egyszer győztes, de már kiesett csapatok halmazát a *Győzött* és a *Kiesett* halmazok metszete (közös része) adja meg:

```
Ki: Győzött ∩ Kiesett halmaz elemei
```

A c) feladat megoldásánál vegyük észre, hogy a kérdezett csapatok csak azok közül kerülhetnek ki, melyek még versenyben vannak! Először határozzuk meg minden ilyen csapat esetén azokat a csapatokat, melyeket az adott csapat közvetve vagy közvetlenül legyőzött! Gyűjtsük ezen csapatok sorszámát a *Vesztesek* halmazba!

VÁLTOZÓ Vesztesek MINT Halmaz(Elemtípus: Egész)

Jelölje a *Győztes* változó egy, még versenyben lévő csapat sorszámát. A *Vesztesek* halmazhoz adjuk hozzá a *Legyőzte*(*Győztes*) halmaz elemeit (ezek közvetlenül a *Győztes* csapattól kaptak ki)! Majd a halmazba került elemekre ismételjük meg ugyanezt az eljárást! Azaz adjuk hozzá a *Vesztesek* halmazhoz a benne lévő csapatok által legyőzött csapatok sorszámait (közvetve kaptak ki a *Győztes* csapattól)! Ezt az eljárást mindaddig ismételjük, amíg végig nem érünk a halmazon.

Egyszerűbben kódolható algoritmushoz jutunk, ha először magát a *Győztes* csapatot is hozzáadjuk a *Vesztesek* halmazhoz (majd a végén kivonjuk belőle). Így a *Vesztesek* halmaz összes elemére végre kell hajtani a vázolt eljárást.

Módszerünket egy konkrét példán keresztül érthetjük meg a legjobban. Tekintsük a feladat számpéldájában szereplő, 4. számú csapatot! A *Vesztesek* halmaz első eleme a 4. Hozzáadjuk a halmazhoz az első elem által legyőzött csapatokat, majd a 2. elem által legyőzött csapatokat stb.

Lépés	*Vesztesek halmaz elemei*	*Teendő*
előkészítés	–	hozzáadjuk a *Győztes* (4.) csapatot
1.	4	hozzáadjuk az első elem (4. csapat) által legyőzött csapatokat
2.	4 1 3	hozzáadjuk a második elem (1. csapat) által legyőzött csapatokat
3.	4 1 3 2	hozzáadjuk a harmadik elem (3. csapat) által legyőzött csapatokat (nincs ilyen)
4.	4 1 3 2	hozzáadjuk a negyedik elem (2. csapat) által legyőzött csapatokat (nincs ilyen)
	4 1 3 2	nincs több elem, kivesszük az első elemet (a 4. csapatot)
	1 3 2	megkaptuk a *Vesztesek* halmazt

Gondoljuk meg, hogy eljárásunk miért fog biztosan véget érni!

A *Vesztesek* halmazt előállító algoritmus:

```
' A Vesztesek halmaz elemeinek meghatározása
Vesztesek.Töröl()
Vesztesek.Add(Győztes)
I = 0
CIKLUS AMÍG I < Vesztesek.Elemszám
    Vesztesek = Vesztesek ∪ Legyőzte(Vesztesek(I))
    I = I + 1
CIKLUS VÉGE
Vesztesek.Kivesz(Győztes)
```

Ezt az eljárást a versenyben lévő minden csapatra elvégezzük. Közben a c) kérdésnek megfelelően meghatározzuk azt a csapatot, amelynél a legnagyobb a *Vesztesek* halmaz elemszáma.

```
VÁLTOZÓ Max, Melyik MINT Egész
Max = –∞
CIKLUS MINDEN Győztes-re a Versenyben halmazban
    A Vesztesek halmaz elemeinek meghatározása (lásd fent)
    HA Vesztesek.Elemszám > Max AKKOR
        Max = Vesztesek.Elemszám
        Melyik = Győztes
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
Ki: Melyik
```

A teljes megoldást a *kieses* program tartalmazza. Jegyezzük meg a c) feladatnál bemutatott algoritmust, melynek segítségével a gráf egy adott csúcsából kiindulva sorra vettük az élek mentén elérhető csúcsokat (szélességi bejárás)!

Javasoljuk az olvasónak, hogy oldja meg a 2012-es OKTV 2. fordulójának nagyon hasonló, 2. feladatát (*Verseny*).

> C++-ban csak a soros tárolók garantálják azt, hogy a később hozzáadott elemek a már bent lévők mögé kerülnek. Az asszociatív jellegű tárolók esetén ez a sorrendiség függ a tároló megvalósításától, így erre nem szabad építeni a megoldás algoritmusát.

Feladatok a halmazokra a 2. és 3. korcsoport számára

Névnapok

Nemes Tihamér OITV 1996. 2. forduló 2. korcsoport 2. feladat

A *nevnap.be* állomány (naptár) legfeljebb 365 sorból áll. *K*-adik sora az év *K*-adik napjára eső névnapokat tartalmazza. Ha egy napra csak egyetlen névnap esik, akkor a sorban egyetlen név van; ha több, akkor a soron belül az egyes neveket egy-egy szóköz választja el. Ugyanazon a napon 5-nél több névnap nem lehet.

Készíts programot *nevnap* néven, amely

a) megadja, hogy hány név szerepel a naptárban;
b) beolvas egy nevet a billentyűzetről, s kiírja, hogy az adott név az év hányadik napján fordul elő a naptárban (ha többször is előfordul, akkor az összeset megadja)!

Megjegyzés: írjuk meg úgy a programot, hogy a billentyűzetről való beolvasás helyett a keresett név a *nevnap.be* fájl legelső sora (tehát ezt követi a naptár legfeljebb 365 sora a névnapokkal)! A *nevnap.ki* fájl első sora tartalmazza az a) feladat megoldását, második sora pedig a b) feladatnak megfelelő napok sorszámát, egymástól egy-egy szóközzel elválasztva!

Példa:

```
nevnap.be                nevnap.ki
Amália                   4
Nóra                     2 3
Bálint Amália
Amália
Ádám
```

Megoldás

Az összes nevet az *Összes* halmazban gyűjtjük, a névnapokat pedig a *Naptár* tömbben, melynek minden egyes eleme az adott napra eső névnapok halmaza. A tömb elemeit (a halmazokat) a beolvasásnál hozzuk létre.

```
VÁLTOZÓ Naptár(1…365) MINT Halmaz(Elemtípus: Sztring)
VÁLTOZÓ Összes MINT Halmaz(Elemtípus: Sztring)
Be: Név
N = 0
CIKLUS AMÍG van még név a fájlban
    N = N + 1
    Naptár(N) = Új Halmaz(Elemtípus: Sztring)
    Be: sor
    Naptár(N).Add(a sorban szereplő nevek)
    Összes.Add(a sorban szereplő nevek)
CIKLUS VÉGE
```

A nevek számát az *Összes* halmaz elemszáma adja meg:

```
Ki: Összes.Elemszám
```

A megadott név névnapjait ciklussal gyűjtjük ki:

```
CIKLUS I=1-től N-ig
    HA Naptár(I).Tartalmazza(Név) AKKOR Ki: I
CIKLUS VÉGE
```

A teljes megoldást a *nevnap* program tartalmazza.

Javasoljuk az Olvasónak, hogy a megoldáshoz mellékelt ne*vnapok.txt* fájl[6] segítségével készítse el a programot az igazi naptárban szereplő névnapokra is. Az állomány minden sorának elején szerepel a dátum (hónap, nap, szóközzel elválasztva), majd az adott napra eső névnapok. Bizonyos nevek végén a csillag azt jelöli, hogy az a nap a „fő” névnap. A kiírásnál a megadott név fő névnapját tegyük külön sorba, a napok sorszáma helyett pedig szerepeltessük a dátumot!

Játék

Nemes Tihamér OITV 2010. 3. forduló 2. korcsoport 1. feladat

Egy *N* napos játékversenyen, ahol nem kötelező minden nap játszani, 3 játékos (*A, B, C*) vesz részt. Nem szerencsések azok a napok, amikor pontosan 2 játékos vesz részt a versenyen, mert ekkor közös stratégiát alkothatnak a harmadik ellen.

Készíts programot *jatek* néven, amely megadja, hogy mikor voltak ilyen helyzetek!

A *jatek.be* szöveges állomány első sorában a napok száma van ($1 \le N \le 1000$). A második sorban az *A*, a harmadikban a *B*, a negyedikben pedig a *C* játékos leírása szerepel. Mindhárom sor első száma ($1 \le M \le N$) azt jelenti, hogy a játékos a verseny hány szakaszában vett rész. Ezt *M* számpár követi: melyek első tagja a szakasz első napjának sorszáma, a második pedig a szakasz hossza. A számok között egy-egy szóköz van.

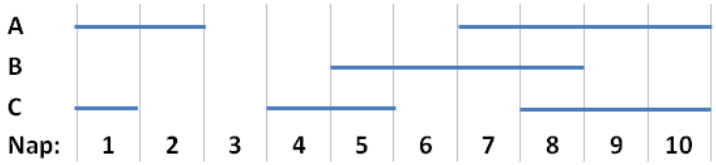
A *jatek.ki* szöveges állomány első sorába azon intervallumokat kell írni, amikor *A* és *B*, a második sorba azokat, amikor *A* és *C*, a harmadikba pedig azokat, amikor *B* és

[6] Forrás: http://www.naptarak.com

C játékos volt kettesben! Minden sor az ilyen intervallumok számával kezdődjön, majd az intervallumok kezdete és vége kövesse, kezdet szerint növekvő sorrendben! A számok közé mindenhova egy-egy szóközt kell írni!

Példa:

```
jatek.be                    jatek.ki
10                          1 7 7
2 1 2 7 4                   2 1 1 9 10
1 5 4                       1 5 5
3 1 1 4 2 8 3
```



Megoldás

A feladatot úgy oldjuk meg, hogy könnyen általánosítható legyen tetszőleges számú játékosra. A játékosok betűjelét a *Játékosok* tömb tartalmazza. Több játékos esetén csupán ezt a tömböt kell módosítani:

```
VÁLTOZÓ Játékosok(0…2) MINT Karakter = ("A", "B", "C")
```

A *Játékosok* tömb maximális indexét a továbbiakban *MaxJ*-vel jelöljük.

Az adatokat a *Napok* tömbben gyűjtjük. A *Napok*(*I*) megadja azoknak a játékosoknak a halmazát, akik az *I*. napon részt vettek a játékban:

```
VÁLTOZÓ Napok(1…N) MINT Halmaz(Elemtípus: Egész)
```

A szakaszok beolvasása előtt létrehozzuk a halmazokat:

```
CIKLUS I=1-től N-ig
   Napok(I) = Új Halmaz(Elemtípus: Egész)
CIKLUS VÉGE
```

A beolvasásnál minden játékost hozzáadunk a *Napok* tömb megfelelő halmazaihoz:

```
CIKLUS Játékos=0-tól MaxJ-ig
   Be: szakaszok száma
   CIKLUS Szakasz=1-től szakaszok számáig
      Be: Kezd, Hossz
      CIKLUS Nap=Kezd-től Kezd+Hossz–1-ig
         Napok(Nap).Add(Játékosok(Játékos))
      CIKLUS VÉGE
   CIKLUS VÉGE
CIKLUS VÉGE
```

A feldolgozás során minden egyes játékospárra megkeressük azokat az intervallumokat, amikor csak ők ketten vettek részt a versenyen. Ehhez felhasználjuk a játékospárokat tartalmazó *Résztvevők* halmazt.

Az intervallumok keresésénél minden egyes játékospárra végignézzük a napokat. Ha elérkezünk egy olyan naphoz, amikor csak ők ketten vesznek részt a versenyen,

akkor megkeressük ennek az időszaknak a végét. Feljegyezzük azt is, hogy hány ilyen intervallumot találtunk.

```
CIKLUS I=1-től MaxJ–1-ig ' az egyik játékos
    Résztvevők.Töröl()
    Résztvevők.Add(Játékosok(I))
    CIKLUS J=I+1-től MaxJ-ig ' a másik játékos
        Résztvevők.Add(Játékosok(J))
        Db = 0
        Nap = 1
        CIKLUS AMÍG Nap ≤ N
            HA Napok(Nap) == Résztvevők AKKOR
                Db = Db + 1
                Kezd = Nap
                CIKLUS AMÍG Nap ≤ N ÉS# Napok(Nap) == Résztvevők
                    Nap = Nap + 1
                CIKLUS VÉGE
                Vég = Nap – 1 ' a ciklusban túlléptünk rajta
                Ki: Kezd, Vég
            ELÁGAZÁS VÉGE
            Nap = Nap + 1
        CIKLUS VÉGE
        Ki: Db
        Résztvevők.Kivesz(Játékos(J))
    CIKLUS VÉGE
CIKLUS VÉGE
```

A kiírásnál gondoskodnunk kell arról, hogy a *Db* változó értéke a szakaszok felsorolása elé kerüljön. Ennek egy lehetséges módját a teljes megoldással együtt a *jatek* program tartalmazza.

Javasoljuk az Olvasónak, hogy oldja meg a 2010-es verseny 3. fordulójának 3. korcsoportjából az ehhez nagyon hasonló 1. feladatot (*Játékos*).

Barátok

Nemes Tihamér OITV 2003. 3. forduló 2. korcsoport 1. feladat

Egy osztályba *N* tanuló jár. Ismerünk tanulópárokat, akik barátai egymásnak. A barátság ún. tranzitív kapcsolat, azaz ha *A* barátja *B*-nek és *B* barátja *C*-nek, abból következik, hogy *A* is barátja *C*-nek. A barátság kapcsolat szimmetrikus is, azaz, ha *A* barátja *B*-nek, akkor *B* is barátja *A*-nak.

Írj programot *baratok* néven, amely megadja, hogy az osztály hány baráti csoportra bontható!

A *baratok.be* szöveges állomány első sorában a tanulók *N* ($2 \le N \le 100$) és az ismert baráti kapcsolatok *M* száma ($0 \le M \le 10000$) van, egy szóközzel elválasztva. A következő *M* sor mindegyikében egy-egy számpár van, két tanuló sorszáma, egy szóközzel elválasztva, akikről tudjuk, hogy barátok.

A *baratok.ki* szöveges állomány első sorába azt a *K* számot kell írni, ahány baráti csoportra az osztályt bontani lehet. Ha tudjuk, hogy *A* és *B* barátok, akkor ugyanazon csoportba kell tartozniuk, ha pedig nem barátok, akkor különbözőkbe. Mindegyikbe a baráti csoportba tartozó tanulók sorszámát kell írni, egy-egy szóközzel elválasztva,

növekvő sorrendben. A sorokat a csoport legkisebb sorszámú tagja szerint növekvően kell kiírni.

Példa:

```
baratok.be          baratok.ki
9 6                 3
1 3                 1 3 5 7 8 9
3 5                 2
4 6                 4 6
7 9
8 9
1 7
```

Megoldás

A feladat megoldásához felhasználjuk a halmazok unióját. Az ugyanazon csoportba tartozó (egymással barátkozó) diákokat egy-egy rendezett halmazban tároljuk, melyeket a *Csoportok* halmaz tartalmaz:

```
VÁLTOZÓ Csoportok MINT Halmaz(Elemtípus: RendezettHalmaz(Elemtípus: Egész))
```

Először minden diákot külön halmazba teszünk:

```
CIKLUS I=1-től N-ig
    Csoportok.Add(Új RendezettHalmaz(Elemtípus: Egész))
    Csoportok(I–1).Add(I) ' a halmazelemek indexelése 0-val kezdődik
CIKLUS VÉGE
```

Beolvassuk a diákpárokat, megkeressük, majd összevonjuk az őket tartalmazó halmazokat. Mivel valamelyik halmaz biztosan tartalmazza a keresett diákot, a keresés ciklusfeltétele egyszerűsödik (lineáris keresés helyett kiválasztás):

```
CIKLUS I=1-től M-ig
    Be: Diák1, Diák2
    ' Megkeressük a két diákot tartalmazó halmazt:
    Halmaz1 = 0
    CIKLUS AMÍG NEM Csoportok(Halmaz1).Tartalmazza(Diák1)
        Halmaz1 += 1
    CIKLUS VÉGE
    Halmaz2 = 0
    CIKLUS AMÍG NEM Csoportok(Halmaz2).Tartalmazza(Diák2)
        Halmaz2 += 1
    CIKLUS VÉGE
    ' Összevonjuk a két diákot tartalmazó halmazt:
    HA Halmaz1 ≠ Halmaz2 AKKOR
        Csoportok(Halmaz1) = Csoportok(Halmaz1) ∪ Csoportok(Halmaz2)
        Csoportok.Töröl(Csoportok(Halmaz2))
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

Gondoljuk meg, miért kell törölnünk a feltételes elágazásban a *Csoportok*(*Halmaz2*) halmazt!

A *Csoportok* halmaz alapján már könnyű elkészíteni a *baratok.ki* fájlt. A teljes megoldást a *baratok* program tartalmazza. Ha az Olvasó ismeri a predikátumfüggvé-

nyek alkalmazásának módját, akkor módosítsa a programot úgy, hogy metódushívással helyettesíti a kiválasztások ciklusát!

Régió

Informatika OKTV 2003. 3. forduló 3. korcsoport 1. feladat

Egy megyén belül a településeket régiókba szeretnék csoportosítani. Ismerjük az egyes települések koordinátáit. Két település távolságán a koordináta-különbségeik abszolút értékének összegét értjük, azaz $TÁVOLSÁG((x,y),(a,b))=|x-a|+|y-b|$.

Két települést azonos régióba teszünk, ha egyikből a másikba el lehet jutni a régió településein keresztül úgy, hogy az egymást követő települések távolsága legfeljebb T kilométer.

Írj programot *regio* néven, amely megadja, hogy a települések hány régiót alkotnak, és mely települések tartoznak egy régióba!

A *regio.be* szöveges állomány első sorában a városok N száma ($2 \le N \le 100$) és a régióba kerülés határát jelentő T távolság ($1 \le T \le 100$) van. A következő N sor mindegyikében egy-egy számpár van, az adott város x és y koordinátája (0 és 1000 közötti egész számok), egy szóközzel elválasztva.

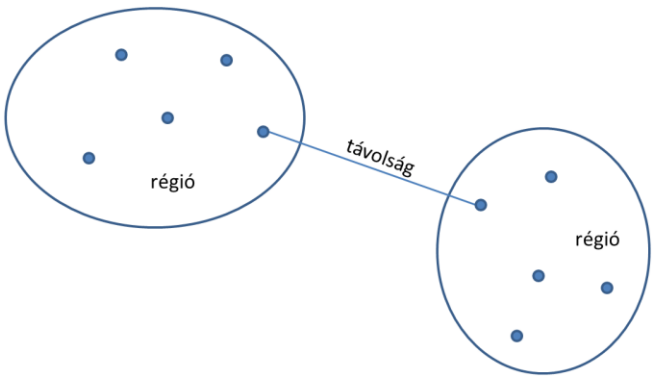
A *regio.ki* szöveges állomány első sorába azt a legkisebb K számot kell írni, ahány régióba lehet besorolni a településeket. A következő K sorba az egyes régiókat kell írni, tetszőleges sorrendben. Egy sorba a régióba tartozó települések sorszámát kell írni, egy-egy szóközzel elválasztva, növekvő sorrendben.

Példa:

```
regio.be
6 50
100 100
100 220
100 120
300 100
120 140
310 90
```

```
regio.ki
3
1 3 5
2
4 6
```

Megoldás



Nevezzük el két régió távolságának a két, egymáshoz legközelebbi, de különböző régiókban lévő településeik távolságát!

Megoldásunk a következő gondolatmeneten alapul. Először minden települést külön régiónak tekintünk. Ha két régió távolsága legfeljebb T, akkor összevonhatók egyetlen régióvá (gondoljuk meg, hogy miért). Ezért páronként megvizsgáljuk a régiók távolságát, és elvégezzük a lehetséges összevonásokat.

Az egyes régiókhoz tartozó települések sorszámait halmazokban tároljuk. Ezeket a halmazokat a *Régiók* halmazban helyezzük el.

```
VÁLTOZÓ Régiók MINT Halmaz(Elemtípus: RendezettHalmaz(Elemtípus: Egész))
```

A települések koordinátáit a *Koordináták* kétdimenziós tömbbe olvassuk be:

```
VÁLTOZÓ Koordináták(1…N, 1…2) MINT Egész
VÁLTOZÓ N, T MINT Egész
Be: N, T, Koordináták(,)
```

A beolvasás után létrehozzuk a régiók egyelőre egy elemű halmazait:

```
CIKLUS I=1-től N-ig
    Régiók.Add(Új RendezettHalmaz(Elemtípus: Egész))
    Régiók.Utolsó.Add(I)
CIKLUS VÉGE
```

A régiók egymástól mért távolságát két, egymásba ágyazott ciklussal vizsgáljuk meg. Visszafelé haladunk, miközben minden egyes régiót összehasonlítunk a halmazban mögötte lévőkkel (nagyobb indexűekkel). Így elkerüljük a már összevont régiók ismételt vizsgálatát (gondoljuk meg, hogy miért).

Mivel az összevonások miatt változhat a régiók (halmazok) száma, ezért a vizsgálat során nem használhatunk számlálós ciklust.

Összevonás esetén ne felejtsük el az egyik halmazt törölni a régiók közül!

```
VÁLTOZÓ I, J MINT Egész
I = Régiók.Elemszám – 2 ' a halmazelemek indexelése 0-val kezdődik
CIKLUS AMÍG I ≥ 0
    ' Az I-edik régió távolságát összehasonlítjuk az összes, nála nagyobb indexűvel
    J = I + 1
    CIKLUS AMÍG J < Régiók.Elemszám
        HA Távolság(Régiók(I), Régiók(J)) ≤ T AKKOR
            Régiók(I) = Régiók(I) ∪ Régiók(J)
            Régiók.Kivesz(Régiók(J))
        EGYÉBKÉNT
            J = J + 1 ' továbblépünk
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
    I = I – 1
CIKLUS VÉGE
```

Gondoljuk meg, hogy a $J = J+1$ utasítás miért egy feltételes elágazás *EGYÉBKÉNT* ágában található! Melyik régióra hivatkozik a *J* index az aktuális régió törlése után?

A két régió távolságát meghatározó *Távolság* függvény a feladat szövege alapján:

```
FÜGGVÉNY Távolság(H1, H2 MINT RendezettHalmaz(Elemtípus: Egész))
                                                    MINT Egész
    VÁLTOZÓ Min, Táv MINT Egész
    Min = +∞
    CIKLUS MINDEN Elem1-re a H1-ben
        CIKLUS MINDEN Elem2-re a H2-ben
            HA Elem1 ≠ Elem2 AKKOR
                Táv = Abs(Koordináták(Elem1, 1) – Koordináták(Elem2, 1)) +
                      Abs(Koordináták(Elem1, 2) – Koordináták(Elem2, 2))
                HA Táv < Min AKKOR Min = Táv
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
    CIKLUS VÉGE
    Távolság = Min
FÜGGVÉNY VÉGE
```

A válaszokat a *Régiók* halmaz elemszáma, illetve az egyes elemhalmazainak az elemei szolgáltatják. A teljes megoldást a kiírással együtt a *regio* program tartalmazza.

Az algoritmus hatékonyságát nagymértékben csökkenti, hogy ugyanazokkal az argumentumokkal sokszor meghívja a *Távolság* függvényt (vizsgáljuk meg, miért). Írjuk meg a programot úgy, hogy halmazok helyett tömbben tároljuk az egyes városok régiójának sorszámát! A régiók összevonását a sorszámok módosításával végezzük el!

Iskola/Tanár

Nemes Tihamér OITV 2013. 2. forduló 2. korcsoport 1. feladat: Iskola
Informatika OKTV 2013. 2. forduló 3. korcsoport 1. feladat: Tanár

A két, egymáshoz nagyon hasonló feladat megoldását összevonva mutatjuk be.

Egy iskola tanárairól tudjuk, hogy mikor, milyen órát tartanak. A tanárokat, a tantárgyakat, a hét napjait és a napokon belüli órákat sorszámukkal azonosítjuk. Készíts programot *iskola/tanar* néven, amely megadja:

Iskola/a)	minden napra az aznap órát tartó tanárok számát;
Tanár/a)	minden napra a szabadnapos tanárok számát;
Iskola/b)	azt a tantárgyat, amit a legtöbb tanár tanít;
Tanár/b)	azt a tanárt, akinek a legkevesebb lyukasórája van (lyukasóra: aznap előtte is van órája valamikor és utána is van órája valamikor);
Iskola/c)	azt a tanárt, akinek a legtöbb lyukasórája van (lyukasóra: aznap előtte is van órája valamikor és utána is van órája valamikor);
Tanár/c)	az adott *T* tanárt egész héten helyettesíteni tudó tanárt (ha lehetséges, akkor úgy, hogy szabadnapján senkit ne kelljen behívni az iskolába);
Iskola/d)	az adott *T* tanárt egész héten helyettesíteni tudó tanárt;
Tanár/d)	adott *T* tanárt a hét *H* napján helyettesítő tanárokat úgy, hogy minden óráján szakos helyettesítés legyen.

> A versenybizottság utólagos tájékoztatása szerint a Tanár/c) feladatnál több megoldás esetén elegendő egyet megadni. A Tanár/d) feladat esetén, ha ugyanaz a tanár helyettesít minden órát, akkor ugyanazt kell többször kiírni. De lehet olyan teszteset is, amikor minden órát más tanár helyettesít. Illetve –1-et kell kiírni akkor, ha van olyan óra, amit nem tudnak helyettesíteni.

Az *iskola.be* szöveges állomány első sorában a tanárok száma ($1 \le N \le 100$), a tantárgyak száma ($1 \le M \le 100$) és egy tanár sorszáma van ($1 \le T \le N$), egy-egy szóközzel elválasztva. A *tanar.be* fájlban ezek után áll még egy nap sorszáma ($1 \le H \le 5$). A következő sorok mindegyikében 4 egész szám van, egy-egy szóközzel elválasztva: tanár sorszáma, a tanított tantárgy sorszáma, nap (1 és 5 közötti egész szám), óra (0 és 8 közötti egész szám). Például, 3 7 2 0 azt jelenti, hogy a harmadik tanár a hetedik tantárgyat a hét második napján a nulladik órában tanítja.

Az *iskola.ki*, illetve *tanar.ki* szöveges állományba négy sort kell írni! Az első sorba az a), a másodikba a b), a harmadikba a c), a negyedikbe pedig a d) részfeladat eredményét. Ha több megoldás van, akkor az elsőt (a legkisebb sorszámút) kell megadni. Ha nincs megoldás – a c) és d) részfeladatban –, akkor –1-et kell kiírni! Az első sorban 5 szám szerepeljen, egy-egy szóközzel elválasztva!

A *tanar.ki* fájl negyedik sorában az első szám a helyettesített órák száma legyen, amit a helyettesítő tanárok sorszámai követnek, órák szerinti sorrendben, egy-egy szóközzel elválasztva!

Példa:

```
iskola.be
3 4 1
1 1 1 6
1 1 2 2
1 2 1 3
2 1 2 2
2 2 3 1
3 4 1 2
3 2 1 4
3 3 2 1
```

```
iskola.ki
2 3 1 0 0
2
1
3
```

1. tanár	1. nap	2. nap	3. nap	2. tanár	1. nap	2. nap	3. nap	3. tanár	1. nap	2. nap	3. nap
0. óra				0. óra			T2	0. óra			
1. óra				1. óra				1. óra		T3	
2. óra		T1		2. óra		T1		2. óra	T4		
3. óra	T2			3. óra				3. óra			
4. óra				4. óra				4. óra	T2		
5. óra				5. óra				5. óra			
6. óra	T1			6. óra				6. óra			
7. óra				7. óra				7. óra			
8. óra				8. óra				8. óra			

```
tanar.be
3 4 1 1
1 1 1 6
1 1 2 2
1 2 1 3
2 1 2 3
2 2 3 1
3 4 1 2
3 2 1 4
3 3 2 1
```

```
tanar.ki
1 0 2 3 3
2
3
2 2 2
```

1. tanár	1. nap	2. nap	3. nap	2. tanár	1. nap	2. nap	3. nap	3. tanár	1. nap	2. nap	3. nap
0. óra				0. óra				0. óra			
1. óra				1. óra			T2	1. óra		T3	
2. óra		T1		2. óra				2. óra	T4		
3. óra	T2			3. óra		T1		3. óra			
4. óra				4. óra				4. óra	T2		
5. óra				5. óra				5. óra			
6. óra	T1			6. óra				6. óra			
7. óra				7. óra				7. óra			
8. óra				8. óra				8. óra			

Megoldás

A feladat klasszikus példája az elemi algoritmusok alkalmazására vonatkozó programoknak. A megoldáshoz számos lineáris keresés, számlálás, maximum- és minimumkiválasztás tartozik. A lineáris keresést általában feltételes ciklussal végezzük, amit a verseny hevében könnyű elrontani. Elfelejtjük inicializálni vagy léptetni a ciklusváltozót, rosszul írjuk fel az ismétlési vagy kilépési feltételt stb.

Az alábbiakban a megoldást halmazok segítségével határozzuk meg. Számlálós ciklusokkal végigmegyünk az összes tanár összes napjának összes óráján, és a megfelelő eseteket „beledobáljuk” egy halmazba. Módszerünk sokkal rosszabb hatásfokúnak tűnik, mint a feltételes ciklusok alkalmazása. Ez azonban nem egészen igaz. Mivel a napok és órák száma konstans érték, mind a kétféle megoldás $O(n)$ futásidejű.[7] Ha a keresett érték a sorozat elején található, akkor a feltételes ciklus gyorsan véget ér, de ha a végén lévő elemet keressük, akkor gyakorlatilag ugyanannyi ideig tart, mint a számlálós ciklussal végzett vizsgálat. Átlagosan (egyenletes eloszlású, véletlenszerű adatokkal) fele annyi időt vesz igénybe a feltételes ciklus, mint a számlálós. Ez azonban nem tekinthető lényeges különbségnek a futásidők hosszában.

Mint a Bevezetésben említettük, egy versenyen (vagy akár az érettségin) a hatékony módszerek helyett/mellett előnyben részesítjük a gyorsan megtervezhető és gyorsan begépelhető (!) programokat. ☺

Lássuk ezek után a feladat megoldását! A tanárok órarendjét háromdimenziós tömbben tároljuk. Az első index a tanár, a második a nap, a harmadik pedig az óra sor-

[7] A futásidő becslését lásd a *Programozási ismeretek haladóknak* tankönyv 43. leckéjében!

száma. A tömbelem értéke adja meg a tantárgyat (számokkal reprezentálva). Ha nincs órája a tanárnak az adott napon és időpontban, azt 0-val jelezzük.

```
VÁLTOZÓ Órák(1…N, 1…5, 0…8) MINT Egész
Be: az Órák tömb elemeinek értéke
```

Iskola/a)
Tanár/a)

A két feladat egymáshoz nagyon hasonló, itt a szabadnapos tanárok számának meghatározását mutatjuk be. Mivel a Tanár/c) feladatban szükségünk lesz arra, hogy az egyes napokon mely tanárok szabadnaposak, a tárolásra tömböt használunk. A tömb *I* indexű eleme megadja az *I*. napon szabadnapos tanárok halmazát:

```
VÁLTOZÓ Szabadnapos(1…5) MINT Halmaz(Elemtípus: Egész)
```

A tömbelemeket képező halmazok létrehozása után minden napra feltöltjük a halmazt az összes tanárral, és kivesszük belőle azokat a tanárokat, akiknek aznap van órájuk:

```
CIKLUS Nap=1-től 5-ig
   Szabadnapos(Nap) = {1, 2, …, N}
   CIKLUS Tanár=1-től N-ig CIKLUS Óra=0-tól 8-ig
      HA Órák(Tanár, Nap, Óra) ≠ 0 AKKOR Szabadnapos(Nap).Kivesz(Tanár)
   CIKLUS VÉGE (Óra) CIKLUS VÉGE (Tanár)
   Ki: Szabadnapos(Nap).Elemszám
CIKLUS VÉGE (Nap)
```

A rövid és egyszerű kód önmagáért beszél. Megspóroltuk a feltételes ciklus szervezésével kapcsolatos tennivalókat, a ciklusfeltételek megfogalmazását, a ciklusváltozó léptetését többdimenziós tömb (egymásba ágyazott ciklusok) esetén.

Iskola/b)

Az előző részfeladathoz hasonló logikát követünk. Végignézzük az összes tanár összes napjának összes óráját, közben egy tömbben kigyűjtjük, hogy az egyes tantárgyakat mely tanárok tanítják. A tömb *I* indexű eleme az *I*. tantárgyat tanító tanárok halmaza:

```
VÁLTOZÓ Tanítják(0...M) MINT Halmaz(Elemtípus: Egész)
```

A 0 indexű elem felhasználásával elkerülhetjük a feltételes elágazást a kigyűjtésben (az *Órák* tömb 0 értékű elemét is „tantárgynak” tekintjük).

A kigyűjtést végző ciklusok:

```
CIKLUS Tanár=1-től N-ig CIKLUS Nap=1-től 5-ig CIKLUS Óra=0-tól 8-ig
   Tanítják(Órák(Tanár, Nap, Óra)).Add(Tanár)
CIKLUS VÉGE (Óra) CIKLUS VÉGE (Nap) CIKLUS VÉGE (Tanár)
```

A választ a *Tanítják* tömbben a legtöbb elemet tartalmazó halmaz elemszáma szolgáltatja.

Halmazok felhasználásával és számlálós ciklussal ismét megnyerően egyszerű kódot kaptunk.

Tanár/b)
Iskola/c)

A két feladat csak a minimum-, illetve maximumkiválasztásban különbözik egymástól.

Először megszámoljuk, hogy az egyes tanároknak hány lyukasórájuk van. Ezek számát a *DbLyukasóra* tömbben gyűjtjük, melynek *I* indexű eleme az *I*. tanár lyukasóráinak számát tartalmazza:

```
VÁLTOZÓ DbLyukasóra(1...N) MINT Egész
```

Egy tanár lyukasóráinak számát egy adott napon a következőképpen határozzuk meg. Az *Órái* rendezett halmazba „beledobáljuk” az összes órájának a sorszámát (nem a tantárgyat!) az adott napon. Ha volt legalább két órája, akkor a lyukasórák számát az

```
Órái.Utolsó – Órái.Első + 1 – Órái.Elemszám
```

kifejezés adja meg (ellenőrizzük egy konkrét példán keresztül!). Azaz:

```
VÁLTOZÓ Órái MINT RendezettHalmaz(Elemtípus: Egész)
CIKLUS Tanár=1-től N-ig CIKLUS Nap=1-től 5-ig
   Órái.Töröl()
   CIKLUS Óra=0-tól 8-ig
      HA Órák(Tanár, Nap, Óra) ≠ 0 AKKOR Órái.Add(Óra)
   CIKLUS VÉGE (Óra)
   HA Órái.Elemszám > 1 AKKOR
      DbLyukasóra(Tanár) += Órái.Utolsó – Órái.Első + 1 – Órái.Elemszám
   ELÁGAZÁS VÉGE
CIKLUS VÉGE (Nap) CIKLUS VÉGE (Tanár)
```

Ezek után már csak a minimum-, illetve maximumkiválasztás van hátra.

Ha eddig nem tette volna meg, akkor most mindenképpen javasoljuk az Olvasónak, hogy oldja meg ezt a feladatot a „klasszikus” módon, tömbökkel, lineáris keresésekkel. ☺

Tanár/c)
Iskola/d)

Mivel az Iskola/d) feladat szigorúbb követelményeket támaszt, ezért ezt oldjuk meg, de eközben megkapjuk a Tanár/c) feladat megoldását is.

Először az összes tanárt tartalmazó halmazból kivonjuk azokat a tanárokat, akiknek van ütköző órájuk a *T* tanárral. Így megkapjuk azokat a tanárokat, akik tudják helyettesíteni.

```
VÁLTOZÓ Tudhelyettesíteni MINT RendezettHalmaz(Elemtípus: Egész) = {1, 2, …, N}
CIKLUS Tanár=1-től N-ig CIKLUS Nap=1-től 5-ig CIKLUS Óra=0-tól 8-ig
   HA Órák(Tanár, Nap, Óra) ≠ 0 ÉS Órák(T, Nap, Óra) ≠ 0 AKKOR
      Tudhelyettesíteni.Kivesz(Tanár)
   ELÁGAZÁS VÉGE
CIKLUS VÉGE (Óra) CIKLUS VÉGE (Nap) CIKLUS VÉGE (Tanár)
```

Némileg gyorsíthatjuk a végrehajtást, ha a ciklus előtt a *T* tanárt kivesszük a *Tudhelyettesíteni* halmazból, és a cikluson belül is kihagyjuk a vizsgálatból.

Ezzel a Tanár/c) feladatot megoldottuk, a *Tudhelyettesíteni* halmaz bármely eleme megfelel (ha nem üres).

Az Iskola/d) feladat megoldását akkor folytatjuk, ha a *Tudhelyettesíteni* halmaz nem üres (egyébként nem tudjuk helyettesíteni minden óráján a *T* tanárt). Lehet, hogy mégis lesz olyan helyettesítő tanár, akinek a szabadnapján kell bejönnie, ezért félretesszük például a halmaz első elemét (lehet, hogy a halmaz a végére ki fog ürülni):

```
VÁLTOZÓ Tartalék MINT Egész = Tudhelyettesíteni.Első
```

Ezek után a *Vanórája* halmazban összegyűjtjük, hogy a *T* tanárnak mely napokon vannak órái:

```
VÁLTOZÓ Vanórája MINT RendezettHalmaz(Elemtípus: Egész)
CIKLUS Nap=1-től 5-ig CIKLUS Óra=0-tól 8-ig
   HA Órák(T, Nap, Óra) ≠ 0 AKKOR Vanórája.Add(Nap)
CIKLUS VÉGE (Óra) CIKLUS VÉGE (Nap)
```

Majd a helyettesíteni tudó tanárok halmazából kivonjuk azoknak a halmazát, akiknek ezeken a napokon szabadnapjuk van (hiszen a szabadnapjukon kellene bejönniük helyettesíteni). Ezen tanárok halmazát minden napra a Tanár/a) feladatban már meghatároztuk (a *Szabadnapos* tömb elemei).

```
CIKLUS MINDEN Nap-ra a Vanórája halmazban
   Tudhelyettesíteni = Tudhelyettesíteni – Szabadnapos(Nap)
CIKLUS VÉGE
```

Ha a *Tudhelyettesíteni* halmaz nem üres, akkor kiírhatjuk például az első elemét. Egyébként pedig a *Tartalék* tanár lesz a megoldás.

Tanár/d)

A feladatot lényegében az előző gondolatmenethez hasonlóan oldjuk meg. Először a *Szakok* tömbben kigyűjtjük, hogy melyik tanár milyen szakokkal rendelkezik (milyen tantárgyakat tanít). A tömb *I* indexű eleme megadja az *I*. tanár szakjainak a halmazát:

```
VÁLTOZÓ Szakok(1…N) MINT Halmaz(Elemtípus: Egész)
CIKLUS Tanár=1-től N-ig CIKLUS Nap=1-től 5-ig CIKLUS Óra=0-tól 8-ig
   HA Órák(Tanár, Nap, Óra) ≠ 0 AKKOR
      Tantárgy = Órák(Tanár, Nap, Óra)
      Szakok(Tanár).Add(Tantárgy)
   ELÁGAZÁS VÉGE
CIKLUS VÉGE (Óra) CIKLUS VÉGE (Nap) CIKLUS VÉGE (Tanár)
```

Kigyűjtjük továbbá a feladatban megadott *H* napon minden egyes órában az akkor lyukasórás tanárokat:

```
VÁLTOZÓ Lyukasórás(0…8) MINT Halmaz(Elemtípus: Egész)
CIKLUS Óra=0-tól 8-ig CIKLUS Tanár=1-től N-ig
   HA Órák(Tanár, H, Óra) == 0 AKKOR Lyukasórás(Óra).Add(Tanár)
CIKLUS VÉGE (Tanár) CIKLUS VÉGE (Óra)
```

Végül végignézzük az órákat, és megnézzük, hogy az adott órában van-e a lyukasórás tanárok között megfelelő szakos. Ha igen, akkor tudjuk helyettesíteni a *T* tanárt.

A feladat minden órára csak egyetlen helyettesítő tanár meghatározását várja el. Mi ennél többet teszünk: meghatározzuk az összes megfelelő tanárt úgy, hogy rendezett

halmazban gyűjtjük őket, minden egyes órának megfelelően. A halmazokat a *Helyettesít* tömbben tároljuk:

```
VÁLTOZÓ Helyettesít(0..8) MINT RendezettHalmaz(Elemtípus: Egész)
```

Közben számoljuk azt is, hány órája volt a *T* tanárnak (*DbT*), és ebből hány órát sikerült helyettesíteni (*Db*, ezt ki kell írni a *tanar.ki* fájlba)

```
DbT = 0, Db = 0
CIKLUS Óra=0-tól 8-ig
    Helyettesít(Óra) = Új RendezettHalmaz(Elemtípus: Egész)
    Tantárgy = Órák(T, H, Óra)
    HA Tantárgy ≠ 0 AKKOR ' van órája a T tanárnak, keresünk helyettesítő tanárt
        DbT = DbT + 1
        CIKLUS MINDEN Tanár-ra a Lyukasórás(Óra) halmazban
            HA Szakok(Tanár).Tartalmazza(Tantárgy) AKKOR ' tudja helyettesíteni
                Helyettesít(Óra).Add(Tanár)
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
        HA Helyettesít(Óra).Elemszám > 0 AKKOR Db = Db + 1
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A *T* tanárt akkor sikerül minden óráján helyettesíteni, ha *Db* = *DbT*. Ekkor az egyes órákban választhatjuk például a *Helyettesít* halmazok első elemét helyettesítő tanárnak.

A teljes megoldást az *iskola*, illetve *tanar* programok tartalmazzák.

A magyarázó szövegtől eltekintve a fenti algoritmusok nagyon rövidek. Ugyanezt a gondolatmenetet kellene végigvinni a klasszikus, lineáris keresésekre, feltételes ciklusokra épülő megoldás esetén is. Halmazok és számlálós ciklusok alkalmazásával azonban sokkal egyszerűbben, gyorsabban kódolható megoldáshoz jutottunk.

Fogadás

Nemes Tihamér OITV 1999. 3. forduló 2. korcsoport 3. feladat

Byteland ország követsége fogadást rendezett. A fogadásra *N* számú ország nagykövetét hívták meg. A vendégek különböző időpontokban érkeztek, és minden vendég azonos ideig volt jelen a fogadáson. A házigazda feljegyezte, hogy ki-kivel találkozott a fogadás során. Az előbb érkezett *A* vendég találkozott a később érkezett *B* vendéggel, ha *A* később távozott, mint *B* érkezett. Azt is tudjuk, hogy az első vendég érkezésétől az utolsó távozásáig minden időpontban jelen volt legalább egy vendég.

A feladat annak kiderítése, hogy a vendégek milyen sorrendben érkezhettek a fogadásra.

Írj programot *fogad* néven, amely kiszámítja a vendégek egy lehetséges érkezési sorrendjét (nem az érkezési időpontokat!).

A *fogad.be* állomány első sora a vendégek ($2<N\leq200$) számát tartalmazza. A második sorban a fogadás alatt találkozott vendégpárok száma áll ($1\leq M\leq10000$). A következő *M* sor mindegyike egy *X Y* számpárt ($1\leq X, Y\leq N$) tartalmaz egy szóközzel elválasztva, ami azt jelenti, hogy az *X* és az *Y* vendég találkozott a fogadás során.

A *fogad.ki* állomány első és egyetlen sorába *N* számot kell írni egy-egy szóközzel elválasztva, ami a vendégek egy lehetséges érkezési sorrendjét mutatja.

Példa:

```
fogad.be              fogad.ki
4                     1 2 4 3     (3 4 2 1 is jó megoldás)
3
1 2
2 4
3 4
```

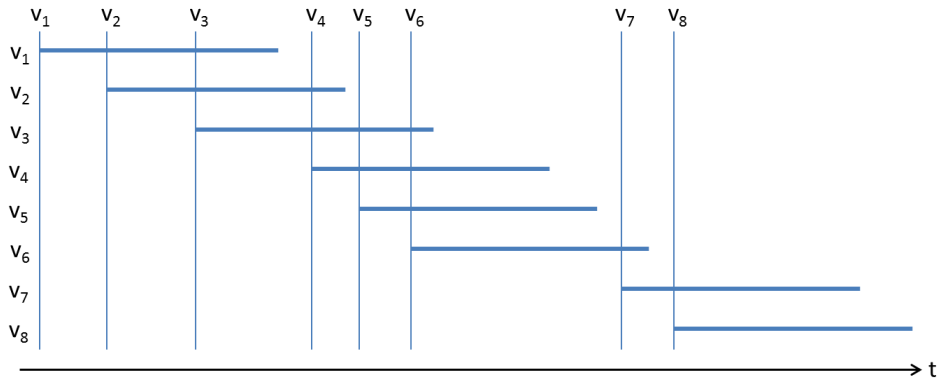
Megoldás

A feladat szövege szerint két vendég „találkozása” azt jelenti, hogy volt olyan pillanat, amikor mindketten ott voltak a fogadáson.

Hogyan választhatjuk ki az elsőként érkező vendéget, és hogyan határozhatjuk meg a sorrendet? Az alábbi gondolatmenetben egyelőre azonosnak tekintjük azokat a vendégeket, akik egyszerre érkeztek (így egyszerre távoztak), de ez nem befolyásolja a kialakuló algoritmust.

Legyenek az egymás után érkező vendégek rendre: v_1, v_2, …, v_n, tehát:
$t(v_1) < t(v_2) < \ldots, < t(v_n)$, ahol $t(v_i)$ az i. vendég érkezési ideje.

Az alábbi ábra példaként azokat az időtartamokat ábrázolja, amikor az egyes vendégek jelen voltak a fogadáson. Az érkezési időpontokat függőleges vonallal jelöltük.



A vendégek jelenléte a fogadáson (8 vendég esetén)

Az ábra alapján gyűjtsük táblázatba a találkozásokat! Egyszerűsíti a gondolatmenetet, ha úgy tekintjük, hogy minden vendég saját magával is találkozott.

Minden v vendéghez hozzunk létre egy $T(v)$ halmazt, amely azokat a vendégeket tartalmazza, akikkel v találkozott (saját magát is beleértve). A halmazokat a $T(1…N)$ tömbben tároljuk. A tömbelemek indexe megfelel a vendég sorszámának. A fenti példában $T(v_1) = \{v_1, v_2, v_3\}$. A halmazok elemeit szemléletesen leolvashatjuk a következő táblázatból.

Ki – Kivel	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	×	×	×					
v_2	×	×	×	×				
v_3	×	×	×	×	×	×		
v_4		×	×	×	×	×		
v_5			×	×	×	×		
v_6			×	×	×	×	×	
v_7						×	×	×
v_8							×	×

A találkozások táblázata (pirossal jelölve az első három halmaz közös része)

Vegyük észre, hogy az elsőként érkező vendég $T(v_1)$ halmaza része bármely olyan vendég halmazának, akivel találkozott a fogadáson (a táblázatban pirossal jelölve)! Hiszen ha v_1 találkozott v_i-vel, akkor a v_i vendégnek is találkoznia kellett v_1-gyel. A v_i vendég azonban később jött, mint v_1, tehát később is ment el (hiszen mindegyik vendég ugyanannyi ideig tartózkodott a fogadáson). Ezért minden további vendéggel találkozott, akivel v_1 találkozott.

Ezzel a tulajdonsággal csak az első (és az utolsó – lásd később) vendég rendelkezik, a többi vendég pedig nem. Ha ugyanis a v_i vendég nem először (és nem utoljára) érkezett, akkor a feladat szövege alapján találkoznia kellett előtte és utána érkező vendégekkel is. De csak akkor lehettek a vendégek folyamatosan jelen a fogadáson, ha az előtte érkező vendég találkozott olyan vendéggel, akivel az utána érkező már nem, és fordítva. Ezért a $T(v_i)$ halmaz nem lehet része minden olyan vendég halmazának, akivel a v_i találkozott.

Ha időben megfordítva vizsgáljuk az érkezéseket és távozásokat, akkor az előbbi tulajdonsággal az utolsó vendég is rendelkezik. Ez csupán annyit jelent, hogy egy lehetséges érkezési sorrend fordítottja is megoldása a feladatnak.

Az első vendéget tehát úgy találhatjuk meg, ha megkeressük, melyik az a $T(v)$ halmaz, amelyik része bármely benne lévő vendég halmazának. Ehhez megkeressük, hogy melyik az a v_i vendég, amelynél a $T(v_i)$-ben lévő vendégek halmazainak a metszete megegyezik a $T(v_i)$ halmazzal:

```
Vendég = 0
Megvan = Hamis
CIKLUS AMÍG NEM Megvan
    Vendég += 1
    Metszet = T(Vendég)
    CIKLUS I=1-től N-ig
        HA I ≠ Vendég ÉS T(I).Tartalmazza(Vendég) AKKOR
            Metszet = Metszet ∩ T(I)
        ELÁGAZÁS VÉGE
        Megvan = (Metszet == T(Vendég))
CIKLUS VÉGE
```

A ciklusból való kilépés után az elsőként érkező vendég sorszámát a *Vendég* változó jelzi.

Mivel a feladat feltételei alapján biztosan van megoldás, ezért lineáris keresés helyett kiválasztást végeztünk. Az algoritmust tovább gyorsíthatjuk, ha a belső ciklus

feltételes elágazásából elhagyjuk az $I \neq Vendég$ vizsgálatot. Így ugyan egy felesleges halmazműveletet is elvégez a program, a reláció viszont csak egyetlen egyszer nem teljesül, ezért kár lenne a többi esetben megvizsgálni.

Ezek után hogyan állapíthatjuk meg az érkezések lehetséges sorrendjét? Az egymás után érkező vendégek sorszámát gyűjtsük össze az *S* halmazban! Itt felhasználjuk, hogy a halmaz elemeire indexükkel hivatkozhatunk, és az elemek indexe jelzi a hozzáadás sorrendjét.

A halmaz első eleme nyilván az első vendég, akinek a sorszámát már tudjuk:

```
S.Add(Vendég)
```

Jelöljük *Utolsó*-val egy adott pillanatban az utoljára érkezett vendég sorszámát! Eddig:

```
Utolsó = Vendég
```

Tekintsük azon vendégek *Következők* halmazát, akik az eddigi *Utolsó* után érkeztek, de még találkoztak vele! Nyilván $Következők = T(Utolsó) - S$, hiszen az *S* a már megérkezett vendégeket tartalmazza. Képezzük a *Következők* halmaz elemeire is a $T(v_i) - S$ különbségeket ($v_i \in Következők$). Ezek azok a vendégek, akikkel a *Következők* halmazban lévő vendégek találkoztak, de az *Utolsó* után érkeztek.

Az alábbi táblázat a feldolgozás egy pillanatnyi állapotát mutatja. Az *S* halmaz elemeit zölddel jelöltük, $Utolsó = v_3$. A *Következők* halmaz elemeit kék, a $T(v_i) - S$ halmazok elemeit pedig piros szín jelzi.

Ki – Kivel	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	×	×	×					
v_2	×	×	×	×				
v_3	×	×	×	×	×	×		
v_4		×	×	×	×	×		
v_5			×	×	×	×		
v_6			×	×	×	×	×	
v_7						×	×	×
v_8							×	×

Az S (zöld), a Következők (kék) és a $T(v_i) - S$ (piros) halmazok elemei a fogadás egy időpontjában

A táblázat alapján a következőket állapíthatjuk meg.

1. A *Következők* halmazban lévő vendégek közül az jött később, aki több, később érkező vendéggel találkozott. A v_6 vendég például később jött, mint a v_4 vagy a v_5.
2. Ha ugyanannyi később jövő vendéggel találkoztak, akkor az jött előbb, aki több, előbb érkező vendéggel találkozott. Ez egyben azt is jelenti, hogy összességében is több vendéggel találkozott. Például a v_4 vendég előbb jött, mint a v_5.
3. Ha ugyanannyi előbb, és ugyanannyi később jövő vendéggel találkoztak, akkor nem tudjuk megállapítani a sorrendet, hiszen pontosan ugyanazokkal a vendégekkel találkoztak.

Feladatunk a $T(v_i) - S$ halmazok közül a legkisebb elemszámút kiválasztani (ahol $v_i \in$ *Következők*). Ha pedig több ilyen is van, akkor azt választjuk, amelyiknek több elemet tartalmaz a $T(v_i)$ halmaza.

Az algoritmus:

```
CIKLUS I=2-től N-ig
    Következők = T(Utolsó) – S
    Min = +∞
    CIKLUS MINDEN Vendég-re a Következők-ben
        Elemszám = (T(Vendég) – S).Elemszám
        HA Elemszám < Min AKKOR
            Utolsó = Vendég
            Min = Elemszám
        EGYÉBKÉNT HA Elemszám == Min AKKOR
            HA T(Vendég).Elemszám > T(Utolsó).Elemszám AKKOR
                Utolsó = Vendég
            ELÁGAZÁS VÉGE
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
    S.Add(Utolsó)
CIKLUS VÉGE
```

A teljes megoldást a *fogad* program tartalmazza.

Mint már utaltunk rá, ha találunk egy megoldást, akkor a fordított sorrend is megoldása a feladatnak. Gondoljuk meg, mitől függ, hogy egy adott érkezési sorrendet vagy pedig a fordítottját kapjuk-e meg! Mitől függ azoknak a vendégeknek a sorrendje a megoldásban, akik pontosan ugyanazokkal a vendégekkel találkoztak (saját magukat is beleértve)?

További feladatok a halmazokra

OITV 1995-2f1	Kocka
OITV 1995-2f2	Állatok
OITV 1996-2f2	Bűvös négyzet
OITV 2003-3f1	Verseny
OITV 2004-3f2	Rémhír
OITV 2006-3f2	Ismerősök
OITV 2006-3f2	Csoportok
OITV 2007-1f2	Unió
OITV 2011-3f2	Csapat
OITV 2014-3f1	Lottó
OKTV 1996-3f3	Maga mindent kétszer mond
OKTV 2006-3f3	Részhalmazok
OKTV 2010-3f3	Játékos
OKTV 2011-3f3	Csoportbeosztás
OKTV 2012-2f3	Verseny

Megjegyezzük, hogy halmazokat a további leckékben is gyakran alkalmazunk.

3. Asszociatív tömbök

Kapcsolódó lecke:
Programozási ismeretek haladóknak (Műszaki Kiadó, 2012)
22. Asszociatív adattípusok

Asszociatív tömböket főleg olyan feladatok megoldásában használunk, melyekben gyakran kell keresni egy-egy elemet, gyakran kell módosítani az elem tulajdonságait. Az asszociatív tömb segítségével kiküszöböljük a sok lineáris keresést. Az elemre a kulcs alapján közvetlenül hivatkozhatunk. Tartsuk szem előtt, hogy a kulcsok egyedi értékek, különböző elemek nem rendelkezhetnek egyforma kulccsal! Objektum-orientált módszerek alkalmazása nélkül kulcsként általában csak egyszerű típust adhatunk meg. Az elemek értéke (*Value* tulajdonsága) azonban már tetszőleges, akár összetett típusú is lehet.

> A C++ STL-ben a szótár (*map*) a kulcsok szerint rendezetten tárolja az elemeket. Mivel a rendezettség megtartása eléggé időigényes, a C++11 szabvány rendezettség nélküli változattal bővíti a sablontárat (*unordered_map*). Ha egy szótárban ugyanazzal a kulccsal több bejegyzést kell elhelyeznünk, használhatjuk a hagyományos C++ nyelv *multimap*, illetve a C++11 *unordered_multimap* konténereit.
> A C++ *map* és *unordered_map* tároló elemei korlátozás nélkül módosíthatók, azonban ezt a megoldások C++ változataiban nem használtuk ki.

> A Visual Basic-ben az asszociatív tömbök (*Dictionary*, *SortedDictionary*) *ElementAt* metódusa, illetve a ciklusnak az elemre hivatkozó iterátora csak az elem lekérdezésére alkalmas, módosítására nem használhatjuk. A módosítás/törlés mikéntjét az *Almák* feladatnál mutatjuk be.

Az úgynevezett **multihalmaz** adatszerkezetet is megvalósíthatjuk asszociatív tömbbel. Multihalmaznak nevezzük a halmaz adatszerkezetet olyan általánosítását, amelyben minden elemhez tartozik egy úgynevezett multiplicitás.[8] Az elem multiplicitása megadja, hogy az elem hányszor (hány példányban) része a halmaznak. Ha asszociatív tömbbel realizáljuk a multihalmazt, akkor egy tömbelem kulcsa jelölheti a halmaz elemét, a tömbelem értéke pedig a multiplicitást.

> A C++ nyelv szabványos sablontára külön konténereket (*multiset*, *unorderd_multiset*) biztosít a multihalmazok kezelésére.

Asszociatív tömböt alkalmazhatunk a gráfok tárolására, ha a csúcsok sorszámok helyett egyéb (például sztring típusú) azonosítóval rendelkeznek.

A továbbiakban a .NET nyelvhasználatának megfelelően az asszociatív tömböt szótárnak nevezzük.

> Visual Basic-ben a *Szótár*(*I*) az *I* kulccsal rendelkező elem értékét (*Value*) jelzi. Ha számlálós ciklussal szeretnénk feldolgozni a szótárobjektumot, akkor használjuk a *Szótár*.*ElementAt*(*I*) hivatkozást!

[8] A multihalmaz ettől eltérő értelmezései is előfordulnak az informatikában.

Hamupipőke

Nemes Tihamér OITV 1994. 2. forduló 1. korcsoport 1. feladat

Hamupipőke gonosz mostohájától azt a feladatot kapta, hogy különböző színű lencséket válogasson szét. Az egyező színűeket azonos tálkába kell tennie. Előre nem tudja, hogy hányféle színű lencse lesz, ez csak feldolgozás közben derül ki. Készíts *lencse* néven programot, amellyel segíthetsz neki!

A program olvassa be a billentyűzetről egyenként az egyes lencsék színeit, majd írja ki, hogy összesen hányféle lencse volt, s milyen színűből mennyit kell kiválogatni! A beolvasás végét egy üres sor jelzi.[9]

Példa:

Bemenet	*Kimenet*
sárga	A színek száma: 3
sárga	fehér lencse: 1 db
zöld	sárga lencse: 3 db
sárga	zöld lencse: 2 db
fehér	
zöld	
(üres sor)	

Megoldás

A lencsék adatait olyan rendezett szótárban tároljuk, melyben az elem kulcsa a szín, értéke pedig a darabszám:

```
VÁLTOZÓ Lencsék MINT RendezettSzótár(Kulcstípus: Sztring, Értéktípus: Egész)
```

A beolvasásnál minden színt hozzáadunk a szótárhoz, ha még nincs benne. Aztán pedig ha benne volt, ha nem, megnöveljük az adott színű lencsék darabszámát:

```
Be: Szín
HA NEM Lencsék.Kulcsként Tartalmazza(Szín) AKKOR Lencsék.Add(Szín, 0)
Lencsék(Szín) = Lencsék(Szín) + 1
```

Ezek után nincs más dolgunk, mint kiírni a szótár elemszámát, majd minden egyes elem kulcsát (szín) és értékét (darabszám).

A teljes megoldást a *lencse* program tartalmazza. Szokásunk szerint javasoljuk az Olvasónak, hogy készítse el a programot a „klasszikus” módon is, statikus tömbökkel és lineáris keresésekkel.

Vegyük észre, hogy a *Lencsék* szótárt multihalmaznak tekinthetjük. A kulcsok (színek) alkotják a halmaz elemeit, a szótárelem értéke pedig megadja a multiplicitást (az adott színből hány darab lencse van a tálban).

[9] A 2. korcsoport hasonló feladatában fájlból olvasás és képernyőre írás szerepelt.

Programozási verseny

Nemes Tihamér OITV 2000. 2. forduló 1. korcsoport 3. feladat

Egy programozási versenyen minden versenyző választhat egy programozási nyelvet, amin dolgozni fog. Írjunk programot *verseny* néven, amely a versenyzők adatait dolgozza fel! Programunknak először a versenyzők *N* számát kell beolvasnia, majd pedig a versenyzők nevét, illetve a választott programozási nyelvet. A program határozza meg

a) a használt nyelvek számát;
b) az egyes nyelveket választó versenyzők számát;
c) az egyes nyelveket választó versenyzők nevét!

Példa:

Bemenet	*Kimenet*
`3`	`2 nyelv`
`Kiss Péter`	`Pascal: 2 Logo: 1`
`Pascal`	`Pascal: Kiss Péter, Kovács Anna`
`Nagy Tamás`	`Logo: Nagy Tamás`
`Logo`	
`Kovács Anna`	
`Pascal`	

Megoldás

A feladat megoldása szemléletes példáját adja annak az esetnek, amikor egy jól megválasztott adatszerkezet a beolvasást követően azonnal szolgáltatja a megoldást. Nincs szükség semmilyen feldolgozásra!

Az adatokat szótárban tároljuk. Az elemek kulcsa a programozási nyelv, értéke pedig egy olyan halmaz, amely az adott nyelvet választó versenyzőket tartalmazza.

```
VÁLTOZÓ Nyelvek MINT Szótár(Kulcstípus: Sztring,
                            Értéktípus: Halmaz(Elemtípus: Sztring))
```

A beolvasás során a megadott nyelv halmazához hozzáadjuk a versenyző nevét. Előtte azonban meg kell vizsgálnunk, hogy szerepelt-e már a nyelv az adatok között. Ha nem, akkor először létrehozzuk a hozzá tartozó halmazt:

```
VÁLTOZÓ Versenyzők MINT Halmaz(Elemtípus: Sztring)
CIKLUS I = 1-től N-ig
    Be: Név, VálasztottNyelv
    HA Nem Nyelvek.Kulcsként Tartalmazza(VálasztottNyelv) AKKOR
        Versenyzők = Új Halmaz(Elemtípus: Sztring)
        Nyelvek.Add(VálasztottNyelv, Versenyzők)
    ELÁGAZÁS VÉGE
    Nyelvek(VálasztottNyelv).Add(Név)
CIKLUS VÉGE
```

A *Nyelvek*(*VálasztottNyelv*) egy halmaz, amihez hozzáadhatunk elemet (a versenyző nevét).

Ügyeljünk arra, hogy a *Versenyzők* halmazt a cikluson belül hozzuk létre, különben az összes szótárelemhez ugyanaz a halmaz tartozna!

Mint említettük, a feladatban szereplő kérdésekre további algoritmusok alkalmazása nélkül, csupán a szótár tulajdonságaival és elemeinek kiírásával válaszolhatunk.

A választott nyelvek száma egyszerűen: *Nyelvek.Elemszám*

Az egyes nyelveket választó versenyzők számát a halmazok elemszáma adja meg:

```
CIKLUS MINDEN Nyelv-re a Nyelvek-ben
    Ki: Nyelv.Kulcs, Nyelv.Érték.Elemszám
CIKLUS VÉGE
```

Végül az egyes nyelveket választó versenyzők nevét a halmazelemek felsorolásával kapjuk meg:

```
CIKLUS MINDEN Nyelv-re a Nyelvek-ben
    Ki: Nyelv.Kulcs ' a programozási nyelv
    CIKLUS MINDEN Versenyző-re a Nyelv.Érték-ben ' a Nyelv.Érték egy halmaz
        Ki: Versenyző
    CIKLUS VÉGE
CIKLUS VÉGE
```

A teljes megoldást a *verseny* program tartalmazza.

Az Olvasónak javasoljuk a 2007-es verseny 3. fordulójának 1. korcsoportjából az ehhez nagyon hasonló 3. feladat (*Nyelvek*) önálló megoldását.

Alma

Nemes Tihamér OITV 2012. 3. forduló 1. korcsoport 3. feladat

Egy almatermelő *N* fajta almát termel, ismerjük, hogy melyik fajtából mennyit. Egy kereskedő *M* fajta almát szeretne venni tőle, azt is tudjuk, hogy melyik fajtából mennyit. Készítsünk programot *alma* néven, amely megadja, hogy a termelőnek melyik fajtából mennyi marad, valamint hogy a kereskedő melyik fajtából mennyit tud vásárolni!

Példa:

Bemenet

```
Termelő fajtái száma: 3
1. fajta neve: jonagold
1. fajta mennyisége: 100
2. fajta neve: golden
2. fajta mennyisége: 30
3. fajta neve: idared
3. fajta mennyisége: 500
```

```
Kereskedő fajtái száma: 2
1. fajta neve: golden
1. fajta mennyisége: 50
2. fajta neve: starking
2. fajta mennyisége: 100
```

Kimenet

```
Termelőnél marad:
jonagold 100
idared 500
```

```
Kereskedő vesz:
golden 30
```

Megoldás

A termelő és a kereskedő adatait egy-egy szótárban tároljuk. Az elemek kulcsa az almafajta neve, értéke pedig a mennyiség az adott fajtából.

```
VÁLTOZÓ Termelő MINT Szótár(Kulcstípus: Sztring, Értéktípus: Egész)
VÁLTOZÓ Kereskedő MINT Szótár(Kulcstípus: Sztring, Értéktípus: Egész)
```

A feladat megoldásához módosítani kell a szótárelemek értékét, esetleg törölni azokat a fajtákat, amelyek elfogytak. A .NET-ben erre sem a számlálós, sem az iterátoros ciklus nem alkalmas. A feldolgozást kétféleképpen végezhetjük:

1. Felhasználjuk a szótár *Kulcsok* (illetve *Elemek*) kollekcióját, melyet tömbbé alakítunk, majd a tömbelemekre futtatjuk a ciklust.
2. Kigyűjtjük egy halmazba a módosításra/törlésre kerülő szótárelemek kulcsait, majd a halmazelemekre futtatjuk a ciklust.

A két módszer egyenértékű egymással. Az alábbiakban mindkét módszert bemutatjuk.

A kereskedő nem tud olyan fajtát vásárolni, amilyennel a termelő nem rendelkezik, ezért ezeket a fajtákat töröljük a *Kereskedő* szótárból. Ehhez a *Kulcsok* kollekció segítségével készítünk egy tömböt, amely a kulcsokat (fajtákat) tartalmazza, majd ennek alapján végezzük el a törlést (1. módszer):

```
VÁLTOZÓ Fajták() MINT Sztring
Fajták = Kereskedő.Kulcsok.Tömbként()
CIKLUS MINDEN Fajta-ra a Fajták-ban
    HA NEM Termelő.Tartalmazza(Fajta) AKKOR Kereskedő.Kivesz(Fajta)
CIKLUS VÉGE
```

Ügyeljünk arra, hogy a *Fajták* tömb mindvégig az eredeti kulcsokat tartalmazza, és a törlések során nem módosul!

Megjegyezzük, hogy feltétlenül szükség van a *Fajták* tömbre. A törlés miatt a *Kulcsok* kollekciót nem használhatjuk a ciklusfejben, tehát a következő ciklus futási hibához vezet:

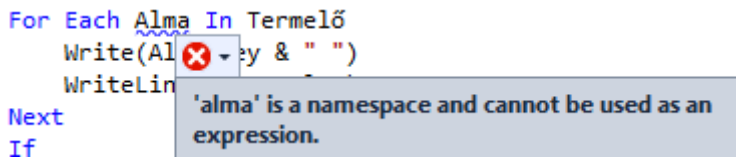
```
CIKLUS MINDEN Fajta-ra a Kereskedő.Kulcsok-ban
    HA NEM Termelő.Tartalmazza(Fajta) AKKOR Kereskedő.Kivesz(Fajta)
Ciklus Vége
```

A mennyiségek módosításához halmazokba gyűjtjük azokat a fajtákat, melyekből több van, illetve azokat, melyekből legfeljebb annyi van, mint amennyit a kereskedő venne (2. módszer):

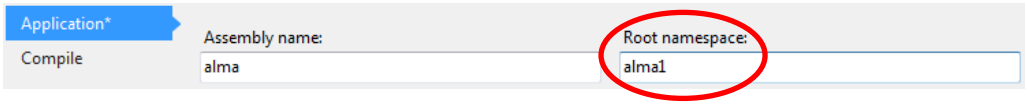
```
VÁLTOZÓ Sok, Kevés MINT Halmaz(Elemtípus: Sztring)
Sok = Új Halmaz(Típus: Sztring)
Kevés = Új Halmaz(Elemtípus: Sztring)
CIKLUS MINDEN Alma-ra a Kereskedő-ben
    Fajta = Alma.Kulcs ' nem szükséges, de így áttekinthetőbbek az utasítások
    HA Termelő(Fajta) > Kereskedő(Fajta) AKKOR Sok.Add(Fajta)
    EGYÉBKÉNT Kevés.Add(Fajta)
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

Emlékezzünk arra, hogy a *Termelő*(*Fajta*), illetve a *Kereskedő*(*Fajta*) a fajtából rendelkezésre álló mennyiségeket jelenti, így a feltételes elágazásokban ezeket hasonlítjuk össze!

> Mivel a Visual Basic a projekt (program) nevét egy névtér azonosítójának tekinti, az *alma* nevű projektben nem használhatjuk ki egy *Alma* nevű változó automatikus deklarációját (például iterátorként). Vagy válasszunk másik változónevet, vagy pedig módosítsuk a projekthez tartozó névtér nevét például *alma1*-re (*Project*/*alma Properties*, *Application* panel, *Root namespace* szövegdoboz)! Az elemek kezeléséhez deklarálhatunk egy *Alma* nevű, *KeyValuePair*(*Of Integer, List*(*Of Integer*)) típusú változót.



Az iterátor azonosítójának ütközése a névtér nevével (Visual Basic)



A névtér nevének módosítása az ütközés elkerüléséhez (Visual Basic)

A következő lépésben csökkentjük azon fajták mennyiségét, amelyekből több van, mint amennyit a kereskedő venni akar:

```
CIKLUS MINDEN Fajta-ra a Sok-ban
    Termelő(Fajta) = Termelő(Fajta) – Kereskedő(Fajta) ' a mennyiséget módosítja!
    ' ennyivel kevesebb marad a termelőnek
CIKLUS VÉGE
```

Ha egy fajtából legfeljebb annyi van, amennyi a kereskedőnek kellene, akkor a termelő átadja az összes almáját. Nem marad neki semennyi, ezért töröljük ezt a fajtát:

```
CIKLUS MINDEN Fajta-ra a Kevés-ben
    Kereskedő(Fajta) = Termelő(Fajta) ' a mennyiséget másolja át
    Termelő.Kivesz(Fajta)
CIKLUS VÉGE
```

Az 1. módszernél tehát kigyűjtjük az összes elemet, majd a feldolgozásnál vizsgáljuk meg az egyes elemek tulajdonságait. A 2. módszernél pedig csak a megfelelő tulajdonságú elemeket gyűjtjük ki, és ezek mindegyikére végrehajtjuk a szükséges műveletet.

A feladatból hátra van még a kiírás. A teljes megoldást az *almak* program tartalmazza.

Vegyük észre, hogy a *Termelő* és a *Kereskedő* szótárakat multihalmaznak tekinthetjük. A fajták alkotják a halmazok elemeit, multiplicitásuk pedig megadja az egyes fajták mennyiségét.

Szólánc

Nemes Tihamér OITV 2004. 3. forduló 1. korcsoport 2. feladat

A szólánc játékot úgy játsszák, hogy valaki mond egy szót, a következőnek a szó utolsó betűjével kezdődő szót kell mondania, az őt követőnek a második szó utolsó betűjével kezdődőt és így tovább. A szólánc szavait azonban valaki összekeverte, de tudjuk, hogy ezek a szavak csak egyféle sorrendben rakhatók össze, ezért a sorrendjük biztosan helyreállítható.

Készítsünk *szolanc* néven programot, amely beolvassa a szólánc szavainak a számát ($1 \le N \le 100$), majd az N darab összekevert szót (mindegyik legfeljebb 20 karakteres), s ezek alapján kiírja a szóláncot a helyes sorrendben!

Példa:

Bemenet

```
N=4
Szavak: agár körte málna retek
```

Kimenet

```
Szólánc: málna agár retek körte
```

> Megjegyzés: a program értékeléséhez használt tesztadatok[10] és a versenybizottság által kiadott megoldás[11] szerint is egy megadott betűvel csak egyetlen szó kezdődhet, illetve végződhet. Ez azonban nem szükséges ahhoz, hogy egyértelmű legyen a sorrend. Például az *AB, BA, AC* szavakból csak egyféleképpen alkotható szólánc, ugyanakkor két szó kezdődik *A* betűvel. Érdekes módon, a feladat szövege szerint a szavak száma akár 100 is lehetne.

Megoldás

A fenti megjegyzés alapján a szavak kezdőbetűi tekinthetők kulcsnak. A szavakat ezért beletesszük egy olyan szótárba, melyben a kulcs a kezdőbetű, az érték pedig maga a szó lesz:

```
VÁLTOZÓ Szavak MINT Szótár(Kulcstípus: Karakter, Értéktípus: Sztring)
CIKLUS I=1-től N-ig
    Be: Szó
    Szavak.Add(Szó.Elsőbetű, Szó)
CIKLUS VÉGE
```

A szólánc kezdőszava olyan szó lehet, melynek első betűje nem szerepel utolsó betűként. Ha nincs ilyen, akkor a szólánc körbemehet, tehát bármelyik szóval kezdhetjük.

Az első szó kiválasztásához a kezdőbetűket és az utolsó betűket kigyűjtjük egy-egy halmazba, majd meghatározzuk a két halmaz különbségét:

```
VÁLTOZÓ Kezdőbetűk, Utolsóbetűk MINT Új Halmaz(Elemtípus: Karakter)
CIKLUS MINDEN Szó-ra a Szavak-ban
    Kezdőbetűk.Add(Szó.Elsőbetű)
    Utolsóbetűk.Add(Szó.Utolsóbetű)
CIKLUS VÉGE
Kezdőbetűk = Kezdőbetűk – Utolsóbetűk ' halmazok különbsége
HA Kezdőbetűk.Elemszám > 0 AKKOR Szólánc(1) = Szavak(Kezdőbetűk.Első)
EGYÉBKÉNT Szólánc(1) = Szavak.Első.Érték ' bármelyiket választhatnánk
ELÁGAZÁS VÉGE
```

A szólánc ezek után egy ciklussal elkészíthető. A következő szó mindig az lesz, amelynek kulcsa (első betűje) az előző szó utolsó betűjével egyezik meg:

```
Ciklus I=2-től N-ig
    Szólánc(I) = Szavak(Szólánc(I–1).Utolsóbetű)
Ciklus vége
```

A teljes megoldást a *szolanc* program tartalmazza.

---

[10] http://nemes.inf.elte.hu/2004/Nt04-3m1.doc

[11] Horváth Gyula–Zsakó László szerk.: Programozási versenyfeladatok tára III. (Neumann János Számítógép-tudományi Társaság, 2006, 235. old.)

Titkosírás

Nemes Tihamér OITV 2012. 3. forduló 1. korcsoport 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 szereplő betűpárt tesszük. Feltehetjük, hogy csak az angol ábécé kisbetűit használjuk.

Készítsünk *titkos* néven programot, amely két funkcióra képes:

a) beolvas egy szót és kiírja titkosítva;

b) beolvas egy titkosított szót és kiírja a visszafejtését!

Példa:

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

Megoldás

A feladatnak látszólag semmi köze a szótárobjektumhoz. Jól rávilágít azonban arra a programtervezési stílusra, amelynél összetett keresések helyett inkább megpróbálunk egy adatszerkezetet hozzáigazítani a problémához. A szótárobjektum szinte minden olyan esetben segít, melynek során gyakran kell keresnünk egy kulcsnak tekinthető (azaz nem ismétlődő) adatot.

A titkosítás táblázatát sztringtömbben tároljuk, melynek elemei a táblázat sorait tartalmazzák. Ezzel nemcsak a begépelést gyorsítjuk, de a keresést is egyszerűsítjük. A sorok és oszlopok indexelését a *Sor*, illetve az *Oszlop* sztring karakterei alapján végezzük.

```
VÁLTOZÓ Sor MINT Sztring = "abc"
VÁLTOZÓ Oszlop MINT Sztring = "abcdefghi"
VÁLTOZÓ Betű() MINT Sztring = {"ioqhbfylw", "nragsktez", "dupxcjvm "}
```

Az egyes betűk kódját egy betű–kód párokat tartalmazó szótárobjektumban tartjuk nyilván:

```
VÁLTOZÓ Kód MINT Szótár(Kulcstípus: Karakter, Értéktípus: Sztring)
```

A szótárobjektumot két, egymásba ágyazott ciklussal töltjük fel (a sztringek karaktereinek indexelése 0-val kezdődik):

```
CIKLUS I=0-tól Sor.Hossz–1-ig
    CIKLUS J=0-tól Oszlop.Hossz–1-ig
        Kód.Add(Betű(I)(J), Sor(I) & Oszlop(J)) ' összefűzzük a két karaktert
    CIKLUS VÉGE
CIKLUS VÉGE
```

Vegyük észre, hogy a *Betű*(*I*) egy sztring, ezért így hivatkozunk a *J*-edik karakterére: *Betű*(*I*)(*J*)!

A titkosítás ezek után már nagyon könnyen elvégezhető. Egy ciklussal végigmegyünk a beolvasott *Szó* karakterein, és mindegyikhez, mint kulcshoz, kiírjuk a *Kód* szótár hozzá tartozó *Érték*-ét:

```
Be: Szó
CIKLUS MINDEN Karakter-re a Szó-ban
    Ki: Kód(Karakter)
CIKLUS VÉGE
```

A visszafejtéshez nincs szükségünk szótárra, a kódolt szó *I*-edik és *I*+1-edik karaktere alapján meghatározzuk a *Betű* tömb sor-, illetve oszlopindexét. A teljes megoldást lásd a *titkos* programban! A program egyszerre végzi el a beolvasott karaktersorozat kódolását és visszafejtését (ha lehet).

Kémek

Nemes Tihamér OITV 2009. 3. forduló 1. korcsoport 3. feladat

Egy *N* tagú kémszervezetben mindenkinek több közvetlen beosztottja lehet, s mindenkinek ismerjük a közvetlen felettesét. A tagokat 1-től *N*-ig sorszámozzuk. Egyetlen tagnak nincs felettese, ő a főnök.

Írj programot *kemek* néven, amely beolvassa a tagok számát ($1 \le N \le 100$) és a tagpárokat (kinek, ki a közvetlen felettese), majd

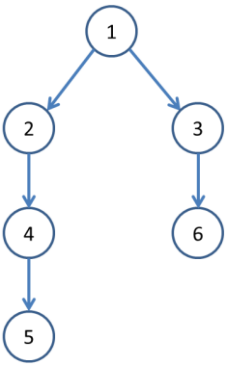
a) megadja a főnököt;

b) megad egy tagot, akinek a legtöbb közvetlen beosztottja van;

c) megad egy tagot, aki a „legmesszebb” van a főnöktől!

Példa:[12]

Bemenet	*Kimenet*
`6` (létszám)	`1` (főnök)
`2 1` (beosztott, főnök)	`1` (legtöbb közvetlen beosztott)
`3 1` (stb.)	`5` (legmesszebb a főnöktől)
`6 3`	
`5 4`	
`4 2`	



Megoldás

A főnök–beosztott kapcsolatot ábrázolhatjuk gráffal (irányított gráf, fagráf). A gráf csúcsai felelnek meg a tagoknak, az irányított él a felettestől a beosztott felé mutat.

A feladatot úgy oldjuk meg, hogy alkalmazható legyen nevekkel azonosított tagok esetére is. Ezért a gráf tárolására statikus tömb helyett szótárt használunk. Az elemek kulcsa a tagok azonosítója, az elemek értéke pedig a beosztottak halmaza (szomszédsági „lista”):

```
VÁLTOZÓ Beosztottak MINT Szótár(Kulcstípus: Egész,
                                Értéktípus: Halmaz(Elemtípus: Egész))
```

[12] A formátumot némileg egyszerűsítettük.

Egyszerűbb lesz a megoldás, ha minden személynek feljegyezzük a főnökét is. Ebben a szótárban az elemek kulcsa szintén a tagok azonosítója, az elemek értéke pedig a főnökök sorszáma.

```
VÁLTOZÓ Felettese MINT Szótár(Kulcstípus: Egész, Értéktípus: Egész)
```

A beolvasásnál ügyeljünk arra, hogy ha egy felettes még nem szerepel a *Beosztottak* szótár kulcsai között, akkor adjuk hozzá a szótárhoz egy üres halmazzal!

```
VÁLTOZÓ N MINT Egész
VÁLTOZÓ Felettes, Tag MINT Egész
Be: N
CIKLUS I=1-től N–1-ig
    Be: Tag, Felettes
    HA NEM Beosztottak.Kulcsként Tartalmazza(Felettes) AKKOR
        Beosztottak.Add(Felettes, Új Halmaz(Elemtípus: Egész))
    ELÁGAZÁS VÉGE
    Beosztottak(Felettes).Add(Tag)
    Felettese.Add(Tag, Felettes)
CIKLUS VÉGE
```

Gondoljuk meg, hogy *N* helyett miért *N*–1 a ciklusváltozó felső határa!

A kémszervezet főnöke az a személy, akinek nincs felettese, tehát nem szerepel kulcsként a *Felettesek* szótárban:

```
VÁLTOZÓ I, Főnök MINT Egész
I = 1
CIKLUS AMÍG I ≤ N ÉS# Felettese.Kulcsként Tartalmazza(I)
    I = I + 1
CIKLUS VÉGE
Főnök = I
Ki: Főnök
```

A lineáris keresés ciklusában nem szükséges az $I \leq N$ vizsgálat. Gondoljuk meg, hogy miért!

A legtöbb közvetlen beosztottal rendelkező tag vizsgálata maximumkiválasztást igényel. Egy kémszervezet azonban akár egyetlen tagból is állhat. Ezért először a *Főnök*-öt tekintjük a keresett személynek.

```
VÁLTOZÓ Max, Melyik MINT Egész
Melyik = Főnök ' lehet, hogy nincs több tag
Max = 0
CIKLUS MNDEN Elem-re a Beosztottak-ban
    HA Elem.Érték.Elemszám > Max
        Max = Elem.Érték.Elemszám
        Melyik = Elem.Kulcs
    ELGAZÁS VÉGE
CIKLUS VÉGE
Ki: Melyik
```

Közben kihasználtuk, hogy a *Beosztottak* szótár egy elemének kulcsa a tag azonosítója, értéke pedig a beosztottjainak a halmaza.

Végül a főnöktől legmesszebb lévő tag kereséséhez minden tagra meghatározzuk a főnöktől mért távolságot. Ehhez elindulunk a tagtól, és a feletteseken át lépegetve megszámoljuk, hány lépésben érünk el a főnökhöz.

```
VÁLTOZÓ Táv(N) MINT Egész
CIKLUS I=1-től N-ig
    Táv(I) = 0
    Felettes = I
    CIKLUS AMÍG Felettes ≠ Főnök
        Táv(I) = Táv(I) + 1
        Felettes = Felettese(Felettes)
    CIKLUS VÉGE
CIKLUS VÉGE
```

Ezek után maximumkiválasztással kapjuk meg a c) kérdésre a választ.

```
Max = –∞
CIKLUS I=1-től N-ig
    HA Táv(I) > Max AKKOR
        Max = Táv(I)
        Melyik = I
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
Ki: Melyik
```

A teljes megoldást a *kemek* program tartalmazza.

Megoldásunkban néhány helyen kihasználtuk, hogy egymást követő egész számokkal azonosítjuk a tagokat. Módosítsuk a megoldást úgy, hogy sztringazonosítók esetén is működjön!

Ez a példa nagyon jó lehetőséget biztosít a szótárobjektumok alkalmazásának gyakorlására. Gondoljuk át alaposan az algoritmus minden lépését!

Oldjuk meg a feladatot szótárobjektumok helyett statikus tömbökkel is!

Ember

Nemes Tihamér OITV 2013. 3. forduló 1. korcsoport 1. feladat

Ismerjük N emberről, hogy ki szülője kinek ($1 \le N \le 100$). Az embereket az 1 és N közötti sorszámukkal azonosítjuk.

Készíts *ember* néven programot, amely megadja, hogy

a) kinek van a legtöbb gyereke;

b) kinek van a legtöbb unokája;

c) hány embernek nincs unokája!

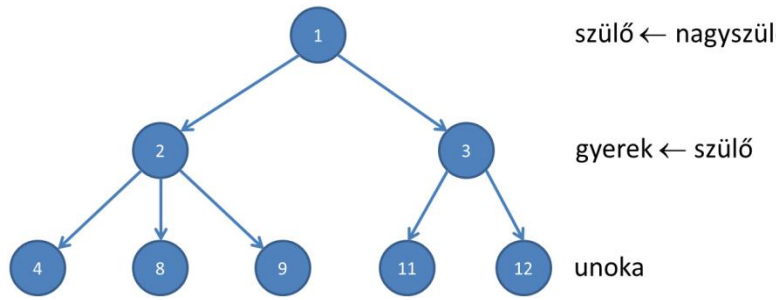
Példa:

Bemenet		*Kimenet*	*Magyarázat*
`N=12`			
`szülő:`	`gyerek:`	`A: 4`	az 5, 6, 7, 10 sorszámú személy
1	2	`B: 1`	a 4, 8, 9, 11, 12 sorszámú személy
1	3	`C: 10`	csak 1-nek és 2-nek van
2	4		
2	8		
2	9		
4	5		
4	6		
4	7		
4	10		
3	11		
3	12		

Megoldás

Az adatokat szemléltethetjük körökből és vonalakból álló ábrával, azaz gráffal (irányított gráf, fagráf). A gráf csúcsai jelképezik az egyes embereket, az irányított élek pedig a szülő→gyerek kapcsolatot.

A szülő gyerekének a gyereke – a szülőnek, mint nagyszülőnek az unokája



Tároljuk a gráfot szótárobjektumban! Az elemek kulcsa az emberek azonosítója (ami lehetne sztring is), az elemek értéke pedig az adott kulcshoz tartozó gyerekek halmaza:

```
VÁLTOZÓ Szülők MINT Szótár(Kulcstípus: Egész,
                           Értéktípus: Halmaz(Elemtípus: Egész))
```

Egyszerűbb lesz a feldolgozás, ha minden emberre összegyűjtjük az unokáik halmazát:

```
VÁLTOZÓ Nagyszülők MINT Szótár(Kulcstípus: Egész,
                               Értéktípus: Halmaz(Elemtípus: Egész))
```

Először létrehozzuk a szótárakhoz tartozó halmazokat:

```
CIKLUS I=1-től N-ig
    Szülők(I) = Új Halmaz(Elemtípus: Egész)
    Nagyszülők(I) = Új Halmaz(Elemtípus: Egész)
CIKLUS VÉGE
```

A gyerekek halmazait az adatok beolvasásakor töltjük fel:

```
Be: Szülő, Gyerek
Szülők(Szülő).Add(Gyerek)
```

Az unokák halmazait a következőképpen határozzuk meg. Minden egyes szülőre a gyerekének a gyerekeit hozzáadjuk a szülő, mint nagyszülő unokáinak a halmazához. Áttekinthetőbb kódot kapunk, ha bevezetjük a *Nagyszülő* változót, illetve a *Gyerekek* és *Unokák* halmazokat.

```
VÁLTOZÓ Nagyszülő MINT Egész
VÁLTOZÓ Gyerekek, Unokák MINT HALMAZ(Elemtípus: Egész)
CIKLUS MINDEN Elem-re a Szülők-ben
    Szülő = Elem.Kulcs
    Gyerekek = Elem.Érték
    Nagyszülő = Szülő ' a szülőből lesz a nagyszülő (ha a gyereknek vannak gyerekei)
    CIKLUS MINDEN Gyerek-re a Gyerekek-ben
        HA Szülők.Kulcsként Tartalmazza(Gyerek) AKKOR
            ' a gyerek szerepel a szülők között
            Unokák = Szülők(Gyerek) ' a gyerek gyerekei
            Nagyszülők(Nagyszülő) = Nagyszülők(Nagyszülő) ∪ Unokák
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
CIKLUS VÉGE
```

Ezek után már csak azt kell megállapítanunk, hogy a *Szülők*, illetve *Nagyszülők* szótárban melyik elemhez tartozó halmaznak maximális az elemszáma. Amelyik nagyszülőhöz tartozó halmaz üres, annak nincs unokája.

A teljes megoldást az *ember* program tartalmazza. Válaszoljunk a kérdésekre statikus tömbök alkalmazásával is!

További feladatok az asszociatív tömbökre

OITV 1996-2f1	Hét törpe
OITV 1997-3f1	Állatkert
OITV 1998-2f1	Állatkertek
OITV 1998-3f1	Karácsonyfa
OITV 2004-2f1	Legolcsóbbak
OITV 2007-1f2	Multihalmaz
OITV 2007-3f1	Nyelvek
OITV 2008-1f2	Mik a hibák?
OKTV 1996-1f3	Multihalmazok
OKTV 2007-1f3	Táblázat
OKTV 2007-1f3	Multihalmaz
OKTV 2008-1f3	Mik a hibák?
OKTV 2008-2f3	Város

4. Dinamikus tömbök

Kapcsolódó lecke:
Programozási ismeretek haladóknak (Műszaki Kiadó, 2012)
19. A dinamikus tömb

A dinamikus tömbök használata felgyorsítja a programírást, és lerövidíti a kódot. Általában hatékonyabb programot eredményez, mintha statikus tömböt alkalmaznánk, főleg az elemek törlésénél vagy új elemek beillesztésénél. Gyakran nem kell foglalkoznunk az aktuális elem indexelésével sem. Az új elemet a dinamikus tömb végéhez fűzzük hozzá, a feldolgozást az első elemmel kezdjük, amit majd – lehetőség szerint – törlünk. Így mindig az első elem kerül sorra. (A törlés hatékonyságára a sor adatszerkezet tárgyalásánál, illetve a befejező részben térünk vissza.)

A továbbiakban a .NET nyelvhasználatának megfelelően a dinamikus tömböt listának nevezzük. Ne keverjük össze a láncolt lista adatszerkezettel!

> C++ nyelven dinamikus tömbként javasoljuk a *vector* típus, illetve a gyorsabban bővíthető, kétvégű sor, a *deque* típus használatát.
> A *forward_list* láncolt listát, a *list* pedig kétirányú láncolt listát valósít meg. Ezek metódusait is bemutatjuk a példaprogramok között.

> Visual Basic-ben két, érték típusú elemeket tartalmazó, felsorolható kollekció egyenlőségének vizsgálatára használtjuk a *Kollekció1*.*SequenceEquals*(*Kollekció2*) metódushívást. A visszatérési érték igaz, ha a kollekciók azonos indexű elemei páronként megegyeznek egymással.

Egyes programozási nyelvek ismerik a rendezett lista adattípust. A .NET-ben a rendezett lista kulcs–érték párokból áll. A kulcsok egyedi értékek, különböző elemek nem rendelkezhetnek egyforma kulccsal! Objektumorientált módszerek alkalmazása nélkül kulcsként általában csak egyszerű típust adhatunk meg. Az elemek értéke (*Value* tulajdonsága) azonban már tetszőleges, akár összetett típusú is lehet. A rendezett lista alkalmazását a *Tipp* és a *Csapat* feladatban mutatjuk be.

> A .NET-ben a rendezett lista lényegében rendezett tömbnek tekinthető, a rendezett szótár ugyanakkor a bináris keresőfát valósítja meg.
> Hasonlóságok a rendezett lista és a rendezett szótár között:
> - A kulcsok egyedi értékek!
> - Az elemeket a kulccsal indexelhetjük (de az elem lekérdezésénél az *ElementAt* függvény paramétereként 0-tól *Count*–1-ig terjedő indexet is alkalmazhatunk).
> - Hasonló tulajdonságokkal, hasonló metódusokkal rendelkeznek.
> - A keresés mindkettőben $O(\log n)$ futásidejű.
>
> Eltérések a rendezett lista és a rendezett szótár között:
> - A lista kevesebb memóriát foglal el, mint a szótár.
> - Rendezetten érkező adatok esetén a lista gyorsabban épül fel [$O(1)$].
> - Rendezetlen adatok beillesztése és törlése gyorsabb a szótár esetén [$O(\log n)$, a listára jellemző $O(n)$-nel szemben!].

Szétválogatás

A dinamikus tömbök alkalmazásának technikáját egy alapvető algoritmuson keresztül mutatjuk be. Válogassuk szét a *Diákok.txt* fájlban[13] lévő adatokat külön a fiúk és külön a lányok listájába!

A tároláshoz struktúrát használunk[14] (lásd: *Programozási ismeretek kezdőknek*, 193. oldal):

```
STRUKTÚRA TDiák
    VÁLTOZÓ Név MINT Sztring, Magasság MINT Egész, Tömeg MINT Valós
    VÁLTOZÓ Fiú MINT Logikai
STURKTÚRA VÉGE
```

A lista nevét többes számban, egy elem azonosítóját pedig egyes számban deklaráljuk. Így jól követhető, könnyen értelmezhető kódot kapunk, amely a begépelést és a hibakeresést is segíti:

```
VÁLTOZÓ Diákok, Fiúk, Lányok MINT Lista(Elemtípus: TDiák)
VÁLTOZÓ Diák MINT TDiák
```

A szétválogatás szinte automatikusan történik, minden különösebb adminisztráció (az indexek módosítása) nélkül:

```
CIKLUS AMÍG Diákok.Elemszám > 0
    HA Diákok.Első.Fiú AKKOR Fiúk.Add(Diákok.Első)
    EGYÉBKÉNT Lányok.Add(Diákok.Első)
    ELÁGAZÁS VÉGE
    Diákok.Kivesz(0)
CIKLUS VÉGE
```

A teljes kódot a *szetvalogat* program tartalmazza.

> A C++ nyelvű megoldások szemléltetik az ASCII (ANSI) és UTF-8 kódolású forrásfájlok, illetve adatfájlok beolvasását, kiírását. Az UTF-8 aláírással (BOM-mal) ellátott változatát alkalmaztuk. Ne felejtsük el a program forráskódját tartalmazó állomány megnyitása előtt beállítani a fejlesztőrendszerben a kódolást (a Code::Blocks-ban például *Settings*/*Editor*, *General settings*/*Other settings*, *Encoding*: Windows-1250 vagy UTF-8)!

Általában egy új elem hozzáadása a lista végéhez gyors, az első elem törlése viszont meglehetősen lassú művelet. Ha lényeges szempont a végrehajtási idő csökkentése, vagy szükség van az eredeti listára, akkor hagyjuk el az elemek törlését, és ciklussal dolgozzuk fel a listát! Ilyenkor – lehetőség szerint – használjunk iterátoros ciklust! Így szintén megspóroljuk az index alkalmazását, egyszerűsítjük a programot.

A bemenő adatok átmeneti tárolására lista helyett hatékonyabban alkalmazhatunk sor adatszerkezetet (lásd a *8. A verem és a sor* leckét). A hatékonyság kérdéseire és az egyes műveletek végrehajtási idejére a befejező részben térünk vissza.

---

[13] A fájlt a *Programozási ismeretek* tankönyv forrásfájljai között találjuk a könyv webhelyén: www.zmgzeg.sulinet.hu/programozas

[14] A struktúra ismeretének hiányában szétválogathatjuk például csak a neveket.

> C++-ban a *vector*, a *queu* és a *list* konténerek *front()* és *back()* metódusai az adatstruktúra első, illetve utolsó elemére hivatkozó referenciával térnek vissza, így ezek az elemek meg is változtathatók.

> Visual Basic-ben a *First* és a *Last* metódusok függvények, így csak az elemek lekérdezésére alkalmasak, módosításukra nem használhatók. Ha azonban a lista hivatkozás típusú elemeket tartalmaz (például listákat), akkor már módosíthatjuk is a listaelemeket: *Lista*.*First*.*Add*(*új elem*) stb.

Javít

Nemes Tihamér OITV 2005. 2. forduló 2. korcsoport 2. feladat

A helyesírás szabályai szerint szövegben írásjelek elé nem teszünk szóközt (vessző, pont, kérdőjel, felkiáltójel, pontosvessző, kettőspont), utána viszont teszünk. Ugyancsak nem teszünk szóközt a nyitó zárójelek után és a csukó zárójelek elé (háromféle van: kerek, szögletes és kapcsos). A nyitó zárójel elé kell tenni szóközt és a csukó után is kell, hacsak nem írásjel követi. Ha egyszerre két szabályt kellene alkalmazni (pl. írásjel után kell, csukó zárójel előtt nem kell szóköz), akkor a tiltó szabály az erősebb.

Készíts programot *javit* néven, amely egy szöveget átalakít a helyesírás szabályai szerint!

A *javit.be* szöveges állományban egyetlen sor van (legfeljebb 10000 karakter), amely a hibásan beírt szöveget tartalmazza.

A *javit.ki* szöveges állományba egyetlen sort kell írni, a javított szöveget.

Példa:

```
javit.be
Ez egy hibásan ,rosszul( rossz zárójelezéssel)írt szöveg   .

javit.ki
Ez egy hibásan, rosszul (rossz zárójelezéssel) írt szöveg.
```

Megoldás

A beillesztések és törlések miatt különösen hatékonyan oldhatjuk meg ezt a feladatot lista adatszerkezet alkalmazásával.

A beolvasott szöveg karaktereit listában tároljuk:

```
VÁLTOZÓ Szöveg MINT Lista(Elemtípus: Karakter)
VÁLTOZÓ Sor MINT Sztring
Be: Sor
CIKLUS MINDEN Karakter-re a Sor-ban
   Szöveg.Add(Karakter)
CIKLUS VÉGE
```

Az egyes írásjeleket, zárójeleket konstansok segítségével azonosítjuk:

```
KONSTANS Írásjel MINT Sztring = ",.?!;:"
KONSTANS Nyitó MINT Sztring = "([{"
KONSTANS Csukó MINT Sztring = ")]}"
```

Sorra vesszük a szöveg karaktereit (a lista elemeit). Egyszerűbb lesz a ciklus felírása, ha minden karaktert a rákövetkezővel hasonlítunk össze. A szöveg végéhez ütközőt illesztünk, így nem hivatkozunk már nem létező elemre.

```
Szöveg.Add("x")
```

A törlések/beszúrások miatt nem használhatunk számlálós ciklust (változik közben az elemszám). A gyengébb szabályokkal kezdjük a vizsgálatot.

```
VÁLTOZÓ I MINT Egész
I = 0 ' a listaelemek indexelése 0-val kezdődik
CIKLUS AMÍG I < Szöveg.Elemszám  – 1
   HA Szöveg(I) ≠ " " ÉS Nyitó.Tartalmazza(Szöveg(I+1)) AKKOR
      ' a nyitó zárójel előtt nincs szóköz, beillesztjük a szóközt
      Szöveg.Beilleszt(I+1, " ")
   EGYÉBKÉNT HA Csukó.Tartalmazza(Szöveg(I)) ÉS Szöveg(I+1) ≠ " " AKKOR
      ' a csukó zárójel után nincs szóköz, beillesztjük a szóközt
      Szöveg.Beilleszt(I+1, " ")
   ELÁGAZÁS VÉGE
```

Az erősebb szabályokat újabb feltételes elágazással ellenőrizzük (nem pedig *EGYÉBKÉNT HA* ággal), mert felülírhatják a gyengébb szabályok módosításait.

```
   HA Szöveg(I) == " " ÉS Írásjel.Tartalmazza(Szöveg(I+1)) AKKOR
      ' a szóközt írásjel követi, töröljük a szóközt
      Szöveg.Kivesz(I indexű elem)
   EGYÉBKÉNT HA …
      …
   EGYÉBKÉNT
      I = I + 1
   ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

Nem adtuk meg minden szabályhoz az ellenőrzés feltételét és a szükséges módosítást, ezeket az olvasóra bízzuk. ☺ Közben figyeljünk az aktuális elem indexére (törlés, beillesztés)! Az *I* növelése miért az *EGYÉBKÉNT* ágban szerepel?

A ciklus után töröljük a szöveg végére helyezett ütközőt. Töröljük a szóközt is, ami akkor kerül oda, ha a szöveg írásjellel végződik.

```
Szöveg.Kivesz(utolsó elem)
HA Szöveg.Utolsó == " " AKKOR
   Szöveg.Kivesz(utolsó elem)
ELÁGAZÁS VÉGE
CIKLUS MINDEN Karakter-re a Szöveg-ben
   Ki: Karakter
CIKLUS VÉGE
```

A teljes megoldást a *javit* program tartalmazza. Figyelmesen tanulmányozzuk a ciklusban a feltételes elágazásokat és értékadó utasításokat!

Érdemes megírni a programot úgy is, hogy lista helyett a használt programozási nyelv sztring típusú változójában tároljuk és módosítjuk a szöveget.

> Visual Basic-ben az ékezetes karakterek helyes kezeléséhez a fájl megnyitásánál adjuk meg a kódolás módját, például:
>
> ```
> Dim Fájlbe As New IO.StreamReader(elérési út,
>                                   Text.Encoding.Default)
> Dim Fájlki As New IO.StreamWriter(elérési út, False,
>                                   Text.Encoding.Default)
> ```

Kiszámolós

Nemes Tihamér OITV 1999. 3. forduló 1. korcsoport 1. feladat

Egy kiszámolós játékban *N* gyerek körbe áll a lenti ábrának megfelelően.

A kiszámolás az elsőnél kezdődik, majd minden *K*-adikat kell kihagyni úgy, hogy végül csak egyetlen egy gyerek maradjon. Először tehát a *K*. marad ki, majd a 2*K*., ... Ha az utolsóhoz értünk, a kör tovább folytatódik.

Készíts programot *kiszamol* néven, amely beolvassa a gyerekek számát ($1<N\leq100$) és hogy minden hányadikat kell kihagyni ($K\geq1$), majd kiírja képernyőre a kiszámolós játékban kiesőket, majd pedig a végén megmaradt gyerek sorszámát!

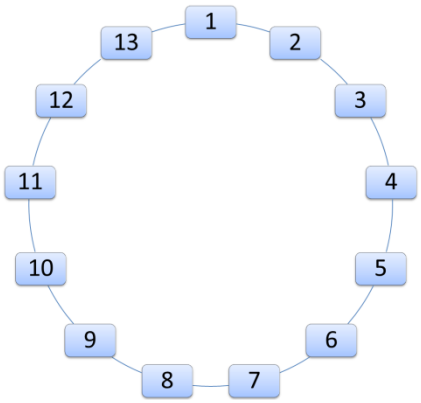
Példa:

```
N=13
K=6
A kimaradók sorban: 6, 12, 5, 13, 8, 3, 1, 11, 2, 7, 4, 10
Végül megmaradt: 9
```

Megoldás

A kiesések miatt ezt a feladatot is hatékonyan oldhatjuk meg lista adatszerkezet alkalmazásával.

A sorszámokat listában tároljuk. Az *N* és a *K* értékének beolvasása után feltöltjük a listát az 1-től *N*-ig terjedő egész számokkal.




```
VÁLTOZÓ Kör MINT Lista(Elemtípus: Egész)
VÁLTOZÓ N, K MINT Egész
Be: N, K
CIKLUS I=1-től N-ig
    Kör.Add(I)
CIKLUS VÉGE
```

A következő gyerek sorszámát összeadással és maradékos osztással határozhatjuk meg. Közben kihasználjuk, hogy a listaelemek indexelése 0-val kezdődik. Az aktuális indexhez hozzáadunk *K*–1-et (az aktuális elem az első!), majd az összeget maradékosan elosztjuk az aktuális elemszámmal. Az osztás maradéka lesz a következő kieső elem indexe:

Következő = (*Előző* + *K* – *1*) *Mod Kör.Elemszám*

A törlés miatt a kiszámolás ugyanazzal az indexszel folytatódik, mint amelyikkel a törölt elem rendelkezett (ennek a helyére került az újabb elem). Ha éppen az utolsó elemet töröltük, akkor pedig a lista első (0 indexű) elemével kezdődik a kiszámolás.

A módszer működését a következő példán ellenőrizhetjük. Legyen $N = 5$, $K = 4$. Először egy ábra alapján határozzuk meg az egymás után kieső diákok sorszámát, majd számoljuk ki az indexeket (a kezdő diák oszlopát sárgával, a kieső diák oszlopát pedig szürkével jelöltük):

1. lépés

Index:	0	1	2	3	4
Diák:	1	2	3	4	5

Kezdőindex: 0
Kieső diák indexe: (0 + 4 – 1) Mod 5 = 3

2. lépés

Index:	0	1	2	3
Diák:	1	2	3	5

Kezdőindex: 3
Kieső diák indexe: (3 + 4 – 1) Mod 4 = 2

3. lépés

Index:	0	1	2
Diák:	1	2	5

Kezdőindex: 2
Kieső diák indexe: (2 + 4 – 1) Mod 3 = 2

4. lépés

Index:	0	1
Diák:	1	2

Kezdőindex: 0
Kieső diák indexe: (0 + 4 – 1) Mod 2 = 1

Megmaradó elem:

Index:	0
Diák:	1

Ha megértettük a számítás menetét, akkor az algoritmust már nagyon könnyen felírhatjuk.

```
VÁLTOZÓ Következő MINT Egész
Következő = (K – 1) Mod N
CIKLUS AMÍG Kör.Elemszám > 1
    Ki: Kör(Következő)
    Kör.Kivesz(Következő indexű elem)
    Következő = (Következő + K – 1) Mod Kör.Elemszám
CIKLUS VÉGE
Ki: Kör.Első
```

A játék végén a *Kör* lista egyetlen (0 indexű) eleme marad meg.
A teljes megoldást a *kiszamol* program mutatja be.

Oldjuk meg a feladatot statikus tömb alkalmazásával is! Hogyan kezelhetjük a kieső diákokat?

Általánosítsuk a fenti gondolatmenetet! Vegyük sorra egy *N* elemű sorozat elemeiből minden *K*-adikat (tehát először a *K*-adik elemet) úgy, hogy az utolsó után ismét az első következik (cirkuláris bejárás). Ha az elemek sorszámozása (indexelése) 0-val kezdődik, akkor az elsőként sorra következő elem indexe:

Első = *K* – *1*

A további elemek indexét a következő képlettel határozzuk meg:

Következő = (*Előző* + *K*) *Mod N*

Néhány számpélda segítségével próbáljuk ki a képlet helyességét! Miben tér el ez a képlet a feladat megoldásában alkalmazott képlettől? Mi az eltérés oka? Hogyan kellene módosítanunk a képletet, ha 0 helyett 1-gyel indulna az elemek indexelése?

Kártya

Nemes Tihamér OITV 2003. 3. forduló 1. korcsoport 3. feladat

Francia kártyával sokféle játékot játszhatunk. Az egyik játékban a játékosok az osztásnál kapott 14 lapból letehetnek kártyahármasokat: vagy egymást követő azonos színű lapokat (pl. pikk 9, 10, bubi), vagy pedig azonos értékű lapokat (pl. pikk 3, kőr 3, treff 3).

Készíts programot *kartya* néven, amely egyesével beolvassa egy játékos treff, pikk, kőr, valamint káró színű kártyáit növekvő sorrendben, majd megadja, hogy milyen kártyahármasokat tehet le!

A kártyák színe: *treff*, *pikk*, *kőr*, *káró*; értéke pedig *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, *10*, *bubi*, *dáma*, *király*, *ász* lehet.

Példa:

Bemenet	*Kimenet*
`treff: 2 6 7 8`	`treff 6 7 8`
`pikk: 8 9 10`	`pikk 8 9 10`
`kőr: 9 10 dáma`	`pikk 10 kőr 10 káró 10`
`káró: 2 5 10 ász`	

Megoldás

A sorozatok kereséséhez az egymást követő értékeket *Sztring* típusú tömbben tároljuk:

```
VÁLTOZÓ Értékek(0…12) MINT Sztring =
                    {"2", "3", "4", "5", "6", "7", "8", "9", "10", "bubi", "dáma", "király", "ász"}
```

A színeket nem adjuk meg előre, mert nem biztos, hogy mindegyik szerepelni fog a kártyák között, és a sorrendjük is változhat a beolvasásnál.

```
VÁLTOZÓ Színek MINT Lista(Elemtípus: Sztring)
```

Minden egyes szín-lista az adott színű lapok értékének listáját tartalmazza. Ezeket a listákat szintén listában tároljuk:

```
VÁLTOZÓ Lapok MINT Lista(Elemtípus: Lista(Elemtípus: Sztring))
```

A beolvasás algoritmusa:

```
VÁLTOZÓ Szín, Érték MINT Sztring
CIKLUS AMÍG van adat a bemeneten
    Be: Szín
    Színek.Add(Szín)
    Lapok.Add(Új Lista(Elemtípus: Sztring)) ' létre kell hozni a listát!
    CIKLUS az adott színű lapokra
        Be: Érték
        Lapok.Utolsó.Add(Érték)
    CIKLUS VÉGE
CIKLUS VÉGE
```

> Visual Basic-ben az *IO.TextReader* osztály *In* osztálytulajdonsága képviseli a standard inputot. Ennek *Peek* metódusával vizsgálhatjuk meg, hogy van-e még adat a bemeneten, például: `Do While In.Peek <> -1 …` Az *In* azonban a nyelv egyik kulcsszava. Azonosítóként használva szögletes zárójelbe kell tennünk: `[In].Peek`. A zárójelezést a kódszerkesztő automatikusan elvégzi.

Bár a feladat nem kérte, mi megszámoljuk a sorozatokat. Előfordulhat ugyanis, hogy nem találunk egyetlen sorozatot sem. Ekkor egy erre utaló üzenetet írunk ki megoldásként.

```
VÁLTOZÓ N MINT Egész ' a talált sorozatok száma
N = 0
```

Először az egyszínű sorozatokat keressük. Minden egyes színre végigmegyünk a kapott lapokon. Minden egyes lapra megnézzük, hogy az *Értékek* sorozatában a következő két elem is szerepel-e a lapok között.

```
VÁLTOZÓ Kezd MINT Egész
CIKLUS I=0-tól Színek.Elemszám–1-ig ' ciklus a színekre
    CIKLUS I=0-tól Lapok(Szín).Elemszám–2-ig ' ciklus a lapokra
        Kezd = Értékek.Indexe(Lapok(Szín)(I))
        HA Lapok(Szín).Tartalmazza(Értékek(Kezd+1))
            ÉS Lapok(Szín).Tartalmazza(Értékek(Kezd+2))
        AKKOR ' van három egymást követő érték a lapok között
            N = N + 1
            Ki: Színek(Szín)
            CIKLUS J=I-től I+2-ig
                Ki: Lapok(Szín)(J)
            CIKLUS VÉGE
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
CIKLUS VÉGE
```

A megoldás során feltételeztük, hogy a listaelemek indexelése 0-val kezdődik.

> Visual Basic-ben nem egyszerűsíthetjük a forráskódot minősítőblokkal (*With … End With*), ha egy lista által tartalmazott lista elemeire közvetlenül az indexükkel hivatkozunk. Nem hagyhatjuk el például a *Lapok*(*Szín*)(*I*) hivatkozásból a *Lapok*(*Szín*) minősítést. Ezért a közvetlen hivatkozás helyett az *Item* (vagy *ElementAt*) metódust hívjuk: `Lapok(Szín).Item(I)`
> Így már használhatunk minősítőblokkot (lásd a program forráskódját).

Ha nem kaptunk legalább háromféle színű lapot, akkor vége a programnak:

```
HA Színek.Elemszám < 3 AKKOR
   HA N == 0 AKKOR Ki: nincs kártyahármas
   PROGRAMBÓL KILÉP
ELÁGAZÁS VÉGE
```

Ha pontosan háromféle színű lapot kaptunk, akkor megkeressük a lehetséges sorozatokat. Ha azonban négyféle színű lapunk van, akkor ezek közül minden lehetséges szín-hármasra el kell végeznünk a vizsgálatot. Vonjuk össze a két esetet! Négy szín esetén a színekre vonatkozó ciklusból kihagyunk egy-egy színt, melynek indexét a *Kimarad* ciklusváltozó jelzi (a lépésköz lehetne +1 is):

```
VÁLTOZÓ Vég MINT Egész
Kezd = 0
Vég = 3
CIKLUS Kimarad = Vég-től Kezd-ig LÉPÉSKÖZ –1 ' a kimaradó szín indexe
   CIKLUS Szín = 0-tól Színek.Elemszám–1-ig
      HA Szín ≠ Kimarad AKKOR
         vizsgálat
      ELÁGAZÁS VÉGE
   CIKLUS VÉGE
CIKLUS VÉGE
```

ahol *Vég* = 3, *Kezd* = 0. Így elérjük, hogy minden lehetséges színhármast megvizsgálunk (gondoljuk meg, miért).

Ha csak háromféle színű lappal rendelkezünk, akkor egyik színt sem kell kihagyni. Ezt úgy érjük el, hogy a fenti ciklusban a *Kezd* és a *Vég* változót is 3-mal tesszük egyenlővé. Mivel ilyen indexű szín most nincsen, ezért egyetlen szín sem marad ki. A külső ciklus viszont egyszer lefut, mert a ciklusváltozó kezdő- és végértéke megegyezik egymással. Azaz:

```
HA Színek.Elemszám == 3 AKKOR Kezd = 3
EGYÉBKÉNT Kezd = 0
ELÁGAZÁS VÉGE
Vég = 3
```

Térjünk vissza a sorozatot kereső eljárásra. Minden lehetséges értékre megszámoljuk, hogy hányféle színű lappal rendelkezünk az adott értékből. Ha ez éppen három, akkor találtunk egy ilyen sorozatot. Közben felhasználjuk a színek kihagyására vonatkozó megfontolásainkat.

```
VÁLTOZÓ Db MINT Egész
CIKLUS MINDEN Érték-re az Értékek-ben
   CIKLUS Kimarad = Vég-től Kezd-ig LÉPÉSKÖZ –1
      Db = 0
      CIKLUS Szín = 0-tól Színek.Elemszám–1-ig
         HA SZÍN == Kimarad AKKOR SZÁMLÁLÓS CIKLUS FOLYTATÁS[15]
         HA LAPOK(Szín).Tartalmazza(Érték) AKKOR Db = Db + 1
      CIKLUS VÉGE
```

[15] *continue, Continue For*

```
        HA Db == 3 AKKOR ' kiírjuk a talált sorozatot
            N = N + 1
            CIKLUS Szín = 0-tól Színek.Elemszám–1-ig
                HA Szín == Kimarad AKKOR SZÁMLÁLÓS CIKLUS FOLYTATÁS
                Ki: Szín, Érték
            CIKLUS VÉGE
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
CIKLUS VÉGE
HA N == 0 AKKOR Ki: nincs kártyahármas a lapok között
```

Javasoljuk az Olvasónak, hogy alaposan gondolja végig az egyes színek kihagyását végző ciklusokat, valamint a feltételes elágazásokat. Természetesen készíthettünk volna egy ciklust a három szín, egy másikat pedig a négy szín esetére.

A teljes megoldást a *kartya* program tartalmazza.

Vitorlás

Nemes Tihamér OITV 2008. 3. forduló 2. korcsoport 2. feladat

Egy vitorlásversenyen K futamot rendeznek, naponta legfeljebb 1 futamot. Nem rendezhető futam, ha nagyon kicsi a szélsebesség, vagy ha nagy vihar van. Ha ilyen nap fordul elő, akkor szünnapot tartanak. Ismerjük N napra a szélsebesség előrejelzést. Az a cél, hogy a versenyt a lehető legkevesebb szünnappal rendezzék meg.

Készíts programot *vitorlas* néven, amely megadja, hogy mely napokon kezdődhet a verseny úgy, hogy a lehető legkevesebb szünnap legyen!

A *vitorlas.be* szöveges állomány első sorában a napok száma ($1 \le N \le 10000$), a futamok száma ($1 \le K \le 100$), valamint a legkisebb és a legnagyobb szélsebesség van, amikor még versenyt lehet rendezni ($0 \le$ *Minsebesség, Maxsebesség* ≤ 200). A következő N sorban egy-egy napra a szélsebesség előrejelzése található ($0 \le$ *sebesség* ≤ 200 km/óra).

A *vitorlas.ki* szöveges állomány első sorába a lehetséges verseny kezdőnapok M számát kell írni, a második sorba pedig az M kezdőnap sorszámát! Ha nincs K napon megfelelő szélsebesség, akkor az eredmény egyetlen sorába 0-t kell írni!

Példa:

```
vitorlas.be
8 4 10 100
0
30
30
80
110
70
60
40
```

```
vitorlas.ki
3
2 3 4
```

Megoldás

A beolvasás során csak azoknak a napoknak a sorszámát tároljuk a *Futamok* listában, amelyeken lehet futamot rendezni:

```
VÁLTOZÓ Futamok MINT Lista(Elemtípus: Egész)
CIKLUS I = 1-től N-ig
    Be: Szélsebesség
    HA MinSebesség ≤ Szélsebesség ÉS Szélsebesség ≤ MaxSebesség AKKOR
        Futamok.Add(I)
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A kiválasztott versenynapok sorszámát a *Versenynapok* listában gyűjtjük:

```
VÁLTOZÓ Versenynapok MINT Lista(Elemtípus: Egész)
```

A feldolgozás során vesszük a *Futamok* lista első *K* elemét a 0 indexűtől a *K*–1 indexűig (ezeken a napokon lehetséges futamot tartani). A közben eltelt napok száma:

```
Eltelt = Futamok(K–1) – Futamok.First + 1
```

tehát

```
Szünnap = Eltelt – K
```

nap nem felel meg (egy számpéldán keresztül gondoljuk meg, hogy miért!).

Ha a *Szünnap* értéke megegyezik az eddig talált legkisebb értékkel, akkor ráakadtunk egy újabb jó időintervallumra. Első napjának sorszámát hozzáadjuk a *Versenynapok* listájához. Ha pedig a *Szünnap* értéke kisebb, mint az eddigi legkisebb érték, akkor még jobb (kevesebb szünnappal rendelkező) időtartamot találtunk. Ekkor feljegyezzük az újabb minimumot, és töröljük (újra kezdjük) az eddigi *Versenynapok* listát:

```
HA Szünnap == MinSzünnap AKKOR
    Versenynapok.Add(Futamok.Első)
EGYÉBKÉNT HA Szünnap < MinSzünnap AKKOR
    MinSzünnap = Szünnap
    Versenynapok.Töröl()
    Versenynapok.Add(Futamok.Első)
ELÁGAZÁS VÉGE
Futamok.Kivesz(0)
```

Az elágazás után ne felejtsük el törölni a *Futamok* lista első (éppen feldolgozott) elemét!

A fenti eljárást addig ismételjük, amíg a *Futamok* lista legalább *K* elemet tartalmaz. A teljes megoldást a *vitorlas* program tartalmazza.

A módszer előnyeit akkor látjuk igazán, ha statikus tömbökkel is megoldjuk ezt a feladatot.

Sorozat

Nemes Tihamér OITV 2006. 2. forduló 2. korcsoport 3. feladat

Adott egy egész számokból álló sorozat.

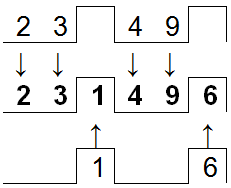
Készíts programot *sorozat* néven, amely meghatároz két olyan monoton növekvő sorozatot, amelyekből fésűs egyesítéssel megkapható a bemeneti sorozat![16]

A *sorozat.be* szöveges állomány első sorában a bemeneti sorozat elemeinek *N* száma ($0 \le N \le 1000$) van. A második sor pontosan *N* egész számot tartalmaz egy-egy szóközzel elválasztva, a bementi sorozatot. A sorozat minden *x* elemére teljesül, hogy ($0 \le x \le 30000$).

A *sorozat.ki* szöveges állomány első és egyetlen sora a 0 0 számpárt tartalmazza, ha a bemeneti sorozat nem állítható elő két monoton növekvő sorozat fésűs egyesítéseként. Egyébként az első sorban a fésűs egyesítéshez kiszámított első sorozat *K* elemszáma álljon! A második sor pontosan *K* számot tartalmazzon, az első sorozat elemeit, egy-egy szóközzel elválasztva! A harmadik sor tartalmazza a második sorozat *L* elemszámát! A negyedik sor pontosan *L* számot tartalmazzon, a második sorozat elemeit, egy-egy szóközzel elválasztva! A kiszámított sorozatok egyike üres is lehet, ekkor a 0 számot és üres sort kell kiírni. A bemeneti sorozat minden eleme pontosan az egyik kiszámított sorozat eleme. Több megoldás esetén bármelyik megadható.

Példa:

```
sorozat.be          sorozat.ki
6                   4
2 3 1 4 9 6         2 3 4 9
                    2
                    1 6
```



1. megoldás

A sorozat elemeit a *Sorozat* listában tartjuk nyilván:

```
VÁLTOZÓ Sorozat MINT Lista(Elemtípus: Egész)
```

A két monoton sorozatot könnyebben tudjuk kezelni, ha listáikat statikus tömbben tároljuk. Egyszerűbb lesz a feldolgozás, ha ütközőket helyezünk el a listák elején:

```
VÁLTOZÓ Monoton(1…2) MINT Lista(Elemtípus: Egész)
CIKLUS I=1-től 2-ig
    Monoton(I) = Új Lista(Elemtípus: Egész)
    Monoton(I).Add(–∞) ' ütköző
CIKLUS VÉGE
```

A *Sorozat* lista következő elemét először ahhoz a monoton sorozathoz próbáljuk meg hozzáadni, amelynek nagyobb szám áll a végén (tulajdonképpen a mohó algoritmust alkalmazzuk). Utána a másik monoton sorozattal próbálkozunk. Ha egyikhez sem sikerül hozzáadni (mert nagyobb elemek állnak az utolsó helyen), akkor nincs megoldása a feladatnak.

[16] Monoton növekvő sorozat: a következő elem nem kisebb, mint az előző (lehetnek egyenlők is).

```
CIKLUS AMÍG Sorozat.Elemszám > 0
    Kisebb = 1
    Nagyobb = 2
    HA Monoton(Kisebb).Utolsó > Monoton(Nagyobb).Utolsó AKKOR
        Kisebb = 2, Nagyobb = 1
    ELÁGAZÁS VÉGE
    HA Sorozat.Első ≥ Monoton(Nagyobb).Utolsó AKKOR
        Monoton(Nagyobb).Add(Sorozat.Első)
        Sorozat.Kivesz(0)
    EGYÉBKÉNT HA Sorozat.Első ≥ Monoton(Kisebb).Utolsó AKKOR
        Monoton(Kisebb).Add(Sorozat.Első)
        Sorozat.Kivesz(0)
    EGYÉBKÉNT ' egyikhez sem adható hozzá, nincs megoldás
        Sorozat.Töröl()
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A megoldás hiányának jelzésére bevezethettünk volna egy logikai változót is. Helyette töröltük a *Sorozat* listát, ami leállítja a ciklust.

A kiírás előtt ne feledjük eltávolítani az ütközőket! Mivel a ciklus mindenképpen törli a *Sorozat* listát, a kiírásnál a *Monoton* sorozatok elemszámának vizsgálatával dönthetjük el, hogy kaptunk-e megoldást.

A teljes megoldást a *sorozat-1* program tartalmazza.

2. megoldás

A monoton listákat tömb helyett listában tároljuk:

```
VÁLTOZÓ Monoton MINT Lista(Elemtípus: Lista(Elemtípus: Egész))
```

A létrehozás és az ütközők elhelyezése után az előző megoldáshoz hasonló ciklus következik. A két monoton listát azonban hivatkozás típusú változókkal kezeljük:

```
VÁLTOZÓ Kisebb, Nagyobb MINT Lista(Elemtípus: Egész)
```

Ezeket a listákat nem kell létrehozni, mert a változók már létező listákra fognak mutatni (így nem foglalnak el újabb helyet a memóriában). A hivatkozások hozzárendelésénél (és a program további részében is) kihasználjuk, hogy a *Monoton* lista csak két elemet (listát) tartalmaz:

```
Kisebb = Monoton.Első
Nagyobb = Monton.Utolsó
HA Kisebb.Utolsó > Nagyobb.Utolsó AKKOR
    Kisebb = Monoton.Utolsó
    Nagyobb = Monoton.Első
ELÁGAZÁS VÉGE
```

A folytatás az előző megoldás gondolatmenetét követi. A teljes megoldást a *sorozat-2* program tartalmazza. Vegyük észre, hogy a programban egyáltalán nem használtunk indexet!

Tipp

Nemes Tihamér OITV 2002. 2. forduló 2. korcsoport 1. feladat

Egy szerencsejátékban a résztvevők sorban egymás után egy-egy 1 és *M* közötti számot tippelnek. Az nyer, aki elsőként tippelt arra a számra, amelyre a résztvevők közül a legtöbben tippeltek. Ha több ilyen szám is van, akkor az összes ilyen szám első tippelője nyer.

Írj programot *tipp* néven, amely fogadja, és feldolgozza a tippeket, majd megadja a nyertes játékos sorszámát, a nyertes számot, és azt, hogy hányan választották ezt a számot!

A *tipp.be* állomány első sorában a versenyzők száma ($1 \le N \le 5000$) és a tipp maximális értéke ($1 \le M \le 1000000$) van, egyetlen szóközzel elválasztva. A következő *N* sorban van az egyes versenyzők tippje.

A *tipp.ki* állományba annyi sort kell írni, ahány győztes van (tippjük szerint növekvő sorrendben). Minden sorban három szám legyen egy-egy szóközzel elválasztva: a győztes sorszáma, a tippje, valamint az, hogy hányan tippelték ezt a számot!

Példa:

```
tipp.be                 tipp.ki
6 100                   3 15 2
25                      2 20 2
20
15
15
30
20
```

Megoldás

A tippeket olyan rendezett listában tároljuk, melyben az elemek kulcsa a tipp maga, az érték pedig az adott tippet tippelők listája. Így nagyon könnyen össze tudjuk gyűjteni azokat a résztvevőket, akik ugyanarra a számra tippeltek.

```
VÁLTOZÓ Tippek MINT RendezettLista(Kulcstípus: Egész,
                                   Értéktípus: Lista(Elemtípus: Egész))
```

A beolvasás során, ha még nem létezik az adott tipp listája, akkor létrehozzuk. Majd beletesszük a tippelő sorszámát:

```
CIKLUS Sorszám = 1-től N-ig
    Be: TippBe
    HA NEM Tippek.Kulcsként Tartalmazza(TippBe) AKKOR
        Tippek(TippBe) = Új Lista(Elemtípus: Egész)
    ELÁGAZÁS VÉGE
    Tippek(TippBe).Add(Sorszám)
CIKLUS VÉGE
```

Ezek után már csak azt kell megállapítanunk, hogy melyik lista tartalmazza a legtöbb elemet:

```
VÁLTOZÓ Legtöbb MINT Egész = 0
CIKLUS MINDEN Tipp-re a Tippek-ben
    HA Tipp.Érték.Elemszám > Legtöbb AKKOR Legtöbb = Tipp.Érték.Elemszám
CIKLUS VÉGE
```

Ne feledjük el, hogy a *Tipp* elem *Érték*-e az adott *Kulcs*-ra tippelők listája!

A rendezett kiírás nem jelent gondot, hiszen a tippeket eleve rendezetten tároljuk. Minden olyan tipp listájából kiíratjuk az első elemet (ő tippelte először ezt a számot), amelyben az elemek száma megegyezik a *Legtöbb* változó értékével. A kulcs megadja magát a tippet, a *Legtöbb* változó pedig a tippelők számát:

```
CIKLUS MINDEN Tipp-re a Tippek-ben
    HA Tipp.Érték.Elemszám == Legtöbb AKKOR
        Ki: Tipp.Érték.Első, Tipp.Kulcs, Legtöbb
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A teljes megoldást a *tipp* program tartalmazza. (Visual Basic-ben a *Tipp* változó használata miatt a névtér nevét *tipp1*-re módosítottuk.)

A feladatot igazán érdemes megoldani statikus tömbökkel is! ☺

Csapat

Nemes Tihamér OITV 2011. 3. forduló 2. korcsoport 4. feladat

Egy kiránduláson résztvevő tanulókból két csapatot kell képezni. A két csapatot úgy kell képezni, hogy ha *X* és *Y* nem szeretik egymást, akkor különböző csapatba kerüljenek. Mindét csapat legalább egy tanulót tartalmazzon!

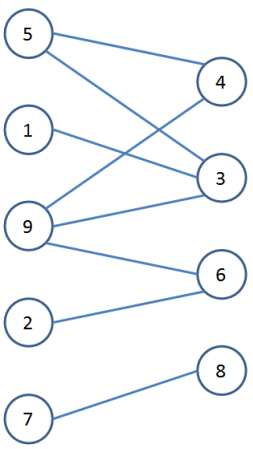
Készíts programot *csapat* néven, amely kiszámít egy, a feltételeknek megfelelő csapatbeosztást!

A *csapat.be* szöveges állomány első sorában két egész szám van egy szóközzel elválasztva, a tanulók *N* száma ($1 \le N \le 200$) és azon párok *M* ($1 \le M \le 20000$) száma, akik nem szeretik egymást. A tanulókat az *1, ..., N* számokkal azonosítjuk. A következő *M* sor mindegyike két egész számot tartalmaz (egy szóközzel elválasztva), *X Y* ($1 \le X, Y \le N$), ami azt jelenti, hogy *X* és *Y* nem szeretik egymást.

A *csapat.ki* szöveges állományba két sort kell kiírni, soronként a két csapat tagjait, egy-egy szóközzel elválasztva! Ha nincs megoldása a feladatnak, akkor az első és egyetlen sorba a –1 számot kell kiírni. A sorban a számok tetszőleges sorrendben kiírhatók. Több megoldás esetén bármelyik megadható.

Példa:

```
csapat.be          csapat.ki
9 8                1 2 5 7 9
1 3                3 4 6 8
3 9
9 4
5 4
2 6
7 8
3 5
9 6
```



Megoldás

Az adatokat gráffal ábrázolhatjuk. A gráf csúcsai jelképezik a diákokat, élei pedig a közöttük fennálló „nem szereti” kapcsolatot. Mivel az ellenszenv kölcsönös, irányítatlan gráfot kapunk. Feladatunk a csúcsokat két olyan csoportba sorolni, hogy csak az ellenkező csoportba tartozó csúcsokat kösse össze él.

A beolvasott adatokat rendezett listában tároljuk. Az elemek kulcsa a tanulók azonosítója (sorszáma), értéke pedig azon tanulók sorszámainak halmaza, akiket a kulcsként megadott tanuló nem szeret (szomszédsági „lista”). A kialakuló csapatok tagjait rendezett halmazokban tartjuk nyilván:

```
VÁLTOZÓ Nemszeret MINT RendezettLista(Kulcstípus: Egész,
                                      Értéktípus: Halmaz(Elemtípus: Egész))
VÁLTOZÓ EgyikCsapat, MásikCsapat MINT RendezettHalmaz(Elemtípus: Egész)
```

Beolvasáskor ügyeljünk arra, hogy egy élt mindkét csúcshoz hozzárendeljük (irányítatlan gráf)! Ha valamelyik tanuló még nem szerepel kulcsként a *Nemszeret* listában, akkor a hozzá tartozó halmazt is létre kell hozni:

```
Be: N, M
CIKLUS I=1-től M-ig
    Be: Ki, Kitnem
    HA NEM Nemszeret.Kulcsként Tartalmazza(Ki) AKKOR
        Nemszeret(Ki) = Új Halmaz(Elemtípus: Egész)
    ELÁGAZÁS VÉGE
    Nemszeret(Ki).Add(Kit)
    HA NEM Nemszeret.Kulcsként Tartalmazza(Kit) AKKOR
        Nemszeret(Kit) = Új Halmaz(Elemtípus: Egész)
    ELÁGAZÁS VÉGE
    Nemszeret(Kit).Add(Ki)
CIKLUS VÉGE
```

A csapatokat tartalmazó halmazok feltöltését a következőképpen végezzük.

Tegyük a *Nemszeret* lista első diákját az *EgyikCsapat* halmazba, a számára ellenszenves diákokat pedig a *MásikCsapat* halmazba! Ezután töröljük a lista első elemét (amit így feldolgoztunk).

Most a *MásikCsapat* halmazba került diákok számára ellenszenves diákokat tegyük az *ElsőCsapat* halmazba! Minden egyes diákot, a számára ellenszenves diákok elhelyezése után törlünk a listából.

A továbbiakban a fenti lépéseket ismételjük az *ElsőCsapat* tagjaival, majd újra a *MásikCsapat* tagjaival stb. Közben természetesen csak azokkal a diákokkal foglalkozunk, akik még szerepelnek a *Nemszeret* listában (tehát nem helyeztük el a számukra ellenszenves diákokat egyik halmazban sem).

Ha már nem változnak a halmazok (így a *Nemszeret* lista sem), akkor minden eddigi diák „ellenszenv” kapcsolatait feldolgoztuk, véget ért az ismétlés.[17]

Ha a kapcsolatokat ábrázoló gráf több, különálló részre (komponensre) esik szét (lásd a feladatban szereplő példa 7-es és 8-as diákját), akkor még marad elem a *Nemszeret* listában. A fenti eljárást ezért addig ismételjük, amíg ki nem ürül a lista. Az algoritmus vázlatosan:

[17] Lényegében a gráf szélességi bejárását végezzük el.

```
ciklus, amíg van eleme a Nemszeret listának
    a lista első diákját hozzáadjuk az EgyikCsapat halmazhoz
    ciklus, amíg változik a Nemszeret lista elemszáma
        az EgyikCsapat halmazban szereplő minden egyes diákra
            a számára ellenszenves diákokat hozzáadjuk a MásikCsapat halmazhoz,
            majd a feldolgozott diákot töröljük a Nemszeret listából
        a MásikCsapat halmazban szereplő minden egyes diákra
            a számára ellenszenves diákokat hozzáadjuk az EgyikCsapat halmazhoz,
            majd a feldolgozott diákot töröljük a Nemszeret listából
    ciklus vége
ciklus vége
```

Javasoljuk az Olvasónak, hogy a feladatban szereplő példára papíron kövesse végig a csapatok kialakítását.

Az algoritmus pontos felírásához bevezetjük az *Elemszám* segédváltozót, amelynek segítségével figyeljük a *Nemszeret* lista elemszámának változását. A halmazok feltöltésénél pedig ügyelünk arra, hogy csak olyan diákkal foglalkozzunk, aki még szerepel a listában:

```
VÁLTOZÓ Elemszám MINT Egész
CIKLUS AMÍG Nemszeret.Elemszám > 0
    Ki = Nemszeret.Első.Kulcs
    EgyikCsapat.Add(Ki)
    CIKLUS
        Elemszám = Nemszeret.Elemszám ' félretesszük az elemszámot
        CIKLUS MINDEN Ki-re az EgyikCsapat-ban
            HA Nemszeret.Kulcsként Tartalmazza(Ki) AKKOR
                CIKLUS MINDEN Kitnem-re a Nemszeret(Ki)-ben
                    MásikCsapat.Add(Kitnem)
                CIKLUS VÉGE
                Nemszeret.Kivesz(Ki)
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
        CIKLUS MINDEN Ki-re a MásikCsapat-ban
            HA Nemszeret.Kulcsként Tartalmazza(Ki) AKKOR
                CIKLUS MINDEN Kitnem-re a Nemszeret(Ki)-ben
                    EgyikCsapat.Add(Kitnem)
                CIKLUS VÉGE
                Nemszeret.Kivesz(Ki)
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
    ISMÉTLÉS AMÍG Elemszám > Nemszeret.Elemszám
    CIKLUS VÉGE
CIKLUS VÉGE
```

A feladatnak akkor van megoldása, ha az *EgyikCsapat* és a *MásikCsapat* halmazoknak nincs közös eleme. Ezt a metszet helyett az elemszámok összegezésével is ellenőrizhetjük. A kiírás ennek megfelelően:

```
HA EgyikCsapat.Elemszám + MásikCsapat.Elemszám < N AKKOR
    Ki. –1 ' nincs megoldás
EGYÉBKÉNT
    Ki: a két halmaz elemei
ELÁGAZÁS VÉGE
```

A két halmaz közös elemének megjelenését már a feltöltést végző ciklusokban figyelhetjük. A közös elem megjelenésekor nem kell folytatnunk a feldolgozást, hiszen nincs megoldás. Alakítsuk át ennek megfelelően a fenti algoritmust!

Célszerű megírni a programot úgy, hogy a két csapat tagjait tartalmazó halmazokat tömbben/listában tároljuk, és ciklussal rövidítjük a forráskódot. Készítsük el a program olyan változatát is, amelyben nem töröljük a *Nemszeret* lista elemeit! Hogyan tarthatjuk nyilván ebben az esetben a már feldolgozott elemeket?

A *Programozási ismeretek versenyzőknek* példatár *Páros gráfok* fejezete bemutatja a feladat megoldását a gráf párosságának ellenőrzésével és a csúcsok szétválogatásával.

További feladatok a dinamikus tömbökre

Dinamikus tömböt szinte minden programban alkalmazunk. Itt csak néhány olyan feladatot sorolunk fel, amely különösen alkalmas ezen adatszerkezet gyakorlására.

OITV 2000-2f2	Gombaszedés
OITV 2002-3f2	Felvételi
OITV 2003-3f2	Futó
OITV 2005-1f2	Lista
OITV 2010-3f2	Játék
OKTV 2010-3f3	Játékos

5. Struktúrák

Kapcsolódó leckék:
Programozási ismeretek (Műszaki Kiadó, 2011)
9. Játék a véletlennel (érték és hivatkozás típusú változók)
33. Rekordok és struktúrák
Programozási ismeretek haladóknak (Műszaki Kiadó, 2012)
16. Adattípusok a programozási nyelvekben

A programozási nyelvek – a statikus tömböket kivéve – általában nem rendelkeznek „többdimenziós” kollekciókkal. Néhány kivételtől eltekintve, a többdimenziós statikus tömbök egyes dimenziói sem lehetnek egymástól eltérő típusúak. Azonos típusú dimenziók esetén pedig nehézkes – többek között – a sorok átrendezése. Ezeket a problémákat küszöböli ki a struktúrák használata.

Mivel a struktúrával új típust definiálunk, a struktúra azonosítóját *T* betűvel fogjuk kezdeni. A struktúra típusú változót egyes számban, az ilyen elemeket tartalmazó kollekció nevét pedig többes számban deklaráljuk, például:

```
STRUKTÚRA TDiák …
VÁLTOZÓ Diák MINT TDiák
VÁLTOZÓ Diákok MINT Lista(Elemtípus: TDiák)
```

Az alábbiakban elsősorban a .NET programozási környezet sajátosságait tartottuk szem előtt. Egyéb esetben célszerű utánanézni a használt nyelv jellemzőinek.

> C++ ban a struktúra változók azonosságának gyors vizsgálatára a *memcmp()* függvényt használhatjuk (*string.h*), melynek hívásakor a két változó címe mellett a struktúra bájtokban kifejezett méretét (*sizeof*) is meg kell adnunk.

> A Visual Basic-ben két, csak érték típusú mezőket tartalmazó, struktúra típusú változó egyenlőségét a *Változó1.Equals*(*Változó2*) metódushívással vizsgálhatjuk. A visszatérési érték igaz, ha a megfelelő mezők értéke páronként megegyezik egymással. (Az összehasonlítás a memóriabájtok alapján történik.)

Struktúra típusú elemeket tartalmazó kollekciók

A struktúrák alkalmazásánál ügyeljünk arra, hogy a struktúrák érték típusú változók! Azaz a *Diák2 = Diák1* értékadás átmásolja a *Diák1* struktúra mezőinek értékét a *Diák2* struktúra megfelelő mezőibe. Ha azonban valamely mező hivatkozás típusú (például kollekció), akkor csak a hivatkozást másolja át. Ennek következményeit később részletezzük.

Statikus tömbök esetén a struktúra típusú elemeket a szokásos módon használhatjuk, akár a kifejezésekben, akár az értékadásnál. Például:

```
VÁLTOZÓ Diákok(100) MINT TDiák
Diákok(1).Név = "Sipos Zsolt"
```

Más kollekcióknál viszont a kollekció elemeire való hivatkozás (például az indexükkel) valójában egy metódushívást takar[18], amely az elem értékével tér vissza, így nem állhat az értékadás bal oldalán. Ezért hibás a következő értékadás:

```
VÁLTOZÓ Diákok MINT Lista(Elemtípus: TDiák)
…
Diákok(I).Név = "Sipos Zsolt"
```

Ekkor az értékadást (illetve egy mező módosítását) például segédváltozó útján[19] végezhetjük el:

```
Diák = Diákok(I)
Diák.Név = "Sipos Zsolt"
Diákok(I) = Diák
```

Ugyanakkor a *Diákok*(*J*) = *Diákok*(*I*) típusú értékadás továbbra is helyesen működik, mert ilyenkor a program átmásolja a *Diákok*(*I*) struktúra mezőinek értékét a *Diákok*(*J*) struktúrába.

> A Visual Basic-ben két, struktúrákat tartalmazó kollekció egyenlőségének vizsgálatára használhatjuk a *Kollekció1*.*SequenceEqual*(*Kollekció2*) metódushívást, ha a struktúrák csak érték típusú mezőket tartalmaznak. A metódushívás eredménye *True*, ha a kollekciók azonos indexű tagjaiban a struktúrák mezőinek értéke páronként megegyezik egymással.

Olimpia

Nemes Tihamér OITV 2008. 3. forduló 1. korcsoport 3. feladat

Az olimpiai játékokon *M* ország vesz részt ($1 \le M \le 100$), *N* versenyszámban versenyeznek a résztvevők ($1 \le N \le 1000$). Minden versenyszámban 1 arany-, 1 ezüst-, valamint 1 vagy 2 bronzérmet adnak ki (kieséses versenyek esetén a döntőbe jutásért küzdők közül mindkét vesztes bronzérmet kap). Az országokat 1 és *M* közötti sorszámukkal azonosítjuk.

Készíts programot *olimpia* néven, amely az eredmények ismeretében előállítja az olimpia éremtáblázatát! Az éremtáblázat aranyérmek száma szerint csökkenő sorrendű legyen! Azonos aranyéremszám esetén a több ezüst-, azonos ezüstérem szám esetén a több bronzérem döntsön! Ha mindhárom éremből ugyanannyi van, akkor a kisebb sorszámú legyen elöl!

Példa:[20]

Bemenet	*Magyarázat*
`5 3`	*M* és *N* értéke
`2 3 2`	arany-, ezüst-, bronzérmes az 1. számban
`1 2 3`	arany-, ezüst-, bronzérmes a 2. számban
`5 2 2 3`	arany-, ezüstérmes és bronzérmesek a 3. számban

[18] A *Lista*(*I*) például a *Lista*.*Item*(*I*) rövidített írásmódja.

[19] Felhasználhatjuk továbbá a struktúra konstruktorát, vagy a definíciót bővíthetjük az értékadást elvégző függvénnyel.

[20] A bemenet formátumát némileg egyszerűsítettük.

Kimenet

```
2. ország: 1 A 2 E 2 B
1. ország: 1 A
5. ország: 1 A
3. ország: 1 E 2 B
```

Megoldás

Az országok eredményeit az *Országok* tömbben tároljuk. A tömbelemek struktúrák, melyek mezői megadják az arany, ezüst illetve bronzérmek számát. A rendezés miatt az ország azonosítóját (sorszámát) is felvesszük a struktúra mezői közé.

```
STRUKTÚRA TOrszág
    VÁLTOZÓ Sorszám, Arany, Ezüst, Bronz MINT Egész
STRUKTÚRA VÉGE
VÁLTOZÓ Országok(1…M) MINT TOrszág
VÁLTOZÓ N, M MINT Egész
```

Az *M* és *N* értékének beolvasása után feltöltjük a *Sorszám* mezőket:

```
Be: M, N
CIKLUS I=1-től M-ig
    Országok(I).Sorszám = I
CIKLUS VÉGE
```

Az érmek számának beolvasásánál segédváltozókat használunk. Az *AranyBe* jelenti például annak az országnak a sorszámát, amely az adott versenyszámban aranyérmet nyert. Így az *Országok* tömbben az *AranyBe* indexű elem (struktúra) *Arany* mezőjének értékét növeljük meg eggyel! Közben ügyeljünk arra, hogy esetenként két bronzérmet is kiadtak!

```
VÁLTOZÓ AranyBe, EzüstBe, BronzBe MINT Egész
CIKLUS I=1-től N-ig
    Be: AranyBe, EzüstBe, BronzBe
    Országok(AranyBe).Arany = Országok(AranyBe).Arany + 1
    Országok(EzüstBe).Ezüst = Országok(EzüstBe).Ezüst + 1
    Országok(BronzBe).Bronz = Országok(BronzBe).Bronz + 1
    HA van még adat a sorban AKKOR
        Be: BronzBe
        Országok(BronzBe).Bronz = Országok(BronzBe).Bronz + 1
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

Ha nem tartjuk zavarónak, akkor az *AranyBe* stb. változónév helyett használjuk egyszerűen az *Arany* azonosítót! A program ugyanis megkülönbözteti az *Arany* nevű változót egy struktúra *Arany* nevű mezőjétől. Ez utóbbira mindig a struktúraváltozó nevével minősítve hivatkozunk (*Országok*(*I*).*Arany*). Így a mezőnév meg fog egyezni annak a változónak a nevével, amelyet a mező értékadó utasításában használunk. Ezzel a hibalehetőségek számát csökkentjük (vagy növeljük? ☺):

```
Be: Arany ' az aranyérmet nyert ország sorszáma (indexe az Országok tömbben)
Országok(Arany).Arany = Országok(Arany).Arany + 1
```

Feladatunk az *Országok* tömb elemeinek rendezése a feladatban szereplő szempontok szerint. A többszempontú (több kulcs szerint történő) rendezést a következő gondolatmenet alapján hajtjuk végre.

Két ország összehasonlításánál
ha az aranyérmek alapján eldönthető a sorrend, akkor
    szükség szerint cserélünk
egyébként ha az aranyérmek száma megegyezik, akkor
    ha az ezüstérmek alapján eldönthető a sorrend, akkor
        szükség szerint cserélünk
    egyébként ha az ezüstérmek száma megegyezik, akkor
        ha a bronzérmek száma alapján eldönthető a sorrend, akkor
            szükség szerint cserélünk
        egyébként a sorszámok alapján döntünk a cseréről.

Közben ügyeljünk arra, hogy az érmek alapján csökkenő, a sorszámok szerint viszont növekvő sorrendet kell kialakítani!

Mivel a csere utasításai több helyen is szerepelnének az algoritmusban, először csak egy logikai változóban jegyezzük fel a szükségességét, majd az összehasonlítások után hajtjuk végre a cserét.

Az egyszerű cserés rendezés[21] algoritmusa így:

```
VÁLTOZÓ Ország MINT TOrszág ' segédváltozó a cseréhez
VÁLTOZÓ Csere MINT Logikai
CIKLUS I=1-től M–1-ig
    CIKLUS J=I+1-től M-ig
        Csere = Hamis
        HA Országok(I).Arany < Országok(J).Arany AKKOR
            Csere = Igaz
        EGYÉBKÉNT HA Országok(I).Arany == Országok(J).Arany AKKOR
            HA Országok(I).Ezüst < Országok(J).Ezüst AKKOR
                Csere = Igaz
            EGYÉBKÉNT HA Országok(I).Ezüst == Országok(J).Ezüst AKKOR
                HA Országok(I).Bronz < Országok(J).Bronz AKKOR
                    Csere = Igaz
                EGYÉBKÉNT HA Országok(I).Bronz == Országok(J).Bronz AKKOR
                    HA Országok(I).Sorszám > Országok(J).Sorszám AKKOR ' növekvő!
                        Csere = Igaz
                    ELÁGAZÁS VÉGE
                ELÁGAZÁS VÉGE
            ELÁGAZÁS VÉGE
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
CIKLUS VÉGE
HA Csere AKKOR
    Ország = Országok(I)
    Országok(I) = Országok(J)
    Országok(J) = Ország
ELÁGAZÁS VÉGE
```

[21] Lásd például: *Programozási ismeretek*, 244. oldal.

Vegyük észre, hogy struktúra alkalmazásával leegyszerűsödött a csere végrehajtása! Gondoljuk meg, miért vizsgáljuk egy-egy érem esetén az egyenlőséget is, mielőtt még rátérnénk a következő éremre!

A teljes megoldást az *olimpia* program tartalmazza. A programban a tömbmetódussal történő rendezést is bemutatjuk.

Alma

Nemes Tihamér OITV 2004. 2. forduló 2. korcsoport 2. feladat

Egy piacon *M* árus *N* egymást követő napon árul almát. Az árusok különböző napokon kezdenek almát árulni, s ettől kezdve, amíg más árat nem választanak, ugyanazon az áron adják az almát. Arra vagyunk kíváncsiak minden napon, hogy aznap mely *K* árustól lehet a legolcsóbban almát venni.

Írj programot *alma* néven, amely megadja minden napra, hogy aznap mely *K* árustól lehet a legolcsóbban almát venni, ha van aznap egyáltalán *K* árus!

Az *alma.be* szöveges állomány első sorában az árusok *M* száma ($1 \le M \le 100$), a napok *N* száma ($1 \le N \le 1000$), és a *K* értéke ($1 \le K \le M$) van egy-egy szóközzel elválasztva. Az állomány további sorai egy-egy árus érkezését írják le. Mindegyik sorban három szám van egy-egy szóközzel elválasztva: az érkezés napja ($1 \le nap \le N$, a sorok eszerint növekvő sorrendben jönnek), az árus sorszáma ($1 \le sorszám \le M$) és az általa árult alma ára attól a naptól kezdve ($0 < ár \le 1000$).

Az *alma.ki* szöveges állományba pontosan *N* sort kell írni, az *i*-edik sorba az *i*-edik napon legolcsóbb *K* árus sorszámát, növekvő sorrendben, egy-egy szóközzel elválasztva. Ha aznap nem árult almát *K* árus, akkor a sorba egyetlen 0-t kell kiírni.

Példa:

alma.be	alma.ki	*Magyarázat*
6 8 3	0	az első napon nincs 3 árus
1 1 100	2 3 6	
1 2 90	1 3 6	
2 6 80	1 3 6	a negyedik napon nem jött újabb árus
2 3 70	1 3 6	az ötödik napon nem jött újabb árus
2 5 120	1 3 6	a hatodik napon nem jött újabb árus
3 1 60	3 4 6	
3 4 100	3 4 6	a nyolcadik napon nem jött újabb árus
7 1 120		
7 4 75		

Megoldás

Az árusok adataihoz struktúrát definiálunk:

```
STRUKTÚRA TÁrus
    VÁLTOZÓ Sorszám, Érkezés, Ár MINT Egész
STRUKTÚRA VÉGE
```

Az árusok adatait listában tároljuk. A feldolgozás során ugyancsak listába gyűjtjük az árusokat, az aktuális árak szerint rendezve:

```
VÁLTOZÓ Árusok, Rendezett MINT Lista(Elemtípus: TÁrus)
```

Nem használhatunk rendezett listát, mert az ár nem tekinthető kulcsnak. A rendezést a lista feltöltése során magunknak kell elvégeznünk. A rendezett listában ugyan nem lenne szükségünk az érkezés időpontjára, de egy újabb struktúra definiálásával, a mezők egyenkénti átmásolásával csak elbonyolítanánk a programot.

A beolvasás során segédváltozó segítségével adunk értéket az *Árusok* lista elemeinek:

```
VÁLTOZÓ M, N, K MINT Egész
VÁLTOZÓ Árus MINT TÁrus
Be: M, N, K
CIKLUS AMÍG van adat a fájlban
   Be: Érkezés, Sorszám, Ár
   Árus.Érkezés = Érkezés, Árus.Sorszám = Sorszám, Árus.Ár = Ár
   Árusok.Add(Árus)
CIKLUS VÉGE
```

> C++ 11-ben a struktúra változók inicializálása egyszerűen elvégezhető az inicializációs lista felhasználásával: `arus = {temp[1], temp[0], temp[2]};`

> Visual Basic-ben segédváltozó helyett a *New* operátort is alkalmazhatjuk:
> ```
> Árus = New TÁrus With _
>             {.Érkezés = Temp(0), .Sorszám = Temp(1), .Ár = Temp(2)}
> ```

Felhívjuk a figyelmet arra, hogy a *TÁrus* struktúra érték típusú mezőket tartalmaz. Ennek a megállapításnak majd a kollekciókat tartalmazó struktúránál lesz fontos szerepe (lásd a következő szakaszt).

A beolvasás után minden egyes napra feldolgozzuk az aznap érkező, illetve árat módosító árusokat, majd elvégezzük a kiírást. Az árust a feldolgozása után – szokás szerint – töröljük az *Árusok* listából. Így mindig az első listaelemet kell megvizsgálnunk:

```
VÁLTOZÓ I, ÚjÁr MINT Egész ' segédváltozók az Árus feldolgozásához
CIKLUS Nap = 1-től N-ig
   CIKLUS AMÍG Árusok.Elemszám > 0 ÉS# Árusok.Első.Érkezés = Nap
      az Árus feldolgozása
      Árusok.Kivesz(0)
   CIKLUS VÉGE
   Kiírás
CIKLUS VÉGE
```

Egy árus feldolgozása a következőket jelenti. Megkeressük, hogy szerepel-e már az árus a *Rendezett* listában.[22] Ha igen, akkor töröljük, mert lehet, hogy eddig olcsóbban adta az almát, ezután pedig drágábban fogja adni. Áttekinthetőbbé tesszük a programot, ha változókkal helyettesítjük a mezőkre történő hivatkozást:

[22] A keresés ciklusát a későbbiekben metódushívással helyettesíthetjük.

```
' Az Árus feldolgozása
Árus = Árusok.Első
ÚjÁr = Árus.Ár
I = 0 ' lineáris keresés
CIKLUS AMÍG I < Rendezett.Elemszám
                                        ÉS# Rendezett(I).Sorszám ≠ Árus.Sorszám
    I = I + 1
CIKLUS VÉGE
HA I < Rendezett.Elemszám AKKOR Rendezett.Kivesz(I)
```

Ezek után az árust hozzáadjuk az árusokhoz úgy, hogy rendezett maradjon a lista:

```
I = 0
CIKLUS AMÍG I < Rendezett.Elemszám ÉS# Rendezett(I).Ár < ÚjÁr
    I = I + 1
CIKLUS VÉGE
Rendezett.Beilleszt(I, Árus)
```

Vegyük észre, hogy algoritmusunk az első árusra is jól működik, mert a *Beilleszt* metódus a lista végéhez szintén hozzá tudja fűzni az új elemet (ekkor *I* = *Elemszám*)!

Mivel a kiírást a sorszámok szerint rendezve kell elvégezni, deklarálunk egy rendezett halmazt. Ebbe beletesszük a *Rendezett* lista első *K* elemét (ha van annyi), majd kiírjuk a halmazelemeket. Ne felejtsük el a halmazt törölni, mert napról napra, ismételten sor kerül a kiírásra!

```
VÁLTOZÓ Sorszámok MINT RendezettHalmaz(Elemtípus: Egész)
HA Rendezett.Elemszám < K AKKOR
    Ki: 0 ' eddig nem volt legalább K árus
EGYÉBKÉNT
    Sorszámok.Töröl()
    CIKLUS I=0-tól K–1-ig
        Sorszámok.Add(Rendezett(I).Sorszám)
    CIKLUS VÉGE
    Ki: a Sorszámok halmaz elemei
ELÁGAZÁS VÉGE
```

A teljes megoldás az *alma* program tartalmazza. Javasoljuk az Olvasónak, hogy készítse el a program olyan változatát, amelyben a beolvasás során végzi el a feldolgozást (tehát nem tárolja az adatokat az *Árusok* listában).

Kollekciókat tartalmazó struktúra

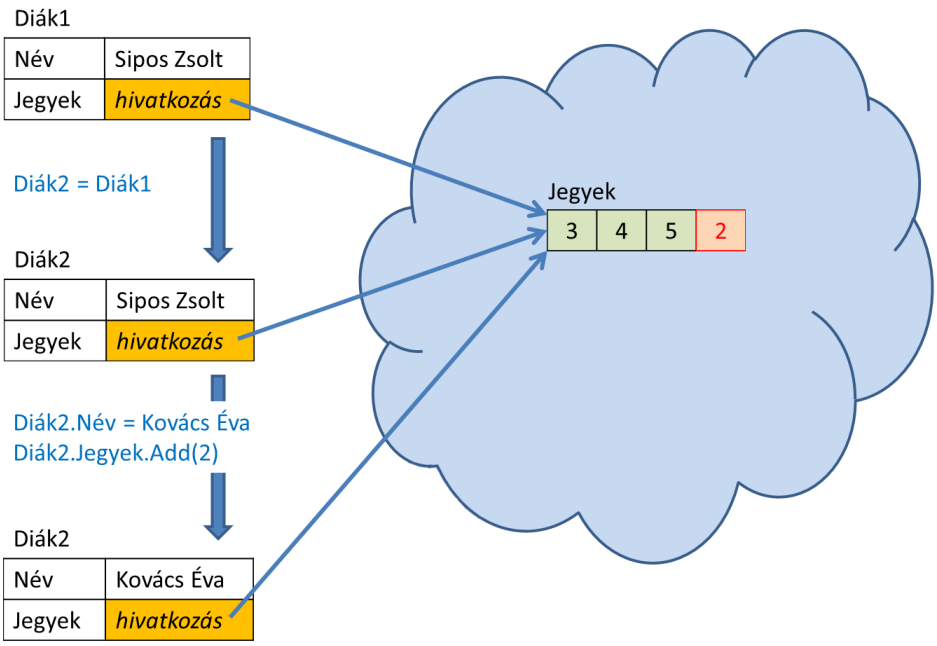
Ha a struktúra rendelkezik kollekció típusú mezővel, akkor tartsuk szem előtt a következőket!

A struktúrával egy új típust hozunk létre, tehát a definíciójában

- nem adhatjuk meg a statikus tömbök méretét és a kollekciók kapacitását;
- nem hozhatjuk létre az objektumokat (nem használható a *New* operátor);
- nem adhatjuk meg továbbá a változók (mezők) kezdőértékét (elemi típusú mezőnél sem)!

A kollekció létrehozását a struktúra típusú változó deklarálása után, de első használata előtt kell elvégezni.

> Visual Basic-ben statikus tömböt a *ReDim* utasítással hozhatunk létre.



Értékadás érték-, illetve hivatkozás típusú mezőnél

A kollekciókra hivatkozás típusú változók mutatnak. Ezért a struktúrák közötti értékadásnál csak az érték típusú mezők másolódnak át! A kollekció típusú mezők esetén a hivatkozás kerül átadásra, azaz mindkét struktúra mezője ugyanarra a kollekcióra mutat!

Tegyük fel például, hogy a *Diák1* és *Diák2* struktúra típusú változó a diák nevét (sztring) és jegyeit (lista) tartalmazza (lásd az alábbi ábrát)! Ekkor a *Diák2 = Diák1* értékadás után a név módosítása az adott változóra, a jegyek módosítása azonban mindkét diákra vonatkozik. A jegyek listája csak egy példányban létezik a memóriában!

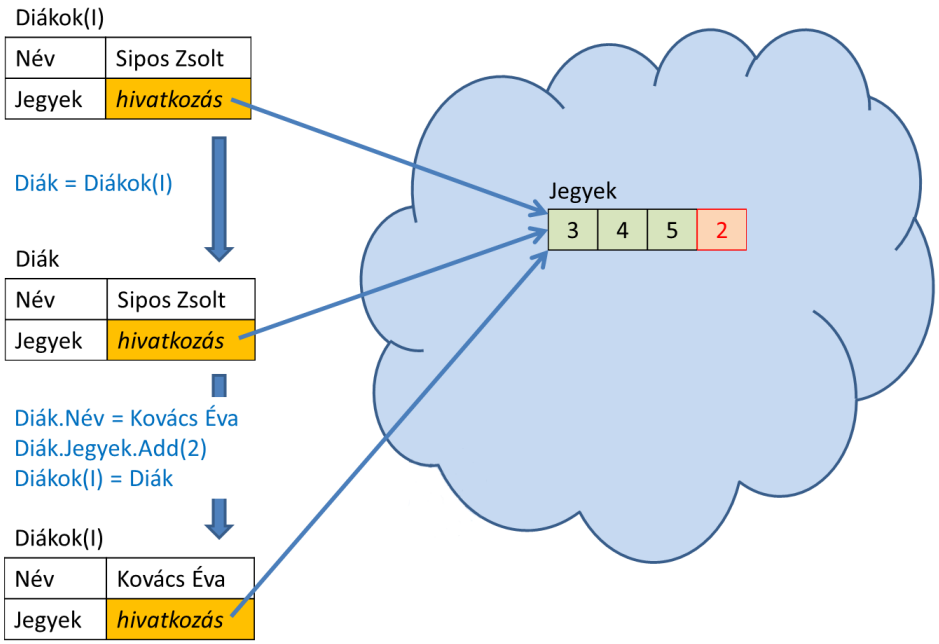
A helyes megoldáshoz az értékadás után létre kell hozni az új kollekciót, majd az eredeti kollekció elemeit át kell másolni az új kollekcióba.

> Visual Basic-ben a statikus tömböknél a *ReDim* utasítást és a *CopyTo* példánymetódust használhatjuk. Az *Array.Copy* osztálymetódus többféle paraméterezéssel is hívható. Listák (és más kollekciók) esetén a létrehozáskor megadhatjuk az inicializáláshoz használt kollekciót, például:
> `Diák2.Jegyek = New List(Of Integer)(Diák1.Jegyek)`

Egyszerűbb a helyzet, ha egy struktúra mezőit segédváltozók útján módosítjuk. Ekkor nincs semmi egyéb tennivalónk, mert a segédváltozó hivatkozás típusú mezője eleve az eredeti kollekcióra mutat (lásd az ábrát a következő oldalon). Ha csak a kollekció típusú mezőt módosítjuk, akkor nincs is szükség segédváltozóra. Ugyanis ilyenkor nem a mezőt módosítjuk, hanem magát a kollekciót. A

```
Diák = Diákok(I)
Diák.Jegyek.Add(2)
Diákok(I) = Diák
```

utasítások helyett használhatjuk a következő értékadást (lásd még a megjegyzést az 55. oldalon):



Értékmódosítás segédváltozóval érték-, illetve hivatkozás típusú mezőnél

Diákok(I).Jegyek.Add(2)

> Visual Basic-ben a struktúrák egyenlőségét vizsgáló *Equals* metódus a hivatkozás típusú mezőknél csak a hivatkozások azonosságát jelzi (tehát nem hasonlítja össze elemről-elemre a kollekciókat).

Verseny

Nemes Tihamér OITV 2010. 3. forduló 1. korcsoport 2. feladat

Egy pontozásos versenyen N résztvevő indul ($1 \le N \le 100$). A versenyzőket M pontozó pontozza ($3 \le M \le 10$). A pontozók azonban befolyásolhatnák a verseny végeredményét (a saját országbeli versenyzőknek nagyobb, a más országbeli versenyzőknek pedig kisebb pontot adva), ezért a verseny szervezői úgy döntöttek, hogy az M pontszám közül a legkisebbet és a legnagyobbat egy versenyzőnél sem veszik figyelembe, azaz mindenkit M–2 pontszám összegével értékelnek.

Készíts programot *verseny* néven, amely

a) kiírja minden versenyzőre, hogy mely pontozók pontszámát hagyják ki;
b) kiszámítja és kiírja minden versenyző pontszámát;
c) kiírja a verseny végeredményét (a versenyzőket pontszám szerint csökkenő sorrendben, holtverseny esetén azonos helyezési számmal)!

Példa:

Bemenet

```
N=4, M=4
1. versenyző pont: 8 2 6 6
2. versenyző pont: 5 5 5 5
3. versenyző pont: 4 5 6 7
4. versenyző pont: 5 4 6 5
```

Kimenet

```
Kihagyott pontozók:
1. versenyző: 1 2
2. versenyző: 1 2        (bármely kettő jó)
3. versenyző: 1 4
4. versenyző: 2 3
Pontszámok:
1. versenyző: 12 pont
2. versenyző: 10 pont
3. versenyző: 11 pont
4. versenyző: 10 pont
Sorrend:
1. helyezett: 1. versenyző 12 pont
2. helyezett: 3. versenyző 11 pont
3. helyezett: 2. versenyző 10 pont
3. helyezett: 4. versenyző 10 pont
```

Megjegyzés: a feladat szövegéből nem derül ki, de ha például két 3. helyezett van, akkor a következő versenyző nem a 4., hanem az 5. helyezett lesz (lásd a megoldási útmutatót).

1. Megoldás tömbökkel

A *TVersenyző* struktúra a sorszámon és az összpontszámon kívül a versenyző pontjait tároló tömböt is tartalmazza:

```
STRUKTÚRA TVersenyző
    VÁLTOZÓ Sorszám, Pontok(), Összpont MINT Egész
STRUKTÚRA VÉGE
```

A versenyzőket statikus tömbben tartjuk nyilván:

```
VÁLTOZÓ Versenyzők(1…N) MINT Egész
```

A beolvasásnál ügyeljünk arra, hogy létre kell hozni az egyes versenyzők pontszámait tartalmazó tömböket:

```
CIKLUS I=1-től N-ig
    Versenyzők(I).Sorszám = I
    Versenyzők(I).Pontok = Új Egész(1…M)
    Be: Versenyzők(I).Pontok tömb elemei
CIKLUS VÉGE
```

Minden egyes versenyzőre a legkisebb, illetve legnagyobb pontszámot adó pontozót hagyjuk ki az összegezésből. A „kihagyást” úgy végezzük, hogy a pontszámot átírjuk 0-ra, így nem befolyásolja az összeget. Közben gondoskodnunk kell arról, hogy feltétlenül két különböző pontozót hagyjunk ki. Ezért a minimumkiválasztásnál megengedjük az egyenlőséget. A kihagyott pontozókat mindjárt ki is írjuk:

```
VÁLTOZÓ Min, Max, MelyikMin, MelyikMax MINT Egész
CIKLUS I=1-től N-ig
    Max = –∞
    Min = +∞
    CIKLUS J=1-től M-ig
        HA Versenyzők(I).Pontok(J) > Max AKKOR Max = Pontok(J), MelyikMax = J
        ELÁGAZÁS VÉGE
        HA Versenyző(I).Pontok(J) ≤ Min AKKOR Min = Pontok(J), MelyikMin = J
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
    Versenyzők(I).Pontok(MelyikMin) = 0
    Versenyzők(I).Pontok(MelyikMax) = 0
    Ki: I, MelyikMin, MelyikMax
CIKLUS VÉGE
```

Gondoljuk meg, hogy egyforma pontszámok esetén melyik két pontozó pontszámait töröljük! Miért nem alkalmazhatunk a második feltételes elágazás helyett egy *EGYÉBKÉNT HA* ágat?

Az egyes versenyzők összpontszámát ezek után a *Pontok* tömbök elemeinek összeadásával kapjuk meg. Ennek megírását az Olvasóra bízzuk. ☺

A c) feladathoz az összpontszámok alapján csökkenő sorrendbe rendezzük a *Versenyzők* tömböt. A kiírásnál külön jegyezzük a helyezést, amit csak akkor módosítunk, ha az aktuális versenyző pontszáma nem egyezik meg az előző versenyző pontszámával. Ilyenkor ugrik a helyezésszám az aktuális versenyzőnek megfelelő indexre. Az összehasonlítás miatt az első versenyzőt külön kiírjuk:

```
Helyezés = 1
Ki: Helyezés, Versenyzők(1).Sorszám, Versenyzők(1).Összpont
CIKLUS I=2-től N-ig
    HA Versenyzők(I).Összpont < Versenyzők(I–1).Összpont AKOR Helyezés = I
    ELÁGAZÁS VÉGE
    Ki: Helyezés, Versenyzők(I).Sorszám, Versenyzők(I).Összpont
CIKLUS VÉGE
```

A teljes megoldást a *verseny-1* program tartalmazza. Hogyan módosítsuk a programot, ha a helyezések sorszáma a holtverseny után is folyamatosan nő (például több 3. helyezés esetén is a 4. helyezés következik)?

2. Megoldás listákkal

Tömb helyett listát használunk mind a struktúrában az egyes versenyzők pontszámainak a tároláshoz, mind pedig maguknak a versenyzőknek a nyilvántartásához:

```
STRUKTÚRA TVersenyző
    VÁLTOZÓ Sorszám, Összpont MINT Egész
    VÁLTOZÓ Pontok MINT Lista(Elemtípus: Egész)
STRUKTÚRA VÉGE

VÁLTOZÓ Versenyzők MINT Lista(Elemtípus: TVersenyző)
```

A pontszámokat tartalmazó listákat a beolvasásnál hozzuk létre. Az értékadáshoz segédváltozót használunk:

```
VÁLTOZÓ Versenyző MINT TVersenyző
CIKLUS I=1-től N-ig
    Versenyző.Sorszám = I
    Versenyző.Pontok = Új Lista(Elemtípus: Egész)
    Be: Versenyző.Pontok lista elemei
    Versenyzők.Add(Versenyző)
CIKLUS VÉGE
```

Vegyük észre, hogy a ciklusban minden egyes versenyzőhöz új listát hozunk létre!

A legkevesebb és legtöbb pontot adó pontozó sorszámát az 1. megoldáshoz hasonlóan választjuk ki. Pontjaikat töröljük a *Pontok* listából. Azonban ügyelnünk kell a sorrendre! Célszerű először a hátrébb álló (nagyobb indexű) listaelemet törölni:

```
CIKLUS MINDEN Versenyző-re a Versenyzők-ben
    Min, Max, MelyikMin, MelyikMax meghatározása (lásd az 1. megoldásban)
    Egyik = Min(MelyikMin, MelyikMax)
    Másik = Max(MelyikMin, MelyikMax)
    Versenyző.Pontok.Kivesz(Másik indexű elem)
    Versenyző.Pontok.Kivesz(Egyik indexű elem)
    Ki: Versenyző.Sorszám, Egyik+1, Másik+1
CIKLUS VÉGE
```

Gondoljuk meg, hogy a kiírásnál miért adtunk hozzá 1-et az *Egyik* és a *Másik* változó értékéhez!

A pontszámok összegzése a *Pontok* listák elemeinek összeadását jelenti. Az *Összpont* mező módosításához a mellékelt megoldásban segédváltozót használunk.

A rendezés a tömbelemek rendezéséhez hasonló módon történik. A cserénél alkalmazott értékadások során a *Pontok* listákra mutató hivatkozások cserélődnek, így helyes eredményt kapunk:

```
HA Versenyzők(I).Összpont < Versenyzők(J).Összpont AKKOR ' csere
    Versenyző = Versenyzők(I)
    Versenyzők(I) = Versenyzők(J)
    Versenyzők(J) = Versenyző
ELÁGAZÁS VÉGE
```

Célszerű egy ábra segítségével végigkövetni a hivatkozások cseréjét!

A helyezések kiírása pontosan megfelel a tömböknél alkalmazott algoritmusnak.

A teljes megoldást a *verseny-2* program tartalmazza. A megoldásban nem használtuk ki, hogy minden versenyző ugyanannyi pontozótól kapott pontszámot.

Falu

Nemes Tihamér OITV 2011. 2. forduló 2. korcsoport 1. feladat

Ismerjük egy megye települései (falvak, városok) közötti utak hosszát. Zsákfalunak nevezzük azt a falut, ahova csak egyetlen út vezet (és onnan tovább már nem lehet menni, csak visszafelé). A településeket sorszámmal azonosítjuk.

Készíts programot *falu* néven, amely megadja:

a) a zsákfalvak számát;
b) azt a települést, ahova a legtöbb út vezet szomszédos településről;
c) az egymáshoz legközelebbi 2, nem szomszédos települést!

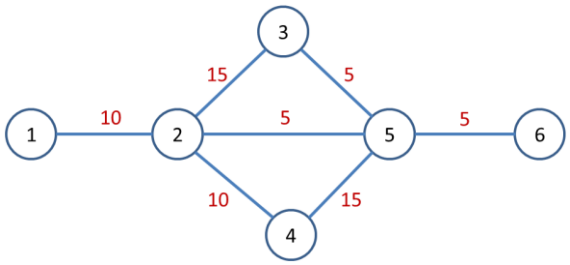
A b) és c) feladatnál több megoldás esetén bármelyik megadható.

A *falu.be* szöveges állomány első sorában a települések ($2 \le N \le 1000$) és az utak száma van ($1 \le M \le 100000$), egy szóközzel elválasztva. A következő *M* sor mindegyikében három egész szám van, egy-egy szóközzel elválasztva: egy-egy út két végpontjának sorszáma és a köztük levő út hossza.

A *falu.ki* szöveges állomány első sorába a zsákfalvak számát; a második sorba a legtöbb utas település sorszámát (ha több van, bármelyik megadható), a harmadikba pedig a két legközelebbi település sorszámát (szóközzel elválasztva, ha több egyforma távolság is van, bármelyik településpár megadható) kell írni!

Példa:

```
falu.be          falu.ki
6 7              2
1 2 10           2
2 3 15           3 6
2 4 10
2 5 5
3 5 5
4 5 15
5 6 5
```



Megjegyzés: a feladat szövege nem tér ki arra, hogy mi a teendő, ha a c) kérdésnek nincsen megoldása (például egyáltalán nincsenek nem szomszédos települések). A versenyen alkalmazott tesztadatoknál nem fordult elő ilyen eset, ezért az alábbiakban nem veszünk figyelembe ilyen bemeneteket.

Megoldás

Ennek a feladatnak az első korcsoportos változatát már bemutattuk a halmazoknál. Most a kérdések megválaszolásához nem elegendő a távolságokat tárolni, szükség van a települések sorszámára is. Mivel egy település két különböző szomszédja lehet tőle ugyanolyan távolságban, ezért halmaz helyett listát használunk.

A feladatnak megfelelő, irányítatlan, súlyozott gráfot statikus tömbben tároljuk. Az *I* indexű tömbelem az *I* sorszámú település szomszédjainak a tulajdonságait tartalmazza. A *Sorszámok* listában a szomszédos települések sorszámait (a kiinduló élek végpontjait), a *Távok* listában pedig ugyanilyen sorrendben ezen települések távolságait jegyezzük fel.

```
STRUKTÚRA TSzomszédok
    VÁLTOZÓ Sorszámok, Távok MINT Lista(Elemtípus: Egész)
STRUKTÚRA VÉGE
VÁLTOZÓ FALUK(1…N) MINT TSzomszédok
VÁLTOZÓ N, M MINT Egész
```

A *Faluk*(2) tömbelem listái például:

Index:	0	1	2	3
Szomszédok:	1	3	4	5
Távok:	10	15	10	5

A továbbiakban tartsuk szem előtt, hogy egy szomszéd indexe a *Sorszámok* listában nem egyezik meg magának a szomszédnak a sorszámával (azonosítójával)! A *Faluk*(2).*Szomszédok* listában a 4-es település indexe például 2.

A beolvasásnál ügyeljünk arra, hogy egy utat mindkét településhez felvegyünk a listákba (irányítatlan gráf)! Még a beolvasás előtt a listákat is létrehozzuk minden településnél.

```
VÁLTOZÓ Falu1, Falu2, Táv MINT Egész
Be: N, M
CIKLUS I=1-től N-ig
    Faluk(I).Sorszámok = Új Lista(Elemtípus: Egész)
    Faluk(I).Távolságok = Új Lista(Elemtípus: Egész)
CIKLUS VÉGE
CIKLUS I=1-től M-ig
    Be: Falu1, Falu2, Táv
    Faluk(Falu1).Sorszámok.Add(Falu2)
    Faluk(Falu1).Távok.Add(Táv)
    Faluk(Falu2).Sorszámok.Add(Falu1)
    Faluk(Falu2).Távok.Add(Táv)
CIKLUS VÉGE
```

Azok a települések számítanak zsákfalunak, amelyek listái csak egyetlen elemből állnak (egyetlen szomszédjuk van). A legtöbb úttal (szomszéddal) pedig az a település rendelkezik, amelynél a legnagyobb a *Sorszámok* (vagy a *Távok*) lista elemszáma. Az algoritmusok megírását az Olvasóra bízzuk. ☺

Lássuk az egymáshoz legközelebbi, két, nem szomszédos település kiválasztását! Ehhez először meghatározzuk minden településre a legközelebbi második szomszédjának a sorszámát, illetve távolságát. Az adatokat egy *TSzomszédok* típusú változóban tároljuk. Így nincs szükségünk újabb típus definiálására. Mivel a listaelemek indexelése általában 0-val kezdődik, kezdetben egy fiktív elemet adunk a listákhoz. Ezzel elérjük, hogy az I-edik listaelem az I-edik település legközelebbi második szomszédjának az adatait tartalmazza.

```
VÁLTOZÓ MinTáv MINT TSzomszédok
MinTáv.Sorszámok = Új Lista(Elemtípus: Egész)
MinTáv.Sorszámok.Add(0)
MinTáv.Távok = Új Lista(Elemtípus: Egész)
MinTáv.Távok.Add(+∞)
```

Az algoritmus további részéhez segédváltozókat deklarálunk. Így tesszük áttekinthetőbbé és érthetőbbé az eljárást.

```
VÁLTOZÓ Szomszéd, SzomszédTáv ' a közvetlen szomszéd sorszáma és távolsága
VÁLTOZÓ Szomszéd2, Szomszéd2Táv ' a második szomszéd sorszáma és távolsága
```

A *MinTáv* listáinak a feltöltéséhez sorra vesszük az egyes településeket. Minden településnél sorra vesszük a szomszédokat. Minden szomszédnál sorra vesszük ezek szomszédjait, azaz az aktuális település második szomszédjait. Meghatározzuk a második szomszédok távolságát az aktuális településtől, és ezek közül a legközelebbit adjuk hozzá a *MinTáv* listáihoz.

Közben az *I, J, K* ciklusváltozókkal a *Sorszámok* listák elemeit indexeljük. A megfelelő elemek értékét (a település azonosítóját, illetve távolságát) a segédváltozókban tároljuk.

Ügyeljünk arra, hogy a második szomszédok közé ne vegyük fel sem az aktuális települést (bár a szomszédjának a szomszédja), sem pedig ennek közvetlen szomszédjait! A feladat gráfjában például a 2. településnek az 5. település a második szomszédja (a 3. szomszéd szomszédja), ugyanakkor közvetlenül is a szomszédja.

Az algoritmus:

```
VÁLTOZÓ Min, Melyik MINT Egész
CIKLUS I=1-től N-ig ' ciklus a településekre
    Min = +∞
    Melyik = 0
    CIKLUS J=0-tól Faluk(I).Sorszámok.Elemszám–1-ig ' ciklus a szomszédokra
        Szomszéd = Faluk(I).Sorszámok(J) ' a szomszéd indexe a Faluk tömbben
        SzomszédTáv = Faluk(I).Távok(J) ' a szomszéd távolsága
        CIKLUS K=0-tól Faluk(Szomszéd).Sorszámok.Elemszám–1-ig
            ' ciklus a második szomszédokra
            Szomszéd2 = Faluk(Szomszéd).Sorszámok(K)
                ' a második szomszéd indexe a Faluk tömbben
            HA Szomszéd2 == I AKKOR
                ' a második szomszéd megegyezik az aktuális településsel, kihagyjuk
                SZÁMLÁLÓS CIKLUS FOLYTATÁS[23]
            ELÁGAZÁS VÉGE
            HA Faluk(I).Sorszámok.Tartalmazza(Szomszéd2) AKKOR
                ' a második szomszéd közvetlen szomszédja az I. településnek
                SZÁMLÁLÓS CIKLUS FOLYTATÁS
            ELÁGAZÁS VÉGE
            Szomszéd2Táv = Faluk(Szomszéd).Távok(K)
                ' a második szomszéd távolsága
            HA Szomszéd2Táv + SzomszédTáv < Min AKKOR
                Min = Szomszéd2Táv + SzomszédTáv
                Melyik = Szomszéd2
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
    CIKLUS VÉGE
    MinTáv.Távok.Add(Min)
    MinTáv.Sorszámok.Add(Melyik)
CIKLUS VÉGE
```

A c) kérdésre a választ a *MinTáv.Távok* lista minimuma adja meg:

```
Min = +∞
CIKLUS I=1-től N-ig ' a 0 indexű, fiktív elemet nem vesszük figyelembe
    HA MinTáv.Távok(I) < Min AKKOR
        Min = MinTáv.Távok(I)
        Melyik = I
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
Ki: Melyik, MinTáv.Sorszámok(Melyik)
```

[23] *continue, Continue For*

A teljes megoldást a *falu* program tartalmazza.

Ez a feladat kiváló lehetőséget nyújt a struktúrák és listák gyakorlására. Alaposan gondoljuk át minden utasítását! Figyeljük meg a ciklusváltozók szerepét, kezdő- és végértékét!

Javasoljuk az Olvasónak, hogy végezze el a következő módosításokat.

- Küszöbölje ki a *Szomszéd*, *SzomszédTáv*, *Szomszéd2*, *Szomszéd2Táv* segédváltozókat!
- A számlálós ciklusok helyett alkalmazzon iterátoros (*FOR EACH*) ciklusokat!
- Egészítse ki a programot úgy, hogy jól működjön akkor is, ha nincs megoldás!
- Kezelje azt az esetet is, amikor két falut több út köt össze (közvetlenül)!
- Alakítsa át az adatszerkezetet! A *TSzomszédok* struktúra helyett definiálja a *TSzomszéd* struktúrát, amely egyetlen szomszéd adatait tartalmazza (*Sorszám*, *Távolság*)! A *Faluk* tömb elemei legyenek *TSzomszéd* típusú elemeket tartalmazó listák (szomszédsági listák)!

További feladatok a struktúrákra

Az alábbi táblázatban olyan feladatok szerepelnek, melyek megoldása nem igényel különösebben bonyolult algoritmust. Így a struktúrák alkalmazására tudunk koncentrálni.

OITV 1999-3f2	Hálózati felügyelőprogram
OITV 2001-3f1	Bob
OITV 2003-3f1	Kártya
OITV 2009-2f1	Törtek
OITV 2010-2f1	Kártya
OITV 2010-2f2	Játék (szimuláció)
OITV 2011-3f1	Lövészverseny
OITV 2013-3f2	Épület
OKTV 2010-2f3	Játék (szimuláció)
OKTV 2013-3f3	Ház
OKTV 2014-2f3	Játéktábla

6. Lambdakifejezések

Kapcsolódó leckék:
Programozási ismeretek (Műszaki Kiadó, 2011)
24. Elemi algoritmusok megvalósítása tömbmetódusokkal
33. Rekordok és struktúrák
42. Rendezés osztálymetódussal

Ebben a leckében mondatszerű leírások helyett Visual Basic forráskódokkal szemléltetjük a lambdakifejezések használatát, de a mintaprogramok C++ nyelven is letölthetők a könyv webhelyéről (www.zmgzeg.sulinet.hu/programozas)

Függvényargumentumok

A *Programozási ismeretek* tankönyv 24. leckéjében bemutattuk, hogyan lehet predikátumfüggvények alkalmazásával kibővíteni a tömbmetódusok használhatóságát. Ezekre a függvényekre – többnyire – akkor van szükségünk, amikor a számlálás vagy a lineáris keresés metódusát nem egyszerűen a kollekció elemeire, hanem egy, az elem alapján felírt logikai kifejezésre alkalmazzuk. Hasonló szerepet töltenek be a transzformációs (vagy szelektor-) függvények például az összegezés, illetve a maximum- és minimumkiválasztás metódusaiban. A 42. leckében láthattuk azt is, hogyan készítsünk komparáló függvényt a rendezési metódusokhoz.

Az említett példák közös jellemzője, hogy függvény szerepel a metódusok argumentumai között. A függvényre memóriacíme alapján hivatkozunk, melyet az *AddressOf* operátorral határozunk meg, például:

```
Dim Magasság() As Integer = {…}
PárosElemekSzáma = Magasság.Count(AddressOf Páros)
```

A metódus működéséhez definiálnunk kell a *Páros* függvényt, amely logikai értéket ad vissza:

```
Function Páros(Elem As Integer) As Boolean
  Páros = (Elem Mod 2 = 0)
End Function
```

A program a *Magasság* tömb összes elemére meghívja a függvényt (azaz a függvénynek átadja az egyes elemek értékét). A *Count* metódus a *True* visszatérési értéket szolgáltató (azaz a páros) elemeket számolja meg.

Ebben az esetben a függvénytörzs csak egy egyszerű értékadást tartalmaz. A forráskód azonban jóval hosszabb, mondhatni sok „felesleges” körítés tartozik hozzá. Ez a helyzet nagyon gyakran előfordul a fent említett többi metódusnál is.

Egysoros lambdakifejezések

Lambdakifejezések használatával nagymértékben egyszerűsíthetjük a forráskódot. Lambdakifejezésnek nevezzük azt a függvényt vagy eljárást, melyet – a megfelelő szabályok betartásával – azonosító nélkül, egyszerűsített szintaxissal írunk be a forráskódba. A lambdakifejezés egy vagy több sorból állhat.

Itt nem tárgyaljuk részletesen a lambdakifejezésekkel kapcsolatos tudnivalókat. A továbbiakban csak néhány egyszerű alkalmazásukat mutatjuk be.

A páros elemek számlálása lambdakifejezéssel megvalósítva:

```
PárosElemekSzáma = Magasság.Count( _
                       Function(Elem As Integer) Elem Mod 2 = 0)
```

Mint látjuk, külön függvény definiálása helyett a függvényt a metódus argumentumának helyére írjuk be. Figyeljük meg az egysoros lambdakifejezések használatára vonatkozó alapvető szabályokat:

- nem írunk azonosítót a *Function*/*Sub* kulcsszó után;
- nem adjuk meg a függvény visszatérési értékének típusát;
- a függvény törzse egyetlen kifejezésből áll, melynek értéke lesz a függvény visszatérési értéke;
- nem rendeljük hozzá értékadó utasítással a visszatérési értéket a függvénynévhez (illetve nem használjuk a *Return* utasítást);
- az eljárás törzse egyetlen utasításból áll;
- a definíciót nem zárjuk le az *End Function*/*Sub* kulcsszavakkal.

A kódot tovább egyszerűsíthetjük a paraméter típusának elhagyásával:

```
PárosElemekSzáma = Magasság.Count( _
                                   Function(Elem) Elem Mod 2 = 0)
```

Ekkor a típust a fordítóprogram állapítja meg.[24] Több paraméter esetén vagy mindegyiknek meg kell adni a típusát, vagy pedig egyetlen paraméter típusát sem adhatjuk meg. Bonyolultabb esetben a program nem tud megbirkózni a típusok illesztésével. Ebben az esetben a többsoros lambdakifejezések formáját alkalmazzuk (lásd később).

Ha ugyanazt az alprogramot több helyen is hívjuk, akkor a kifejezést **lokális** (!) „változóként” definiálhatjuk, a fenti szabályok betartásával:

```
Dim PárostSzámlál = Function(Elem As Integer) Elem Mod 2 = 0
```

A *Count* metódus hívása ebben az esetben:

```
PárosElemekSzáma = Magasság.Count(PárostSzámlál)
```

Figyeljük meg, hogy az argumentumnál nem írtuk ki az *AddressOf* operátort!

A *Programozási ismeretek* tankönyv 24. leckéjének 4. gyakorlatában szereplő transzformációs függvény a magasságok átlagától való átlagos eltérést határozta meg. Ugyanez lambdakifejezéssel:

```
Átlag = Magasság.Average()
ÁtlagosEltérés = Magasság.Average( _
                         Function(Elem) Math.Abs(Elem - Átlag))
```

Függvény helyett eljárást például a *ForEach* metódus hívásakor alkalmazunk. A *ForEach* a kollekció összes elemére végrehajtja az eljárásban szereplő utasítást:

```
Array.ForEach(Tömb, Sub(Elem) Console.WriteLine(Elem))
```

Az argumentumként hívott eljárást – nemcsak a lambdakifejezéseknél – gyakran akcióeljárásnak nevezzük. Ne feledjük el azonban, hogy **a metódusok érték szerint adják át az alprogramoknak az argumentumokat, tehát akcióeljárásban az érték típusú tömbelemek módosítására nincs lehetőségünk!**

[24] Bonyolultabb esetben előfordulhat a késői kötés alkalmazása is.

> Ha C++-ban a függvény paraméterét hivatkozás típusú változóként deklaráljuk, akkor módosíthatjuk a tömbelemeket (lásd az *Egysoros.cpp* példaprogramot).

Lambdakifejezések alkalmazásával elérjük, hogy nincs szükség külön alprogram (függvény vagy eljárás) definiálására, illetve a függvényértéket szolgáltató kifejezés pontosan ott szerepel a forráskódban, ahol szükség van rá. Így áttekinthetőbbé tesszük a programot. Lambdakifejezések természetesen nemcsak a tömbmetódusoknál, hanem a többi kollekció metódusainál is alkalmazhatók.

Az If függvény

Előfordulhat, hogy egy feltételes elágazás miatt nem tudunk egysoros lambdakifejezést készíteni még akkor sem, ha az elágazás viszonylag egyszerű lenne. Többek között ezért vezették be a Visual Basic-ben is az *If* függvényt. A függvény szintaxisa:

```
If(feltétel, kifejezés1, kifejezés2)
```

A függvényérték a *kifejezés1*-gyel egyenlő, ha a feltétel igaz, illetve a *kifejezés2*-vel, ha a feltétel hamis. Mint látjuk az *If* függvény megfelel az Excel *HA* függvényének (továbbá a C++ és a C# feltételes operátorának).

Példaként határozzuk meg az egész számokat tartalmazó *Számok* tömb páros elemeinek négyzetösszegét (a *Sum* függvényt példánymetódusként kell hívni):

```
Dim Számok() As Integer = {…}
Dim PárosNégyzetÖsszeg As Integer
PárosNégyzetÖsszeg = Számok.Sum( _
                  Function(Elem) If(Elem Mod 2 = 0, Elem ^ 2, 0))
```

Az *If* függvények egymásba is ágyazhatók.

Megjegyezzük, hogy a Visual Basic régebbi, *IIf* függvénye a visszatérési érték meghatározása előtt mindkét ágat kiértékeli, így a következő függvényhívás futási hibához vezet:

```
Dim X As Integer = 0
WriteLine(IIf(X = 0, 0, 1 \ X))
```

Az *If* függvény azonban a feltétel logikai értéke alapján csakis az egyik ágat értékeli ki, tehát nem okoz futási hibát.

Többsoros lambdakifejezések

Előfordulhat, hogy az alprogram több utasítást tartalmaz. A *Programozási ismeretek* tankönyv 42. leckéjének 5. gyakorlatában például a neveket tartalmazó tömb rendezését a nevek hossza alapján végeztük el:

```
Array.Sort(Név, AddressOf Hasonlít)
```

Ehhez komparáló függvényt használtunk, melynek definíciója:

```
Function Hasonlít(Név1 As String, Név2 As String) As Integer
  Dim Hossz1, Hossz2 As Integer
  Hossz1 = Név1.Length : Hossz2 = Név2.Length
  If Hossz1 < Hossz2 Then
    Hasonlít = -1
  ElseIf Hossz1 = Hossz2 Then
    Hasonlít = 0
```

```
  Else
    Hasonlít = 1
  End If
End Function
```

Többsoros lambdakifejezés alkalmazásával a megoldás:

```
Dim Hasonlít = _
            Function(Név1 As String, Név2 As String) As Integer
              Dim Hossz1, Hossz2 As Integer
              Hossz1 = Név1.Length : Hossz2 = Név2.Length
              If Hossz1 < Hossz2 Then
                Return -1
              ElseIf Hossz1 = Hossz2 Then
                Return 0
              Else
                Return 1
              End If
            End Function
…
Array.Sort(Név, Hasonlít)
```

Többsoros lambdakifejezés esetén tehát az alprogramot egy lokális (!) változóhoz hasonló módon definiáljuk. Az alprogram törzsét az *End Function*/*Sub* kulcsszavakkal zárjuk.

Megjegyezzük, hogy az előző példában szereplő, *Hasonlít* függvényt Visual Basic-ben egyszerűbben is megadhatjuk. A komparáló függvény visszatérési értéke ugyanis –1 helyett tetszőleges negatív, +1 helyett pedig tetszőleges pozitív szám lehet. Így a rendezés:

```
Dim Hasonlít = _
            Function(Név1 As String, Név2 As String) As Integer
              Return Név1.Length - Név2.Length
            End Function
Array.Sort(Név, Hasonlít)
```

Vagy még egyszerűbben:

```
Array.Sort(Név, Function(Név1, Név2) Név1.Length – Név2.Length)
```

A lambdakifejezések a matematikában használatos lambda-kalkulusról kapták a nevüket, bár csak távoli rokonságban állnak vele. A Visual Basic-ben (pontosabban a .NET-ben) lehetővé teszik a LINQ kifejezések egyszerűsítését – valójában ez volt bevezetésük oka. Ezen kapcsolatok részleteire azonban itt nem térünk ki. Az érdeklődő Olvasónak Timothy Ng cikkét ajánljuk az MSDN Magazine 2007. szeptemberi számából (lásd az irodalomjegyzéket). A lambdakifejezések gyakorlati alkalmazását a következő leckében mutatjuk be.

7. Algoritmusokat megvalósító metódusok

Kapcsolódó leckék:
Programozási ismeretek (Műszaki Kiadó, 2011)
24. Elemi algoritmusok megvalósítása tömbmetódusokkal
33. Rekordok és struktúrák
42. Rendezés osztálymetódussal
Programozási ismeretek haladóknak (Műszaki Kiadó, 2012)
19. A dinamikus tömb
34. A mohó algoritmus

A kollekciók metódusainak egy része elemi algoritmusokat, illetve rendezést valósít meg. Ezek segítségével nagymértékben egyszerűsíthetjük a kódot, lerövidíthetjük a program megírásához szükséges időt.

Az alábbiakban néhány versenyfeladat megoldásával szemléltetjük a metódusok használatát.

A metódusok használata

A *Programozási ismeretek* tankönyvben csak a statikus tömbök metódusainak alkalmazását mutattuk be. A legtöbb tömbmetódust megosztott (shared) metódusként, az osztály nevével minősítve kell hívni. Más kollekciók esetén (halmaz, lista stb.) általában példánymetódusként, azaz az objektum nevével minősítve hívhatók. A statikus tömbök metódusait a *tombmetodusok*, a dinamikus tömbök (Visual Basic: listák) metódusait a *listametodusok* program szemlélteti. Javasoljuk az Olvasónak ezek áttekintését. A többi kollekció hasonló metódusokkal rendelkezik.

> C++-ban a tömbmetódusok (*tombmetodusok*) alkalmazása mellett bemutatjuk a vector (*vektormetodusok*) és a list (*listametodusok*) adattípus metódusainak használatát.

Ügyeljünk arra, hogy a metódusok argumentumaiban (beleértve a lambdakifejezéseket is)

a) nem tudunk hivatkozni a kollekció más elemeire (indexeikre);
b) nem módosíthatjuk magát az elemet (érték szerinti paraméterátadás)!

Összetett típusú elemek esetén az összehasonlítást igénylő metódusoknál alkalmazzunk komparáló függvényt![25] Nem kívánt eredményre vezethet, ha egy lambdakifejezésben (például akcióeljárásban) módosítjuk magát a kollekciót (elem törlése, beillesztése).

Figyeljünk arra, hogy a *Find, FindAll* metódusoknál a keresett tulajdonságot szerepeltetjük a predikátumfüggvényben! Ha viszont ciklussal keresünk, akkor a tulajdonság negáltját kell megadni ismétlési feltételként.

[25] Implementálhatjuk az *IComparable* interfészt is (lásd: *Programozási ismeretek haladóknak*, 72. oldal).

> Visual Basic-ben a halmaz/rendezett halmaz kollekció nem rendelkezik *IndexOf*, *FindIndex* vagy *Find* metódussal. Ez utóbbi helyett használjuk a *First*(*predikátumfüggvény*) vagy *FirstOrDefault*(*predikátumfüggvény*) metódust. Mindkét függvény visszatérési értéke maga a halmazelem (nem pedig az indexe). A *First* futási hibát okoz, ha nem talál megfelelő elemet, a *FirstOrDefault* ilyenkor a típus alapértelmezett értékét adja vissza.

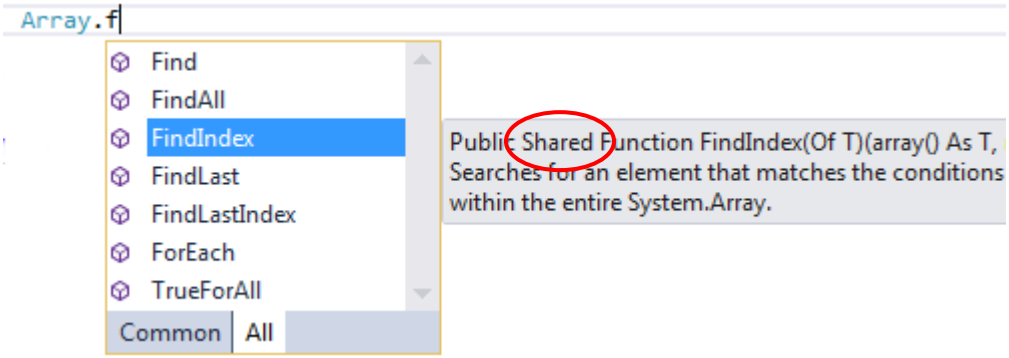
A metódusok teljes listáját a Visual Basic súgójában, illetve a www.cplusplus.com webhelyen találjuk. Ez utóbbinál az egyes kollekciók és metódusok leírását a *Reference*/*Containers* hivatkozás követésével érjük el.

A Visual Basic online súgójában a kollekció típusa mellett célszerű feltüntetni a *class visual basic* kulcsszavakat is, például: *list class visual basic*. Így gyorsabban jutunk eredményre. A találati listából válasszuk a *System.Collections.Generic* (típusos kollekció) jelölésű hivatkozást!

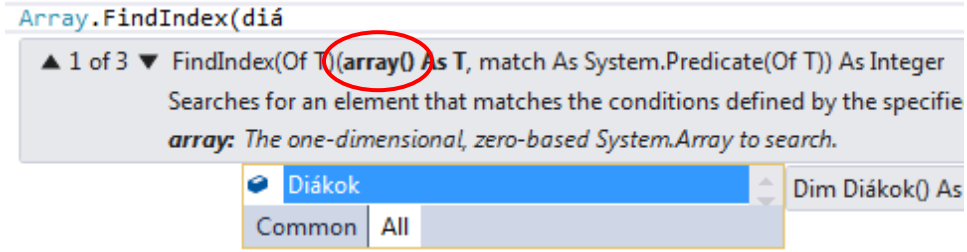
A statikus tömbök megosztott (shared) metódusait egy S betű jelöli a metódusok listájában. Hívásuk:

```
Array.Metódusnév(tömbnév, további argumentumok)
```

Az intelligens súgó szintén jelzi a megosztott besorolást, illetve első argumentumként a tömb (array) megadását kéri.



Megosztott (Shared) metódus jelzése a Visual Basic intelligens súgójában



A tömböt (array) a megosztott metódus első argumentumaként kell megadni

Települések

Nemes Tihamér OITV 2014. 1. forduló 1. korcsoport 4. feladat

Ismerjük Magyarország összes településének távolságát Kukutyintól és Piripócstól. Készíts programot *telepules* néven, amely megadja

a) a Kukutyintól legmesszebb levő település sorszámát (ha több megoldás van, akkor közülük a legkisebb sorszámút);
b) azon települések számát, amelyek közelebb vannak Kukutyinhoz, mint Piripócshoz;
c) a Piripócshoz 100 kilométernél közelebbi azon települések sorszámát, amelyek Kukutyintól legalább 100 kilométerre vannak (sorszám szerint növekvő sorrendben)!

A bemenet első sorában a települések száma (≥1), alatta soronként egy-egy település távolsága Kukutyintól, illetve Piripócstól (egy szóközzel elválasztva). A kimenetre kell kiírni a fenti kérdésekre, feladatokra adott válaszokat.

Példa:

Input (billentyűzet)	*Output (képernyő)*	
`6`	`4`	az 1. részfeladat válasza
`42 15`	`3`	a 2. részfeladat válasza
`110 20`	`2 6`	a 3. részfeladat válasza
`125 160`		
`166 180`		
`42 100`		
`110 39`		

Megoldás

A metódusok hatékonyságát először egy nagyon egyszerű feladaton keresztül mutatjuk be.

A települések adatait listában tároljuk, mert a metódushívások egyszerűbbek, mint a statikus tömbnél. Ráadásul nem kell foglalkoznunk a feltöltetlen (például a 0 indexű) tömbelemekkel.

A távolságokat struktúrában tároljuk, mert két lista (tömb) elemeit nem tudjuk kezelni (például összehasonlítani) a metódusokkal. A távolságok mellett a települések sorszámát is felvesszük a struktúra mezői közé.

```
STRUKTÚRA TTelepülés
    VÁLTOZÓ Sorszám, Kukutyin, Piripócs MINT Egész
STRUKTÚRA VÉGE
VÁLTOZÓ Települések MINT Lista(Elemtípus: TTelepülés)
VÁLTOZÓ N MINT Egész

Be: N
Be: a Települések lista elemei
```

Az a) kérdésre maximumkiválasztással válaszolunk.

```
VÁLTOZÓ Max, Melyik MINT Egész
Max = Települések.Max(FÜGGVÉNY(Elem) Elem.Kukutyin)
Melyik = Települések.IndexKeres(FÜGGVÉNY(Elem) Elem.Kukutyin == Max)
Ki: Települések(Melyik).Sorszám
```

A *Max* metódus meghatározza a *Kukutyintól* mért távolságok maximumát, az *IndexKeres* metódus pedig azt az indexet, amelynek megfelelő listaelem *Max* távolságra van *Kukutyintól*. Ennek sorszámát írjuk ki.

Egyszerűbb programhoz jutunk, ha az index helyett magát a *TTelepülés* típusú elemet keressük meg:

```
VÁLTOZÓ Település MINT TTelepülés
Település = Települések.Keres(FÜGGVÉNY(Elem) Elem.Kukutyin == Max)
Ki: Település.Sorszám
```

A b) kérdésre a megszámolást végző metódus hívásával válaszolunk:

```
VÁLTOZÓ Db MINT Egész
Db = Települések.Megszámol(FÜGGVÉNY(Elem) Elem.Kukutyin < Elem.Piripócs)
Ki: Db
```

A metódushívás visszatérési értékét egyből ki is írhattuk volna.

> Visual Basic-ben (illetve a .NET-ben) a lista *Count* metódusával csak a lista elemszámát határozhatjuk meg. Predikátumfüggvényt csak a *LongCount* metódusnál alkalmazhatunk.

A c) kérdésnek megfelelő településeket a kiválogatást végző metódussal gyűjthetjük össze. Ha a feladat további részében még szükség lenne rájuk, akkor egy újabb listában tároljuk a megfelelő sorszámokat:

```
VÁLTOZÓ Lista MINT Lista(Elemtípus: TTelepülés)
Lista = Települések.MindetMegkeres(FÜGGVÉNY(Elem)
                                   Elem.Piripócs < 100 ÉS Elem.Kukutyin ≥ 100)
Lista.MindenElemre(ELJÁRÁS(Elem) Ki: Elem.Sorszám)
```

Ha csak a sorszámok kiírására van szükség, akkor nem készítünk újabb listát:

```
Települések.MindenElemre(ELJÁRÁS(Elem)
            Ki: HA(Elem.Piripócs < 100 ÉS Elem.Kukutyin ≥ 100, Elem.Sorszám, ""))
```

Itt emlékeztetünk a *HA* (*If*) függvény működésére (lásd az előző leckében).

A teljes megoldást a *telepules* program tartalmazza.

Írjuk meg a programot az elemi algoritmusok szokásos (ciklussal történő) kódolásával is!

Vitorlás

Nemes Tihamér OITV 2008. 2. forduló 2. korcsoport 4. feladat

Egy vitorlásversenyen N futamot rendeznek, melyek mindegyikében az első K helyezettet értékelik. Az első helyezett K, a második K–1, a harmadik K–2, … pontot kap. Az összetett versenyben mindenkinek az L legmagasabb pontszámát veszik figyelembe. A versenyen M versenyző vett részt, 1 és M közötti sorszámukkal azonosítjuk őket. A helyezést ezen pontszámok összege alapján csökkenő sorrendben határozzák meg. Ha két versenyzőnek ugyanannyi pontja lenne, akkor az kerül előbbre, akinek több első helyezése van; ha ugyanannyi első helyezésük van, akkor a második helyezések száma dönt és így tovább. Ha két versenyző ebben is egyforma, akkor a sorrendjük tetszőleges lehet.

Készíts programot *vitorlas* néven, amely megadja azok névsorát, akik mindegyik futamon az első K hely valamelyikén végeztek, valamint add meg a verseny végeredményét!

A *vitorlas.be* szöveges állomány első sorában N ($0<N\leq100$), K ($3\leq K\leq10$), L ($2\leq L\leq N$) és M ($1\leq M\leq1000$) értéke van, egyetlen szóközzel elválasztva. Ezt követi N sor az egyes futamok sorrendjével. Minden sorban K versenyző sorszáma van, helyezésük szerint csökkenő sorrendben, egy-egy szóközzel elválasztva.

A *vitorlas.ki* állomány első sorába azon versenyzők X számát kell írni, ahányan mindegyik futamban az első K helyezés valamelyikét érték el, a következő sorba pedig az X darab ilyen versenyzők sorszámát, egy-egy szóközzel elválasztva, tetszőleges sorrendben! A harmadik az összes, pontot szerzett versenyző Y számát kell írni, a következő Y sorba pedig ezek sorszámát és összpontszámát, pontszám szerint csökkenő sorrendben!

Példa:

```
vitorlas.be
5 5 3 15
1 2 3 4 5
2 4 6 8 3
3 6 9 12 4
5 4 3 2 1
1 4 5 2 3
```

```
vitorlas.ki
2
3 4
9
4 12
1 11
2 11
3 11
5 9
6 7
9 3
8 2
12 2
```

de az utolsó két helyen jó lenne:

```
12 2
8 2
```

Megoldás

Egy versenyző adatait struktúrában, a versenyzőket pedig listában tároljuk:

```
STURKTÚRA TVersenyző
    VÁLTOZÓ Sorszám, Összpontszám MINT Egész
    VÁLTOZÓ Helyezések MINT Lista(Elemtípus: Egész)
STRUKTÚRA VÉGE
VÁLTOZÓ Versenyzők MINT Lista(Elemtípus: TVersenyző)
VÁLTOZÓ N, K, L, M MINT Egész

Be: N, K, L, M
```

Egy-egy futam eredményének beolvasása után megkeressük, hogy az egyes helyezéseket elért versenyzők szerepelnek-e már a versenyzők listájában. Ha nem, akkor hozzáadjuk a versenyzőt a *Versenyzők* listához. Majd hozzáadjuk a helyezést is a *Helyezések* listájához:

```
Be: egy futam eredménye (a versenyzők sorszáma a helyezések sorrendjében)
CIKLUS Helyezés=1-től K-ig
    SorszámBe = a Helyezés-t elért versenyző sorszáma
    Melyik = Versenyzők.IndexKeres(Függvény(Elem) Elem.Sorszám == SorszámBe)
    HA nincs ilyen versenyző AKKOR ' létrehozzuk
        Melyik = Versenyzők.Elemszám ' az új listaelem lesz a versenyző
        Versenyző.Sorszám = SorszámBe
        Versenyző.Helyezések = Új Lista(Elemtípus: Egész)
        Versenyzők.Add(Új TVersenyző(Sorszám = SorszámBe,
                                     Helyezések = Új Lista(Elemtípus: Egész)))
    ELÁGAZÁS VÉGE
    Versenyzők(Melyik).Helyezések.Add(Helyezés)
CIKLUS VÉGE
```

A fenti ciklust megismételjük minden futamra, amely a *vitorlas.be* fájlban szerepel.

A programnak szinte a leghosszabb része volt a beolvasás. A listametódusok felhasználásával nagyon könnyen válaszolhatunk a kérdésekre.

Azok a versenyzők értek el minden versenyen legalább *K*-adik helyezést, akiknél a *Helyezések* listája pontosan *N* elemet tartalmaz. Számuk és sorszámaik:

```
Ki: Versenyzők.Elemszám(Függvény(Elem) Elem.Helyezések.Elemszám == N)
Versenyzők.MindenElemre(
      Eljárás(Elem) HA Elem.Helyezések.Elemszám == N AKKOR Ki: Elem.Sorszám))
```

A versenyzők számát (*Y*) a *Versenyzők* lista elemszáma adja meg:

```
Ki: Versenyzők.Elemszám
```

Az összpontszámok meghatározásához rendezzük a helyezéseket nagyság szerint növekvő sorrendbe, majd legfeljebb *L* helyezést hagyunk meg bennük (a rendezés miatt a legjobbakat):

```
Versenyzők.MindenElemre(Eljárás(Elem) Elem.Helyezések.Rendez())
Versenyzők.MindenElemre(
     Eljárás(Elem) HA Elem.Helyezések.Elemszám > L AKKOR
        Elem.Helyezések.Kivesz(L-től kezdve Elem.Helyezések.Elemszám–L darabot))
```

Mint tudjuk, a *MindenElemre* metódussal maguk a kollekcióelemek nem módosíthatók, és az akcióeljárás is érték szerint kapja meg az argumentumot. Az *Elem.Helyezések* mező azonban csak egy hivatkozást tartalmaz a megfelelő listára. Magát a hivatkozást nem is változtatjuk meg. A helyezések listáját így a hivatkozás birtokában tudjuk rendezni, illetve belőle elemeket törölni!

A törlés következében az összpontszám meghatározásához már minden megmaradt elemet felhasználunk a *Helyezések* listában. Sajnos az értékadás miatt nem alkalmazhatjuk a *MindenElemre* metódust:

```
CIKLUS I=0-tól Versenyzők.Elemszám–1-ig
   Versenyző = Versenyzők(I) ' a listaelemet segédváltozóval módosítjuk
   Versenyző.Összpontszám = Versenyző.Helyezések.Összead(
                                                  Függvény(Elem) K – Elem + 1)
   Versenyzők(I) = Versenyző
CIKLUS VÉGE
```

Gondoljuk meg, miért a *K–Elem*+1 értékeit adtuk össze a metódussal!

Az összpontszámok kiírása előtt rendezzük a versenyzők listáját! A rendezéshez komparáló függvényt definiálunk. Ha két versenyző összpontszáma különbözik, akkor a nagyobb összpontszámú áll elöl. Ha az összpontszámok megegyeznek, akkor megkeressük a *Helyezések* listáiban az első különböző párt, és ezek döntik el a sorendet.

```
FÜGGVÉNY Hasonlít(V1, V2 MINT TVersenyző) MINT Egész
   HA V1.Összpontszám ≠ V2.Összpontszám AKKOR
      Visszatérési érték: V2.Összpontszám – V1.Összpontszám ' csökkenő sorrend
   ELÁGAZÁS VÉGE
   Meddig = Minimum(V1.Helyezések.Elemszám, V2.Helyezések.Elemszám)
   Hely = 0
   CIKLUS AMÍG Hely < Meddig ÉS# V1.Helyezések(Hely) == V2.Helyezések(Hely)
      Hely = Hely + 1
   CIKLUS VÉGE
   HA Hely < Meddig AKKOR ' volt eltérő sorszámú helyezés
      Visszatérési érték: V1.Helyezések(Hely) – V2.Helyezések(Hely)
      ' (növekvő sorrend)
   EGYÉBKÉNT ' egyformának tekintjük őket
      Visszatérési érték: 0
   ELÁGAZÁS VÉGE
FÜGGVÉNY VÉGE
```

A függvény definíciójában kihasználtuk, hogy a legtöbb programozási nyelvben a visszatérési érték megadásával (*Return*) ki is lépünk a függvényből. A minimális elemszám meghatározása pedig egyszerűsítette a ciklus kilépési feltételét. Enélkül a program minden egyes ismétlésnél mindkét lista elemszámát lekérte volna.

Ezek után már csak a rendezés és a kiírás marad hátra:

```
Versenyzők.Rendez(a Hasonlít függvény szerint)
Versenyzők.MindenElemre(Eljárás(Elem) Ki: Elem.Sorszám, Elem.Összpontszám)
```

A teljes megoldást a *vitorlas* program tartalmazza. Figyeljük meg, hogy a beolvasásnál szereplő cikluson kívül csak egyetlen rövid és egyszerű ciklust kellett megírnunk egy meglehetősen összetett feladat algoritmusában!

Olimpiai láng

Nemes Tihamér OITV 2009. 2. forduló 2. korcsoport 4. feladat

Az olimpiai lángot egy kiindulási városból a cél városba kell eljuttatni. A két város távolsága *K* kilométer. A szervezők meghirdették, hogy olyan futók jelentkezését várják, akik pontosan *H* kilométert futnak az olimpiai lánggal. Sok futó jelentkezett, mindegyik megadta, hogy hányadik kilométertől vállalja a futást. A szervezők ki akarják választani a jelentkezők közül a lehető legkevesebb futót, akik végigviszik a lángot. Ha egy futó az *x* kilométertől fut, akkor minden olyan futó át tudja venni tőle a lángot, aki olyan *z* kilométertől vállalja a futást, hogy $z \le x + H$.

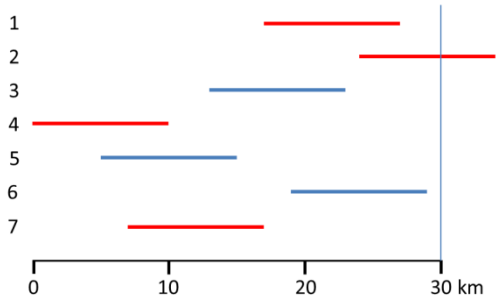
Írj programot *lang* néven, amely kiszámítja, hogy legkevesebb hány futó kell ahhoz, hogy a láng eljusson a cél városig!

A *lang.be* szöveges állomány első sorában három egész szám van, a két város távolsága ($10 \le K \le 10000$), a jelentkezett futók száma ($2 \le N \le 30000$) és a lefutandó kilométer ($1 \le H \le 100$). A további *N* sor mindegyikében egy egész szám van ($0 \le x \le K$) ami azt jelenti, hogy egy futó az *x*-edik kilométertől vállalja a láng továbbítását. A futókat a sorszámukkal azonosítjuk, a bemenet *j*+1-edik sorában a *j*-edik futó adata van. Feltételezhetjük, hogy a láng eljuttatható a cél városig a jelentkezett futókkal.

A *lang.ki* szöveges állomány első sorába a láng célba juttatásához minimálisan szükséges futók *M* számát kell írni! A második sor pontosan *M* számot tartalmazzon (egy-egy szóközzel elválasztva), azon futók sorszámát, akik teljesítik a feladatot! Tehát a felsorolásban a *j*-edik futó a *j*+1-edik futónak adja át a lángot. Több megoldás esetén bármelyik megadható.

Példa:

```
lang.be        lang.ki
30 7 10        4
17             4 7 1 2
24
13
0
5
19
7
```



Megoldás

Akkor van szükség a legkevesebb résztvevőre, ha minden futó a lehető legtovább fut. Azaz az általa elérhető futók közül azzal váltjuk le, aki a legtávolabbról indul.[26]

Rendezzük a jelentkezőket az indulás helye szerint növekvő sorrendbe! Először kiválasztjuk az első jelentkezőt. Majd mindig megkeressük azt a jelentkezőt, akit még éppen el tud érni az előzőleg választott futó, amíg csak be nem érnek a célba.

Az egyes futók jellemzéséhez struktúrát definiálunk:

```
STRUKTÚRA TFutó
    VÁLTOZÓ Sorszám, Honnan MINT Egész
STRUKTÚRA VÉGE
```

[26] Ezzel a mohó algoritmust alkalmazzuk.

A jelentkezőket és a kiválasztott futókat egyaránt listában tároljuk:

```
VÁLTOZÓ Jelentkezők MINT Lista(Elemtípus: TFutó)
VÁLTOZÓ Futók MINT Lista(Elemtípus: TFutó)
VÁLTOZÓ K, N, H MINT Egész
Be: K, N, H, Jelentkezők lista elemei
```

A beolvasás után a jelentkezőket az indulási hely szerint rendezzük, majd ütközőt helyezünk a lista végére. Így a továbbiakban egyszerűbb lesz a keresés.

```
Jelentkezők.Rendez(Függvény(Elem1, Elem2) Elem1.Honnan – Elem2.Honnan)
Futó.Sorszám = N + 1, Futó.Honnan = K + H + 1 ' ütköző
Jelentkezők.Add(Futó)
```

A feladat szövege szerint van megoldás, tehát a rendezett listában az első futó a 0 km-től indul. Vele kezdünk:

```
Futók.Add(Jelentkezők.Első)
Jelentkezők.Kivesz(0 indexű elem)
```

Amíg csak el nem érnek a célba, mindig megkeressük a következő jelentkezőt, akit még el tud érni az előzőleg választott futó:

```
VÁLTOZÓ Következő, Meddig MINT Egész
Meddig = Futók.Utolsó.Honnan + H ' eddig tud elfutni az utoljára választott futó
CIKLUS AMÍG Meddig < K
    Következő = Jelentkezők.IndexKeres(Függvény(Elem) Elem.Honnan>Meddig) – 1
    ' (a keresésnél talált jelentkezőt megelőző jelentkezőnek kell indulnia)
    Futók.Add(Jelentkezők(Következő))
    Jelentkezők.Kivesz(0-tól Következő-ig) ' így gyorsabb lesz a következő keresés
    Meddig = Futók.Utolsó.Honnan + H
CIKLUS VÉGE
```

A teljes megoldást a *lang* program tartalmazza.

Elkerülhetjük a beolvasott adatok rendezését, ha egy statikus (!) tömbben feljegyezzük, hogy melyik jelentkező indulna az *I*-edik km-től. Írjuk meg így is a programot!

További feladatok a metódusokra

Egy gyakorlott programozó egyetlen program megírása során sem nélkülözheti az algoritmusokat megvalósító metódusokat. Bármely feladat megoldásában a hasznunkra válnak. Ezért itt olyan versenyfeladatokat válogattunk össze, melyek megoldása fejleszti programozási készségünket, de nem igénylik különösebb stratégia alkalmazását.

OITV 1995-2f2	Fenyőfa
OITV 1999-2f2	Család
OITV 1999-3f2	Hálózat
OITV 2000-2f2	Zárnyitogató
OITV 2000-2f2	Lift
OITV 2000-3f2	Vállalatok
OITV 2001-2f2	Jelek
OITV 2003-2f2	Harmadolás
OITV 2006-2f2	Csőposta
OITV 2007-2f2	Hírek
OITV 2009-3f2	Kísérlet
OITV 2010-1f1	Eső
OITV 2011-1f1	Balaton
OITV 2011-2f1	Sípálya

8. A verem és a sor

Kapcsolódó leckék:
Programozási ismeretek (Műszaki Kiadó, 2011)
51. Dátum és idő típusú változók
Programozási ismeretek haladóknak (Műszaki Kiadó, 2012)
17. A verem
18. A sor

A verem és a sor adatszerkezetet a *Programozási ismeretek haladóknak* tankönyvben főleg összetett algoritmusok felírásánál (például a gráfok bejárásánál) használtuk. Önálló alkalmazásuk a versenyfeladatokban ritkábban fordul elő.

Valójában nem lenne szükség erre a két kollekcióra, hiszen a lista segítségével mindkettőjük működését szimulálhatjuk. A verem vagy a sor azonban egyszerűbbé teszi az adminisztrációt (nem kell megjelölnünk a listaelem indexét), esetenként pedig rövidebb ideig tart egy-egy művelet végrehajtása a listához viszonyítva.

A sorból például gyorsabban lehet kivenni (törölni) elemet, mint a listából. Ezért célszerű sort használni a lista helyett mindazokban a programokban, melyekben a lista csak a beolvasott adatok átmeneti tárolását szolgálja. A metódusok végrehajtási idejére a befejező leckében térünk vissza.

A továbbiakban – a tankönyvben bemutatott modellnek megfelelően – feltételezzük, hogy a *Sorból*/*Veremből* műveletek olyan függvények, melyek visszatérési értéke a kollekció soron következő eleme. Ezek a műveletek el is távolítják az elemet a kollekcióból.

> A C++-ban a verem (*stack*) és a sor (*queue*) nem teljes értékű konténerek, csupán konténer adaptációk. Létrehozásukkor megadhatjuk, hogy melyik szekvenciális konténerre (*vector*, *list*, *deque*) épülve jöjjenek létre. A konténer adaptációk korlátozott metóduskészlettel rendelkeznek. Például az első/legfelső elem eltávolításához először a *front()*/*top()* függvénnyel ki kell olvasnunk az elemet, majd pedig a *pop()* függvénnyel eltávolíthatjuk (!) azt. A fenti részben a *deque* a kétvégű sor (double ended queue) szekvenciális konténert jelöli.

Nyelv

Nemes Tihamér OITV 2011. 2. forduló 1. korcsoport 3. feladat

Egy programozási nyelven az elágazások az IF szóval kezdődnek és a FI szóval végződnek. Minden program legalább egy, legfeljebb 100 szóból áll.

Készíts programot *nyelv* néven, amely beolvas szavanként egy szöveget, majd megadja, hogy az elágazások egymásba ágyazása helyes-e! Ha nem helyes, akkor megadja az első hiba okát is. (Például hibás egymásba ágyazás az alábbi: … IF … FI … FI … IF …)

Megjegyzés: a többi szó bármi lehet, ellenőrzésükkel nem kell foglalkozni.

Példa:

Bemenet	*Kimenet*
`Szavak száma: 7`	`Hibás: FI utasítás hiányzik`
`1. szó: IF`	
`2. szó: ALMA`	
`3. szó: IF`	
`4. szó: FI`	
`5. szó: FI`	
`6. szó: IF`	
`7. szó: BARACK`	

Megoldás

Valójában elegendő lenne az IF és FI szavakat számlálni, de gyakorlásként veremmel oldjuk meg a feladatot.

A beolvasásra kerülő szavakat a *Szavak* listában tároljuk:

```
VÁLTOZÓ Szavak MINT Lista(Elemtípus: Sztring)
VÁLTOZÓ N MINT Egész
Be: N, Szavak lista elemei
```

Sorra vesszük a *Szavak* lista elemeit. Ha IF következik, akkor beletesszük egy verembe. Ha FI következik, akkor

– üres verem esetén hibát találtunk (hiányzik egy IF utasítás);

– nem üres verem esetén kivesszük (és eldobjuk) az utoljára behelyezet IF-et.

Az egyéb szavakat nem vesszük figyelembe a feldolgozásnál.

```
VÁLTOZÓ Verem MINT Verem(Elemtípus: Sztring)
Verem.Töröl()
CIKLUS MINDEN Szó-ra a Szavak-ban
    ELÁGAZÁS a Szó SZERINT
        "IF" ESETÉN
            Verem.Verembe("IF")
        "FI" ESETÉN
            HA Verem.Elemszám == 0 AKKOR
                Ki: IF utasítás hiányzik
                PROGRAMBÓL KILÉP
            EGYÉBKÉNT
                Verem.Veremből()
            ELÁGAZÁS VÉGE
        ' egyébként eldobjuk a Szó-t
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

Hibátlan program esetén a ciklus végeztével kiürül a verem. Egyébként hiányzik a programból legalább egy FI utasítás.

```
HA Verem.Elemszám > 0 AKKOR
    Ki: FI utasítás hiányzik
    PROGRAMBÓL KILÉP
ELÁGAZÁS VÉGE
```

Ha találunk valamilyen hibát, akkor kiugrunk a programból. Ezért az algoritmus utolsó utasítását csak helyes program esetén érjük el:

Ki: Helyes program

Célszerű néhány tesztesetre papíron, ceruzával végigkövetni az algoritmust. Közben figyeljük a verem tartalmát!

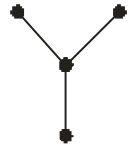
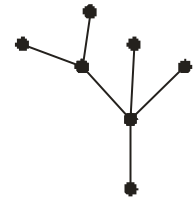
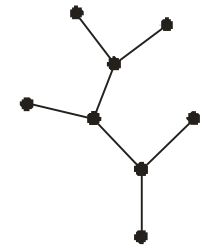
A megoldást a *nyelv* program tartalmazza. Beolvasás közben nagybetűssé alakítottuk a szavakat (biztos, ami biztos ☺).

Írjuk meg úgy is a programot, hogy verem alkalmazása helyett beolvasás közben számoljuk az IF és FI szavakat!

Fa

Nemes Tihamér OITV 2011. 2. forduló 2. korcsoport 2. feladat

Minden fát leírhatunk egy karaktersorozattal. Ebben a leírásban X betűk és zárójelek fognak szerepelni. Az X ágat jelent, az ágak végi elágazásokat pedig zárójelbe tesszük.

			
X	X(X)(X)	X(X(X)(X))(X)(X)	X(X(X)(X(X)(X))(X)

Írj programot *fak* néven, amely megadja

a) a fa magasságát (a földtől milyen messze van a legmesszebb levő ágvég);

b) a fa elágazásainak a számát (a törzs nem számít elágazásnak);

c) egy helyen a legnagyobb elágazásszámot!

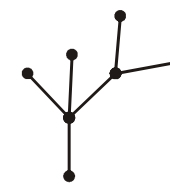
A *fak.be* szöveges állomány egyetlen sorában a fát leíró szöveg van (hossza legfeljebb 10000 karakter).

A *fak.ki* szöveges állomány első sorába a fa magasságát, a második sorába a fa elágazásai számát, a harmadik sorába pedig a legnagyobb elágazásszámot kell kiírni!

Példa:

```
fak.be
X(X)(X)(X(X)(X))
```

```
fak.ki
3
5
3
```



Megoldás

A megoldás előtt gondosan tanulmányozzuk a feladat szövegében szereplő példákat! Ellenőrizzük az ábrákon az alábbi gondolatmeneteket! Az algoritmusokat kövessük végig papíron, ceruzával! Közben figyeljük a vermek tartalmát!

A fát leíró szöveget a *Leírás* sztringben tároljuk. Egyszerűsítjük az algoritmusokat, ha az egész sztringet egy további zárójelpárba foglaljuk. Így minden ág zárójelen belül helyezkedik el.

```
VÁLTOZÓ Leírás MINT Sztring
Be: Leírás
Leírás = "(" & Leírás & ")"
```

A legkönnyebben a b) kérdésre válaszolhatunk. Az elágazások száma megegyezik a nyitózárójelek számával.

```
VÁLTOZÓ DbElágazás MINT Egész
DbElágazás = Leírás.Megszámol(Függvény(Elem) Elem == "(") – 1
```

Az 1-et az utólag hozzáfűzött zárójelpár miatt vontuk ki.

Az a) kérdés megválaszolásához értelmeznünk kell a fa magasságát, amit nem definiál a feladat szövege. Tekintsük magasságnak a földtől az egyes csúcsokig található ágak (*X*-ek) számának a maximumát!

Az ábrákból és a példából arra következtethetünk, hogy egymás után több ág (*X* karakter) nem fordul elő elágazás nélkül. Ebben az esetben az *X*-ek száma megegyezik a nyitó zárójelek számával (gondoljuk meg, hogy miért). Elegendő tehát az egymást követő, lezárás nélkül megnyitott zárójelek maximális számát meghatározni.

A versenybizottság által a megoldások értékeléséhez használt tesztesetek között azonban előfordult olyan bemenet, amelyben több *X* karakter szerepelt egymás után. Ez természetesen megnöveli a fa magasságát, hiszen hallgatólagosan az ágakat egyforma hosszúnak tekintjük.[27]

Határozzuk meg a fa magasságát mindkét értelmezés szerint!

1. Az egyszerre megnyitott (és le nem zárt) zárójelek számának a maximuma

Vegyük sorra a fa leírásában szereplő karaktereket! Ha nyitó zárójel következik, azt helyezzük egy verembe! Végzárójel estén pedig vegyük le a verem tetején lévő nyitó zárójelet, ugyanis ez volt a végzárójel párja (miért?)! Közben figyeljük a verem méretét, mert ez határozza meg az aktuális magasságot!

```
VÁLTOZÓ Fa MINT Verem(Elemtípus: Karakter)
VÁLTOZÓ Magasság MINT Egész ' a fa magassága
Magasság = 0
CIKLUS MINDEN Karakter-re a Leírás-ban
   ELÁGAZÁS Karakter SZERINT
      "(" ESETÉN
         Fa.Verembe(Karakter)
      ")" ESETÉN
         HA Magasság < Fa.Elemszám AKKOR Magasság = Fa.Elemszám
         Fa.Veremből()
   ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

Számítsuk ki a fa magasságát verem felhasználása nélkül, a zárójelek számlálásával is!

[27] Ennek ellenére a versenybizottság az ilyen bemenetnél is a megnyitott és le nem zárt zárójelek maximális számát fogadta el helyes megoldásként!? Lásd még ugyanebben a fordulóban az OKTV 2. feladatát!

2. A földtől a csúcsokig az egymást követő X-ek számának a maximuma

A magasság meghatározásához képzeljük el, hogy elindulunk a fa tövétől a csúcsokig! Minden kezdőzárójelnél elágazást találunk, ennek egyik ágán megyünk tovább, egészen a csúcsig. Közben számoljuk az ágakat (az *X*-eket). Egy csúcshoz érve az *X*-ek száma megadja a csúcs távolságát a gyökértől.

Ezt követően a csúcstól elindulunk visszafelé az előző elágazásig (az előző kezdőzárójelig). Minden lépésnél (*X*-nél) kivonunk 1-et az aktuális magasságból. Ha visszaértünk az elágazáshoz, akkor ismét elindulunk felfelé, a következő csúcsig. Ezt az eljárást folytatjuk, amíg csak van elágazás. A csúcsok távolságának a maximuma adja a fa magasságát.

A bejárást könnyebben követhetjük, ha a *Fa* veremben tároljuk a talált kezdőzárójeleket és az *X*-eket. Felfelé haladáskor elhelyezzük ezeket a karaktereket a veremben, visszafelé lépkedéskor pedig kivesszük a veremből. A csúcs elérést végzárójel jelzi.

```
VÁLTOZÓ Fa MINT Verem(Elemtípus: Karakter)
VÁLTOZÓ DbX MINT Egész ' az egymást követő X-ek száma
VÁLTOZÓ Magasság ' a fa magassága
DbX = 0
Magasság = 0
CIKLUS MINDEN Karakter-re a Leírás-ban
   ELÁGAZÁS Karakter SZERINT
      "(" ESETÉN ' kezdődik egy elágazás
         Fa.Verembe(Karakter)
      "X" ESETÉN
         Fa.Verembe(Karakter)
         DbX = DbX + 1
      ")" ESETÉN ' befejeződik egy elágazás
         HA DbX > Magasság AKKOR Magasság = DbX
         CIKLUS AMÍG Fa.Tető ≠ "(" ' visszalépünk az előző kezdőzárójelig
            DbX = DbX – 1
            Fa.Veremből() ' kivesszük az X-et (és eldobjuk)
         CIKLUS VÉGE
         Fa.Veremből()' kivesszük a kezdőzárójelet (és eldobjuk)
   ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A c) kérdés megválaszolásához figyeljük meg az elágazásokat! Minden elágazást egy zárójelpár határoz meg. Ezért megszámoljuk, hogy egy zárójelben hány zárójelpár van közvetlenül (!) benne. Azaz a közvetlenül benne lévő zárójelpárok tartalmát már nem vesszük figyelembe. A ((…)(…)(…)) fa esetén például az elágazások száma 3, akármi is áll a pontok helyén.

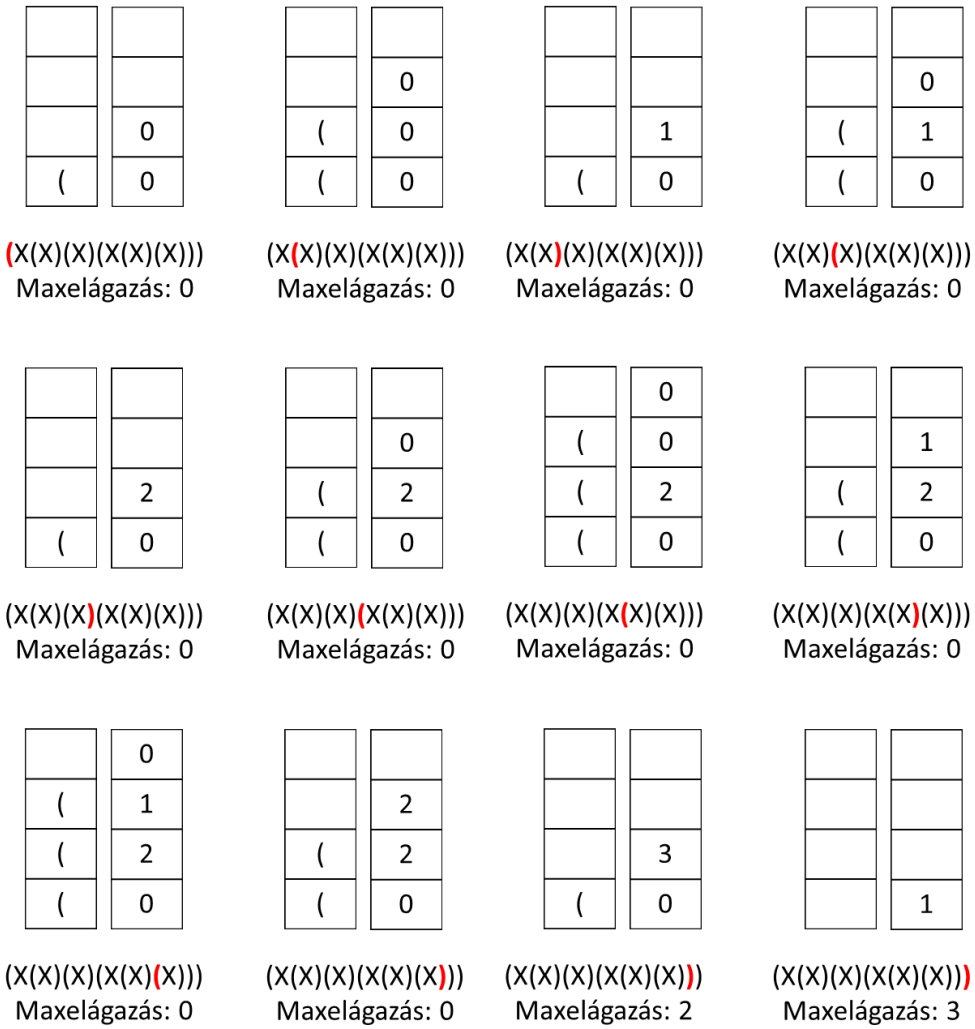
A közvetlenül egy zárójelpárban lévő zárójelpárok számának maximuma adja meg az egy helyen található a legnagyobb elágazásszámot.

Az a) feladat megoldásához hasonlóan bejárjuk a fát. A *Fa* verembe helyezzük a megkezdett zárójelpárokat. Most azonban nincs szükségünk az *X* karakterekre, ezeket nem vesszük figyelembe. Így egy végzárójel elérése esetén biztosan a verem tetején található a hozzá tartozó kezdőzárójel.

Ha kezdőzárójelhez érkezünk, akkor elkezdjük számolni az utána következő zárójelpárokat. Előfordulhat azonban, hogy egy ilyen zárójelpáron belül újabb zárójelpár

található. Ezért a darabszámokat az *Elágazások* veremben tároljuk. Ha újabb zárójel-pár kezdődik, akkor egy 0-t helyezünk az *Elágazások* verem tetejére, és minden egyes végzárójelnél ennek értékét növeljük meg.

Kövessük végig az alábbi ábra segítségével a fa bejárását és a vermek tartalmának alakulását!



A Fa és az Elágazások verem tartalma, illetve a Maxelágazás változó értéke a zárójelek feldolgozása során (az aktuális zárójelet pirossal jelöltük)

Az algoritmus:

```
VÁLTOZÓ Elágazások MINT Verem(Elemtípus: Egész)
' Elágazások: a közvetlenül egy zárójelen belüli elágazások száma
VÁLTOZÓ MaxElágazás MINT Egész
Fa.Töröl() ' nem kellene, mert kiürült a verem
MaxElágazás = 0
Elágazások.Verembe(0) ' ütköző
```

```
CIKLUS MINDEN Karakter-re a Leírás-ban
    ELÁGAZÁS Karakter SZERINT
        "(" ESETÉN ' kezdődik egy elágazás
            Fa.Verembe(Karakter)
            Elágazások.Verembe(0)
        ")" ESETÉN ' befejeződik egy elágazás
            HA Elágazások.Tető() > MaxElágazás AKKOR
                MaxElágazás = Elágazások.Tető()
            ELÁGAZÁS VÉGE
            ' Levesszük a befejezett zárójelpár által tartalmazott zárójelpárok számát:
            Elágazások.Veremből()
            Elágazások.Verembe(Elágazások.Veremből() + 1)
                ' eggyel több elágazást tartalmazott az előző zárójelpár
            Fa.Veremből() ' levesszük a kezdőzárójelet
        ' Karakter == "X" esetén nem csinálunk semmit (továbblépünk a ciklusban)
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A feladat megoldását a *Magasság*, *DbElágazás* és *MaxElágazás* változók értéke adja meg. A programot a kiírással együtt a *fak* program tartalmazza.

Módosítsuk a fenti algoritmust úgy, hogy a három ciklust egyetlen ciklusba vonjuk össze! Írjuk meg a programot vermek helyett a zárójelek és az *X*-ek számlálásával!

Oldjuk meg a 2011-es OKTV 2. fordulójának nagyon hasonló, 2. feladatát (*Fa*)!

Benzinkút

Nemes Tihamér OITV 1997. 2. forduló 2. korcsoport 2. feladat

Egy benzinkútnál *K* töltőhelyen lehet tankolni. A kúthoz összesen *N* autó érkezik, melyek különböző ideig foglalják el a töltőhelyeket. Az érkező autók egy szabad töltőhelyhez állnak. Ha minden hely foglalt, akkor várakoznak, majd érkezési sorrendben állnak a szabaddá vált kutakhoz.

A *benzin.be* állomány első sorában *N* ($1 \le N \le 100$) és *K* ($1 \le K \le 10$) értéke van, szóközzel elválasztva. A következő *N* sor az egyes autók érkezési (óra, perc), illetve tankolási (perc) idejét tartalmazza.

Készíts *benzin* néven programot, amely a *benzin.ki* állomány egy-egy sorába kiírja, hogy az egyes autók – érkezési idejük sorrendjében – mikor hagyják el a benzinkutat (óra, perc)!

Példa:

```
benzin.be          benzin.ki
3 2                7 5
6 30 35            6 45
6 35 10            6 55
6 40 10
```

Megoldás

A feladattal azt szemléltetjük, hogyan lehet lista helyett sor adatszerkezetben tárolni a beolvasott adatokat.

A megoldás során feltételezzük, hogy az autók érkezési idejük sorrendjében szerepelnek a *benzin.be* fájlban. Ez a feltétel teljesült a versenybizottság által használt tesztadatoknál.

Az autók adataihoz struktúrát definiálunk. Mivel az autókat sor adatszerkezetben tároljuk, a struktúrába felvesszük az autó sorszámát is:

```
STURKTÚRA TAutó
    VÁLTOZÓ Sorszám MINT Egész
    VÁLTOZÓ Érkezés MINT Időpont
    VÁLTOZÓ Tankolás MINT Egész ' a tankolás időtartama percben
STRUKTÚRA VÉGE

VÁLTOZÓ Autók MINT Sor(Elemtípus: TAutó)
```

Ha a tanult programozási nyelv nem ismeri az *Időpont* típust, akkor számolhatunk percekben is.

A töltőhelyeket olyan listában tartjuk nyilván, amely az onnan való távozási időket tartalmazza:

```
VÁLTOZÓ Töltőhelyek MINT Lista(Elemtípus: Időpont)
```

Az autók távozási időpontjait rendezett listában gyűjtjük. Az elem kulcsa az autó sorszáma, értéke pedig a távozás időpontja lesz:

```
VÁLTOZÓ Távozások MINT RendezettLista(Kulcstípus: Egész, Értéktípus: Időpont)
```

A beolvasásnál feltöltjük a sort:

```
Be: N, K
VÁLTOZÓ Autó MINT TAutó
CIKLUS I=1-től N-ig
    Be: érkezés időpontja, tankolás időtartama (percben)
    Autó.Sorszám = I
    Autó.Érkezés = érkezés időpontja
    Autó.Tankolás = tankolás időtartama
    Autók.Sorba(Autó)
CIKLUS VÉGE
```

A sor elején álló autókat beengedjük a töltőhelyekre. Közben ügyelünk arra, hogy esetleg kevesebb autó érkezik, mint ahány hely a rendelkezésünkre áll:

```
VÁLTOZÓ Időpont MINT Időpont
CIKLUS AMÍG Töltőhely.Elemszám < K ÉS Autók.Elemszám > 0
    Autó = Autók.Sorból
    Időpont = Autó.Érkezés + Autó.Tankolás ' a távozás időpontja
    Töltőhelyek.Add(Időpont)
    Távozások.Add(Autó.Sorszám, Időpont)
CIKLUS VÉGE
```

A következő autó (ha van) arra a töltőhelyre állhat be, ahonnan a leghamarabb távozik az ott lévő autó. Távozási időpontját úgy határozzuk meg, hogy a tankolás időtartamát hozzáadjuk

a) az érkezési időpontjához, ha ebben az időpontban már elment a töltőhelyről az autó;
b) a töltőhelyen álló autó távozási időpontjához, ha az érkezéskor még nem ment el a töltőhelyről.

Mindezt addig ismételjük, amíg van még autó a sorban. Közben ne felejtsük el feljegyezni a *Távozások* listában a távozó autók adatait, illetve felülírni a töltőhelyre beálló autó távozási időpontját:

```
VÁLTOZÓ Hová MINT Egész ' ide fog beállni a következő autó
CIKLUS AMÍG Autók.Elemszám > 0
    Időpont = Töltőhelyek.Min()
    Hová = Töltőhelyek.IndexKeres(Időpont)
    Autó = Autók.Sorból()
    HA Autó.Érkezés > Időpont
        ' az autó később érkezett, mint ahogy az előző autó elment a töltőhelyről
        Időpont = Autó.Érkezés
    ELÁGAZÁS VÉGE
    Időpont = Időpont + Autó.Tankolás
    Távozások.Add(Autó.Sorszám, Időpont)
    Töltőhelyek(Hová) = Időpont
CIKLUS VÉGE
```

Mivel a távozó autók adatait a sorszám szerint rendezett listában tartjuk nyilván, a végén nincs más dolgunk, mint kiírni a *Távozások* lista elemeinek értékét. A teljes megoldást a *benzin* program tartalmazza.

Huszár

Nemes Tihamér OITV 2009. 2. forduló 2. korcsoport 3. feladat

Egy sakktáblára elhelyezünk egy huszárt. A sakktábla 8*8-as négyzet. A huszár „lóugrásban” lép, azaz vagy vízszintes irányban lép egyet és függőlegesen kettőt, vagy pedig fordítva.

Készíts programot *huszar* néven, amely egy adott pozícióra elhelyezett huszár esetén megadja, hogy a huszár legfeljebb *N* lépés alatt mely pozíciókra juthat el!

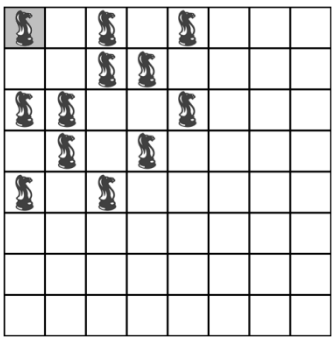
A *huszar.be* szöveges állomány egyetlen sorában a huszár sorindexe ($1 \le SOR \le 8$) és oszlopindexe ($1 \le OSZLOP \le 8$), valamint a lépések maximális száma ($1 \le N \le 6$) van, egy-egy szóközzel elválasztva.

A *huszar.ki* szöveges állományba 8 sort kell írni, mindegyikben pontosan 8 karakter legyen! Az *i*-edik sor *j*-edik oszlopába *-ot kell kiírni, ha a huszár ezt a pozíciót legfeljebb *N* lépés alatt elérheti, egyébként pedig szóközt!

Példa:

```
huszar.be
1 1 2
```

```
huszar.ki
*.*.*...
..**....
**..*...
.*.*....
*.*.....
........
........
........
```



Megoldás

A sakktáblát a *Tábla* tömbben tároljuk, amit kezdetben szóközökkel töltünk fel:

```
KONSTANS Méret MINT Egész = 8
VÁLTOZÓ Tábla(1…Méret, 1…Méret) MINT Karakter
VÁLTOZÓ Sor, Oszlop MINT Egész
Tábla(,) = " "
Be: N, Sor, Oszlop
```

A huszár pozícióit (a mezők koordinátáit) struktúrában tartjuk nyilván:

```
STRUKTÚRA TMező
    VÁLTOZÓ Sor, Oszlop MINT Egész
STRUKTÚRA VÉGE
```

A huszár egy lépése során a sor 2-vel és az oszlop 1-gyel, illetve a sor 1-gyel és az oszlop 2-vel változhat. Az új pozíció meghatározásához célszerű az összetartozó változásokat egy-egy tömbben tárolni (lásd az ábrát):

	−2 −1		−2 +1	
−1 −2				−1 2
		0 0		
1 −2				1 2
	2 −1		2 1	

```
VÁLTOZÓ Dsor(1…8) MINT Egész = (–2, –2, –1, 1, 2, 2, 1, –1)
VÁLTOZÓ Doszlop(1…8) MINT Egész = (–1, 1, 2, 2, 1, –1, –2, –2)
```

A *Sor*, *Oszlop* mezőn álló huszár egy lehetséges új helyzetét a *Sor* + *Dsor*(*I*), *Oszlop* + *Doszlop*(*I*) kifejezések adják meg ($1 \le I \le 8$).

A bejárás során a huszár minden egyes pozíciója esetén sor adatszerkezetben gyűjtjük össze azoknak a mezőknek a koordinátáit, melyekre továbbléphet a bábu. Majd a sor minden elemével megismételjük ugyanezt az eljárást. Ezért összesen *N* sorra lenne szükségünk. De megtehetjük azt is, hogy csak két sort deklarálunk, amiket aztán felváltva alkalmazunk. A feldolgozás során ugyanis kiürül az aktuális sor, tehát ismét felhasználhatjuk.

A sorok felcseréléséhez (valójában csak a sorokra mutató hivatkozások cseréjéhez) szükségünk van egy segédváltozóra:

```
VÁLTOZÓ Cseresor MINT Sor(Elemtípus: TMező)
```

A sorok feldolgozását *N*-szer végezzük el. A közben érintett mezőket *-gal jelöljük. A második sorba csak azokat a pozíciókat tesszük bele, melyeken még nem jártunk (nincsenek *-gal megjelölve). Ezzel jelentős mértékben gyorsítjuk a bejárást.

Kiindulásként a kezdőmező pozícióját helyezzük el a sorban.

```
VÁLTOZÓ Sor1, Sor2 MINT Sor(Elemtípus: TMező)
VÁLTOZÓ Mező MINT TMező ' segédváltozó
Sor1.Sorba((Sor, Oszlop))
Tábla(Sor, Oszlop) = "*"
```

```
CIKLUS I=1-től N-ig
    CIKLUS AMÍG Sor1.Elemszám > 0
        Mező = Sor1.Sorból()
        CIKLUS J=1-től 8-ig
            Sor = Mező.Sor + Dsor(J) ' az új pozíció
            Oszlop = Mező.Oszlop + Doszlop(J)
            HA Sor ≥ 1 ÉS# Sor ≤ Méret ÉS# Oszlop ≥ 1 ÉS# Oszlop ≤ Méret ÉS#
                                                        Tábla(Sor, Oszlop) = " "
                Tábla(Sor, Oszlop) = "*"
                Sor2.Sorba((Sor, Oszlop))
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
    CIKLUS VÉGE
    Cseresor = Sor 1 ' a hivatkozások cseréje
    Sor1 = Sor2
    Sor2 = Cseresor
    Sor2.Töröl() ' felesleges, mert a Sor1 kiürült!
CIKLUS VÉGE
```

Ezek után már csak a *Tábla* tömb fájlba írása van hátra. A teljes megoldást a *huszar* program tartalmazza.

Vegyük észre, hogy lényegében egy gráf szélességi bejárását végeztük el. Mi helyettesítette a szélességi bejárás megszokott algoritmusában szereplő halmazt?

Futtassuk a programot nagyméretű, például 1000×1000-es sakktáblára is (legyen a kezdőpozíció 500, 500 és $N = 10000$)! Milyen végrehajtási időt tapasztalunk? Mi történne, ha a *Mező = Sor.Sorból*() utasítás után (tehát nem a legbelső ciklusban) helyeznénk *-ot a mezőre? Próbáljuk ki így is a futásidőt!

A feladat egy másik megoldását a *Programozási ismeretek versenyzőknek* példatár *Rekurzió* leckéje mutatja be. Hasonlítsuk össze a kétféle megoldás algoritmusát, hatékonyságát, futásidejét!

Javasoljuk az Olvasónak, hogy oldja meg a 2009-es OKTV 2. fordulójának nagyon hasonló, 3. feladatát is (*Sakk*).

Verem

Nemes Tihamér OITV 2003. 2. forduló 2. korcsoport 5. feladat

A veremautomata olyan gép, amely a bemenetként kapott számsorozaton az alábbi módon működik. Sorban balról jobbra egyesével olvassa a számsorozatot, és vagy a sorozat aktuális elemével, vagy a verem tetején lévő elemmel végezhet műveletet. Egy lépésben az alábbi három művelet valamelyikét hajthatja végre:

1. A bemenet aktuális elemét kiírja a kimenetre.
2. A bemenet aktuális elemét beteszi a verembe az ott lévő sorozat elé.
3. A verem tetején lévő (a sorozatban első) elemet kiveszi a veremből és kiírja a kimenetre.

Kezdetben a verem üres. Feladatunkban a veremautomatát arra akarjuk használni, hogy bemenetként kap egy számsorozatot, amely az 1, …, N számokat tartalmazza tetszőleges sorrendben, és a kimenetre írja ki az 1, …, M ($1 \le M \le N$) számsorozatot, a lehető legnagyobb M-ig. (A kimenetben minden számnak szerepelnie kell M-ig és sorrendben kell lenniük!)

Írj programot *verem* néven, amely kiszámítja, hogy melyik az a legnagyobb *M* érték, amelyre a veremautomata kimenete az 1, …, *M* sorozat lehet!

A *verem.be* szöveges állomány első sorában a bementi sorozat *N* elemszáma van ($1 \le N \le 10000$). A második sor *N* különböző egész számot tartalmaz egy-egy szóközzel elválasztva. Minden *x* számra teljesül, hogy $1 \le x \le N$.

A *verem.ki* állomány első és egyetlen sorába azt a legnagyobb *M* számot kell írni, amelyre a veremautomata kimenete az 1, …, *M* sorozat lehet!

Példa:

```
verem.be                    verem.ki
10                          8
3 2 1 5 4 6 9 7 10 8
```

Megoldás

A bemenő sorozatot sor adatszerkezettel tároljuk (másként is lehetne). Mivel szükségünk lesz az utoljára kiírt szám ellenőrzésére, a kimenő sorozat elemeit listában gyűjtjük össze. Egyes programozási nyelveknél a sor (illetve a verem) bármely elemét lekérdezhetjük. Ebben az esetben a kimenő sorozatot elhelyezhetnénk egy újabb sorban. A feladat szövegében szereplő vermet szintén deklaráljuk:

```
VÁLTOZÓ SorBe MINT Sor(Elemtípus: Egész)
VÁLTOZÓ Lista MINT Lista(Elemtípus: Egész)
VÁLTOZÓ Verem MINT Verem(Elemtípus: Egész)
```

Az adatok beolvasásakor feltöltjük a *SorBe* sort. A feldolgozás előtt mind a listához, mint a veremhez fiktív kezdőértéket adunk hozzá. Így egyszerűbb lesz a cikluson belüli vizsgálat:

```
Be: N, a SorBe elemei
Lista.Add(0)
Verem.Verembe(0)
```

A feldolgozási ciklus minden egyes lépésében megnézzük, hogy a verem tetején lévő elem kiírható-e a listába.

a) Ha igen, akkor kiírjuk.
b) Ha nem, akkor kiveszünk egy elemet a sorból, és megnézzük, hogy kiírható-e a listába.
   – Ha igen, akkor kiírjuk.
   – Ha nem, akkor betesszük a verembe.

Ezt a folyamatot mindaddig ismételjük, amíg ki nem ürül a *SorBe*:

```
CIKLUS AMÍG SorBe.Elemszám > 0
   HA Verem.Tető == Lista.Utolsó + 1 AKKOR
      Lista.Add(Verem.Veremből())
   EGYÉBKÉNT
      Szám = SorBe.Sorból()
      HA Szám == Lista.Utolsó + 1 AKKOR Lista.Add(Szám)
      EGYÉBKÉNT Verem.Verembe(Szám)
      ELÁGAZÁS VÉGE
   ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A kódoláskor ügyeljünk a vizsgálatok sorrendjére! A fentitől eltérő sorrendet is alkalmazhatunk, de szükség lehet az egymásba ágyazások módosítására.

A sor kiürülése után a veremből a lehetséges értékeket áttesszük a listába:

```
CIKLUS AMÍG Verem.Tető == Lista.Utolsó + 1
    Lista.Add(Verem.Veremből)
CIKLUS VÉGE
```

A kiírás előtt célszerű eltávolítani a listából a fiktív kezdőértéket:

```
Lista.Kivesz(0)
Ki: Lista.Elemszám
```

A teljes megoldást a *verem* program tartalmazza. A teszteléshez az elemszám mellett kiírtuk a lista elemeit is.

További feladatok a veremre és a sorra

A programozási versenyeken közvetlenül a veremre és a sorra vonatkozó feladatok csak ritkán fordulnak elő. Ezen kollekciókkal főleg a különböző programozási stratégiák alkalmazásánál és a gráfok feldolgozásánál találkozunk.

Az alábbi feladatok főleg a sor alkalmazására vonatkoznak. Legtöbbjükben a feladat szövegében szereplő események szimulálására használható fel a sor adatszerkezet.

OITV 1995-2f2	Fenyőfa
OITV 1999-1f2	Kiskapus sor Készítsünk programot, amely megvalósítja a feladatban szereplő adatszerkezetet! Eljáráshívásokkal mutassuk be a működését!
OITV 2013-2f1	Autók (a belépési időket tárolhatjuk sor adatszerkezetben)
OITV 2013-2f2	Autó
OITV 2014-3f2	Párosítás Oldjuk meg a feladatot kétvégű sorral!
OITV 2014-3f2	Metró
OKTV 2009-2f3	Sakk
OKTV 2011-2f3	Fa
OKTV 2013-2f3	Zebra
OKTV 2014-1f3	Növény
OKTV 2014-3f3	Metró

9. A prioritási sor

Kapcsolódó leckék:
Programozási ismeretek haladóknak (Műszaki Kiadó, 2012)
18. A sor
19. A dinamikus tömb

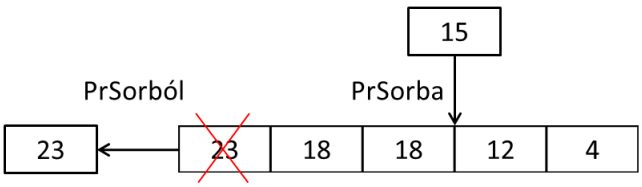
A *Programozási ismeretek haladóknak* tankönyv a prioritási sort csak egy feladat kapcsán említi (138. oldal, 8. feladat). Most részletesebben bemutatjuk a megvalósítását és alkalmazását.

A **prioritási sor** olyan lineáris kollekció, amely prioritással[28] rendelkező elemeket tartalmaz. A sorba helyezésnél az elem a prioritásának megfelelő helyre kerül. Előtte állnak a nagyobb, mögötte pedig a kisebb prioritású elemek. Kivételnél a sor elején álló, tehát a legnagyobb prioritású elemet kapjuk meg.

A prioritásokra léteznie kell rendezési relációnak, azaz egyértelmű sorrendnek.

A prioritást meghatározhatja maga az elem, például elemi típusú adatokat tárolunk nagyság szerinti sorrendben. Ügyeljünk arra, hogy nagyobb prioritáshoz nem feltétlenül tartozik nagyobb, a prioritást kifejező érték! Egy sportversenyen például a jobb helyezés számíthat nagyobb prioritásnak. A távolugrásnál a nagyobb távolság, a futóknál viszont a rövidebb idő jelent jobb helyezést, így nagyobb prioritást.

A prioritási sor gyakran prioritás–érték párokat tartalmaz, a rendezett lista vagy rendezett szótár kulcs–érték párjaihoz hasonlóan. A prioritási sornál azonban megengedjük, hogy a különböző elemeknek egyforma legyen a prioritása, ekkor a kivétel sorrendje tetszőleges.



A prioritási sor műveletei

A prioritási sor legfontosabb műveletei a létrehozás és a törlés mellett:

PrSorba(*Elem*): beilleszti az elemet (prioritás+érték párt) a prioritásnak megfelelő helyre.

PrSorból(): visszatérési értéke megadja a sor elején álló elemet, melyet egyben töröl is a sorból.

A prioritási sor megvalósítása

A C++ nyelv konténer adaptációként tartalmazza a prioritási sort (*priority_queue*). Lekérdezhetjük a sor elemszámát (*size*), illetve hogy üres-e (*empty*). A *push* függvénnyel újabb elemet helyezhetünk a sorba, a *pop* pedig kiveszi (törli) a legnagyobb prioritású elemet. A törlés előtt a legnagyobb prioritású elem értékét a *top* függvénnyel kérdezhetjük le.

[28] A latin *prioritas* szó magyarul elsőbbséget jelent.

Amennyiben a feladatok megoldása során nem elegendőek a *priority_queue* által biztosított műveletek, prioritási sorként alkalmazhatjuk a C++ könyvtár *multimap*, illetve egyszerűbb esetekben a *multiset* konténerét is.

Ha a programozási nyelv nem ismeri a prioritási sort, akkor például dinamikus tömbbel valósíthatjuk meg. Az elemeket olyan struktúrának tekintjük, melynek egyik mezője jelzi a prioritást.

```
STRUKTÚRA TElem
    Prioritás MINT Prioritástípus ' a továbbiakban egész számnak tekintjük
    Érték MINT Értéktípus
STRUKTÚRA VÉGE
```

> Ha a prioritások egymástól különböző, egyedi értékek, akkor a .NET-ben prioritási sorként használhatunk rendezett halmazt vagy rendezett listát. A rendezett listában az elem kulcsa jelenti a prioritást.

A sor létrehozását és törlését, továbbá az elemszám lekérdezését a dinamikus tömbnél megszokott módon végezzük.

A *PrSorba* és a *PrSorból* metódusokat eljárással, illetve függvénnyel helyettesítjük. Általában magát a sort (a dinamikus tömböt) paraméterként adjuk meg:

```
ELJÁRÁS PrSorba(PrSor MINT DinamikusTömb(Elemtípus: TElem),
                                                ÚjElem MINT TElem)
FÜGGVÉNY PrSorból(PrSor MINT DinamikusTömb(Elemtípus: TElem)) MINT TElem
```

Ha a dinamikus tömböt modulszintű változóként deklaráljuk, akkor a *PrSor* paraméter elhagyható.

A Visual Basic-ben (és a .NET-ben) kibővíthetjük a *List* objektumosztály metódusait újabb metódusokkal (részletesebben lásd a *Programozási ismeretek haladóknak* tankönyv 52. oldalán). Mivel a módszer egyszerű, és nem igényli az objektumorientált programozás mélyebb ismeretét, a továbbiakban ezt mutatjuk be.

> Visual Basic-ben a kiterjesztett metódust a forráskódban lássuk el az <*Extension*()> attribútummal! Az attribútum alkalmazásához importáljuk a *System.Runtime.CompilerServices* névteret! A metódus első paramétere *List* típusú legyen! Ennek segítségével hivatkozunk a listára. Ezt a paramétert a hívásnál nem írjuk be az argumentumok közé.

A *PrSorba* eljárásban a paraméterként megadott új elemet a prioritása alapján beillesztjük az eddigi elemek közé. Olyan sorrendet alkalmazunk, hogy a lista végére kerüljön a legnagyobb prioritású (az elsőként kikerülő) elem. Kivételkor ugyanis a lista végén lévő elem sokkal gyorsabban törölhető, mint a lista elején lévő elem.

```
<Kiterjesztett metódus a Lista(Elemtípus: TElem)-hez>
ELJÁRÁS PrSorba(ÚjElem MINT TElem)
    VÁLTOZÓ MelyikElé MINT Egész
    MelyikElé = Lista.IndexKeres(FÜGGVÉNY(Elem) Elem.Prioritás≥ÚjElem.Prioritás)
    HA nincs nagyobb prioritású elem AKKOR
        ' a Lista végére illesztjük be (akkor is, ha üres a Lista)
        Lista.Add(ÚjElem)
    EGYÉBKÉNT
        Lista.Beilleszt(MelyikElé, ÚjElem)
    ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

Az eljárásban természetesen mindig a megfelelő rendezési relációt kell felírnunk.

Mivel a lista rendezett, lineáris keresés helyett célszerű bináris keresést alkalmazni az elem helyének meghatározásánál.

> A Visual Basic (.NET) *FindIndex* metódusa lineáris keresést, *BinarySearch* metódusa pedig bináris keresést végez. Ha a listaelemek összetett típusúak, akkor a *BinarySearch* alkalmazásához implementálnunk kell az *IComparable* interfész *CompareTo* metódusát.

A *PrSorból* függvény visszatérési értéke a lista utolsó eleme. A kiolvasás után töröljük is ezt az elemet:

```
<Kiterjesztett metódus a Lista(Elemtípus: TElem)-hez>
FÜGGVÉNY PrSorból() MINT TElem
    PrSorból = Lista.Utolsó
    Lista.Kivesz(utolsó elem)
FÜGGVÉNY VÉGE
```

Megtehettük volna, hogy az új elemet a lista végére illesztjük be, a legnagyobb prioritású elemet pedig a kivételkor keressük meg. Egy új elem beillesztése a rendezett listába azonban gyorsabb, mint rendezetlen lista esetén a maximális prioritású elem megkeresése és törlése (gondoljuk meg, hogy miért).

A *prsor-1* program bemutatja a fenti metódusok definiálását és használatát. Beolvassa diákok nevét, illetve magasságát, majd kiírja az adatokat a magasságok szerint növekvő sorrendben.[29]

Ha csökkenő sorrendben várjuk az elemeket (a nagyobb érték jelent nagyobb prioritást), akkor a listát növekvő sorrendbe kell rendezni (a végén legyen a legnagyobb, ezt vesszük ki először)! Ha pedig csökkenően rendezzük a listát, akkor az elemeket növekvő sorrendben kapjuk meg.

A *prsor-2* program bemutatja a prioritási sor definícióját struktúrával, a *prsor-3* pedig objektumosztállyal. Mivel a struktúra definíciójában nem hozhatjuk létre a listaobjektumot, ezért konstruktorral készítjük el. A struktúra rendelkezik alapértelmezett (paraméter nélküli) konstruktorral, így csak olyan konstruktort definiálhatunk, amelynek van paramétere (például a lista kapacitása).

Realizáljuk a prioritási sort statikus tömbbel is!

> Visual Basic-ben (.NET-ben) a LINQ *OrderBy*/*OrderByDescending* metódusaival rendezhetjük a sort, így prioritási sort készíthetünk belőle (lásd a *prsor-4* programot). A metódusok a szelektorfüggvény függvényértéke szerint végzik a rendezést. A metódusok visszatérési értéke egy úgynevezett felsoroló objektum, amit vissza kell alakítani sor adatszerkezetté (például egy új sor létrehozásával, a konstruktor segítségével). A módszer azonban meglehetősen rossz hatásfokú, így nem célszerű alkalmazni a prioritási sor megvalósítására.

[29] A *Diákok.txt* állományt a *Programozási ismeretek* tankönyv forrásfájljai között találjuk.

Hidak

Nemes Tihamér OITV 2007. 2. forduló 2. korcsoport 1. feladat

Egy óceáni szigetország N szigetből áll ($1 \le N \le 1000$), amelyek egy egyenes mentén helyezkednek el. A szigeteket hidakkal szeretnék összekötni. A hídépítő vállalatnak annyi pénze van, hogy az egyes tengerszakaszokra összesen K darab pillért építhet ($1 \le K \le 100$), amivel megoldhatja azt, hogy egyes hidakat több darabból építsék össze.

Készíts programot *hidak* néven, amely megadja, hogy mely tengerszakaszokra hány pillért építsenek, ha azt szeretnék, hogy a leghosszabb híddarab a lehető legkisebb legyen!

A *hidak.be* szöveges állomány első sorában a szigetek N száma és a pillérek K száma van egy szóközzel elválasztva. A következő N sorban egy-egy sziget kezdő és végpozíciója található ($0 \le kezdőpozíció < végpozíció \le 1000000$). A pozíciókat az első sziget kezdetétől mérjük, azaz az első sziget kezdőpozíciója biztosan 0.

A *hidak.ki* szöveges állományba annyi sort kell írni, ahány tengerszakaszra helyezünk el pilléreket. Minden sorban a tengerszakasz sorszáma, valamint az oda építendő pillérek száma legyen, sorszám szerint növekvő sorrendben!

Példa:

```
hidak.be          hidak.ki
4 3               2 2
0 20              3 1
25 28
42 44
52 58
```

Megoldás

Mivel a hosszú híddarabok hosszúságát szeretnénk csökkenteni, a pilléreket sorra egymás után a leghosszabb szakaszokhoz fogjuk beilleszteni. [30]

Rendezzük a hidakat a pillérek közötti távolságok szerint csökkenő sorrendbe! Ez először megfelel a hosszok alapján történő rendezésnek, mivel még nincsenek pillérek. A pilléreket sorra egymás után mindig ahhoz a hídhoz helyezzük, amelynél a legnagyobb a pillérek (illetve először a szigetek) közötti távolság. A rendezéshez és a soron következő híd kiválasztásához kézenfekvő prioritási sort használni, melyben egy híd prioritását a pillérek közti távolság adja meg.

A hidak adatait (sorszám, hossz, részek száma) struktúrában tároljuk. A pillérek száma helyett egyszerűbb a hídrészek számát nyilvántartani, ami eggyel több, mint a pillérek száma. A részek számából egyszerűbben határozható meg két pillér távolsága. Ez utóbbit szintén szerepeltetjük a mezők között, így meggyorsítjuk a program futását (nem kell minden összehasonlításnál kiszámítani).

```
STRUKTÚRA THíd
    VÁLTOZÓ Sorszám, Hossz, Rész MINT Egész
    VÁLTOZÓ Táv MINT Valós
STRUKTÚRA VÉGE
```

[30] Ezzel lényegében a mohó stratégiát alkalmazzuk. Lásd: *Programozási ismeretek haladóknak*, 219. oldal.

A hidak adatait prioritási sorba helyezzük, amit listával realizálunk:

```
VÁLTOZÓ Hidak MINT Lista(Elemtípus: THíd)
VÁLTOZÓ N, K MINT Egész
Be: N, K
```

Ügyeljünk arra, hogy az adatfájlban a szigetek adatai találhatók, míg mi a hidak adatait tároljuk! A hidak hosszának meghatározásához először beolvassuk az első sziget kezdetét és végét, majd ezután indítjuk a ciklust:

```
VÁLTOZÓ Kezd, Vég, ElőzőVég MINT Egész
VÁLTOZÓ Híd MINT THíd
Be: Kezd, ElőzőVég ' a Kezd értékét nem használjuk fel (0)
CIKLUS I=1-től N–1-ig ' N szigetet N–1 híd köt össze
    Be: Kezd, Vég
    Híd.Sorszám = I
    Híd.Hossz = Kezd – ElőzőVég
    Híd.Rész = 1
    Híd.Táv = Híd.Hossz ' nincs még pillér
    Hidak.Add(Híd)
    ElőzőVég = Vég
CIKLUS VÉGE
```

A soron következő pillért a leghosszabb hídhoz illesztjük be. A prioritási sorból kivesszük az elején álló (leghosszabb) hidat, majd a *Táv* és a *Rész* mező módosítása után visszatesszük azt. Ezt a folyamatot *K*-szor ismételjük.

```
CIKLUS I=1-től K-ig ' ciklus a pillérekre
    Híd = Hidak.PrSorból()
    Híd.Rész = Híd.Rész + 1 ' eggyel több pillér tartja a hidat
    Híd.Táv = Híd.Hossz / Híd.Rész
    Hidak.PrSorba(Híd)
CIKLUS VÉGE
```

A *PrSorba*, *PrSorból* metódus definícióját lásd az előző szakaszban! A beillesztésnél nagyság szerint növekvő sorrendet használunk a *Táv* mező szerint, hogy a legnagyobb távolság kerüljön a lista végére.

Vegyük észre, hogy a ciklusban tulajdonképpen módosítottuk az elemek prioritását!

A kiírás előtt a sorszámok alapján rendezzük a hidakat:

```
Hidak.Rendez(FÜGGVÉNY(Elem1, Elem2) Elem1.Sorszám – Elem2.Sorszám)
CIKLUS MINDEN Híd-ra a Hidak-ban
    HA Híd.Rész > 1 AKKOR Ki: Híd.Sorszám, Híd.Rész – 1
CIKLUS VÉGE
```

A teljes megoldást a *hidak* program tartalmazza.

Bár most nem okoz gondot, általában nem célszerű valós változók értékét közvetlenül összehasonlítani egymással.[31] Ezt itt úgy kerülhetjük el, hogy átalakítjuk a relációt:

[31] Lásd például: *Programozási ismeretek*, 154. oldal.

$$Elem.Táv \geq ÚjElem.Táv$$

$$\frac{Elem.Hossz}{Elem.Rész} \geq \frac{ÚjElem.Hossz}{ÚjElem.Rész}$$

$$Elem.Hossz \cdot ÚjElem.Rész \geq ÚjElem.Hossz \cdot Elem.Rész$$

Utóbbi alakot használva nincs szükségünk a *Táv* valós típusú változóra. A kifejezés kiértékelése (valójában egy beágyazott cikluson belül!) azonban növeli a program futásidejét.

A prioritási sor metódusainak megírása helyett a rendezett beillesztést és az utolsó elem kiolvasását (majd törlését) végző utasításokat belefoglalhattuk volna a pillérekre vonatkozó ciklusba.

Programunk hatékonyságát csökkenti a listaelemek folyamatos törlése, beillesztése. Írjuk meg úgy, hogy nem rendezzük a listát! Helyette maximumkiválasztással határozzuk meg a következő pillér helyét, majd módosítjuk a listaelem mezőinek értékét. Hasonlítsuk össze a futásidőket!

Lámpák

Informatika OKTV 2003. 2. forduló 3. korcsoport 3. feladat

Egy $N \times M$-es téglalap alakú téren K lámpát helyeztek el. Mindegyiknek ismerjük a helyét. Mindegyik lámpa azt a $H \times H$-s (H páratlan) négyzet alakú területet világítja be, amely átlóinak metszéspontjában áll a lámpa. A világos területek éjszaka is biztonságosak, a sötéteken azonban tanácsosabb nem járni.

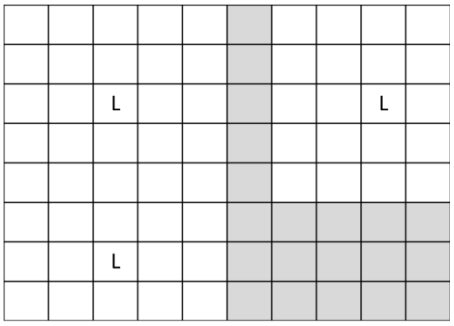
Írj programot *lampak* néven, amely megadja, hogy mekkora a téren sötétben maradt terület (a mezők száma), valamint hogy hogyan menjünk át a tér bal felső sarkából a jobb alsó sarkába a legbiztonságosabban úgy, hogy minden pozícióról a 4 oldalszomszédjára léphetünk, átlósan pedig nem léphetünk!

A *lampak.be* szöveges állomány első sorában a tér sorainak N ($1 \leq N \leq 100$) és oszlopainak M száma ($1 \leq M \leq 100$), valamint a lámpák K száma ($0 \leq K \leq 1000$) és az általuk bevilágított négyzet oldalhossza ($1 \leq H \leq 100$, H páratlan) van. A következő K sor mindegyike egy lámpa helyét tartalmazza, egy számpárt egy szóközzel elválasztva: közülük az első egy lámpát tartalmazó mező sorindexe, a második pedig az oszlopindexe. A sorokat felülről lefelé, az oszlopokat balról jobbra sorszámozzuk.

A *lampak.ki* szöveges állomány első sorába a sötétben maradt mezők számát kell írni. A második sorba azon sötét mezők száma kerüljön, ahányon minimálisan át kell menni, ha a tér bal felső sarkából a jobb alsó sarkába szeretnénk eljutni.

Példa:

```
lampak.be          lampak.ki
8 10 3 5           20
3 3                4
7 3
3 9
```



Megoldás

Rendeljünk hozzá a tér mezőihez 0-t, ha megvilágítja egy lámpa, és 1-et, ha sötétben van. Egy út mentén a hozzá tartozó mezők értékeinek az összege éppen megadja az érintett sötét mezők számát. Legyen ez az összeg az út „hossza”! Ennek minimumát keressük a bal felső mezőről a jobb alsó mezőre vezető utaknál.

Induljunk el a bal felső sarokból! Az aktuális mezőről mindig arra a szomszéd mezőre lépünk, amelyik a „legközelebb” van a kiindulási mezőhöz, azaz a legrövidebb út vezet oda. A jobb alsó mezőre jutva így valóban a legrövidebb, azaz a legkevesebb sötét mezőt tartalmazó utat fogjuk megtalálni.

A keresést megkönnyítjük, ha a bejárt mezőket prioritási sorban tároljuk. Annak a mezőnek nagyobb a prioritása, amely közelebb van a kiindulási mezőhöz (rövidebb úton érhető el). A prioritási sor biztosítja, hogy a mezőket a fenti módon válasszuk ki a keresés során.[32]

A mezőkhöz rendelt értékeket kétdimenziós tömbben tároljuk:

```
VÁLTOZÓ Tér(1…N, 1…M) MINT Egész
VÁLTOZÓ N, M, K, H MINT Egész
Be: N, M, K, H
```

A lámpák pozícióinak beolvasása előtt minden mezőt sötétnek tekintünk:

```
CIKLUS I=1-től N-ig, CIKLUS J=1-től M-ig
    Tér(I, J) = 1
CIKLUS VÉGE (J), CIKLUS VÉGE (I)
```

A pozíciók beolvasásánál módosítjuk a megvilágított, négyzet alakú tartomány mezőinek értékét:

```
VÁLTOZÓ Félhossz, X, Y MINT Egész
Félhossz = H div 2 ' a maradékos osztás hányadosa
CIKLUS I=1-től K-ig
    Be: X, Y ' egy lámpa koordinátái
    CIKLUS Sor=Max(X–Félhossz, 1)-től Min(X+Félhossz, N)-ig
        CIKLUS Oszlop=Max(Y–Félhossz, 1)-től Min(Y+Félhossz, M)-ig
            Tér(Sor, Oszlop) = 0
        CIKLUS VÉGE
    CIKLUS VÉGE
CIKLUS VÉGE
```

Figyeljük meg, hogyan kerültük ki az indexhatárokra vonatkozó feltételes elágazásokat a ciklusváltozók kezdő- és végértékének a megadásánál!

A sötét mezők megszámolását az Olvasóra bízzuk. ☺

Az egyes mezőknek a kezdőponttól mért távolságát a *Táv* tömbben tároljuk. A tömbelemek kezdőértékét –∞-nek tekintjük (a távolság biztosan nemnegatív érték lesz). A –∞ értékű tömbelem jelzi, ha az adott mezőn még nem jártunk.

A szomszédok vizsgálatát egyszerűsítjük, ha a teret határmezőkkel vesszük körül (lásd például *Programozási ismeretek haladóknak*, 186. old.). Ezen mezők értékét tetszőlegesen felvehetjük, csak ne legyen –∞ (nem léphetünk ezekre a mezőkre).

[32] Valójában egy súlyozott gráf szélességi bejárását végezzük el, de sor helyett prioritási sorban tároljuk az aktuális csúcs (mező) szomszédjait.

```
VÁLTOZÓ Táv(0…N+1, 0…M+1)
CIKLUS I=1-től N-ig, CIKLUS J=1-től M-ig
    Táv(I,J) = –∞
CIKLUS VÉGE (J), CIKLUS VÉGE (I)
' A határ mezőire:
CIKLUS I=0-tól N+1-ig
    Táv(I, 0) = –1 ' 0. oszlop
    Táv(I, M+1) = –1 ' M+1. oszlop
CIKLUS VÉGE
CIKLUS J=1-től M-ig
    Táv(0, J) = –1 ' 0. sor
    Táv(N+1), J) = –1 ' N+1. sor
CIKLUS VÉGE
```

A prioritási sorban a mezők koordinátáit és a prioritást jelző távolságot tároljuk. Ehhez struktúrát deklarálunk:

```
STRUKTÚRA TElem
    VÁLTOZÓ X, Y, Távolság MINT Egész
STRUKTÚRA VÉGE

VÁLTOZÓ PrSor MINT PrioritásiSor(Elemtípus: TElem)
```

Egy elemet az *X*, *Y* pár azonosít, a prioritást pedig a *Távolság* határozza meg (a kisebb távolság jelent nagyobb prioritást).

A keresés indításához a prioritási sorba helyezzük a bal felső mezőt, illetve megadjuk a *Táv* tömb kezdőelemének értékét:

```
PrSor.PrSorba({X=1, Y=1, Táv=Tér(1, 1)})
Táv(1, 1) = Tér(1, 1)
```

A bejárás ciklusát addig futtatjuk, amíg el nem érünk a jobb alsó mezőig. Minden egyes lépésben kiveszünk egy elemet a prioritási sorból, majd elhelyezzük a szomszéd mezőket is a sorban. A távolságok számítását eljáráshívással végezzük, melyben megadjuk az aktuális elemet és a szomszédok koordinátáinak eltérését az aktuális csúcs koordinátáitól:

```
VÁLTOZÓ Elem MINT TElem
CIKLUS AMÍG Táv(N, M) == –∞
    Elem = PrSor.PrSorból()
    TávMódosít(Elem, –1, 0) ' fölötte
    TávMódosít(Elem, 0, –1) ' balra
    TávMódosít(Elem, 1, 0) ' alatta
    TávMódosít(Elem, 0, 1) ' jobbra
CIKLUS VÉGE
```

A *TávMódosít* eljárás megvizsgálja, hogy jártunk-e már a megadott szomszéd mezőn. Ha nem, akkor meghatározza a távolságát a kezdőcsúcstól, azaz hozzáadja az aktuális elem távolságához a *Tér* tömbnek a szomszéd csúcshoz tartozó értékét. Ugyanezt az értéket beírja a *Táv* tömbbe is, majd a szomszédot beteszi a prioritási sorba.

```
ELJÁRÁS TávMódosít(Elem MINT TElem, DX, DY MINT Egész)
    VÁLTOZÓ SzomszédElem MINT TElem
    SzomszédElem = Elem
    MINŐSÍT SzomszédElem-mel
        .X = Elem.X + DX
        .Y = Elem.Y + DY
        HA Táv(.X, .Y) = –∞ AKKOR
            .Távolság = .Távolság + Tér(.X, .Y)
            Táv(.X, .Y) = .Távolság
            PrSor.PrSorba(SzomszédElem)
        ELÁGAZÁS VÉGE
    MINŐSÍTÉS VÉGE
ELJÁRÁS VÉGE
```

Ezek után nincs más dolgunk, mint kiírni a *Táv*(*N*, *M*) értékét. A teljes megoldást a *lampak* program tartalmazza. Érdemes megfigyelni az algoritmus működését úgy, hogy az útkeresés ciklusában kiíratjuk a *Táv* tömb elemeit.

Ha a *DX*, *DY* változások lehetséges értékeit tömbökben tároljuk, akkor a szomszédok vizsgálatára az ismételt eljáráshívások helyett alkalmazhatnánk ciklust. Írjuk meg így is a programot!

Hogyan módosítsuk az algoritmust, ha az összes mezőre szeretnénk tudni a legbiztonságosabb út hosszát?

Elem módosítása a prioritási sorban

Esetenként előfordulhat, hogy módosítanunk kell a prioritási sor egy elemét.[33] Ehhez általában az elemek egyértelmű azonosítására van szükség. A módosítás a *TElem* struktúra bármely mezőjére vonatkozhat (akár a prioritásra is).

A módosítást például a *PrSorba* metódus kibővítésével hajtjuk végre. A beillesztés előtt megvizsgáljuk, hogy a sor tartalmazza-e az argumentumként megadott elemet. Ha igen, akkor töröljük a sorból. Ezt követi a beillesztés az aktuális mezőkkel.

Ha az elem prioritása nem változik, akkor felesleges a törlés, elegendő a többi mezőt módosítani. Ha a prioritás változik, akkor a keresést csak lineárisan végezhetjük, hiszen a sort a prioritások szerint rendeztük.

A *Prioritás* és *Érték* mezőkkel rendelkező *TElem* típusú elem prioritásának módosításához például a következőképpen bővítjük ki a *PrSorba* metódust (az elemet az *Érték* mező alapján azonosítjuk):

```
<Kiterjesztett metódus a Lista(Elemtípus: TElem)-hez>
ELJÁRÁS PrSorba(ÚjElem MINT TElem)
    VÁLTOZÓ HolVan, MelyikElé MINT Egész
    ' Töröljük, ha már van a sorban azonos értékű elem (a prioritás módosításánál)
    HolVan = Lista.IndexKeres(FÜGGVÉNY(Elem) Elem.Érték == ÚjElem.Érték)
    HA találtunk ilyen elemet AKKOR
        Lista.Kivesz(HolVan indexű elem)
    ELÁGAZÁS VÉGE
```

[33] Lásd például a Prim-algoritmust a *Programozási ismeretek versenyzőknek* példatár 171–173. oldalán.

```
    ' Az aktuális prioritással beillesztjük a sorba
    MelyikElé = Lista.IndexKeres(FÜGGVÉNY(Elem) Elem.Prioritás≥ÚjElem.Prioritás)
    HA nincs nagyobb prioritású elem AKKOR
        ' a Lista végére illesztjük be (akkor is, ha üres a Lista)
        Lista.Add(ÚjElem)
    EGYÉBKÉNT
        Lista.Beilleszt(MelyikElé, ÚjElem)
    ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

A prioritás módosítását a *prsormodosit* program mutatja be, amely a diákok nevét és magasságát prioritási sorban tárolja a magasság alapján, majd Farsang Jakab magasságát (azaz a prioritását) 174 cm-re módosítja. Vegyük észre, hogy nem használjuk fel az eredeti magasság értékét!

Megjegyezzük, hogy a *Hidak* feladatban valójában az elemek prioritását módosítottuk. Azonban mindig a legnagyobb prioritású elem került sorra, így nem volt szükség a keresésre.

További feladatok a prioritási sorra

Prioritási sorra gyakran a súlyozott gráfokra vonatkozó versenyfeladatok megoldásánál van szükség. Az alábbiakban bejelöltük a gráfalgoritmusok alkalmazását igénylő feladatokat. A gráfalgoritmusok ismertetését lásd a *Programozási ismeretek haladóknak* tankönyvben, illetve a *Programozási ismeretek versenyzőknek* példatárban.

OITV 2000-2f2	Lift
OKTV 1998-3f3	Kommandó (Dijkstra-algoritmus)
OKTV 2000-1f3	Prioritási sor
OKTV 2000-2f3	Lift
OKTV 2000-3f3	Térkép (gráfbejárás)
OKTV 2002-1f3	Prioritási sor
OKTV 2005-3f3	Malom (Prim-algoritmus)

10. Befejezés

Kapcsolódó lecke:
Programozási ismeretek haladóknak (Műszaki Kiadó, 2012)
43. Az algoritmusok bonyolultsága

Az előző leckékben bemutattuk, hogyan lehet a modern programozási nyelvek által biztosított kollekciókat felhasználni a rövidebb, egyszerűbb, könnyebben áttekinthető programok írásához. Többször hangsúlyoztuk, hogy ezek a szempontok gyakran a hatékonyság rovására mennek, azaz a futásidő, a memóriafelhasználás megnő. Általában is találkozunk egymásnak ellentmondó igényekkel. Az adott feladat dönti el, hogy melyiket tekintsük fontosabbnak, hogyan biztosítsuk az egyensúlyt közöttük.

A befejező részben megvizsgáljuk az adatszerkezetek és a futásidő kapcsolatát, majd bemutatunk egy bonyolultabb adatszerkezetet felhasználó alkalmazást.

Az adatszerkezet választásának hatása a futásidőre

A különböző típusú kollekciók polinomiális futásidőben, általában lineáris vagy négyzetes nagyságrendben átalakíthatók egymásba. Ezért – elvileg – az algoritmus futásidejének nagyságrendjét nem befolyásolja lényegesen az alkalmazott adatszerkezet. A megfelelő adattípus inkább az algoritmus bonyolultságát csökkenti.

A megoldáshoz alkalmazott adatszerkezet kiválasztásának további szempontja lehet a kollekció metódusainak hatékonysága. A futásidők nagyságrendjét a .NET 4-re vonatkozóan az alábbi táblázatban foglaltuk össze.

	Count	*Add/Insert/Push/ Enqueue*	*Clear*	*Contains/ Find/Findindex*	*Remove/Pop/ Dequeue*	*ElementAt/Item*
Halmaz	*O*(1)	*O*(1) [*O*(*n*)]	*O*(*n*)	*O*(1)	*O*(1)	n.a.
Rendezett halmaz	*O*(1)	n.a.	*O*(*n*)	*O*(log *n*)	*O*(1)	n.a.
Lista	*O*(1)	Add: *O*(1) [*O*(*n*)] Insert: *O*(n)	*O*(*n*)	*O*(*n*)	*O*(*n*)	*O*(1)
Rendezett lista	*O*(1)	*O*(log *n*) a végéhez, *O*(*n*) belülre	*O*(*n*)	*O*(log *n*) kulcsra, *O*(*n*) értékre	*O*(*n*)	*O*(log *n*), ha már létezik a kulcs
Szótár	*O*(1)	*O*(1) [*O*(*n*)]	*O*(*n*)	*O*(1) kulcsra, *O*(*n*) értékre	*O*(1)	*O*(1)
Rendezett szótár	*O*(1)	*O*(log *n*)	*O*(1)	*O*(log *n*) kulcsra, *O*(*n*) értékre	*O*(log *n*)	*O*(log *n*)
Verem	*O*(1)	*O*(1) [*O*(*n*)]	*O*(*n*)	*O*(*n*)	*O*(1)	Peek: *O*(1)
Sor	*O*(1)	*O*(1) [*O*(*n*)]	*O*(*n*)	*O*(*n*)	*O*(1)	Peek: *O*(1)

Néhány metódus futásidejének nagyságrendje az egyes kollekcióknál (.NET 4)
A futásidő szögletes zárójelben áll, ha közben bővíteni kell a kollekció kapacitását.

A *First()* és a *Last()* metódusok futásidejére sajnos nincs utalás az MSDN-ben.

A szótár és a rendezett szótár közti különbségek az eltérő realizációból erednek. A szótár hasítótáblában kerül tárolásra, a rendezett szótár pedig kiegyensúlyozott fa formájában.

A rendező metódusok futásideje függ az alkalmazott algoritmustól. A .NET gyorsrendező (quick sort) algoritmust használ, melynek átlagos futásideje $O(n \cdot \log n)$, de legrosszabb esetben elérheti az $O(n^2)$-t is.

A futásidőre érzékeny feladatok megoldásánál természetesen nemcsak a nagyságrend, hanem akár egy konstans szorzó is számít. Példaként vizsgáljuk meg 100000 véletlenszerűen választott egész szám páros és páratlan elemekre történő szétválogatásának futásidejét!

1. Tárolás listákban, a feldolgozott elem törlése a lista elejéről

```
VÁLTOZÓ Számok, Páratlan, Páros MINT Lista(Elemtípus: Egész)
CIKLUS AMÍG Számok.Elemszám > 0
   HA Számok.Első páros AKKOR Páros.Add(Számok.Első)
   EGYÉBKÉNT Páratlan.Add(Számok.Első)
   ELÁGAZÁS VÉGE
   Számok.Kivesz(0)
CIKLUS VÉGE
```

2. Tárolás listákban, a feldolgozott elem törlése a lista végéről

```
VÁLTOZÓ Számok, Páratlan, Páros MINT Lista(Elemtípus: Egész)
CIKLUS AMÍG Számok.Elemszám > 0
   HA Számok.Utolsó páros AKKOR Páros.Add(Számok.Utolsó)
   EGYÉBKÉNT Páratlan.Add(Számok.Utolsó)
   ELÁGAZÁS VÉGE
   Számok.Kivesz(Számok.Elemszám–1)
CIKLUS VÉGE
```

3. Tárolás listákban, feldolgozás For ciklussal, törlés nélkül

```
VÁLTOZÓ Számok, Páratlan, Páros MINT Lista(Elemtípus: Egész)
CIKLUS I=0-tól Számok.Elemszám–1-ig
   HA Számok(I) páros AKKOR Páros.Add(Számok(I))
   EGYÉBKÉNT Páratlan.Add(Számok(I))
   ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

4. Tárolás statikus tömbökben, feldolgozás For ciklussal

```
VÁLTOZÓ Számok(100000), Páratlan(100000), Páros(100000) MINT Egész
VÁLTOZÓ Darab1, Darab2 MINT Egész
Darab1 = 0
Darab2 = 0
CIKLUS I=1-től 100000-ig
   HA Számok(I) páros AKKOR
      Darab1 +=1
      Páros(Darab1) = Számok(I)
   EGYÉBKÉNT
      Darab2 +=1
      Páratlan(Darab2) = Számok(I)
   ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A *For* ciklusnál iterátort is használhatunk az elemek indexelése helyett.

100 véletlenszerűen választott adatsorra az átlagos futásidőket Visual Basic illetve C++ esetén az alábbi táblázatok tartalmazzák (a futásidők erősen függnek a hardver- és szoftverkörnyezettől!). Megjegyezzük, hogy a Visual Basic a *List* adatszerkezetet

lényegében tömbként tárolja, míg a *LinkedList* kétirányú lista. A C++-ban a *forward_list* egyirányú, a *list* pedig kétirányú lista.

Adatszerkezet	Törlés a lista elejéről (1)	Törlés a lista végéről (2)	For ciklus indexeléssel (3, 4)	For ciklus iterátorral (3, 4)
List *First*/*Last* *ElementAt*	2250 ms 2240 ms	4,6 ms 6,6 ms	1,2 ms	1,6 ms
LinkedList *First*/*Last* *ElementAt*	2,3 ms 1,6 ms	6,0 ms –	–	5,5 ms
Tömb	–	–	0,7 ms	0,7 ms

Átlagos futásidők a szétválogatásnál (Visual Basic)

Adatszerkezet	Törlés a kollekció elejéről (1)	Törlés a kollekció végéről (2)	For ciklus indexeléssel (3, 4)	For ciklus iterátorral (3, 4)
vector *begin*/*rbegin* *indexelés*	2255 ms 2242 ms	0,7 ms 0,8 ms	0,8 ms	0,9 ms (*)
list (*begin*/*rbegin*)	11 ms	12,6 ms	–	9,5 ms
forward_list (*begin*)	11 ms	–	–	8,7 ms
array	–	–	0,5 ms	0,4 ms (*)
tömb	–	–	0,4 ms	0,4 ms (*)
dinamikus tömb (tömb mutatóval)	–	–	0,5 ms	–

(*): C++11 szabvány

Átlagos futásidők a szétválogatásnál (C++)

Bár „dinamikus tömböt” használva (VB: *list*, C++: *vector*) az 1. módszer futásideje jelentősen nagyobb, de a forráskód rövidebb és egyszerűbb logikájú, különösen a 4. módszerhez viszonyítva. Ismételten kiemeljük, hogy az adott körülmények határozzák meg, mely szempontok szerint optimalizáljuk a programot.

A Nemes Tihamér versenyeken és az OKTV-n általában nyugodtan használhatjuk az 1. módszert. A futásidő még a nagyon sok adatot tartalmazó teszteseteknél is bőven a megengedett 1 percen belül marad (általában csak néhány másodperc). A diákolimpián, a nemzetközi versenyeken vagy gyakran a KöMaL feladatokban azonban néhány másodperces futásidőt várnak el. Ilyenkor alkalmazzunk statikus (vagy dinamikus) tömböt és ciklusváltozót!

Jegyek

A kollekciók használatának bemutatását egy bonyolult adatszerkezet definiálásával, egy elektronikus napló készítésével zárjuk. A naplóban diákok jegyeit tároljuk az egyes tantárgyakból. A diákok nem feltétlenül ugyanazokat a tantárgyakat tanulják, és egy diák tetszőleges számú jegyet kaphat egy-egy tantárgyból. Feljegyezzük az osztályzat megszerzésének dátumát is.

A jegy–dátum párokhoz struktúrát definiálunk (az ábrát lásd a következő oldalon):

```
STRUKTÚRA TOsztályzat
    VÁLTOZÓ Jegy MINT Egész, Dátum MINT DátumIdő
STRUKTÚRA VÉGE
```

Egy tantárgy osztályzatait célszerű dinamikus tömbben tárolni:

```
VÁLTOZÓ Osztályzatok MINT DinamikusTömb(Típus: TOsztályzat)
```

> Ha egy diák ugyanabból a tantárgyból ugyanazon a napon legfeljebb egy jegyet kaphatna, akkor a struktúra és a dinamikus tömb helyett használhatnánk rendezett listát (*SortedList*).

Egy diák tantárgyankénti osztályzatait rendezett szótárban gyűjtjük. A szótárelemek kulcsa a tantárgy neve, értéke pedig az osztályzatok dinamikus tömbje:

```
VÁLTOZÓ Tantárgyak MINT RendezettSzótár(Kulcstípus: Sztring,
                                        Értéktípus: DinamikusTömb(Típus: TOsztályzat))
```

A diákokat a jegyekkel együtt szintén rendezett szótárban tartjuk nyilván. A szótárelemek kulcsa a diák neve, értéke pedig tantárgyainak rendezett szótára:

```
VÁLTOZÓ Diákok MINT RendezettSzótár(Kulcstípus: Sztring,
                                    Értéktípus: RendezettSzótár(Kulcstípus: Sztring,
                                                Értéktípus: DinamikusTömb(Típus: TOsztályzat)))
```

Egy diák egy tantárgyának osztályzatait tartalmazó dinamikus tömbre a következő formában hivatkozhatunk:

```
Diákok(DiákNév)(TantárgyNév)
```

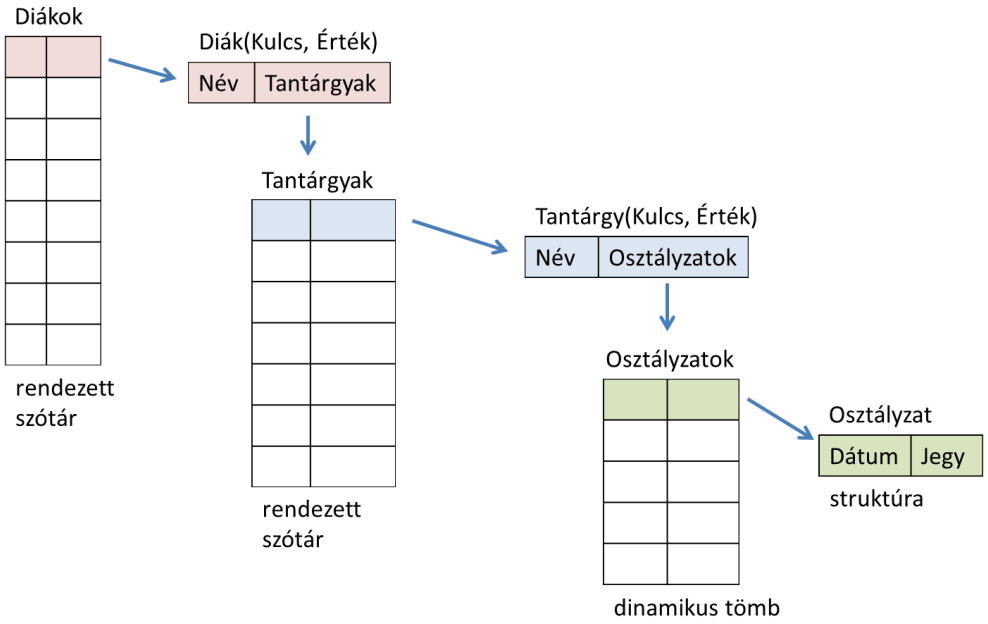
A tömbelemek struktúrák, így a jegyek kiírása például:

```
CIKLUS MINDEN Osztályzat-ra a Diákok(DiákNév)(TantárgyNév)-ben
    Ki: Osztályzat.Dátum, Osztályzat.Jegy
CIKLUS VÉGE
```

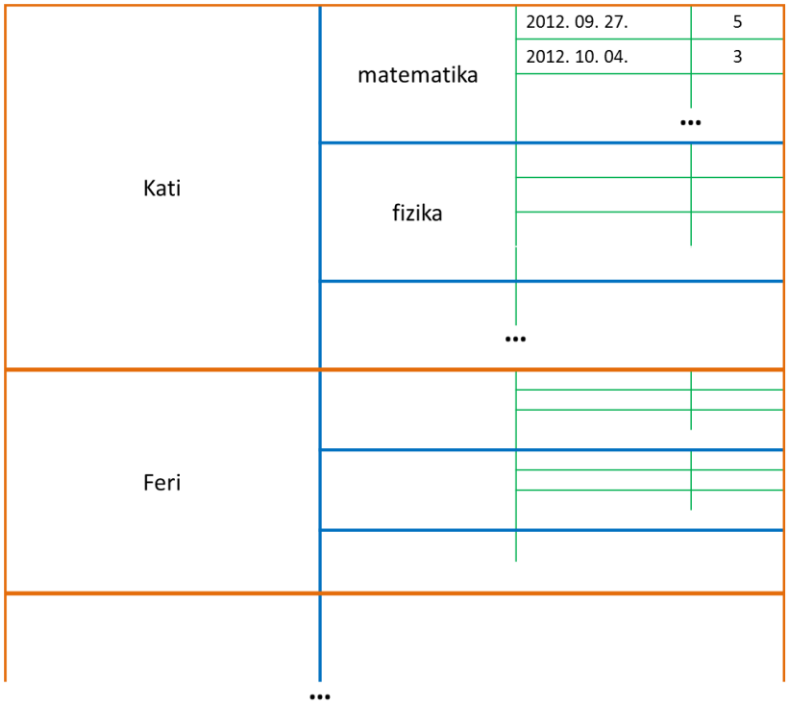
A *jegyek* program tartalmazza a szótárak/tömbök feltöltését, illetve az adatok lekérdezését és kiírását. Figyeljük meg a hivatkozásokat az egyes elemekre! Egészítsük ki a programot az adatok billentyűzetről történő beolvasásával!

> A C++ változatok bemutatják a *széles sztringek* (*wstring*), illetve az UTF-8 kódolású sztring literálok használatát. Bár az ANSI–UTF-8 átalakítást a C++11 szabvány már támogatja, a GCC még nem implementálta. Ezért a konvertáláshoz saját függvényeket definiáltunk.

A programok betöltéséhez állítsuk át a CodeBlocks editorát UTF-8 kódolásra (*Settings*/*Editor*/*Other Settings*, *Ecoding*)! Az ékezetes karakterek akkor jelennek meg helyesen a konzolablakban, ha TrueType betűtípust választunk a megjelenítéshez (*Consolas*, *Lucida Console* stb. az ablakmenü *Tulajdonságok*, illetve *Alapértelmezések* parancsával).



A diákok jegyeinek tárolásához használt adatszerkezetek

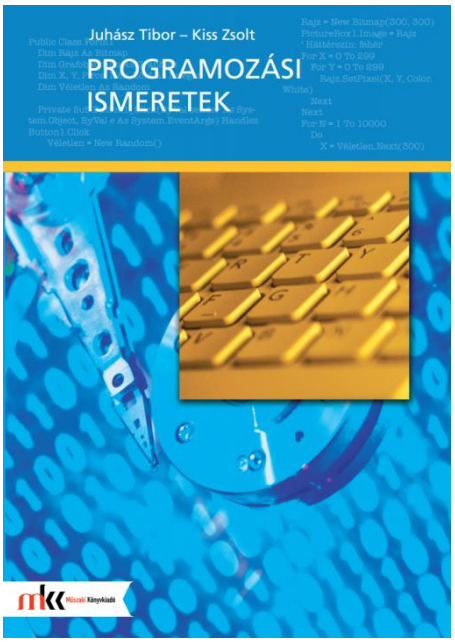


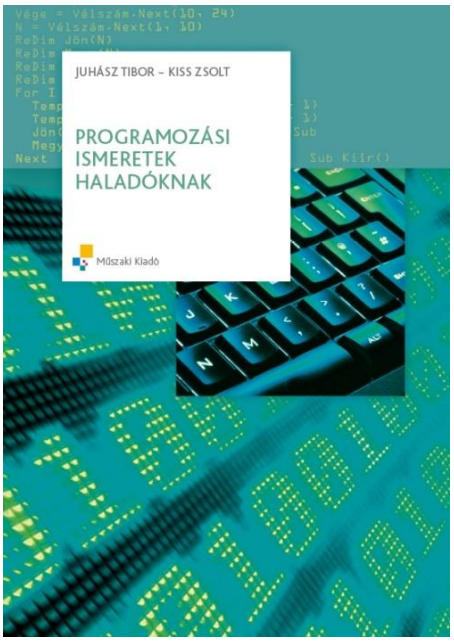
A Diákok szótár Név–Tantárgyak párokat,
a Tantárgyak szótár Tantárgynév–Osztályzatok párokat,
az Osztályzatok dinamikus tömb pedig Dátum–Jegy párokat tartalmaz.

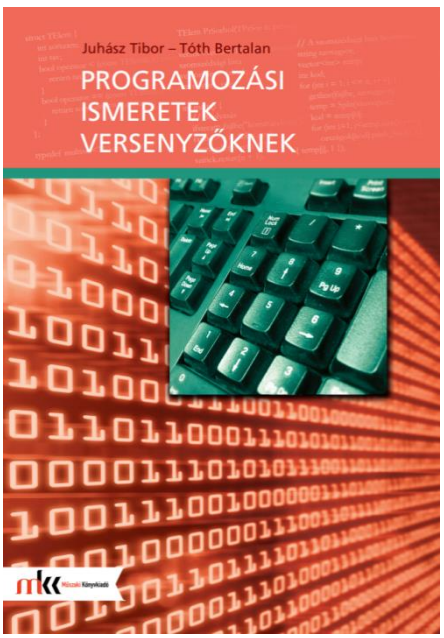
Függelék

A leckékben megoldott versenyfeladatok jegyzéke

A versenyfeladatok jelölése: a tanév 2. félévének éve, forduló, korcsoport.







Juhász Tibor – Kiss Zsolt:
Programozási ismeretek
(Műszaki Kiadó, 2011, 2015)

Juhász Tibor – Kiss Zsolt:
Programozási ismeretek haladóknak
(Műszaki Kiadó, 2012)

Juhász Tibor – Tóth Bertalan:
Programozási ismeretek versenyzőknek
(Műszaki Kiadó, 2015)

www.zmgzeg.sulinet.hu/programozas