Maximumkeresés és rendezés

Maximumkeresés

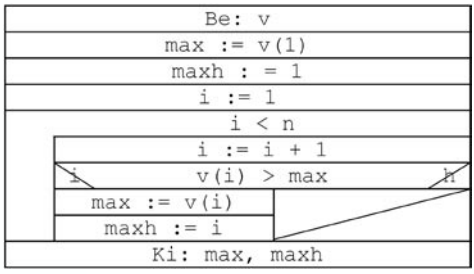
*A **maximumkeresés*** feladata*: Adott az *n* elemű *v* vektor. Határozzuk meg a vektor legnagyobb elemét! Jelöljük ezt az értéket *max*-szal!

Megjegyzés: Természetesen elképzelhető, hogy a vektorban több olyan elem is van, amelynek értéke az a bizonyos *max*.

A megoldás gondolatmenete: Az eljárás hasonlít ahhoz, ahogy pl. egy kutyafalka kiválasztja a legerősebbet. Amíg egy kutya egyedül van, addig ő a legerősebb. Ha csatlakozik hozzá egy másik, megvívnak egymással, s a győztes a legerősebb. A következő csatlakozó már csak a győztessel vív meg, stb.

Számítástechnikai megközelítésben az eljárás így fogalmazható meg: Vesszük az első elemet, és feltesszük, hogy az a legnagyobb. Ezután megvizsgáljuk a következő elemet, s ha az nagyobb az eddigi maximumnál, feltesszük, hogy az a legnagyobb. Ezt az eljárást addig ismételgetjük, amíg van még nem vizsgált eleme a vektornak. Az éppen legnagyobbnak tartott elem a *maxh*-adik, értéke pedig *max*. Az eljárás végén *max* a legnagyobb elem értékét jelenti, *maxh* pedig az elsőnek megtalált *max* értékű elem indexét.

```
Eljárás MaxKer
     Be: v

     max = v(1)
     maxh = 1
     Ciklus i = 2-től n-ig
           Ha v(i) > max akkor
                 max = v(i)
                 maxh = i
           Elágazás vége
     Ciklus vége
     Ki: max, maxh
Eljárás vége
```



Rendezés maximumkereséssel

Igen gyakori feladat, hogy egy vektor elemeit növekvő vagy csökkenő sorba kell rendeznünk. Eddigi ismereteink alapján erre ugyan felhasználhatjuk a *ListBox* vezérlőt, amely a benne elhelyezett elemeket a *Sorted* [szortid] tulajdonságának *True*-ra állításával automatikusan rendezi, de érdemes önállóan is elkészíteni egy rendező eljárást, mivel azt más nyelvek esetében is ki tudjuk használni.

*A **rendezés*** feladata*: Adott az *n* elemű *v* vektor. Rendezzük növekvő sorba az elemeit!

A megoldás gondolatmenete: Válasszuk ki az *n* elemből a legnagyobbat, és cseréljük meg az utolsóval; majd válasszuk ki az első *n–1*-ből a legnagyobbat, és cseréljük meg az utolsó előttivel; majd az első *n–2*-ből válasszuk ki a legnagyobbat, és cseréljük meg az *n-2*-dikkel, stb. A módszer lényege tehát ***rendezés maximumkereséssel****.

Látható, hogy első lépésben helyére kerül a legnagyobb elem, ezért a továbbiakban már nem vizsgáljuk. Ezt követően helyére kerül az utolsó előtti, s ezt a továbbiakban nem vizsgáljuk, stb.

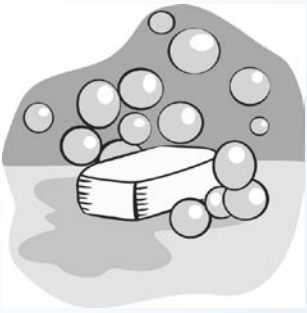
Az eljárás lényege tehát egy külső ciklus, melyben a ciklusváltozó „hátulról halad előre", a ciklusmagban pedig egy maximumkeresés és egy csere szerepel:

```
Eljárás Rendezés_maximumkereséssel

     Ciklus j=n-től 2-ig -1-egyével
           MaxKer (az első j elemben)
           Csere (v(j), maxh)
     Ciklus vége

Eljárás vége
```

Mivel mind a *MaxKer*, mind a *Csere* algoritmusa ismert, ezért ezeket az eljárásban részletezni már újra nem szükséges.

Rendezés egyszerű cserével (buborékmódszer)

A ***rendezés egyszerű cserével**** vagy buborékmódszer* talán a legközismertebb rendezési eljárás, s bár az algoritmus viszonylag lassú, az eljárás könnyen kódolható.



Ha a vektort növekvő módon szeretnénk rendezni, eljárhatunk a következő módon. Hasonlítsuk össze a vektor első és második elemét, s ha az első nagyobb, cseréljük meg őket. Ezt követően hasonlítsuk össze a vektor második és harmadik elemét, s ha a második nagyobb, cseréljük meg őket, és így tovább. Látható, hogy ekkor a vektor legnagyobb eleme biztosan a helyére kerül, még akkor is, ha ő volt az első.

Sajnos ez a többi elemre nem feltétlenül teljesül, tehát ahhoz, hogy a második legnagyobb elem a helyére kerüljön, ismét el kell indulnunk a vektor elejéről. Az első elemet össze kell hasonlítani a másodikkal, a másodikat a harmadikkal, stb., azonban most már elég elmenni az utolsó előtti elemig, hiszen az utolsó a helyén van.