I. Elemi algoritmusok

A számítógépes feladatmegoldás során az algoritmus megtervezésekor bizonyos elemi tevékenységek gyakran felmerülnek megoldandó feladatként. Az ezeket megoldó algoritmusokat mutatjuk meg a következőkben. Az algoritmusokat mondatszerű leírással adjuk meg.

Elsőként olyan algoritmusokkal foglalkozunk, amelyek egy sorozatból egy adatot állítanak elő.

Elemek összegzése

Adott egy *N* elemű *e* számsorozat. Határozzuk meg az elemek összegét! A végeredményt az *s* tartalmazza.

ÖSSZEGZÉS
  s := 0
  CIKLUS *i* **:= 1-TŐL** *N*-**IG**
    s := *s* + *e*[*i*]
  CIKLUS VÉGE
ELJÁRÁS VÉGE

Adott tulajdonság keresése

Adott egy *N* elemű *e* sorozat, és az elemein értelmezett *T* tulajdonság. Döntsük el, hogy van-e legalább egy olyan elem a sorozatban, amely rendelkezik a *T* tulajdonsággal!

ELDÖNTÉS
  talált:=0
  i := 1
  CIKLUS AMÍG *i*<=*N* **ÉS** *e*[*i*] **NEM** *tulajdonságú*
  i := *i*+1
  CIKLUS VÉGE
  HA *i* <= *N* **AKKOR** *talált:*=1
  ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE

Kiválasztás

Adott egy *N* elemű *e* sorozat. Tudjuk, hogy a sorozat valamely eleme rendelkezik a *T* tulajdonsággal. Határozzuk meg az első ilyen elem sorszámát!

KIVÁLASZTÁS
  i := 1
  CIKLUS AMÍG *e*[*i*] **NEM** *tulajdonságú*
    i := *i*+1
  CIKLUS VÉGE
  sorszám := *i*
ELJÁRÁS VÉGE

Maximum kiválasztása

Az előbbi feladat egy speciális eseteként, egy adott sorozat legnagyobb (vagy esetleg legkisebb) elemének megkeresése is gyakran előfordul.

Adott *e* sorozatból keressük ki a legnagyobb elemet!

```
MAXIMUM
   sorszám := 1
   CIKLUS i := 2-TŐL N-IG
      HA e[sorszám] < e[i] AKKOR sorszám := i
      ELÁGAZÁS VÉGE
   CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Megszámlálás

A tulajdonság előfordulása mellett az is kérdés lehet, hogy hány elem rendelkezik ezzel az adott tulajdonsággal. Legyen adott egy *N* elemű *e* sorozat és egy *T* tulajdonság. Határozzuk meg a *T* tulajdonsággal rendelkező elemek számát!

```
MEGSZÁMLÁLÁS
   darab := 0
   CIKLUS i := 1-TŐL N-IG
      Ha e[i] T tulajdonságú AKKOR darab := darab+1
      ELÁGAZÁS VÉGE
   CIKLUS VÉGE
ELJÁRÁS VÉGE
```

II. Keresések

Az előző feladatok összefoglalását adja a keresés. Nem rendelkezünk azzal az ismerettel, hogy egy adott tulajdonságú elem benne van-e a sorozatban. Ha igen, akkor meg akarjuk tudni a sorszámát is. A keresésre két módszer is használatos. A *lineáris keresést* akkor alkalmazzuk, ha a sorozat elemei nem rendezettek a tulajdonságnak megfelelően, míg a *bináris* vagy *logaritmikus keresést* akkor, ha a rendezettség fennáll.

Lineáris keresés

Adott egy *N* elemű e sorozat, és egy *T* tulajdonság. Döntsük el, hogy van-e olyan eleme a sorozatnak, amely rendelkezik a *T* tulajdonsággal, és ha van, akkor hányadik! (Ha nincs ilyen tulajdonságú elem, akkor a sorszám értéke legyen 0.)

```
LINEÁRIS_ KERESÉS
   sorszám := 0
   i :=1
   CIKLUS AMÍG i <= N ÉS e[i] NEM T tulajdonságú
      i := i+1
   CIKLUS VÉGE
   HA i <= N AKKOR sorszám := i
   ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

Strázsás keresés

Adott *N* elemű e sorozatban keressük azt az elemet, amelynek értéke *q*.
Ha a sorozat *N*+1-dik elemeként felvesszük a keresett elemet (*q*-*t*), akkor nem kell figyelni, hogy a sorozat végére értünk-e. Az *N*+1-dik elem "őrködik", és biztosítja, hogy ne haladjunk túl a sorozaton.

```
STRÁZSÁS_KERESÉS
   e(N+1) := q
   sorszám := 0
   i := 1
   CIKLUS AMÍG e[i] <> q
      i := i+1
   CIKLUS VÉGE
   HA i <= N AKKOR sorszám := i
   ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

A strázsás keresés nagy elemszámú sorozatok esetén hatékonyabb az egyszerű lineáris keresésnél, mert a ciklusfeltétel kiértékelése lényegesen rövidebb.

Logaritmikus keresés

A rendezett sorozatban való keresés algoritmusa az előbbitől lényegesen eltérő.
Adott egy növekvő sorrendbe rendezett *N* elemű e sorozat. Keressük meg azt az *i*-t, amelyre *e*[*i*] = *x* (*T* tulajdonság)! Elsőként a teljes intervallum felezőpontjában lévő elemet nézzük meg, hogy egyenlő-e *x*-szel. Ha nem azonos *x* a felezőpontban talált elemmel, akkot két eset lehet. Ha nagyobb, akkor a felső részintervallumot felezzük a következő lépésben, ha kisebb, akkor pedig az alsót. Ezt folytatjuk, míg az intervallumok hossza 1 nem lesz.

```
LOGARITMIKUS_KERESÉS
   sorszám := 0
   alsó := 1
   felső := N
   CIKLUS
      közép:=EgészRész((alsó+felső) /2)
      HA e[közép]< x AKKOR alsó:= közép+1
      ELÁGAZÁS VÉGE
      HA e[közép]< x AKKOR felső:= közép -1
      ELÁGAZÁS VÉGE
   AMÍG alsó <= felső ÉS  e[közép] <> x
   CIKLUS VÉGE
   HA alsó <= felső AKKOR  sorszám := közép
   ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

A következő algoritmus egy sorozatból egy másik sorozatot állít elő: meghatározzuk egy sorozat összes *T* tulajdonságú elemét.

Kiválogatás

Egy sorozat elemei közül azokat válogatjuk ki egy új sorozatba, amelyek megfelelnek egy adott tulajdonságnak. Legyen adott egy *N* elemű *e* sorozat és a *T* tulajdonság. Gyűjtsük ki az *a* sorozatba a *T* tulajdonságú *e*-beli elemeket!

```
KIVÁLOGATÁS
   j := 0
   CIKLUS i-TŐL N-IG
      HA e[i] T tulajdonságú AKKOR j := j+1
                                   a[j] := e[i]
      ELÁGAZÁS VÉGE
   CIKLUS VÉGE
ELJÁRÁS VÉGE
```

III. Rendezések

Gyakran a rendezett elemekkel való műveletvégzés (pl. egy adott tulajdonság keresése) sokkal hatékonyabb, mint rendezetlen elemek esetében. Ezért nagyon fontosak a sorozat elemeit más sorrendbe előállító algoritmusok. Ezeket az algoritmusokat adott tulajdonság szerinti rendezésnek nevezzük. A többféle rendezési módszer közül a legismertebbek algoritmusát adjuk meg.

Két elem cseréjét a következőkben az alábbi módon jelöljük: **EGYIK ↔ MÁSIK**. A csere megvalósítása például a következő módon történhet:

```
EGYIK ↔ MÁSIK
    segéd := egyik
    egyik := másik
    másik := segéd
ELJÁRÁS VÉGE
```

Minimum-elven alapuló rendezés

A minimumkereséses rendezés lényege az, hogy egy adott intervallumban - amely kezdetben a teljes sorozat - megkeressük a legkisebb elemet, s azt tesszük az első helyre. A következő lépésben eggyel szűkítjük az intervallumot (megnöveljük a sorozat alsó határát jelölő indexet), és ismét a legkisebb elemet visszük előre. Végül a rendezendő adatok elfogynak, a sorozat végére érünk. A mindenkori legkisebb elem keresése lineáris módszerrel történik.

```
MINIMUM_ELV
  CIKLUS i:=1-TŐL n-1-IG
    legkisebb := e[i]; sorszáma := i;
    CIKLUS j := i+1-TŐL N-IG
      HA legkisebb > e[j] AKKOR
        legkisebb := e[j]
        sorszáma := j;
        e[sorszáma] ↔ e[i];
      ELÁGAZÁS VÉGE
    CIKLUS VÉGE
  CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Buborék rendezés

A buborék rendezés mindig két szomszédos elemet vizsgál, s ha azok a kívánt rendezettségnek nem felelnek meg, felcseréli őket, majd tovább lépve újabb elempárt vizsgál. Az elejétől a végéig ismételten végigpásztázva az egész sorozatot, eljutunk a rendezett állapotig, amikor már nincs szükség cserére.

```
BUBORÉK_ELV
   CIKLUS i:=1-TŐL N-1-IG
      CIKLUS j:= i-TŐL N-1-IG
         HA e[j] > e[j+1] AKKOR e[j] ↔ e[j+1]
         ELÁGAZÁS VÉGE
      CIKLUS VÉGE
   CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Javított buborékos rendezés

Ha a buborékos rendezésnél egy teljes belső ciklus lefutása alatt egyetlen csere sem volt, akkor a sorozat már rendezett. Ilyenkor feleslegesek a további összehasonlítások.

```
JOBB_BUBORÉK
   csere := 1
   CIKLUS AMÍG csere = 1
      CSERE := 0
      CIKLUS j := 1-TŐL N-1-IG
         HA e[j]>e[j+1] AKKOR e[j] ↔ e[j+1]
                              CSERE :=1
         ELÁGAZÁS VÉGE
      CIKLUS VÉGE
   CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Rendezés a módosított Buborék-elv alapján

Alapelve megegyezik a buborék rendezéssel azzal a különbséggel, hogy a pásztázás nem egyirányú, hanem váltakozó, és a pásztázás intervallumát mindkét irányban folyamatosan csökkentjük. Az első pásztázás után a legnagyobb elem a sorozat végére, vagyis végleges helyére kerül, és az intervallum felső határát eggyel lecsökkentjük. A következő lépésben visszafelé végezzük el a pásztázást, így a legkisebb elem a sorozat elejére kerül, az alsó határt pedig eggyel megnöveljük. A rendezett állapotig elvezető pásztázás mindaddig tart, amíg a pásztázási intervallum le nem csökken két elemre.

```
MÓDOSÍTOTT_BUBORÉK_ELV
    alsó := 2
    felső := N
    segéd := N
    CIKLUS
        CIKLUS j:= felső-TŐL alsó-IG -1-ESÉVEL
            HA e[j-1] > e[j] AKKOR
                e[j-1] ↔ e[j]
                segéd := j
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
        alsó := segéd+1
        CIKLUS j:= alsó -TÓL felső -IG
            HA e[j-1]>e[j]AKKOR
                e[j-1] ↔ e[j]
                segéd := j
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
        felső := segéd -1
    AMÍG alsó > felső
    CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Beszúró rendezés

A rendezés során sorrendbeli hibát keresünk. A már rendezett részsorozatba illesztjük a következő elemet, amíg a teljes sorozat rendezetté válik.

```
BESZÚRÓ_ELV
   CIKLUS i := 2-TŐL N-IG
      segéd := e[i]
      j := i
      CIKLUS AMÍG j > 1 ÉS segéd < e[j-1]
            e[j] := e[j-1]
            j := j-1
      CIKLUS VÉGE
      e[j] := segéd
   CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Összefésülés

Két rendezett sorozat egyesítése oly módon, hogy az egyesítés során keletkező sorozat is rendezett legyen.

A két növekvő sorrendben rendezett sorozat legyen az M elemű a ill. az N elemű b sorozat. Az eredmény sorozat c, elemszáma legfeljebb $M+N$. Ha egy elem mindkét sorozatban szerepel, csak egyszer másoljuk át az új sorozatba. Az összefésülést strázsával valósítjuk meg. A strázsa az adott adattípus legnagyobb eleme lesz.

```
ÖSSZEFÉSÜLÉS
   a[M+1] := strázsa
   b[N+1] := strázsa
   i :=1, j := 1, k := 0
   CIKLUS AMÍG i < M+1 VAGY j < N+1
      k := k+1
      ELÁGAZÁS
         a[ i] < b[ j] ESETÉN c[ k] := a[ i],  i := i+1
         a[ i] = b[ j] ESETÉN c[ k] := a[ i],  i := i+1, j := j+1
         a[ i] > b[ j] ESETÉN c[ k] := b[ j],  j := j+1
      ELÁGAZÁS VÉGE
   CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Rekurzív algoritmusok

Rekurzív algoritmus az, amely közvetlenül vagy közvetve hívja önmagát.
Rekurzív algoritmusok használata akkor lehet indokolt, ha a megoldandó probléma vagy a feldolgozásra váró adatok struktúrája eleve rekurzív. A rekurzív algoritmussal megoldott legismertebb problémák közül a Hanoi tornyait és a gyorsrendezést kell megemlítenünk. Sok olyan feladat is van, amelyet rekurzívan megoldani kényelmes, de nem érdemes.
Ilyenek pl. a faktoriálisok és a Fibonacci-számok előállítása.

Gyorsrendezés (quicksort)

Azért kapta ezt a nevet, mert a tömbökre ismert rendezések közül a leghatékonyabb.
Megalkotója C.A. R. Hoare (1960.).
Az elv a következő:

- osszuk két részre a rendezendő sorozatot úgy, hogy az egyik rész minden eleme kisebb legyen a másik rész minden eleménél!
- a két részre külön-külön ismételjük meg az előző lépést, amíg mindkét rész 0 vagy 1 elemű lesz.

A feldolgozási műveletet tehát egy szétválogatás, amelynek során egy megadott értékhez (pl. a sorozat középső eleméhez) viszonyítva a tőle kisebb elemeket elé, a nagyobbakat mögé helyezzük.
A megvalósítás:

az *a[n]* sorozat kiválasztott eleme legyen *x*. Balról indulva keressük meg az első $a[i] > x$ elemet, majd jobbról kezdve az első olyat, amelyre $a[j] < x$. Cseréljük fel ezt a két tételt, majd folytassuk a keresés-felcserélés műveletet mindaddig, amíg a két irányú pásztázás nem találkozik valahol a tömb közepén. A cserében a kiválasztott elem is részt vesz.

```
QUICKSORT (alsó, felső)
    i := alsó , j := felső  x : =a[(alsó+ felső) /2]
    CIKLUS AMÍG  i < j
            CIKLUS AMÍG a[i] < x
                i := i+1
            CIKLUS VÉGE
            CIKLUS AMÍG a[j] > x
                j := j-1
            CIKLUS VÉGE
            HA i < j AKKOR  a[i]↔ a[j]
                i := i+1, j := j-1
            ELÁGAZÁS VÉGE
    CIKLUS VÉGE
    HA alsó < j AKKOR
                QUICKSORT (alsó, j)
    ELÁGAZÁS VÉGE
    HA  i < felső AKKOR
                QUICKSORT (i, felső)
    ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

Rekurzív adatszerkezetek

Definiálhatók rekurzív módon (pl bináris fa), méretük nem állandó.
Egy bináris fa vagy üres, vagy minden csomópontból 2 részfa ágazik el.

A bináris fákat gyakran alkalmazzák adathalmaz ábrázolására, ahonnan az elemeket azonosító kulcsuk alapján kell előkeresni. Ha egy bináris fát oly módon építünk fel, hogy minden csomópontra a bal részfán szereplő kulcsok kisebbek, a jobb részfa kulcsai pedig nagyobbak, mint a csomópont kulcsa, akkor a fát keresőfának (search tree) nevezzük. Egy keresőfában tetszőleges kulcs érhető el a gyökértől induló keresőútvonalon úgy, hogy minden csomópontból a bal vagy jobb részfa felé haladunk, és a döntéshez csak a csomóponthoz tartozó kulcsot kell vizsgálnunk. (Gyorsabb a keresés, mint a rendezett listában!) A továbbiakban az ilyen keresőfákkal végezhető alapműveleteket (keresés, beszúrás, törlés, bejárások) adjuk meg.
A bináris fa bejárásának azt a folyamatot nevezzük, amikor a fa minden elemét - pontosan egyszer érintve - feldolgozzuk. A gyakorlatban három, a rekurzív definícióra támaszkodó módszer terjedt el. Ezek:

1. Preorder (gyökérkezdő) bejárás:
   - a gyökérelem feldolgozása,
   - a baloldali részfa preorder bejárása,
   - a jobboldali részfa preorder bejárása.

2. Inorder (gyökérközepű) bejárás:
   - a baloldali részfa inorder bejárása,
   - a gyökérelem feldolgozása,
   - a jobboldali részfa inorder bejárása.

3. Postorder (gyökérvégű) bejárás:
   - a baloldali részfa postorder bejárása,
   - a jobboldali részfa postorder bejárása,
   - a gyökérelem feldolgozása.

Ezek a műveletek rendkívül egyszerűek, ezért csak egyiknek adjuk meg a megoldását.
Látható, hogy az inorder bejárás a kulcsok növekvő sorrendje szerinti lesz!

A fa csomópont elemei struktúrák:

```
typedef  struct csomopont* pcsomo;
struct csomopont  {   pcsomo bal;
                      int kulcs;
                      pcsomo jobb;   };

void preorder(pcsomo csomo)
{
    if (csomo != NULL)      {
                write(csomo->kulcs);
                preorder(csomo->bal);
                preorder(csomo->jobb);       }
    }
```

A beszúrást és a törlést meg kell előznie az adott kulcsú elem keresésének. Első esetben a beszúrás helyét, a másodikban pedig a törlendő elemet keressük. A keresést mindig a gyökértől kiindulva végezzük.
Beszúrásnál az új elemet *(x)* levélként illesztjük a fára.

KERES_BESZUR *(x,p)* *p:* az aktuális csomópontra mutat.
HA p = **NULL AKKOR** {x nincs a fán, beiktatás}
KÜLÖNBEN ELÁGAZÁS
$x < p \rightarrow$ kulcs **ESETÉN KERES_BESZUR** (x, $p \rightarrow$ bal)
$x > p \rightarrow$ kulcs **ESETÉN KERES_BESZUR** (x, $p \rightarrow$ jobb)
$x = p \rightarrow$ kulcs **ESETÉN** {Már van ilyen elem a fán!}
ELÁGAZÁS VÉGE
ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE

A törlés általában bonyolultabb feladat, mint a beszúrás. Ha a törlendő kulcsú elem levél vagy olyan csomópont, amelynek csak egy utóda van, akkor egyszerű a megoldás. De ha két utódja van, akkor nehezebb a probléma, hiszen egy pointerrel nem tudunk két irányba mutatni. Ilyenkor a kiiktatott csúcsot vagy a bal részfájának legjobboldalibb elemével (a nála kisebb elemek legnagyobbikával) kell helyettesítenünk, vagy a jobb részfájának legbaloldalibb elemével (a nála nagyobb elemek legkisebbikével) kell helyettesítenünk.
Mindegyiküknek legfeljebb egy utódja van!
A törlés feladatát két rekurzív eljárással oldjuk meg.

A **TOROL** eljárás keresi a törlendő faelemet (kulcsa: x) és három esetet különböztet meg:
1. Nincs olyan elem a fán, amelynek kulcsa = x.
2. Az x kulcsú komponensnek legfeljebb 1 utódja van.
3. Az x kulcsú komponensnek 2 utódja van.

A 3. esetben a **TOROL** eljárás meghívja a másik eljárást, a **POTOL**-t. Ennek az a dolga, hogy az x kulcsú csomópont baloldali részfájának szélső jobboldali elemét megkeresi, és áthelyezi a törlendő csomópontba.
A **POTOL** visszaadja q-ban az áthelyezett elem pointerét. A felszabadítást a **TOROL** végzi.

```
TOROL (x, p)                    p: az aktuális csomópontra mutat.
   HA p = NULL AKKOR {Nincs a fán x kulcsú elem!}
   KÜLÖNBEN
      ELÁGAZÁS
         HA x < p → kulcs AKKOR TOROL (x, p → bal)
         HA x > p → kulcs AKKOR TOROL (x, p → jobb)
         HA x = p → kulcs AKKOR  q := p
            ELÁGAZÁS
               HA q → jobb = NULL AKKOR p:= q → bal
               HA q → bal = NULL AKKOR p:= q → jobb
               HA q → jobb ≠ NULL ÉS q → bal ≠ NULL
                     AKKOR      POTOL (q → bal)
            ELÁGAZÁS VÉGE
            {q felszabadítása}
      ELÁGAZÁS VÉGE
ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE


POTOL (p)
   HA r → jobb ≠ NULL AKKOR POTOL (r → jobb)
   KÜLÖNBEN { q-ba átmozgatjuk r-t }
      q := r, r := r → bal
   ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```