Alapvető algoritmusok

4. előadás

Dr. Pătcaș Csaba

BABEȘ-BOLYAI TUDOMÁNYEGYETEM
Matematika és Informatika Kar

Tartalom

Halmaz-e?

Feladat

Döntsük el egy n elemű a sorozatról, hogy az elemei halmazt alkotnak-e!

- Egy halmaz vagy üres, vagy bizonyos számú elemet tartalmaz.
- Ha egy halmazt sorozattal implementálunk, az elemei különbözőek.
- A döntés eredményét az `ok` kimeneti paraméter tartalmazza.

Megjegyzés

Az itt bemutatott halmazműveletek bármilyen típusú elemeket tartalmazó halmazokra működnek. Az esetek nagy többségében a halmazelemek típusa specifikusabb (pl. egész számok, karakterláncok), ekkor speciális adatszerkezetek felhasználásával (pl. kiegyensúlyozott bináris keresőfák, hasítótáblák) hatékonyabban megvalósíthatóak a bemutatott műveletek.

Halmaz-e?

```
ALGORITMUS Halmaz-e(n, a, ok)
  i = 1
  ok = IGAZ
  AMÍG (ok ÉS (i < n)) végezd el:
    j = i + 1
    AMÍG ((j <= n) ÉS (a[i] != a[j])) végezd el:
    //a nem egyenlő operátort !=-nek írjuk
      j = j + 1
    VÉGE(Amíg)
    ok = (j > n)
    i = i + 1
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

Másolás

Feladat

Adott egy n elemű a sorozat és az elemeken értelmezett f függvény. Másoljuk át az összes elemet egy b sorozatba, úgy, hogy alkalmazzuk rájuk az f függvényt (amely lehet az identitásfüggvény is)!

Másolás

```
ALGORITMUS Másolás(n, a, b)
  MINDEN i = 1, n végezd el:
    b[i] = f(a[i])
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Kiválogatás

Feladat

Adott egy n elemű a sorozat és az elemeken értelmezett T tulajdonság. Válogassuk ki ezeket!

Több változat lehetséges a követelmények függvényében:

- Kigyűjtéssel
- Kiírással
- Helyben (2 változat)
- Kihúzással (segédsorozattal vagy speciális értékkel)

A megoldások a `Keresés` és a `Megszámlálás` programozási tételekből tartalmaznak majd elemeket.

Kiválogatás kigyűjtéssel

- A keresett elemeket (vagy azok sorszámait) kigyűjtjük a `pozíciók` sorozatba.
- A sorozat számosságát a `db` változóban tartjuk nyilván.
- A `db` változó értéke legtöbb n-el lesz egyenlő, mivel előfordulhat, hogy a bemeneti sorozat minden eleme adott tulajdonságú.

Kiválogatás kigyűjtéssel

```
ALGORITMUS Kiválogatás1(n, a, db, pozíciók)
  db = 0
  MINDEN i = 1, n végezd el:
    HA (T(a[i]))
      db = db + 1
      pozíciók[db] = i
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Kiválogatás kiírással

```
ALGORITMUS Kiválogatás2(n, a)
  MINDEN i = 1, n végezd el:
    HA (T(a[i]))
      KI: a[i]
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Kiválogatás helyben

- Ha a T tulajdonságú elemeket meg szeretnénk őrizni a sorozatban és a többbit ki akarjuk zárni, több lehetőségünk van a feladat specifikációjától függően.
- Ha a törlés után nem kötelező, hogy az elemek az eredeti sorrendjükben maradjanak, akkor minden törléskor rámásoljuk a törlendő elemre a sorozat utolsó elemét és csökkentjük 1-el a sorozat méretét.

Kiválogatás helyben

```
ALGORITMUS Kiválogatás3(n, a)
  i = 1
  AMÍG (i <= n) végezd el:
    HA (NEM T(a[i])) akkor
      a[i] = a[n]
      n = n - 1
    KÜLÖNBEN
      i = i + 1
    VÉGE(Ha)
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

Példa: [1 2 2 3 1 4], $T(x) = x$ páratlan

Kiválogatás helyben

- Ha az eredeti sorozatra nincs többé szükség, de szeretnénk megőrizni az elemek eredeti sorrendjét, akkor az eredeti tömb elejére sorakoztatjuk fel a T tulajdonságú elemeket.
- Így a kiválogatott elemekkel felülírjuk az eredeti adatokat.
- Mivel nem használunk másik sorozatot, a kiválogatást helyben végezzük.
- A db az új sorozat számosságát tartja nyilván.

Kiválogatás helyben

```
ALGORITMUS Kiválogatás4(n, a, db)
  db = 0
  MINDEN i = 1, n végezd el:
    HA (T(a[i])) akkor
      db = db + 1
      a[db] = a[i]
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Példa: [1 2 2 3 1 4], $T(x) = x$ páratlan

Kiválogatás segédsorozattal

- Ha a törlés ideiglenes, akkor a `törölt` logikai tömbben tartjuk nyilván, hogy melyik elemet töröltük (ideiglenesen).
- A `törölt` tömb elemei kezdetben mind `HAMIS`ak, majd a megfelelő sorszámú elemek értéke `IGAZ`zá változik.

Kiválogatás segédsorozattal

```
ALGORITMUS Kiválogatás5(n, a, törölt)
  MINDEN i = 1, n végezd el:
    törölt[i] = NEM T(a[i])
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Kiválogatás speciális értékkel

- Egy másik változat nem hoz létre segédsorozatot, tehát helyben dolgozik.
- Nem mozdítja el a helyükről a T tulajdonságú elemeket.
- A nem T tulajdonságú elemek helyére egy speciális értéket tesz.

Kiválogatás speciális értékkel

```
ALGORITMUS Kiválogatás6(n, a)
  MINDEN i = 1, n végezd el:
    HA (NEM T(a[i])) akkor
      a[i] = speciális érték
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Halmazzá alakítás

Feladat

Adott egy n elemű a sorozat. Alakítsuk halmazzá!

- Gyakorlatilag ki kell zárnunk a sorozatból a másodszor, harmadszor stb. megjelenő értékeket.
- Használjuk a `Kiválogatás3` algoritmusban látott ötletet, ahol a törlendő elemeket a sorozat utolsó elemével írtuk fölül.
- A `Halmaz-e` algoritmust használjuk az ismétlődő elemek detektálására.

Halmazzá alakítás

```
ALGORITMUS Halmazzá(n, a)
  i = 1
  j = i + 1
```

Halmazzá alakítás

```
  AMÍG (i < n) végezd el:
    AMÍG (j <= n) ÉS (a[i] != a[j]) végezd el:
      j = j + 1
    VÉGE(Amíg)
    HA (j <= n) akkor
      a[j] = a[n]
      n = n - 1
    KÜLÖNBEN
      i = i + 1
      j = i + 1
    VÉGE(Ha)
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

Példa: [1 2 1 3 3 2 1]

Keresztmetszet

Feladat

Adott az n elemű a és az m elemű b sorozat, melyek nem rendezettek és halmazokat ábrázolnak. Határozzuk meg a halmazok keresztmetszetét, vagyis azt a db elemű c sorozatot, mely a sorozatok közös elemeit tartalmazza!

Keresztmetszet

```
ALGORITMUS Keresztmetszet(n, a, m, b, db, c)
  db = 0
  MINDEN i = 1, n végezd el:
    j = 1
    AMÍG ((j <= m) ÉS (a[i] != b[j])) végezd el:
      j = j + 1
    VÉGE(Amíg)
    HA (j <= m) akkor
      db = db + 1
      c[db] = a[i]
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Példa: $A = [3\ 1\ 2]$, $B = [2\ 5\ 1]$

Egyesítés

Feladat

Adott az n elemű a és az m elemű b sorozat, melyek nem rendezettek és halmazokat ábrázolnak. Határozzuk meg a halmazok egyesítését, vagyis azt a db elemű c sorozatot, mely azokat az elemeket tartalmazza, amelyek legalább az egyik sorozatban megtalálhatóak!

- Mivel a sorozatok nem rendezettek, nem alkalmazhatjuk az összefésülést.
- Előbb a c sorozatba másoljuk az a sorozatot.
- Majd kiválogatjuk a b-ből azokat az elemeket, amelyeket nem találtunk meg az a-ban.

Egyesítés

```
ALGORITMUS Egyesítés(n, a, m, b, db, c)
  Másol(n, a, db, c)
  MINDEN j = 1, m végezd el:
    i = 1
    AMÍG ((i <= n) ÉS (a[i] != b[j])) végezd el:
      i = i + 1
    VÉGE(Amíg)
    HA (i > n)
      db = db + 1
      c[db] = b[j]
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Összefésülés

Feladat

Adott az n elemű a és az m elemű b sorozat, melyek rendezettek. Határozzuk db elemű c sorozatot, mely a két sorozat elemeit tartalmazza és rendezett!

- Az `Egyesítés` és `Keresztmetszet` algoritmusok nem rendezett sorozatokon dolgoztak, bonyolultságuk $O(n^2)$.
- Ha a két sorozat rendezett, ez a két művelet megvalósítható lineáris időben.

Összefésülés

Első változat

1. Elindulunk mindkét sorozatban és összehasonlítjuk az aktuális elemeket, ez alapján döntjük el, hogy melyik kerül az építendő sorozatba.
2. Addig végezzük ezeket a műveleteket, ameddig valamelyik sorozatnak a végére nem érünk.
3. A másik sorozatban megmaradt elemeket átmásoljuk az eredménysorozatba.

Összefésülés

Első változat

```
ALGORITMUS Összefésülés1(n, a, m, b, db, c)
  db = 0
  i = 1
  j = 1
  AMÍG ((i <= n) ÉS (j <= m)) végezd el:
    db = db + 1
    HA (a[i] < b[j]) akkor
      c[db] = a[i]
      i = i + 1
    KÜLÖNBEN
      c[db] = b[j]
      j = j + 1
    VÉGE(Ha)
  VÉGE(Amíg)
```

Összefésülés

Első változat

```
  AMÍG (i <= n) végezd el:
    db = db + 1
    c[db] = a[i]
    i = i + 1
  VÉGE(Amíg)
  AMÍG (j <= m) végezd el:
    db = db + 1
    c[db] = b[j]
    j = j + 1
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

Példa: $A = [1\ 3\ 4\ 7]$, $B = [2\ 5\ 8\ 9\ 10]$

Összefésülés

Második változat

- Akkor használjuk, ha a két sorozat halmazt reprezentál és az eredmény is az kell legyen.
- Az előző algoritmushoz hozzávesszük azt az esetet, amikor két elem egyenlő, ekkor mindkét sorozatban továbblépünk és az értéket természetesen csak egyszer írjuk be a megoldásba.

Összefésülés

Második változat

```
ALGORITMUS Összefésülés2(n, a, m, b, db, c)
  db = 0
  i = 1
  j = 1
  AMÍG ((i <= n) ÉS (j <= m)) végezd el:
    db = db + 1
```

Összefésülés

Második változat

```
  HA (a[i] < b[j]) akkor
    c[db] = a[i]
    i = i + 1
  KÜLÖNBEN
    HA (a[i] = b[j])
      c[db] = a[i]
      i = i + 1
      j = j + 1
    KÜLÖNBEN
      c[db] = b[j]
      j = j + 1
    VÉGE(Ha)
  VÉGE(Ha)
VÉGE(Amíg)
```

Összefésülés

Második változat

```
  AMÍG (i <= n) végezd el:
    db = db + 1
    c[db] = a[i]
    i = i + 1
  VÉGE(Amíg)
  AMÍG (j <= m) végezd el:
    db = db + 1
    c[db] = b[j]
    j = j + 1
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

Összefésülés

Harmadik változat

- Ha szerencsések lettünk volna, $a[n] = b[m]$ lett volna, ekkor a két utolsó `AMÍG` struktúrának egyetlen iterációját sem hajtottuk volna végre.
- Ezt az észrevételt kihasználva, mindkét sorozat végére elhelyezünk egy fiktív elemet, ezt őrszemnek vagy strázsának nevezzük.
- Az őrszemek értéke nagyobb kell legyen mint mindkét sorozat utolsó eleme, ezeket `végtelennel` jelöljük az algoritmusban.
- Egy másik lehetséges megválasztása a strázsa értékeinek: `a[n + 1] = b[m] + 1, b[m + 1] = a[n] + 1`
- Az összefésült sorozat nem fogja tartalmazni a `végtelen` értéket.
- Ha az eredeti sorozatok nem halmazok, vagy elemeik nem diszjunktak, az eredmény sem lesz halmaz.
- Mivel nem kezeljük külön az egyenlőséget, az eredménynek pontosan $n + m$ eleme lesz, vagyis használhatunk `MINDEN` struktúrát.

Összefésülés

Harmadik változat

```
ALGORITMUS Összefésülés3(n, a, m, b, db, c)
  i = 1
  j = 1
  a[n + 1] = végtelen
  b[m + 1] = végtelen
```

Összefésülés

Harmadik változat

```
  MINDEN db = 1, n + m végezd el:
    HA (a[i] < b[j]) akkor
      c[db] = a[i]
      i = i + 1
    KÜLÖNBEN
      c[db] = b[j]
      j = j + 1
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Példa: $A = [1\ 2\ 3]$, $B = [2\ 3\ 5]$

Összefésülés

Negyedik változat

- Gyakorlatilag a második változat őrszemmel való megvalósítása.
- Szintén akkor használjuk mikor a bemeneti sorozatok halmazokat ábrázolnak és az eredménysorozat is halmaz kell legyen.
- Mivel nem tudjuk előre, hogy hány eleme lesz a megoldásnak, `MINDEN` struktúra helyett `AMÍG`-ot használunk.
- Az őrszemek révén az `AMÍG` struktúrát addig hajtjuk végre, ameddig mindkét sorozat végére nem értünk.

Összefésülés

Negyedik változat

```
ALGORITMUS Összefésülés4(n, a, m, b, db, c)
  db = 0
  i = 1
  j = 1
  a[n + 1] = végtelen
  b[m + 1] = végtelen
  AMÍG ((i <= n) VAGY (j <= m)) //nem ÉS!
    db = db + 1
    HA (a[i] < b[j]) akkor
      c[db] = a[i]
      i = i + 1
```

Összefésülés

Negyedik változat

```
    KÜLÖNBEN
      HA (a[i] = b[j]) akkor
        c[db] = a[i]
        i = i + 1
        j = j + 1
      KÜLÖNBEN
        c[db] = b[j]
        j = j + 1
      VÉGE(Ha)
    VÉGE(Ha)
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

Szétválogatás

- A `Kiválogatás` programozási tételben egy sorozatot dolgoztunk fel és abból válogattunk ki bizonyos elemeket.
- A nem kiválogatott elemekkel nem foglalkoztunk, de van amikor azokra is szükségünk van.
- Előfordulhat, hogy két vagy több sorozatba kell szétválogatnunk a bemeneti sorozatot.

Szétválogatás

Első változat

Feladat

Adott az n elemű a sorozat és az elemein értelmezett T tulajdonság. Válogassuk szét az elemeit úgy, hogy a b sorozat tartalmazza az összes T tulajdonságú elemet és a c sorozat a többi elemet!

- A b elemeinek számát dbb-vel, a c elemeinek számát dbc-vel jelöljük.
- Nem tudhatjuk előre az új sorozatok méretét, legrosszabb esetben az eredetivel azonos méretűek lehetnek, ha valamennyi elem „átvándorol” valamelyik sorozatba és a másik üres marad.

Szétválogatás

Első változat

```
ALGORITMUS Szétválogatás1(n, a, dbb, b, dbc, c)
  dbb = 0
  dbc = 0
  MINDEN i = 1, n végezd el:
    HA (T(a[i])) akkor
      dbb = dbb + 1
      b[dbb] = a[i]
    KÜLÖNBEN
      dbc = dbc + 1
      c[dbc] = a[i]
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Szétválogatás

Második változat

- A feladat megoldható egyetlen új sorozattal.
- A kiválogatott elemeket az új sorozat elejére tesszük (balról jobbra haladva), a megmaradtakat a végére (jobbról balra haladva).
- Nem lesz ütközés, mivel pontosan n elemet kell n helyre elhelyezzünk.
- A megmaradt elemek az eredeti sorozatban elfoglalt relatív pozícióik fordított sorrendjében kerülnek az új sorozatba.
- A `b` tömbbe válogatjuk szét az elemeket, a `dbb` változó jelzi a tulajdonsággal rendelkező elemek számát, a `dbc` a megmaradt elemek számát.

Szétválogatás

Második változat

```
ALGORITMUS Szétválogatás2(n, a, dbb, dbc, b)
  dbb = 0
  dbc = 0
  MINDEN i = 1, n végezd el:
    HA (T(a[i])) akkor
      dbb = dbb + 1
      b[dbb] = a[i]
    KÜLÖNBEN
      dbc = dbc + 1
      b[n - dbc + 1] = a[i]
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Szétválogatás

Szétválogatás helyben

- Ha az eredeti sorozatra nincs már szükségünk, a szétválogatást elvégezhetjük helyben.
- Egy lehetőség elindulni párhuzamosan elölről és hátulról a tömbben és keresni olyan elemeket, amelyeket fel kell cserélni.
- Ha találtunk két felcserélendő elemet, akkor felcseréljük őket és folytatjuk a keresést.
- Ha összetalálkoztunk két irányból, befejeztük.

Egy másik ötlet:

1. Lementjük a tömb első elemét egy segédváltozóba.
2. Az utolsó elemtől visszafelé indulva megkeressük az első olyan elemet, amely T tulajdonságú és ezt beírjuk a tömb elejére.
3. Elölről indulva keresünk egy olyan elemet, amely nem T tulajdonságú és ezt beírjuk a tömb végére.
4. A fenti lépéseket addig végezzük, ameddig össze nem találkozunk, a db változó tárolja az utolsó T tulajdonságú elem indexét balról nézve.

Szétválogatás

Harmadik változat

```
ALGORITMUS Szétválogatás3(n, a, db)
  e = 1
  u = n
  segéd = a[e]
  AMÍG (e < u) végezd el:
    AMÍG ((e < u) ÉS NEM T(a[u])) végezd el:
      u = u - 1
    VÉGE(Amíg)
```

Szétválogatás
Harmadik változat

```
    HA (e < u)
      a[e] = a[u]
      e = e + 1
      AMÍG ((e < u) ÉS T(a[e])) végezd el:
        e = e + 1
      VÉGE(Amíg)
      HA (e < u)
        a[u] = a[e]
        u = u - 1
      VÉGE(Ha)
    VÉGE(Ha)
  VÉGE(Amíg)
```

```
  a[e] = segéd
  HA (T(a[e]))
    db = e
  KÜLÖNBEN
    db = e - 1
  VÉGE(Ha)
VÉGE(Algoritmus)
```

Példa: $A = [2\ 4\ 1\ 6\ 5\ 7\ 8]$, $T(x) = x$ páros

Többszörös szétválogatás

- Ha egy sorozatot több részsorozatba kell szétválogatni több tulajdonság alapján, akkor több szétválogatást fogunk végezni.
- Előbb szétválogatjuk a bemeneti sorozatból az első tulajdonsággal rendelkező elemeket a többitől.
- A megmaradt elemeket szétválogatjuk a második tulajdonság alapján és így tovább.

Programozási tételek összeépítése

Előfordulhat, hogy egyszerű egymás utáni alkalmazás helyett, egyszerűbb, rövidebb, hatékonyabb algoritmust tervezhetünk, ha összeépítünk több programozási tételt, mint a következő példa esetén.

Megszámolással való összeépítés

Van-e egy adott sorozatban legalább k darab T tulajdonságú elem? Ha igen, adjuk meg a sorozat k. elemét, mely rendelkezik a T tulajdonsággal!

Tartalom

Algoritmusok és programok fejlesztési módozatai

- A megoldandó feladatot elképzelhetjük egy gyökeres faként, amelynek a gyökere jelképezi a feladatot és alatta helyeszkednek el a részfeladatok.
- Ha ezt a fát fentről lefele építjük fel (járjuk be, implementáljuk), akkor beszélünk top-down típusú programozásról.
- Ha a legegyszerűbb részfeladatok megoldásával kezdjük és ezek összerakásával alakítjuk ki a megoldást, bottom-up típusú programozásról beszélünk.
- A gyakorlatban legtöbbször keverjük a két módszert, de a kezdeti tervezési fázisban általában top-down gondolatmenetet használunk.
- Ezeket a koncepciókat modern software-fejlesztési technológiákban is megtaláljuk (pl. Spring, ”inversion of control”, ”dependency injection” stb.)

A top-down (fentről-lefele) típusú programozás

- Az eredeti feladatot fokozatosan felbontjuk egyre egyszerűbb részfeladatokra.
- Felbontáskor feltételezzük, hogy a részfeladatokat már megoldottuk és a felbontás után fejtjük ki az alprogramokat.
- Amikor olyan részfeladathoz érünk, amelynek megoldása már eléggé egyszerű, alprogramot írunk e rész megoldására.
- Egy-egy alprogram további alprogramokat tartalmazhat, ezek tulajdonképpen a leszármazottai a fában.

Feladat

Adott egy n elemű tömb, mely természetes számokat tartalmaz az $[1, n]$ intervallumból. Határozzuk meg, hogy van-e olyan elem, amely több mint $n/2$ példányban fordul elő a tömbben.

- Ha egy tömbben vagy más adatszerkezetben számoljuk az előfordulásokat, $\Theta(n)$ idő- és memóriabonyolultságú algoritmusunk van.
- A szemináriumon vett jelöltes megoldás ("Boyer–Moore majority vote algorithm") $\Theta(n)$ időbonyolultsággal és $\Theta(1)$ memóriabonyolultsággal rendelkezik, ezt tárgyaljuk a továbbiakban.
- Megjegyezzük, hogy létezik hatékonyabb (probabilisztikus) algoritmus is.

A top-down (fentről-lefele) típusú programozás

Példa: Többségi elem (elnökválasztás)

Vázlatszerűen felírjuk a „nagy” feladat megoldásának lépéseit (a fa gyökere).

```
ALGORITMUS Elnökválasztás()
  Beolvas(n, a)
  jelölt = KeresJelölt(n, a)
  többségiJelölt = EllenőrizJelölt(n, a, jelölt)
  Kiír(többségiJelölt, jelölt)
VÉGE(Algoritmus)
```

A top-down (fentről-lefele) típusú programozás

Példa: Többségi elem (elnökválasztás)

Ezután sorban kifejtjük mindegyik alprogramot (a leszármazottakat).

```
ALGORITMUS Beolvas(n, a)
  BE: n
  MINDEN i = 1, n végezd el:
    BE: a[i]
  VÉGE(Minden)
VÉGE(Algoritmus)
```

```
ALGORITMUS KeresJelölt(n, a)
  jelölt = a[1]
  hány = 1
```

A top-down (fentről-lefele) típusú programozás

Példa: Többségi elem (elnökválasztás)

```
  MINDEN i = 2, n végezd el:
    HA (a[i] = jelölt) akkor
      hány = hány + 1
    KÜLÖNBEN
      HA (hány = 0) akkor
        jelölt = a[i]
        hány = 1
      KÜLÖNBEN
        hány = hány - 1
      VÉGE(Ha)
    VÉGE(Ha)
  VÉGE(Minden)
  VISSZATÉRÍT: jelölt
VÉGE(Algoritmus)
```

A top-down (fentről-lefele) típusú programozás

Példa: Többségi elem (elnökválasztás)

```
ALGORITMUS EllenőrizJelölt(n, a, jelölt)
  hány = 0
  MINDEN i = 1, n végezd el:
    HA (a[i] = jelölt) akkor
      hány = hány + 1
    VÉGE(Ha)
  VÉGE(Minden)
  VISSZATÉRÍT: (hány > n / 2)
VÉGE(Algoritmus)
```

A top-down (fentről-lefele) típusú programozás

Példa: Többségi elem (elnökválasztás)

```
ALGORITMUS Kiír(többségiJelölt, jelölt)
  HA (többségiJelölt)
    KI: jelölt
  KÜLÖNBEN
    KI: 'Nem nyert senki'
  VÉGE(Ha)
VÉGE(Algoritmus)
```

A bottom-up (lentről felfele) típusú programozás

- A top-down stílus ellentétje.
- Megvalósítjuk az alprogramokat, melyek egy-egy részfeladatot oldanak meg és miután készen vannak, összefűzzük őket.
- Tehát az alprogram megírása után következik az őt hívó program vagy alprogram megírása.

A bottom-up (lentről felfele) típusú programozás

Példa: Törzstényezőkre bontás

Feladat

Bontsunk törzstényezőkre k darab természetes számot!

Példa: $12 = 2^2 \cdot 3$

Biztosan szükségünk lesz beolvasásra, úgyhogy azt meg is írhatjuk.

```
ALGORITMUS Beolvas(k, számok)
  BE: k
  MINDEN i = 1, k végezd el:
    BE: számok[i]
  VÉGE(Minden)
VÉGE(Algoritmus)
```

A bottom-up (lentről felfele) típusú programozás

Példa: Törzstényezőkre bontás

Úgy döntünk, hogy egy adott számra a megoldást egy struktúra-tömbben tároljuk, megírjuk ennek a kiíratását.

```
ALGORITMUS KiírTörzstényezők(m, felbontás)
  KI: felbontás[1].tényező + '^' + felbontás[1].kitevő
  MINDEN i = 2, m végezd el
    KI: '*' + felbontás[i].tényező + '^' + felbontás[i].kitevő
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Szükségünk lesz egy egyszerűbb kiíratásra, ha a számnak nincsenek törzstényezői.

```
ALGORITMUS KiírEgyszerű(n)
  KI: n
VÉGE(Algoritmus)
```

A bottom-up (lentről felfele) típusú programozás

Példa: Törzstényezőkre bontás

Az előbbi két kiíratást összekombináljuk egy nagyobba:

```
ALGORITMUS Kiír(n, m, felbontás)
  HA (n > 1)
    KiírTörzstényezők(m, felbontás)
  KÜLÖNBEN
    KiírEgyszerű(n)
  VÉGE(Ha)
VÉGE(Algoritmus)
```

A bottom-up (lentről felfele) típusú programozás

Példa: Törzstényezőkre bontás

Ezután megírjuk magát a felbontást.

```
ALGORITMUS Felbont(n, m, felbontás)
  m = 0
  osztó = 2
  AMÍG (n != 1) végezd el:
    hatvány = 0
    AMÍG (n % osztó = 0) végezd el: //% = maradék operátor (mod)
      hatvány = hatvány + 1
      n = n / osztó
    VÉGE(Amíg)
```

A bottom-up (lentről felfele) típusú programozás
Példa: Törzstényezőkre bontás

```
    HA (hatvány != 0)
      m = m + 1
      felbontás[m].tényező = osztó
      felbontás[m].kitevő = hatvány
    VÉGE(Ha)
    osztó  = osztó + 1
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

A bottom-up (lentről felfele) típusú programozás

Példa: Törzstényezőkre bontás

A felbontást és a kiíratást meg kell hívnunk mind a k darab beolvasott számra.

```
ALGORITMUS Megold(k, számok)
  MINDEN i = 1, k végezd el:
    Felbont(számok[i], m, felbontás)
    Kiír(számok[i], m, felbontás)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Végül meghívjuk a megírt alprogramokat.

```
ALGORITMUS Törzstényezők()
  Beolvas(k, számok)
  Megold(k, számok)
VÉGE(Algoritmus)
```

Látjuk, hogy lentről felfele építettük a megoldást.