Eötvös Loránd Tudományegyetem
Informatikai Kar

Alkalmazott modul: Programozás

3. fejezet

Programozási tételek, rendezések

Giachetta Roberto

A jegyzet az ELTE Informatikai Karának 2015. évi
Jegyzetpályázatának támogatásával készült

Programozási tételek

Algoritmusok és programozási tételek

- Az egyszerű, sorozatokra alkalmazott algoritmusokat nevezzük *programozási tételek*nek, ezek a következők:
  - összegzés, számlálás
  - lineáris keresés
  - maximum keresés, feltételes maximumkeresés
  - bináris keresés
- A programozási tételek absztrakt módon fogalmazzuk meg, majd a feladatnak megfelelő átalakításokkal alkalmazzuk
  - az absztrakt megfogalmazást egész értékű sorozatokon fogalmazzuk meg, de lehetne általánosítani

Programozási tételek

Összegzés

- Az *összegzés programozási tétele* lehetővé teszi tetszőleges sorozat $(a_1, \ldots, a_n)$ adott függvény (f) szerint vett értékének összesítését (sum)

$$sum = \sum_{i=1}^{n} f(a_i)$$

  - az összegzés egy ciklusban történik, az összeget egy külön változóhoz adjuk hozzá minden lépésben, amelyet egy kezdeti értékkel inicializálunk
  - általában az összegző művelet az összeadás, ekkor az összeg változó 0-ról indul
  - a függvény sokszor az identitás, de lehet nagyon összetett is

Programozási tételek

Összegzés

Absztrakt leírás:

$A = (a : \mathbb{Z}^n, sum : \mathbb{Z})$

$f : \mathbb{Z} \to \mathbb{Z}$

$Q = \left(\forall j \in [1..n] : \left(a_j = a'_j\right)\right)$

$R = Q \wedge \left(sum = \sum_{j=1}^{n} f(a_j)\right)$

$sum := 0, i := 1$	
$i \leq n$	
	$sum := sum + f(a_j)$
	$i := i + 1$

Programozási tételek

Összegzés

Megvalósítás:

```
int a[n]; // feldolgozandó sorozat
int sum; // összeg változó

… // a sorozat elemei értéket kapnak

sum = 0; // az összeget kinullázzuk
for (int i = 0; i < n; i++)
{
    sum = sum + f(a);
  // f tetszőleges egész függvény
}
// az eredmény a sum változóban található
```

Programozási tételek

Összegzés

Feladat: Adjuk meg az első *n* természetes szám összegét.

- ehhez az összegzés tételére van szükségünk, a forrás és az eredmény típusa egész, a művelet az összegzés, a kezdőérték nulla
- az értéket nem kell külön beolvasnunk, és nem kell külön eltárolnunk, ezek adott konstansok, csupán az intervallum végét, azaz *n* értékét kérjük be a felhasználótól
- egy számláló ciklusra lesz szükségünk, amely elmegy 1-től az *n* értékéig

Programozási tételek

Összegzés

Megoldás:

```
int main(){
   int sum = 0, n; // inicializálás
   cout << "Kérem n értékét: ";
   cin >> n; // beolvasás

   for (int i = 1; i <= n; i++) // összegzés
      sum += i;

   cout << "Az első " << n << " szám összege: "
        << sum << endl; // eredmény kiírása
   return 0;
}
```

Programozási tételek

Számlálás

- A *számlálás programozási tétele* lehetővé teszi, hogy tetszőleges $(a_1, \ldots, a_n)$ sorozatban megszámoljuk egy adott (logikai) felvételt (β) teljesítő elemek számát (c)

$$c = \sum_{\substack{i=1 \\ \beta(a_i)}}^{n} 1$$

  - a feltétel tetszőleges logikai értékű függvény
  - lényegében az összegzés egy speciális esetét végezzük, ahol vagy 1-t, vagy 0-t adunk hozzá az eddigi összeghez, függően a feltétel teljesülésétől
  - ezt a végrehajtásban elágazás segítségével valósítjuk meg

Programozási tételek

Összegzés

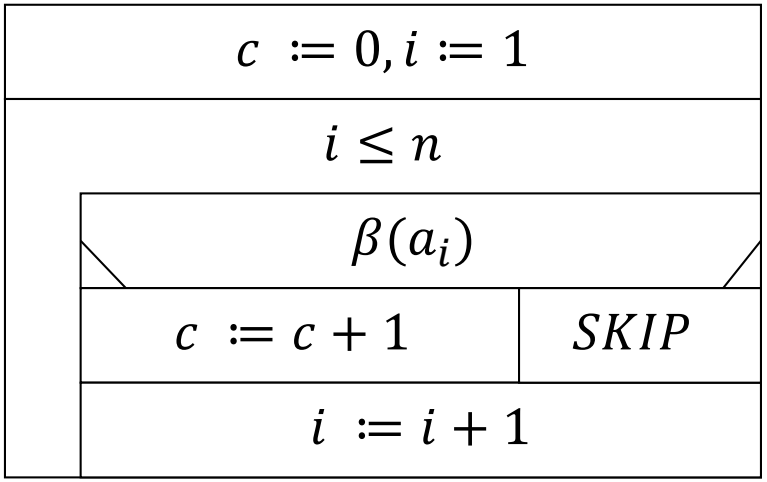
Absztrakt leírás:

$$A = (a : \mathbb{Z}^n, c : \mathbb{N})$$

$$\beta : \mathbb{Z} \to \mathbb{L}$$

$$Q = \left(\forall j \in [1..n] : \left(a_j = a'_j\right)\right)$$

$$R = Q \wedge \left(c = \sum_{\substack{j=1 \\ \beta(a_i)}}^{n} 1\right)$$



Programozási tételek

Összegzés

Megvalósítás:

```
int a[n]; // feldolgozandó sorozat
int c; // számláló változó

… // a sorozat elemei értéket kapnak

c = 0; // az számlálót kinullázzuk
for (int i = 0; i < n; i++)
{
   if (beta(a[i])) // ha teljesül a feltétel
      c++; // növeljük a számlálót
}
// eredmény a c változóban található
```

Programozási tételek

Számlálás

Feladat: Generáljuk 100 véletlen számot 1 és 10 között, és számoljuk meg, hány páratlan van közöttük.

- használjuk a számlálás tételét, ahol a páratlanság biztosítja a feltételt (oszthatóságot a moduló segítségével ellenőrizhetünk, 2-vel osztva 1-t ad maradékul)
- a számot 0-9 közé generáljuk, majd hozzáadunk egyet
- szükségünk lesz egy számláló ciklusra

Megoldás:

```
int main(){
    srand(time(0)); // véletlen generátor indítás
    int nums[100]; // számok tömbje
```

Megoldás:

```
    for (int i = 0; i < 100; i++) // generálás
       nums[i] = rand() % 10 + 1;
          // 1 és 10 közötti szám

    int c = 0; // inicializálás
    for (int i = 0; i < 100; i++) // számlálás
       if (nums[i] % 2 == 1) // ha páratlan a szám
          c++;

    cout << "100 véletlen számból " << c
         << " volt páratlan. " << endl; // kiírás
    return 0;
}
```

Programozási tételek

Lineáris keresés

- A *lineáris keresés* programozási tétele segítségével megállapíthatjuk, hogy van-e egy sorozatban egy adott feltételt teljesítő elem, és amennyiben több van, melyik az első ilyen elem.
  - a teljesüléshez szükségünk van egy logikai változóra (l), és megadhatjuk magát az értéket, vagy a helyét (ind)
  - amennyiben már teljesült a feltétel, akkor nincs értelme végignézni a további értékeket, hiszen a teljesülést nem befolyásolják
  - ezért a ciklust korábban is terminálhatjuk úgy, hogy a logikai változót is bevesszük a ciklusfeltételbe

Programozási tételek

Lineáris keresés

Absztrakt leírás:

$$A = (a : \mathbb{Z}^n, ind : \mathbb{N}, l : \mathbb{L})$$

$$\beta: \mathbb{Z} \to \mathbb{L}$$

$$Q = \left(\forall j \in [1..n]: a_j = a'_j\right)$$

$$R = Q \wedge \left(\begin{array}{c} l = \exists j \in [1..n]: \beta(a_j) \wedge \\ l \to \left(\begin{array}{c} ind \in [1..n] \wedge \beta(a_{ind}) \wedge \\ \forall j \in [1..ind-1]: \neg\beta(a_j) \end{array} \right) \end{array} \right)$$

$l :=\downarrow, i := 1$
$\neg l \wedge i \leq n$
$l := \beta(a_i), ind := i$
$i := i + 1$

Programozási tételek

Lineáris keresés

Megvalósítás:

```
int a[n]; // feldolgozandó sorozat
bool l = false; // teljesülés változója
int ind; // keresett hely változója

… // a sorozat elemei értéket kapnak

for (int i = 0; i < n && !l; i++)
   // a logikai érték is bekerül a feltételbe
{
   l = beta(a[i]) // ha teljesül a feltétel
   ind = i; // növeljük a számot
}
// eredmény az l és ind változókban található
```

Programozási tételek

Lineáris keresés

Feladat: 10 napon át megmértük a hőmérsékletet, mértünk-e fagypont alatti értéket, és ha igen, hányadik napon először.

- valós értékekkel dolgozunk, ugyanakkor az érték helye egész lesz
- lineáris keresést alkalmazunk, a feltétel a negatív érték

Megoldás:

```
int main(){
   float t; // aktuális napi hőmérséklet
   bool l = false;
   int ind;
```

Megoldás:

```
    for (int i = 1; i <= 100 && !l; i++){
       cin >> t; // napi hőmérséklet beolvasása
       l = t < 0; // feltétel ellenőrzés
       ind = i;
    }
    // eredmény függvényében való kiírás
    if (l) // ha teljesült
       cout << "A(z) " << ind << ". napon mértünk
                  először negatív értéket. " << endl;
    else // ha nem teljesült a feltétel
       cout << "Nem mértünk mínusz értéket."
            << endl;
    return 0;
}
```

Programozási tételek

Maximumkeresés

- Egy sorozat (adott szempont szerinti) maximumát a *maximumkeresés* programozási tételével állapíthatjuk meg.
  - megadja a maximum értékét, illetve helyét
  - mindig összehasonlítjuk az új elemet a sorozat eddigi részének maximumával, és ha nagyobb nála, akkor ő lesz az új maximum
  - ehhez a maximum értéket kezdetben valamilyen extremális értéktől indíthatjuk el (pl. ha csak pozitív számok vannak, akkor 0-tól), vagy rögtön ráállítjuk a sorozat első elemére, így a sorozat második elemétől indul a ciklus
  - a tétel könnyen átfogalmazható minimumra

Programozási tételek

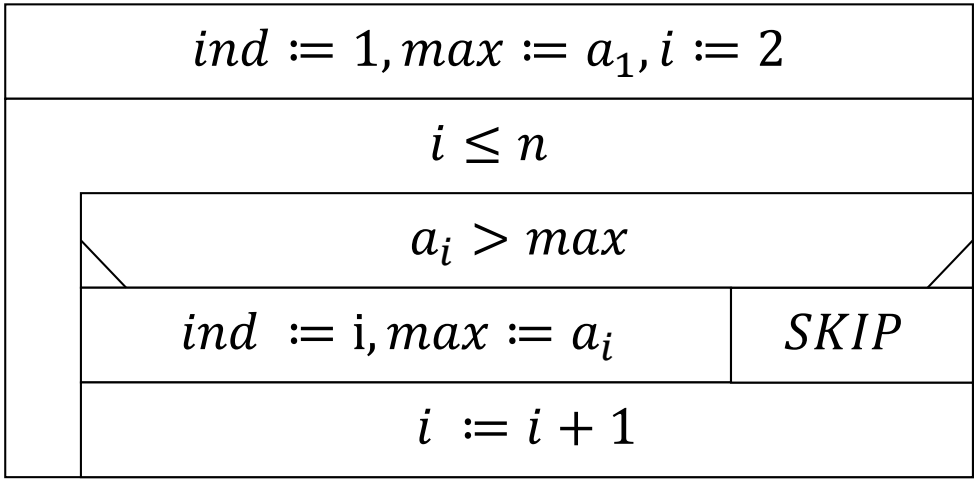
Maximumkeresés

Absztrakt leírás:

$$A = (a : \mathbb{Z}^n, ind : \mathbb{N}, max : \mathbb{Z})$$

$$Q = \left(n \geq 1 \ \wedge \forall j \in [1..n]: \left(a_j = a'_j\right)\right)$$

$$R = Q \wedge \left(max, ind = \max\left(a_j\right)\right)$$



Programozási tételek

Maximumkeresés

Megvalósítás:

```
int a[n]; // feldolgozandó sorozat
… // a sorozat elemei értéket kapnak

int ind = 0, max = a[0];
   // maximum hely és érték változók, az első
   // elemre állnak
for (int i = 0; i < n; i++) {
   if (max < a[i]){ // ha nagyobb elemet találunk
      ind = i;      // újra beállítjuk őket
      max = a[i];
   }
}
// eredmény az ind és max változókban található
```

Programozási tételek

Maximumkeresés

Feladat: Minden nap megmértük a hőmérséklet egészen addig, amíg fagypont alá nem estek. Keressük meg az addig tartó tartomány legnagyobb értékét, és hogy hányadik.

- maximumkeresést alkalmazunk
- előtesztelő ciklusban dolgozunk, amely 0-nál kisebb érték esetén megáll (számolnunk kell a napokat is, hogy megállíthassuk a maximum helyét)

Megoldás:

```
int main(){
   float t; // aktuális napi hőmérséklet
   cin >> t; // első érték beolvasása
   int ind = 1, max = t, i = 1; // inicializálás
```

Programozási tételek

Maximumkeresés

Megoldás:

```
    cin >> t; // beolvassuk a következő értéket
    while (t >= 0) {
       // amíg nem lesz negatív az érték
       i++; // növeljük a sorszámot
       if (t > max) { // új maximum
          ind = i;
          max = t;
       }
       cin >> t; // beolvassuk a következő értéket
    }
    cout << "A maximum: " << max << ", a(z) "
         << ind << ". napon mértünk. " << endl;
    return 0;
}
```

Programozási tételek

Feltételes maximumkeresés

- A maximumkeresés tételének azon változatát, ahol csak a sorozat bizonyos elemei között keresünk maximumot, *feltételes maximumkeresés*nek nevezzük
  - adott egy feltétel, amelyet a sorozat minden elemére megvizsgálunk, ha az elem nem teljesíti a feltételt, akkor nem teszünk semmit
  - nem állíthatjuk rögtön az első elemre a maximumot, hanem csak az első olyan elemre, amely teljesíti ezt a feltételt
  - nem garantált, hogy van olyan eleme a sorozatnak, ami teljesíti a feltételt, ezért egy logikai értékkel jeleznünk kell, hogy találtunk-e ilyet

Programozási tételek

Feltételes maximumkeresés

Absztrakt leírás:

$$A = (a : \mathbb{Z}^n, ind : \mathbb{N}, max : \mathbb{Z}, l : \mathbb{L})$$

$$\beta : \mathbb{Z} \to \mathbb{L}$$

$$Q = \left(n \geq 1 \wedge \forall j \in [1..n] : \left(a_j = a'_j\right)\right)$$

$$R = Q \wedge \left(l, max, ind = \max_{\beta(a_j)}\left(a_j\right)\right)$$

<table>
<tr><td colspan="4">$l := \downarrow, i := 1$</td></tr>
<tr><td colspan="4">$i \leq n$</td></tr>
<tr><td>$\beta(a_i) \wedge \neg l$</td><td colspan="2">$\beta(a_i) \wedge l$</td><td>$\neg\beta(a_i)$</td></tr>
<tr><td rowspan="2">$l := \uparrow, ind := i,$
$max := a_i$</td><td colspan="2">$a > max$</td><td rowspan="2">$SKIP$</td></tr>
<tr><td>$ind := i, max := a_i$</td><td>$SKIP$</td></tr>
<tr><td colspan="4">$i := i + 1$</td></tr>
</table>

Programozási tételek

Feltételes maximumkeresés

Megvalósítás:

```
int a[n]; // feldolgozandó sorozat
… // a sorozat elemei értéket kapnak

bool l = false;
   // kezdetben nincs a feltételnek eleget tevő
   // elem
for (int i = 0; i < n; i++)
{
   if (beta(a[i]) && !l){ // első ilyen elem
       l = true; // mindent be kell állítanunk
       ind = i;
       max = a[i];
   }
```

Programozási tételek

Feltételes maximumkeresés

Megvalósítás:

```
      if (beta(a[i]) && max < a[i]){
         // ha már volt ilyen elem, és az aktuális,
         // feltételt teljesítő elem nagyobb nála
         ind = i;
         max = a[i];
            // újra beállítjuk az indexet és
            // maximumot
      }
   }
   // eredmény az l, ind és max változókban található
```

Programozási tételek

Példa

Feladat: Olvassunk be 10 szót a bemenetről, és adjuk meg a legrövidebb, legalább 10 hosszú szavat.

- használjunk feltételes minimumkeresést, amelynek feltétele, hogy a szó legalább 10 hosszú
- a szavak hosszát hasonlítjuk össze, a legrövidebbet keressük

Megoldás:

```
int main(){
   string word; // beolvasandó szó
   bool l = false;
   int ind;
   string min;
```

Programozási tételek

Példa

Megoldás:

```
for (int i = 1; i <= 10; i++) {
   cin >> word; // szó beolvasása
   if (word.length() >= 10 && !l){
      // hossz ellenőrzése
      l = true;
      ind = i;
      min = word;
   }
   if (word.length() >= 10 &&
       word.length() < min.length()) {
      ind = i; min = word;
   }
}
```

Programozási tételek

Példa

Megoldás:

```
    if (l){
       // találtunk legalább 10 hosszú szavat
       cout << "A legrövidebb megfelelő szó "
            << min << " volt, amelyet " << ind
            << "-ként adtunk meg." << endl;
    }
    else {
       // nem találtunk
       cout << "Nem volt megfelelő szó." << endl;
    }

    return 0;
}
```

Programozási tételek

Bináris keresés

- Amennyiben egy értéket keresünk a bemenő sorozatban, és annak elemei növekvően (vagy csökkenően) rendezettek, lehetőségünk van *bináris* (*logaritmikus*) *keresés*t használni
  - a bináris keresés nem sorrendben dolgozza fel a bemenetet, hanem felhasználja a sorozat rendezettségét
  - vegyünk egy elemet a sorozatból, ha az a keresett elem, akkor végeztünk, ha kisebb, mint a keresett, akkor csak az utána lévő tartományban lehet a keresendő elem, ha pedig nagyobb, akkor az előtte lévő tartományban
  - mindig felezzük a tartományt, és a középső elemet ellenőrizzük le, nem egyezés esetén vagy az első felét, vagy a második felét vesszük a tartománynak, és így tovább

Programozási tételek

Bináris keresés

- Pl. keressük a 73-as elemet az alábbi sorozatban:

```
03 10 18 21 39 40 51 73 76 81 93
                  ↑
```

itt felezünk, a 40 kisebb, mint a 73, ezért következő lépésben a tartomány második felét vizsgáljuk

```
03 10 18 21 39 40 51 73 76 81 93
                        ↑

03 10 18 21 39 40 51 73 76 81 93
                     ↑

03 10 18 21 39 40 51 73 76 81 93
                        ↑
```

Programozási tételek

Bináris keresés

- Ez az algoritmus hatékonyabb, mint a lineáris keresés, mivel esetenként sok elemet is át tud ugrani egyszerre
- A tartományt úgy szabályozzuk, hogy mindig nyilvántartjuk annak alsó, illetve felső határát, és ezt állítjuk a középső elem függvényében
- Az algoritmus addig halad, amíg meg nem találjuk az elemet, vagy üres nem lesz a tartomány (az alsó határ nagyobb, mint a felső határ)
- Csökkenő értékekkel is megfogalmazható, csak fordítani kell a relációkon

Programozási tételek

Bináris keresés

Absztrakt leírás:

$$A = (a : \mathbb{Z}^n, ind : \mathbb{N}, l : \mathbb{L}, h : \mathbb{Z})$$

$$Q = \left(\forall j \in [1..n]: a_j = a'_j \wedge \forall j \in [1..n-1]: a_j \leq a_{j+1}\right)$$

$$R = Q \wedge \begin{pmatrix} l = \exists j \in [1..n]: a_j = h \wedge \\ l \rightarrow (ind \in [1..n] \wedge a_{ind} = h) \end{pmatrix}$$

<table>
<tr><td colspan="4">$lowB := 1, highB := n, l :=\downarrow$</td></tr>
<tr><td colspan="4">$\neg l \wedge lowB \leq highB$</td></tr>
<tr><td colspan="4">$ind := (lowB + highB)/2$</td></tr>
<tr><td>$a_{ind} > h$</td><td>$a_{ind} < h$</td><td colspan="2">$a_{ind} = h$</td></tr>
<tr><td>$highB := ind - 1$</td><td>$lowB := ind + 1$</td><td colspan="2">$l :=\uparrow$</td></tr>
</table>

Programozási tételek

Bináris keresés

Megvalósítás:

```
int a[n]; // feldolgozandó monoton növő sorozat
int h; // keresett érték
… // a sorozat és keresett érték megadása

bool l = false;
in lowB = 0, highB = n-1; // kezdő határok
while (!l && lowB <= highB)
{
   int ind = (lowB + highB) / 2; // középre állás
   if (a[i] < h) highB = ind - 1;
   if (a[i] > h) lowB = ind + 1;
   if (a[i] == h) l = true; // ha megtaláltuk
}
```

Rendezések

Típusai

- A rendezések feladata egy adott sorozat elemeinek növekvő, vagy csökkenő sorrendbe állítása
  - az egy elemű sorozat mindig rendezett, csak a nagyobb számúakat kell rendezni
- A rendezéseknek két típusát tartjuk nyilván:
  - *összehasonlító*: az elemek összehasonlításával állapítja meg az egymáshoz viszonyított sorrendjüket, ez önmagában a sorozat ismeretéve levégezhető
  - *nem összehasonlító*: abszolút sorrendiséget állapít meg kategóriák, és darabszámok megállapításával, amihez az elemeken felül további információk szükségesek

Rendezések

Kiválasztásos rendezés

- A kiválasztásos rendezés kiválasztja a rendezetlen részsorozat minimális (vagy maximális) elemét, és azt a rendezett részsorozat elé (vagy mögé) helyezi
  - kicseréli azt a rendezetlen részsorozat első, vagy utolsó elemével, és növeli a rendezett részsorozat hosszát
  - a rendezetlen részsorozat kezdetben az egész sorozat, majd folyamatosan csökken
  - egy ciklussal növeljük a rendezett részsorozat hosszát, ezt elég az utolsó előtti elemig futtatni (hiszen az egy elemű sorozat mindig rendezett)
  - a kiválasztáshoz szükség van egy minimum-, vagy maximumkeresésre az adott részsorozatban

Rendezések

Kiválasztásos rendezés

- Pl.: `10 03 09 04 15 04 12`

kiválasztjuk a 3-at a rendezetlen részsorozatból, és cserélünk

`03 10 09 04 15 04 12`

a rendezett részsorozat hossza nő

```
03 04 09 10 15 04 12
03 04 04 10 15 09 12
03 04 04 09 15 10 12
03 04 04 09 10 15 12
03 04 04 09 10 12 15
```

Absztrakt leírás:

$$A = (a : \mathbb{Z}^n, ind : \mathbb{N}, l : \mathbb{L}, h : \mathbb{Z})$$

$$Q = \left(\forall j \in [1..n]: a_j = a'_j\right)$$

$$R = \begin{pmatrix} \{a_1, \dots, a_n\} = \{a'_1, \dots, a'_n\} \wedge \\ \forall j \in [1..n-1]: a_j \leq a_{j+1} \end{pmatrix}$$

$j := 1$
$j \leq n - 1$
$ind, max := minSearch(a[j \dots n])$
$swap(a_{ind}, a_j)$
$j := j + 1$

Rendezések

Kiválasztásos rendezés

Megvalósítás:

```
int a[n]; // feldolgozandó monoton növő sorozat
… // a sorozat megadása

for (int j = 0; j < n - 1; j++){
   int ind = minSearch(a, j, n);
   // manimumkeresés a résztömbön

   int temp = a[ind]; // két elem cseréje
   a[ind] = a[j];
   a[j] = temp;
}
```

Rendezések

Kiválasztásos rendezés

Feladat: Olvassunk be 10 sort a bemenetről, és rendezzük őket hossz szerint növekvő sorrendbe.

- használjunk maximumkiválasztásos rendezést a sor hosszára
- mivel a sorokat többször is fel kell dolgozni, mindenképpen el kell tárolnunk őket egy tömbbe

Megoldás:

```
int main(){
   string lines[10]; // a sorok
   for (int i = 0; i < 10; i++) // beolvasás
      getline(cin, lines[i]); // soronként
```

Rendezések

Maximumkeresés

Megoldás:

```
      for (int j = 0; j < 9; j++){ // rendezés
         int ind = j; // maximumkeresés
         for (int i = j; i < 10; i++){
            if (lines[i].length() >
                     lines[ind].length()){
               ind = i;
            }
         }
         string temp = lines[ind]; // elemek cseréje
         lines[ind] = lines[j];
         lines[j] = temp;
      }
      return 0;
   }
```

Rendezések

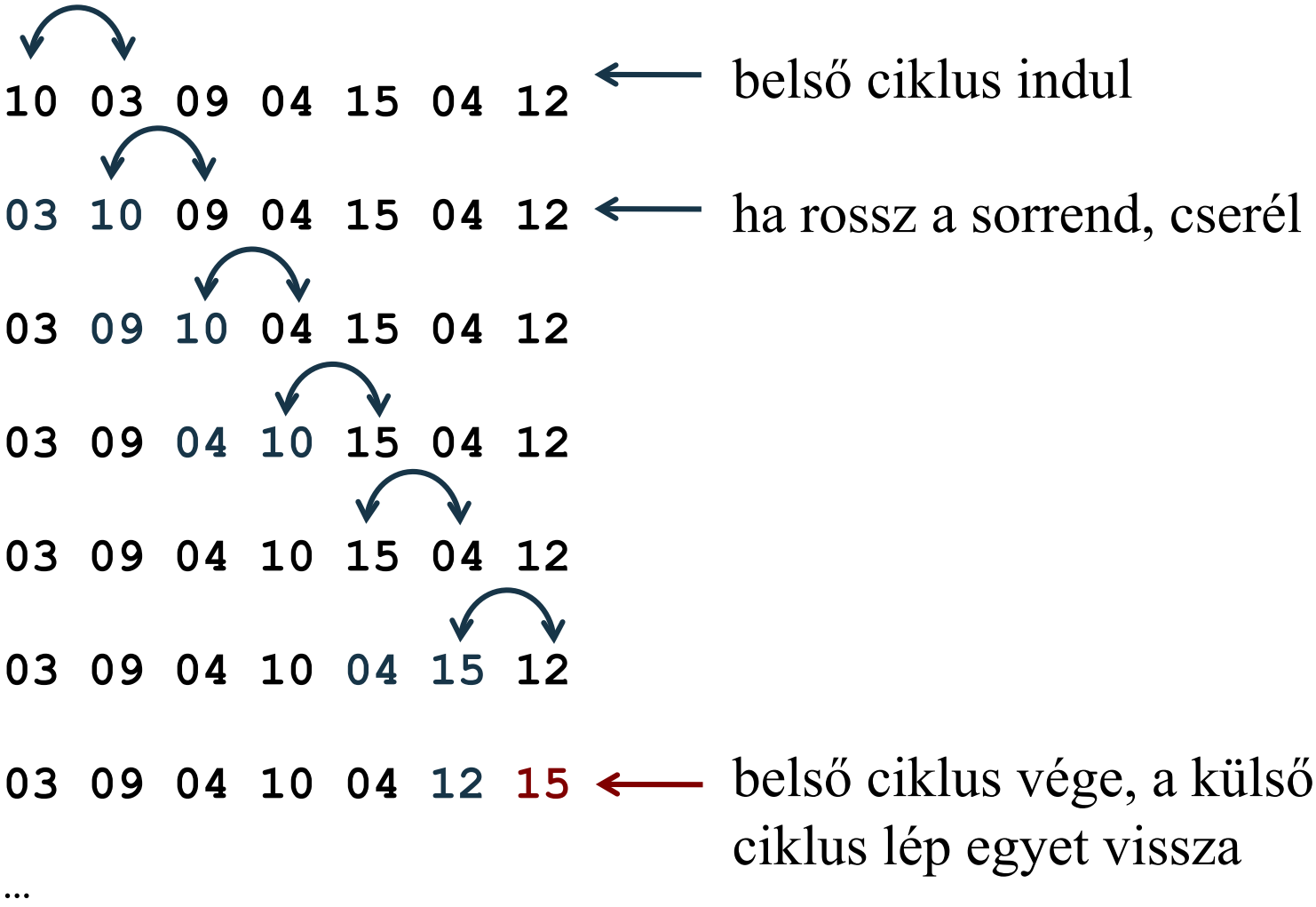
Buborékrendezés

- A buborékrendezés összehasonlít egy elemet a rákövetkezővel, és amennyiben rossz sorrendben vannak, megcseréli őket
  - így az elemek „felbuborékolódnak” a rendezetlen részsorozat végére
  - egy külső ciklus szabályozza, hogy hány rendezetlen elem van még a sorozatban (ez a második elemig halad visszafelé, hiszen az egy hosszú sorozatot már nem kell rendezni)
  - a belső ciklus végighalad a rendezetlen részsorozaton az elejétől annak utolsó előtti eleméig, ellenőrzi, hogy az elemek rossz sorrendben vannak-e, és amennyiben igen, megcseréli őket

Rendezések

Buborékrendezés

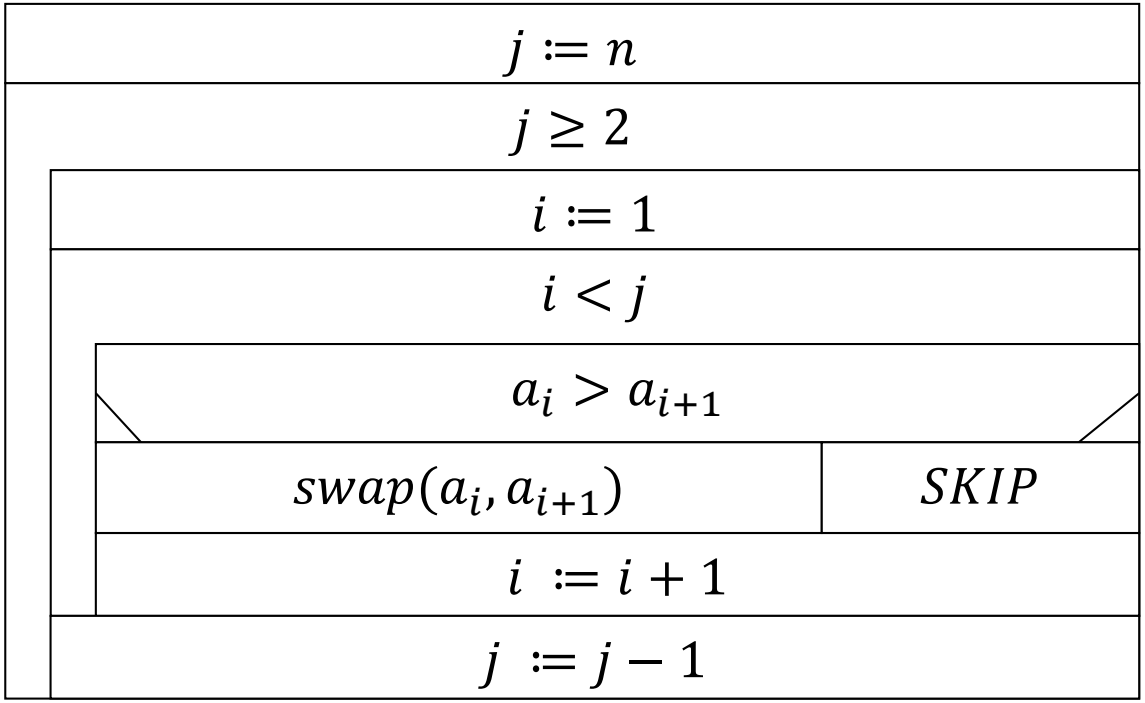
- Pl.:



Rendezések

Buborékrendezés

Absztrakt leírás:



Rendezések

Buborékrendezés

Megvalósítás:

```
int a[n]; // feldolgozandó monoton növő sorozat
… // a sorozat megadása

for (int j = n - 1; j > 0; j--){ // külső ciklus
   for (int i = 0; i < j; i++){ // belső ciklus
      if (a[i] > a[i + 1]){
         // ha rossz a sorrend
         int temp = a[ind]; // két elem cseréje
         a[ind] = a[j];
         a[j] = temp;
      }
   }
}
```