Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Bevezetés a programozásba I

12. gyakorlat

C++: programozási tételek, rendezések

© 2011.11.29. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Programozási tételek
Algoritmusok és programozási tételek

- *Algoritmusnak* nevezzük azt a műveletsorozatot, amely a feladat megoldásához vezet
  - az algoritmusokat célszerű általánosan (absztraktan) megfogalmazni, hogy a változtatások (transzformációk) könnyen véghezvihetők legyenek
  - kész algoritmusokat azért célszerű használni, mert jó, bizonyított megoldását adják a feladatnak
- Az egyszerű, sorozatokra alkalmazott algoritmusokat nevezzük *programozási tételek*nek, ezek a következők:
  - elemenkénti feldolgozás
  - összegzés, számlálás, lineáris keresés, bináris keresés
  - maximumkeresés, feltételes maximumkeresés

Programozási tételek
Feltételes maximumkeresés

- A maximumkeresés tételének azon változatát, ahol csak a sorozat bizonyos elemei között keresünk maximumot, *feltételes maximumkeresés*nek nevezzük
  - pl. keressük meg az egész sorozat legnagyobb páros elemét, keressük meg az origóhoz legközelebb eső pontot, …
  - 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
  - a maximum értékhez használunk egy leképezést a forrás típusról egy eredménytípusra

Programozási tételek
Feltételes maximumkeresés

  - nem garantált, hogy van olyan eleme a sorozatnak, ami teljesíti a feltételt, ezért egy logikai értékkel jeleznünk kell, hogy találtunk-e ilyet
  - tehát az első ilyen elemet rögtön megjelöljük maximumnak, és amennyiben lesz nagyobb, a feltételt teljesítő elem, akkor a maximumot átállítjuk

- *Algoritmus:*

```
bool l = false;
// találtunk-e a feltételt teljesítő elemet
<forrás típusa> item, x; // maximum elem
<eredmény típusa> max; // maximum elem értéke
```


Programozási tételek
Feltételes maximumkeresés

```
while (<nincs vége a sorozatnak>){
   <sorozat aktuális elemének beolvasása x-be>
   if (<x teljesíti a feltételt>)
      if(!l){ // ez az első ilyen elem
         l = true; item= x;
         max = <értékleképezés>(x);
      }
      else { // már volt ilyen elem
         if (<x értéke nagyobb max-nál>){
            item = x;
            max = <értékleképezés>(x);
         }
      }
}
```


Programozási tételek
Példa

Feladat: Keressük meg egy szekvenciális fájlban a legrövidebb szót, amely ‘a’-val kezdődik.

- használjunk feltételes maximum keresést, amelynek feltétele, hogy a szó ‘a’-val kezdődik
- a szavak hosszát hasonlítjuk össze, a legrövidebbet keressük
- a programozási tételt alprogramban valósítjuk meg, amelynek három kimenő értéke lesz (teljesült-e valahol a feltétel, a legrövidebb szó és annak hossza), ezért nem csak a visszatérési értékben, hanem a paraméterekben is adunk vissza értéket (cím szerinti paraméterátadással)

Programozási tételek
Példa

Megoldás:

```
…
/* feltételes minimum keresés, amely megadja a
   legrövidebb 'a'-val kezdődő szót
   bemenet: adatokat tartalmazó fájl
   kimenet: van-e a feltételnek eleget tevő szó
            (visszatérési érték), ha igen, akkor a
            szó és annak hossza (cím szerint
            átadott paraméterben)
*/
bool FeltMinKer(ifstream&, string&, int&);
```


Programozási tételek
Példa

Megoldás:

```
int main (int argc, char* argv[]){
   if (argc > 1){
      ifstream f(argv[1]);
      if (f.good()){
         string szo; int hossz;
         // ezekbe rakjuk az eredményt
         if (FeltMinKer(f, szo, hossz))
            … // találtunk megfelelő elemet
         f.close(); // fájl bezárása
      }
   }
   return 0;
}
```


Programozási tételek
Feltételes maximumkeresés

Megoldás:

```
bool FeltMinKer(ifstream& f, string& elem,
                int& min){
   bool l = false;
   string x; // a forrás típusa szöveg
   f >> x; // előreolvasás

   while(!f.eof()){
      if (x[0] == 'a'){ // a-val kezdődik
         if (!l){ // ez az első ilyen elem
            l = true;
            elem = x;
            min = x.length();
         }
```


Programozási tételek
Feltételes maximumkeresés

Megoldás:

```
         else // már volt a-val kezdődő
            if (x.length() < min){
               elem = x;
               min = x.length();
            }
      }
      f >> x;
   }
   return l;
   // visszaadjuk, hogy találtunk-e ilyen szót
}
```


Programozási tételek
Bináris keresés

- Amennyiben egy értéket keresünk a bemenő adatok sorozatban, és annak elemei növekvően (vagy csökkenően) rendezettek, lehetőségünk van a *bináris keresés*t használni
  - a bináris keresés nem sorrendben dolgozza fel a bemenetet, hanem felhasználja azt, hogy rendezett a sorozat
  - vegyünk egy elemet a sorozatból, ha az az elem a keresett, akkor végeztünk, ha kisebb, mint a keresett, akkor tudjuk, hogy 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 nem elemenkénti feldolgozás, mert nem sorban veszi az elemeket, és egyszerre többet is feldolgozhat
  - ezért 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 ekkor fordítani kell a relációkon

Programozási tételek
Bináris keresés

- *Algoritmus:*

```
bool l = false; // megtaláltuk-e a keresett elemet
int ah = <első index>, fh = <utolsó index>;
// tartomány alsó és felső határa
while (ah <= fh && !l){
   int kozep = (ah + fh) / 2;
   // középső értéket vesszük
   if (<kozep-en lévő elem egyezik a keresettel>)
      l = true;
   if (<kozep-en lévő érték nagyobb>)
      fh = kozep - 1; // a felső határt állítjuk
   if (<kozep-en lévő érték kisebb>)
      ah = kozep + 1; // az alsó határt állítjuk
}
```


Programozási tételek
Bináris keresés

Feladat: Adott egy szekvenciális fájlban egy évfolyam névsora sorba rendezve. Keressük meg, van-e „Nagy” vezetéknevű hallgató.

- feltehetjük, hogy minden név új sorban van
- mivel a fájl rendezett, használhatunk bináris keresést, ehhez azonban először be kell olvasnunk az adatokat egy vektorba, és utána futtatni a keresést
- az összehasonlításhoz vegyük az első 5 karakter összehasonlítását a „Nagy ” szöveggel (így a hasonlóan kezdődő vezetékneveket nem válogatja be)
- kihasználjuk, hogy a `string` összehasonlító operátorai lexikografikus összehasonlítást végeznek

Programozási tételek
Bináris keresés

Megoldás:

```
…
/* bináris keresés, amely "Nagy" vezetéknevű
   hallgatót keres
   bemenet: adatok egy vektorban (nevek)
   kimenet: találtunk-e "Nagy" vezetéknevű
            hallgatót
*/
bool BinarisKereses(vector<string> nevek);
```


Programozási tételek
Bináris keresés

Megoldás:

```
/* adatok beolvasása egy vektorba
   bemenet: az adatokat tartalmazó fájlnév
            (fn) és az üres vektor (nevek)
   kimenet: sikeres volt-e a beolvasás, és, ha
            igen, akkor a nevek vektora feltöltve
*/
bool Beolvas(string fn, vector<string> &nevek);

int main(){
   string fajlnev;
   vector<string> nevek;
   cout << "Adatok fájlja: ";
   cin >> fajlnev;
```


Programozási tételek
Bináris keresés

Megoldás:

```
   if (Beolvas(fajlnev, nevek)){
      if (BinarisKereses(nevek))
         …
   }
   return 0;
}

…
bool BinarisKereses(vector<string> nevek){
   bool l = false;
   int ah = 0, fh = nevek.size() - 1;
   // tartomány alsó és felső határa
```

Programozási tételek
Bináris keresés

Megoldás:

```
while (ah <= fh && !l){
   int kozep = (ah + fh) / 2;
   if (nevek[kozep].substr(0, 5) == "Nagy ")
      // nézzük a szóközt is a szó után
      l = true;
   if (nevek[kozep].substr(0, 5) > "Nagy ")
      // betűrend szerinti összehasonlítás
      fh = kozep - 1;
   if (nevek[kozep].substr(0, 5) < "Nagy ")
      ah = kozep + 1;
}
return l; // visszaadjuk, hogy megtaláltuk-e
}
```


Rendezések
Típusai

- A rendezések feladata egy adott sorozat elemeinek rendezése növekvő, vagy csökkenő sorrendben
  - az egy elemű sorozat mindig rendezett, csak a nagyobb számúakat kell rendezni
- A rendezéseknek két típusát tartjuk nyilván:
  - *nem összehasonlító*: abszolút sorrendiséget állapít meg kategóriák, és darabszámok megállapításával, ez elvégezhető elemenkénti feldolgozás alapján
  - *összehasonlító*: az elemek összehasonlításával állapítják meg az egymáshoz viszonyított sorrendjüket, ez nem végezhető el elemenkénti feldolgozásként, mert egyszerre kell ismerni a teljes sorozatot

Rendezések
Típusai

- Az összehasonlító rendezések három egyszerű algoritmusa:
  - *kiválasztásos rendezés*: kiválasztja a következő elemét a rendezett sorozatnak a nem rendezettek körül, és megcseréli azok sorrendjét
  - *beszúró rendezés*: kiválasztja a következő elemet, majd megkeresi a következő elem megfelelő helyét a sorozatban, és oda beszúrja
  - *buborékrendezés*: páronként összehasonlítja az elemeket, és felcseréli őket, amennyiben rossz a sorrendjük
- Ezek az algoritmusok mind beágyazott ciklussal végzik el a feldolgozást, ezért nem túl hatékonyak, vannak jobb, de bonyolultabb megoldások is (versenyrendezés, kupacrendezés, gyorsrendezés, …)

Rendezések
Kiválasztásos rendezés

- A kiválasztásos rendezés kiválasztja a rendezetlen részsorozat minimális (vagy maximális) elemét, és azt a rendezett részsorozat elé (vagy mögé) helyezi
  - úgy, hogy 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`

Rendezések
Kiválasztásos rendezés

- *Algoritmus:*

```
void KivalasztoRend(<felsorolható típus> elemek){
   <i-vel az első elemre lépünk>
   while (<i-vel nem értük el az utolsó elemet>){
      int ind = MinimumKereses(elemek,
                  <aktuális elem helye>);
      // minimumkeresés az aktuális pozíció után

      int temp = elemek[ind]; // két elem cseréje
      elemek[ind] = items[i];
      elemek[i] = temp;
      <i-vel lépés a következő elemre>
   }
}
```

Rendezések
Kiválasztásos rendezés

```
int MinimumKereses(<felsorolható típus> elemek,
                   int kezdo_pozicio){
   int ind = <kezdo_pozicio rákövetkezője>;
   int i = <kezdo_pozicio második rákövetkezője>;
   while (<i-vel nem értünk a sorozat végére>){
      if (items[ind] > items[i])
         ind = i;
      <i-vel lépés a következő elemre>
   }
   return ind;
}
```

- A maximum-kiválasztásos rendezés esete analóg módon adható meg, de akkor a tömb elejéről építjük fel a rendezett résztömböt, illetve maximumot keresünk

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 pedig 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.:

```
10 03 09 04 15 04 12   ← belső ciklus indul
03 10 09 04 15 04 12   ← ha rossz a sorrend, cserél
03 09 10 04 15 04 12
03 09 04 10 15 04 12
03 09 04 10 15 04 12
03 09 04 10 04 15 12
03 09 04 10 04 12 15   ← belső ciklus vége, a külső ciklus lép egyet vissza
…
```


Rendezések
Buborékrendezés

- *Algoritmus:*

```
void BuborekRend(<felsorolható típus> elemek){
   <i-vel a sorozat utolsó elemére lépünk>
   while (<i-vel nem értük el az első elemet>){
      <j-vel a sorozat első elemére lépünk>
      while (<j-vel nem értük el i-t>){
         if (elemek[j] > elemek[j + 1]){
            int temp = elemek[j];
            elemek[j] = elemek[j + 1];
            elemek[j + 1] = temp;
         }  // csere a rákövetkezővel
         <j-vel a következő elemre lépünk> }
      <i-vel az előző elemre lépünk> }
}
```


Rendezések
Példa

Feladat: Adjuk meg síkbeli pontok sorozatát, és rendezzük azt az origótól való távolság függvényében.

- az adatokat olvassuk be billentyűzetről biztonságosan
- meg kell valósítanunk a síkbeli pont típusát rekord segítségével (a pont az X és Y koordinátából, mint egész számokból fog állni), és ahhoz az összehasonlítás műveletét (< operátor) a rendezéshez, továbbá kényelmi okokból célszerű a beolvasást és kiírást (<<, >> operátorok) is megvalósítani
- használjuk a kiválasztó rendezés algoritmusát, amelyhez így külön meg kell írni a megfelelő minimumkeresést is, a minimumkeresésben fogjuk kihasználni a típusunk operátorát

Rendezések
Példa

Megoldás:

```
…
struct Pont { // pont típusa
   int X;
   int Y;
}; // a pontnak tároljuk a két koordinátáját

/* összehasonlító operátor
   bemenet: két pont (a, b)
   kimenet: igaz, ha a kisebb, mint b */
bool operator<(Pont a, Pont b) {
    return (a.X * a.X + a.Y * a.Y) <
           (b.X * b.X + b.Y * b.Y);
}
```

Rendezések
Példa

Megoldás:

```
/* beolvasó operátor
   bemenet: adatfolyam (i)
   kimenet: az adatfolyam, valamint
            a beolvasott pont (p) */
istream& operator>>(istream& i, Pont& p) {
   string strX, strY;
   stringstream conv;
   i >> strX >> strY; // biztonságos beolvasás
   conv << strX; conv >> p.X; // konverzió
   conv.clear();
   conv.str(strY); conv >> p.Y;
   return i;
}
```


Rendezések
Példa

Megoldás:

```
/* kiíró operátor
   bemenet: adatfolyam (o), valamint a kiírandó
            pont (p)
   kimenet: az adatfolyam */
ostream& operator<<(ostream& o, Pont p) {
    o << "(" << p.X << "," << p.Y << ")";
    return o;
}

/* pontok rendezése kiválasztó rendezéssel
   bemenet: pontok vektora (elemek)
   kimenet: pontok vektora rendezve */
void KivalasztoRendezes(vector<Pont> &elemek);
```


Rendezések
Példa

Megoldás:

```
…
void KivalasztoRendezes(vector<Pont> &elemek) {
    for (int i = 0; i < elemek.size() - 1; i++) {
        int ind = Minimum(elemek, i);

        Pont temp = elemek[ind]; // csere
        elemek[ind] = elemek[i];
        elemek[i] = temp;
    }
}
```