Algoritmi i strukture podataka
9. Čas — Tehnika dva pokazivača, stabilno sortiranje

Sana Stojanović Đurđević

December 7, 2018

Pred vama je prateći materijal koji se može koristiti uz skriptu. Sastoji se od rešenja nekoliko zadataka koji su pređeni na vežbama i predavanjima.

1 Tehnika dva pokazivača

1.1 Par brojeva datog zbira

Problem: Definisati algoritam složenosti $O(n)$ koji određuje koliko u datom rastuće sortiranom nizu dužine n (svi elementi su različiti) postoji parova brojeva na različitim pozicijama čiji je zbir jednak datom broju s.

```
#include <iostream>
#include <vector>

using namespace std;

// rekurzivno resenje
int brojParova(const vector<int>& a, int s, int l, int d) {
  if (l >= d)
    return 0;
  if (a[l] + a[d] > s)
    return brojParova(a, s, l, d - 1);
  else if (a[l] + a[d] < s)
    return brojParova(a, s, l + 1, d);
  else
    return 1 + brojParova(a, s, l + 1, d - 1);
}
int brojParova(const vector<int>& a, int s) {
  int n = a.size();
  return brojParova(a, s, 0, n-1);
}
```

```
// iterativno resenje
int brojParovaDatogZbira(const vector<int>& a, int s) {
  int n = a.size();
  int brojParova = 0;
  int levo = 0, desno = n - 1;
  while (levo < desno)
    if (a[levo] + a[desno] > s)
      desno--;
    else if (a[levo] + a[desno] < s)
      levo++;
    else{
      brojParova++;
      levo++;
      desno--;
    }
  return brojParova;
}
```

```
int main(){
  int n;
  cout << "Unesite n: " << endl;
  cin >> n;
  vector<int> a(n);

  cout << "Unesite elemente: " << endl;
  for(int i=0; i<n; i++)
    cin >> a[i];

  cout << "Unesite broj s: " << endl;
  int s;
  cin >> s;

  cout << "Parova trazenog zbira ima: " << brojParova(a,s) << endl;
  cout << "Iterativno resenje: " << brojParovaDatogZbira(a,s) << endl;

}
```

1.2 Trojka brojeva sa zbirom nula

Problem: Definisati algoritam složenosti $O(n^2)$ koji određuje da li u datom sortiranom nizu dužine n postoji trojka elemenata čiji je zbir 0.

```
#include<iostream>
#include<vector>

using namespace std;

int trojkaZbiraNula(const vector<int>& a) {
  int n = a.size();
  // ukupan broj trojki iji je zbir 0
  int brojTrojki = 0;
  for (int i = 0; i < n - 2; i++) {
    // izracunavamo broj parova u delu niza [i+1, n)
    // ciji je zbir -a[i]
    // posto je ceo niz sortiran, sortiran je i taj deo
    int l = i + 1;
    int d = n - 1;
    while (l < d) {
      if (a[i] + a[l] + a[d] > 0)
        d--;
      else if (a[i] + a[l] + a[d] < 0)
        l++;
      else {
        brojTrojki++;
        l++;
        d--;
      }
    }
  }
  return brojTrojki;
}
```

```
int main(){
  int n;
  cout << "Unesite broj elemenata: " << endl;
  cin >> n;
  vector<int> a(n);
  cout << "Unesite elemente: " << endl;
  for(int i=0; i<n; i++)
    cin >> a[i];

  cout << "Trojki zbira nula ima: " << trojkaZbiraNula(a) << endl;
  return 1;
}
```

1.3 Segmenti niza prirodnih brojeva koji imaju dati zbir

Problem: U datom nizu pozitivnih prirodnih brojeva naći sve segmente (njihov početak i kraj) čiji je zbir jednak datom pozitivnom broju (brojanje pozicija

počinje od nule).

```
#include <iostream>
#include <vector>

using namespace std;

void ispisiSegmenteDatogZbira(const vector<int>& a, int trazeniZbir){
  // granice segmenta
  int i = 0, j = 0;
  int n = a.size();
  // zbir segmenta
  int zbir = a[0];
  while(true) {
    // na ovom mestu vazi da je zbir = sum(ai, ..., aj) i da
    // za svako i <= j' < j vazi sum(ai, ..., aj') < trazeniZbir
    if (zbir < trazeniZbir) {
      // prelazimo na interval [i, j+1]
      j++;
      // ako takav interval ne postoji, zavrsili smo pretragu
      if (j >= n)
        break;
      // izracunavamo zbir intervala [i, j+1]
      // na osnovu zbira intervala [i, j]
      zbir += a[j];
    }
    else
      {
        // ako je zbir jednak trazenom,
        // vazi da je sum(ai, ..., aj) = trazeniZbir
        // pa prijavljujemo interval
        if (zbir == trazeniZbir)
          cout << i << " " << j << endl;
        // prelazimo na interval [i+1, j]
        // izracunavamo zbir intervala [i+1, j]
        // na osnovu zbira intervala [i, j]
        zbir -= a[i];
        i++;
      }
  }
}
```

```
int main(){
  int n;
  cout << "Unesite broj elemenata: " << endl;
  cin >> n;
  vector<int> a(n);
  cout << "Unesite pozitivne prirodne brojeve: " << endl;
  for(int i=0; i<n; i++)
    cin >> a[i];

  cout << "Unesite trazeni zbir: " << endl;
  int zbir;
  cin >> zbir;

  cout << "Segmenti datog zbira su: " << endl;
  ispisiSegmenteDatogZbira(a,zbir);
}
```

1.4 Najkraća podniska koji sadrži sva data slova

Problem: Definisati funkciju koja u datoj niski određuje najkraću podnisku koja sadrži sve karaktere iz datog skupa karaktera.

Na primer, u niski `xCxxBxxBxxAxxBxxCxxxxBxAxAxCxxxBx` tražimo podnisku u kojoj se javljaju slova iz skupa `ABC`.

```
#include <iostream>
#include <vector>
#include <limits>
#include <map>

using namespace std;
```

```
// niska koja se pretrazuje i skup u kome se nalaze trazeni karakteri
int najkracaPodniska(const string& niska, const string& S) {

  // kreiramo vektor relevantnih pozicija
  vector<int> poz_karaktera_iz_S;
  for (int i = 0; i < niska.size(); i++)
    // ako se niska[i] nalazi u skupu S
    if (S.find(niska[i]) != string::npos)
      // zapamti njegovu poziciju i
      poz_karaktera_iz_S.push_back(i);

  // najmanja duzina podniske
  int min_duzina = numeric_limits<int>::max();

  // broj pojavljivanja svakog relevantnog karaktera
  // u trenutnoj podniski
  map<char, int> broj_pojavljivanja_u_podniski;

  // trenutna podniska je odredjena pozicijama [i, j]
  vector<int>::const_iterator i, j;

  for (i = j = poz_karaktera_iz_S.begin();
       j != poz_karaktera_iz_S.end(); j++){
    // trenutnu podnisku prosirujemo do sledece pozicije j i
    // uvecavamo broj pojavljivanja karaktera koji se nalazi na
    // poziciji odredjenoj sa j (jer se i on sada javlja u podniski)
    broj_pojavljivanja_u_podniski[niska[*j]]++;

    // ako mapa ima isto elemenata kao i skup S, onda su svi elementi
    // skupa S prisutni u podniski
    if (broj_pojavljivanja_u_podniski.size() == S.size()) {

      // trazimo najkracu podnisku koji se zavrsava na poziciji
      // odredenoj sa j tako sto podnisku skracujemo sa leve strane
      // dok god se prvi karakter podniske javlja vise puta u njemu
      while (broj_pojavljivanja_u_podniski[niska[*i]] > 1) {
        // podnisku skracujemo i
        // uklanjamo sve karaktere sa pocetka sve do
        // sledeceg karaktera iz skupa S
        broj_pojavljivanja_u_podniski[niska[*i]]--;
        i++;
      }

      // izracunavamo duzinu trenutne podniske
      int duzina = *j - *i + 1;
      // azuriramo minimum ako je to potrebno
      if (duzina < min_duzina)
        min_duzina = duzina;
    }
  }

  // prijavljujemo rezultat
  if (min_duzina != numeric_limits<int>::max())
    return min_duzina;
  else
    return -1;
}
```

```
int main(){
  string niska;
  cout << "Unesite nisku: " << endl;
  cin >> niska;

  string S;
  cout << "Unesite skup karaktera koji trazite: " << endl;
  cin >> S;

  cout << "Duzina najkrace podniske je: ";
  cout << najkracaPodniska(niska, S) << endl;
}
```

2 Sortiranje

2.1 Stabilnost sortiranja

Algoritam sortiranja se naziva stabilnim (engl. stable) ako se prilikom sortiranja, za elemente koji su jednaki po parametru sortiranja, zadržava originalni redosled elemenata.

Funkcija `stable_sort` garantuje stabilnost:
`http://www.cplusplus.com/reference/algorithm/stable_sort/`

```
// stable_sort example
#include <iostream>     // std::cout
#include <algorithm>    // std::stable_sort
#include <vector>       // std::vector

using namespace std;

bool compare_as_ints (double i,double j)
{
  return (int(i)<int(j));
}

int main () {
  double mydoubles[]={3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};

  vector<double> myvector;

  myvector.assign(mydoubles,mydoubles+8);

  cout << "using default comparison:";
  stable_sort (myvector.begin(), myvector.end());
  for (vector<double>::iterator it=myvector.begin();
       it!=myvector.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';

  myvector.assign(mydoubles,mydoubles+8);

  cout << "using 'compare_as_ints' :";
  stable_sort (myvector.begin(), myvector.end(), compare_as_ints);
  for (vector<double>::iterator it=myvector.begin();
       it!=myvector.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';

  return 0;
}
```

```
Possible output:

using default comparison: 1.32 1.41 1.62 1.73 2.58 2.72 3.14 4.67
using 'compare_as_ints' : 1.41 1.73 1.32 1.62 2.72 2.58 3.14 4.67
```

2.2 Sortiranje prebrojavanjem

Kada se zna da svi elementi niza dolaze iz nekog relativno uskog intervala vrednosti, sortiranje je moguće uraditi i efikasnije nego kada se koristi sorti-

ranje upoređivanjem. Jedna ideja je da se sortiranje vrši prebrojavanjem tj. određivanjem broja pojavljivanja svake vrednosti iz dopuštenog intervala vrednosti i da se onda niz rekonstruiše tj. popuni iz početka na osnovu izračunatog broja pojavljivanja. Ako je unapred poznato da je interval vrednosti interval $[0, m]$, tada za prebrojavanje možemo upotrebiti običan niz brojača.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void sortiranjePrebrojavanjem(vector<int>& a, int n) {
  // funkcija vraca iterator na maksimalni element
  int m = *max_element(a.begin(), a.end());

  // brojimo pojavljivanja svakog elementa niza
  vector<int> frekvencije(m+1, 0);
  for (int i = 0; i < n; i++){
    frekvencije[a[i]]++;
  }

  // rekonstruisemo niz iz pocetka
  int k = 0;
  for (int j = 0; j <= m; j++)
    for (int i = 0; i < frekvencije[j]; i++)
      a[k++] = j;
}

int main(){
  int n;
  cout << "Unesite n: " << endl;
  cin >> n;
  vector<int> a(n);

  cout << "Unesite elemente: " << endl;
  for(int i=0; i<n; i++)
    cin >> a[i];

  sortiranjePrebrojavanjem(a, n);
  for(int i=0; i<n; i++)
    cout << a[i] << " ";
  cout << endl;

}
```

2.3 Sortiranje razvrstavanjem

Problem: Državna komisija je napravila spisak svih birača u državi. Potrebno je da se svakoj opštini distribuira spisak birača sa teritorije te opštine, ali tako da redosled ostane isti kakav je na polaznom spisku državne komisije. Na standardni izlaz ispisati spiskove za sve opštine, svaki u posebnom redu. Opštine treba da budu uređene leksikografski, rastuće.

```
#include <iostream>
#include <vector>
#include <map>

using namespace std;

// za svakog biraca poznata je opstina sa koje dolazi i sifra
typedef struct Birac {
  string opstina;
  string sifra;
}Birac;

void sortiraj(vector<Birac>& biraci) {
  // izracunavamo broj biraca iz svake opstine
  // preslikavanje brojBiraca odredjuje broj biraca za dati
  // naziv opstine
  map<string, int> brojBiraca;

  // prolazimo kroz spisak svih biraca
  for (auto birac : biraci)
    // uvecavamo broj biraca na opstini trenutnog biraca
    brojBiraca[birac.opstina]++;

  cout << "Broj biraca po opstinama ";
  cout << "(sortiran leksikografski) je: " << endl;
  for (auto it = brojBiraca.begin(); it != brojBiraca.end(); ++it)
    cout << it->first << " " << it->second << endl;
```

```
  // izracunavamo poziciju u sortiranom nizu na kojoj
  // pocinju biraci sa date opstine
  map<string, int> pozicije;

  // ukupan broj biraca u do sada obradjenim opstinama
  int prethodnoBiraca = 0;
  // prolazimo kroz sve opstine u sortiranom redosledu
  // (abecedno, leksikografski)
  for (auto it : brojBiraca) {
    // tekuca opstina pocinje na poziciji odredjenoj
    // brojem biraca u prethodnim opstinama
    pozicije[it.first] = prethodnoBiraca;
    // uvecavamo broj biraca u prethodnim opstinama za broj
    // biraca u trenutnoj opstini, pripremajuci se za novu
    // iteraciju
    prethodnoBiraca += it.second;
  }

  cout << "Ispisujemo pozicije biraca ";
  cout << "(sortirano po opstinama): " << endl;
  for (auto it : pozicije){
    cout << it.first << " " << it.second << endl;
  }
```

```
  // kreiramo konacan sortirani niz biraca
  vector<Birac> sortirano(biraci.size());
  // prolazimo kroz sve birace iz polaznog niza
  for (auto birac : biraci) {
    // postavljamo biraca na tekuce mesto u njegovoj opstini
    sortirano[pozicije[birac.opstina]] = birac;
    // uvecavamo slobodnu poziciju u toj opstini
    pozicije[birac.opstina]++;
  }
  biraci = sortirano;
}
```

```
int main(){
  int n;
  cout << "Unesite broj biraca: ";
  cin >> n;
  vector<Birac> biraci(n);
  cout << "Unesite birace, prvo opstina pa sifra biraca:" << endl;
  for(int i=0; i<n; i++)
    cin >> biraci[i].opstina >> biraci[i].sifra;

  sortiraj(biraci);
  cout << "Biraci rasporedjeni po opstinama: " << endl;
  for(int i=0; i<n; i++)
    cout << biraci[i].opstina << " " << biraci[i].sifra << endl;
}
```

2.4 Sortiranje višestrukim razvrstavanjem

Problem: Dati niz prirodnih brojeva sortirati primenom sortiranja višestrukim razvrstavanjem.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void sortiranjeRazvrstavanjem(vector<int>& a, int s) {
  // brojimo pojavljivanje svake cifre uz koeficijent s
  // (on je uvek stepen desetke) 10^s
  int broj[10] = {0};
  for (int i = 0; i < a.size(); i++)
    broj[(a[i] / s) % 10]++;

  // pozicije u rezultujucem nizu ispred kojih se zavrsavaju grupe
  // elemenata sa odgovarajucim ciframa uz koeficijent s
  for (int i = 1; i < 10; i++)
    broj[i] += broj[i-1];

  // prepisujemo elemente u pomoni niz
  vector<int> pom(a.size());
  for (int i = a.size() - 1; i >= 0; i--)
    pom[--broj[(a[i] / s) % 10]] = a[i];
  // vraamo elemente nazad u originalni niz
  a = pom;
}
```

```
void sortiranjeVisestrukimRazvrstavanjem(vector<int>& a) {
  // odredjujemo maksimum niza a
  int max = numeric_limits<int>::min();
  for (int x : a)
    if (x > max)
      max = x;

  // na osnovu maksimuma odredjujemo koliko ima najvise cifara
  // sortiramo na osnovu svake cifre krenuvsi od cifara jedinica
  for (int s = 1; max / s > 0; s *= 10)
    sortiranjeRazvrstavanjem(a, s);
}
```

```
int main(){
  int n;
  cout << "Unesite broj elemenata niza: " << endl;
  cin >> n;
  vector<int> a(n);
  cout << "Unesite elemente: " << endl;
  for(int i=0; i<n; i++)
    cin >> a[i];

  sortiranjeVisestrukimRazvrstavanjem(a);
  cout << "Sortirani niz: " << endl;
  for(int i=0; i<n; i++)
    cout << a[i] << " ";
  cout << endl;
}
```