Algoritmi i strukture podataka
8. Čas — Primene sortiranja i binarne pretrage

Sana Stojanović Đurđević

November 25, 2019

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 Korišćenje set i unordered_set sa strukturama podataka

1.1 Set

A function becomes const when the const keyword is used in the functions declaration. The idea of const functions is not to allow them to modify the object on which they are called. It is recommended the practice to make as many functions const as possible so that accidental changes to objects are avoided.

Ovo je neophodno zbog: The value of the elements in a set cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.

Internally, the elements in a set are always sorted following a specific strict weak ordering criterion indicated by its internal comparison object (of type Compare).

By default, comparison object used with set is a less object, which returns the same as operator¡.

This object determines the order of the elements in the container: it is a function pointer or a function object that takes two arguments of the same type as the container elements, and returns true if the first argument is considered to go before the second in the strict weak ordering it defines, and false otherwise.

Two elements of a set are considered equivalent if comparison object returns false reflexively (i.e., no matter the order in which the elements are passed as arguments).

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;

struct Tacka {
  int x, y;

  bool operator < (const Tacka& other) const
   {
     return x < other.x || (x == other.x && y < other.y);
   };
};

int main(){
  int n;
  vector<Tacka> v;
  set<Tacka> s;

  Tacka t;
  int x, y;

  cout << "Unesi koliko ima tacaka: " << endl;
  cin >> n;
  cout << "Unesi tacke: " << endl;
  for(int i=0; i<n; i++){
    cin >> t.x;
    cin >> t.y;
    v.push_back(t);
    s.insert(t);
  }

  cout << "Ukupno ima: " << v.size() << " tacaka" << endl;
  cout << "Od toga imamo: " << s.size() << " razlicitih" << endl;
}
```

Moze se definisati eksterna funkcija poredjenja koja se posle navodi prilikom deklarisanja skupa:

```
#include <iostream>
#include <vector>
#include <set>
#include <unordered_set>

using namespace std;

struct tacka {
  int x, y;
};

// eksterna funkcija poredjenja
struct poredjenjeTacaka
{
    bool operator()(const tacka& lhs, const tacka& rhs) const
    {
        return (lhs.x < rhs.x) || ((!(rhs.x < lhs.x)) && (lhs.y < rhs.y));
    }
};

int main(){
  int n;
  vector<tacka> v;

  // navodi se funkcija poredjenja koja se koristi
  set<tacka,poredjenjeTacaka> s;

  tacka t;
  int x, y;

  cout << "Unesi koliko ima tacaka: " << endl;
  cin >> n;
  cout << "Unesi tacke: " << endl;

  for(int i=0; i<n; i++){
    cin >> t.x;
    cin >> t.y;
    v.push_back(t);
    s.insert(t);
  }

  cout << "Ukupno ima: " << v.size() << " tacaka" << endl;
  cout << "Od toga ima: " << s.size() << " razlicitih" << endl;
}
```

1.2 Unordered set

http://www.cplusplus.com/reference/unordered_set/unordered_set/

Internally, the elements in the `unordered_set` are not sorted in any particular order, but organized into buckets depending on their hash values to allow for fast access to individual elements directly by their values (with a constant average time complexity on average).

They use hash function - a unary function object type that takes an object of the same type as the elements as argument and returns a unique value of type `size_t` based on it. This can either be a class implementing a function call operator or a pointer to a function (see constructor for an example). This defaults to `hash<Key>`, which returns a hash value with a probability of collision approaching `1.0/std::numeric_limits<size_t>::max()`. The `unordered_set` object uses the hash values returned by this function to organize its elements internally, speeding up the process of locating individual elements.

`http://www.cplusplus.com/reference/unordered_set/unordered_set/bucket_count/`

A bucket is a slot in the container's internal hash table to which elements are assigned based on their hash value. The load factor influences the probability of collision in the hash table (i.e., the probability of two elements being located in the same bucket). The container automatically increases the number of buckets to keep the load factor below a specific threshold (its `max_load_factor`), causing a rehash each time an expansion is needed.

```
#include <iostream>
#include <vector>
#include <set>
#include <unordered_set>

using namespace std;

struct Tacka {
  int x, y;

  // unordered_set zahteva implementaciju operatora ==
  bool operator == (const Tacka& other) const {
    if (x == other.x && y == other.y)
      return true;
    return false;
  };
};
```

```
// pomocna struktura za izracunavanje hash funkcije tacke
struct TackaHash{
  size_t operator()(const Tacka& TackaZaHes) const noexcept{
    size_t hash = TackaZaHes.x + 10*TackaZaHes.y;
    return hash;
  }
};

// drugi nacin:
// pakujemo x i y u jedan 64-bitni broj i uzimamo njegov
// ugradjen hash
struct TackaHash64 {
size_t operator()(const Tacka& t) const {
return hash<int64_t>()((int64_t) t.x << 32 | (int64_t) t.y);
}
};
```

```
int main(){
  int n;
  vector<Tacka> v;

  // mora se naglasiti koja hes funkcija se koristi
  unordered_set<Tacka, TackaHash64> un_set;

  Tacka t;
  int x, y;

  cout << "Unesi koliko ima tacaka: " << endl;
  cin >> n;
  cout << "Unesi tacke: " << endl;

  for(int i=0; i<n; i++){
    cin >> t.x;
    cin >> t.y;
    v.push_back(t);
    un_set.insert(t);

   /*  Broj grupa u unordered_set ce uvek biti prost broj
       to je da bi raspodela elemenata po grupama bila
       sto bolja i za losije hash funkcije
    */
    cout << "Bucket count: " << un_set.bucket_count() << endl;
  }

  cout << "Ukupno ima: " << v.size() << " tacaka" << endl;
  cout << "Od toga ima: " << un_set.size() << " razlicitih" << endl;
}
```

1.3 Implementacija set

```
#include <iostream>

using namespace std;

struct Tacka {
int x, y;

bool operator<(const Tacka& b) const {
if(x < b.x)
return true;
if(x > b.x)
return false;
return y < b.y;
}
};

struct Cvor {
Tacka t;
Cvor *l, *d;
};

Cvor* napraviCvor(Tacka t) {
Cvor *p = new Cvor();
p->t = t;
p->l = nullptr;
p->d = nullptr;
return p;
}
```

```
void dodaj(Tacka t, Cvor*& koren) {
if(koren == nullptr)
koren = napraviCvor(t);
else if(t < koren->t)
dodaj(t, koren->l);
else if(koren->t < t)
dodaj(t, koren->d);
}

int prebroj(Cvor* koren) {
if(koren == nullptr)
return 0;
return 1 + prebroj(koren->l) + prebroj(koren->d);
}

void oslobodi(Cvor* koren) {
if(koren != nullptr) {
oslobodi(koren->l);
oslobodi(koren->d);
delete koren;
}
}

int main() {
int n;
cin >> n;

Cvor* koren = nullptr;
for(int i = 0; i < n; i++) {
Tacka t;
cin >> t.x >> t.y;
dodaj(t, koren);
}

cout << "Broj razlicitih tacaka je " << prebroj(koren) << endl;

oslobodi(koren);

return 0;
}
```

1.4 Implementacija unordered_set

```
#include <iostream>
#include <vector>
#include <algorithm>
#define my_prime 1000000009
#define my_count 100000

using namespace std;

struct Tacka {
int x, y;

bool operator==(const Tacka& b) const {
return x == b.x && y == b.y;
}
};
```

```
// my_hash preslikava tacku t u broj iz intervala [0, my_prime - 1]
// pakujemo dva 32-bitna broja u jedan 64-bitan i uzimamo ostatak
// pri deljenju sa prostim brojem my_prime
int my_hash(Tacka t) {
return ((int64_t) t.x << 32 | t.y) % my_prime;
}

// na osnovu hash funkcije racunamo poziciju grupe u koju smestamo
// tacku i dodajemo je u tu grupu ukoliko vec nije dodata
void dodaj(Tacka t, vector< vector<Tacka> >& s) {
int pos = my_hash(t) % s.size();
if(count(s[pos].begin(), s[pos].end(), t) == 0)
s[pos].push_back(t);
}

// racuna koliko je ukupno elemenata u hash tabeli
int prebroj(vector< vector<Tacka> >& s) {
int n = 0;
for(int i = 0; i < s.size(); i++)
n += s[i].size();
return n;
}
```

```
int main() {
int n;
cin >> n;

vector< vector<Tacka> > s(my_count);
for(int i = 0; i < n; i++) {
Tacka t;
cin >> t.x >> t.y;
dodaj(t, s);
}

cout << "Broj razlicitih tacaka je " << prebroj(s) << endl;

return 0;
}
```

2 Funkcije jezika C++, sort i binary_search

Neke od bibliotečkih funkcija koje se koriste za sortiranje i binarnu pretragu su:

sort funkcija `sort` vrši sortiranje neke kolekcije (niza, deka, liste). Parametri su par iteratora koji ograničavaju poluotvoreni raspon niza koji se sortira u rastućem poretku — prvi iterator ukazuje na prvi element, a drugi iterator ukazuje neposredno iza poslednjeg elementa. Ova funkcija nema povratnu vrednost.

Moguće je navesti i treći parametar koji predstavlja funkciju poređenja. Postoji više načina da se to uradi, ali najčešće ćemo koristiti anonimnu (lambda) funkciju. Funkcija proverava da li se njen prvi argument nalazi *ispred* njenog drugog argumenta, u željenom sortiranju.

`http://www.cplusplus.com/reference/algorithm/sort/`

binary_search funkcija `binary_search` vrši binarnu pretragu niza i vraća logičku vrednost tačno ako i samo ako je traženi element prisutan u rasponu koji se pretražuje. Naravno, kao i kod svih varijanti binarne pretrage, potrebno je da je raspon koji se pretražuje sortiran. Parametri su par iteratora koji ograničavaju poluotvoreni raspon i element koji se pretražuje. Funkciju poređenja (ako je ona korišćena prilikom sortiranja niza) moguće je navesti kao četvrti argument funkcije.

Ako nije navedena funkcija poređenja, elementi se porede koristeći operator $<$. Dva elementa se smatraju ekvivalentnim ako važi $(!(a < b)\&\&!(b < a))$. Sličan izraz važi ako se koristi funkcija poređenja.

`http://www.cplusplus.com/reference/algorithm/binary_search/`

lower_bound Funkcija `lower_bound` vraća iterator na prvi element koji je veći ili jednak datom (na prvi element koji *nije* manji od traženog). Parametri

su isti kao i kod funkcije `binary_search`. Ako ne postoji takav element funkcija vraća iterator koji pokazuje na element iza

`http://www.cplusplus.com/reference/algorithm/lower_bound/`

upper_bound Funkcija `upper_bound` vraća iterator na prvi element koji je strogo veći od datog. Parametri su isti kao i kod funkcije `binary_search`. Ako ne postoji takav element funkcija vraća iterator koji pokazuje na element iza poslednjeg.

Za razliku od `lower_bound` gde vrednost na koju iterator pokazuje može biti i jednaka traženoj vrednosti, vrednost na koju pokazuje iterator koji vraća `upper_bound` mora biti veća od tražene vrednosti.

`http://www.cplusplus.com/reference/algorithm/upper_bound/`

equal_range Funkcija `equal_range` vrši binarnu pretragu i vraća par iteratora koji ograničavaju poluotvoreni raspon u kom se nalaze svi elementi ekvivalentni datom. Parametri su isti kao i kod funkcije `binary_search`.

Povratna vrednost je par iteratora od kojih prvi pokazuje na `lower_bound`, a drugi na `upper_bound`.

`http://www.cplusplus.com/reference/algorithm/equal_range/`

2.1 Dužina pokrivača

Problem: Dato je n zatvorenih intervala realne prave. Odredi ukupnu dužinu koju ti intervali pokrivaju, kao i najmanji broj zatvorenih intervala koji pokrivaju isti skup tačaka kao i polazni intervali (oni se mogu dobiti ukrupnjavanjem polaznih intervala).

Da bi se rešio ovaj zadatak, sortiraćemo niz svih granica intervala, kojima ćemo pridružiti oznaku 1 ako predstavljaju početak, odnosno oznaku -1 ako predstavljaju kraj tekućeg intervala. Odnosno sortiraćemo vektor parova kojima je prvi element tekuća koordinata a drugi element vrednost 1 ili -1. Podrazumevano sortiranje za vektore parova jeste da se vrednosti sortiraju prvo po prvom polju, pa nakon toga ako ima više elemenata sa istim prvim poljem, onda se oni sortiraju prema vrednosti drugog polja.

Na osnovu ovako sortiranog niza parova, formiraćemo niz ukrupljenih interavala koji čine pokrivač skupa tačaka pokrivenih polaznim intervalima, na sledeći nain. U toku prolaza kroz sortirani niz, računaćemo broj intervala koji pokrivaju tekuću koordinatu. Taj broj se uvećava za jedan kada se naiđe na početak nekog intervala, a umanjuje se za jedan kada se naiđe na kraj nekog intervala. Tačka u kojoj taj broj sa nule poraste na strogo pozitivan broj, pamti se kao početak ukrupnjenog intervala. Prva naredna tačka u kojoj taj broj padne ponovo na nulu pamti se kao kraj ukrupljenog intervala.

Nakon toga se ukupna dužina koju pokrivaju svi intervali dobija kao suma dužina ukrupljenih intervala.

Sličan zadatak se može naći na:
`https://petlja.org/biblioteka/r/problemi/Zbirka/intervali`

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
typedef vector<pair<double, double>> Intervali;

void ukrupniIntervale(const Intervali& ulazni, Intervali& izlazni) {
  // formiramo niz promena
  vector<pair<double, int>> promene(ulazni.size()*2);
  for
    (int i = 0; i < ulazni.size(); i++) {
    promene[2*i] = make_pair(ulazni[i].first, 1);
    promene[2*i+1] = make_pair(ulazni[i].second, -1);
  }

  // sortiramo niz promena leksikografski,
  // prvo po x koordinati, a zatim po promeni (izlazak pre ulaska)
  sort(begin(promene), end(promene));
  // da li je trenutno zapocet neki izlazni interval koji
  // jos nije zavrsen
  bool zapocetIzlazni = false;
  // pocetak trenutnog izlaznogIntervala (ako je zapocet)
  double xPocetakIzlaznog = 0;
  // broj ulaznih intervala koji obuhvataju trenutnu koordinatu x
  int brojUlaznih = 0;
  // obraujemo jednu po jednu promenu
  int i = 0;
  while (i < promene.size()) {
    // obradjujemo sve intervale koji se zavrsavaju, pa onda one
    // koji pocinju u trenutnoj tacki, azurirajuci broj ulaznih
    // itervala koji sadrze trenutnu tacku
    double xTrenutno = promene[i].first;
    while (i < promene.size() && promene[i].first == xTrenutno)
      brojUlaznih += promene[i++].second;
    // tekuca tacka jeste unutar nekog ulaznog intervala, ali
    // izlazni interval nije zapocet, pa ga zato zapocinjemo
    if (!zapocetIzlazni && brojUlaznih > 0) {
      zapocetIzlazni = true;
      xPocetakIzlaznog = xTrenutno;
    }
    // tekuca tacka nije unutar nijednog ulaznog intervala pa
    // se zapoceti izlazni interval mora da se zavrsi na ovom mestu
    if (zapocetIzlazni && brojUlaznih == 0) {
      izlazni.push_back(make_pair(xPocetakIzlaznog, xTrenutno));
      zapocetIzlazni = false;
    }
  }
}
```

```
int main() {
  // ucitavamo ulazne intervale
  int n;
  cin >> n;
  Intervali ulazni(n);
  for (int i = 0; i < n; i++) {
    double x1, x2;
    cin >> x1 >> x2;
    ulazni[i] = make_pair(x1, x2);
  }
  // ukrupnjavamo ulazne intervale
  Intervali izlazni;
  ukrupniIntervale(ulazni, izlazni);
  // izracunavamo ukupnu duzinu prekrivaca
  int duzina = 0;
  for(auto it : izlazni)
    duzina += it.second - it.first;
  cout << duzina << endl;
  // ispisujemo broj intervala u prekrivacu
  cout << izlazni.size() << endl;
  return 0;
}
```

2.2 Odlični takmičari

Problem: Studenti su se na jednom turniru takmičili iz programiranja i matematike. Takmičar je odličan ako ne postoji drugi takmičar koji je od njega osvojio strogo više poena i iz programiranja i iz matematike. Napisati funkciju koja određuje ukupan broj odličnih takmičara.

Takmičar x je odličan ako ne postoji takmičar y koji ima strogo više poena od njega i iz programiranja i iz matematike, odnosno ako važi: $\neg\exists y : (y.m > x.m \wedge y.p > x.p)$, što je ekvivalentno sa $\forall y : (y.m \leq x.m \vee y.p \leq x.p)$.

Otuda vidimo da ako sortiramo niz takmičara prvo nerastuće po broju poena iz matematike, tekućeg takmiča ne ugrožavaju takmičari koji se nalaze u tom nizu iza njega. Eventualno mogu da ga ugrožavaju takmičari koji se nalaze u tom nizu pre njega, i to samo ako je njihov broj poena iz programiranja veći od tekućeg broja poena iz programiranja. Zbog toga ćemo takmičare sa istim brojem poena iz matematike, sortirati neopadajuće po broju poena iz programiranja.

Otuda vidimo da je, u ovako sortiranom nizu, tekući broj poena iz programiranja dovoljan za proveru da li je tekući takmičar odličan. Ako je tekući broj poena iz programiranja veći od svih prethodnih, onda je taj takmičar odličan. Zbog toga ćemo čuvati podatak o tekućem maksimalnom broju poena iz programiranja.

Malo lakši, sličan zadatak, se može naći na:

https://petlja.org/biblioteka/r/problemi/Zbirka/pobednik_u_dve_discipline

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

using namespace std;

struct Poeni {
  int mat;
  int prog;
};

int odlicniTakmicari(vector<Poeni>& poeni) {

  // sortiramo poene leksikografski,
  // nerastue po matematici i neopadajue po programiranju
  sort(begin(poeni), end(poeni),
       [](Poeni& p1,Poeni& p2) {
    return p1.mat > p2.mat ||
      p1.mat == p2.mat && p1.prog < p2.prog;
       });

  // ukupan broj do sada pronadenih odlicnih takmicara
  int brojOdlicnih = 0;

  // najveci do sada vidjen broj poena iz programiranja
  int maxProg = 0;
  for (int i = 0; i < poeni.size(); i++) {
    // niko od takmicara ispred tekuceg nije osvojio vise poena
    // od maxProg iz programiranja (iako su u matematici mozda
    // imali vise poena od tekueg takmicara)
    // ako je on osvoji bar toliko poena, onda ne postoji takmicar
    // koji od njega ima i strogo vise poena iz matematike
    // i strogo vise poena iz programiranja, pa je tekuci takmicar
    // odlican
    if (poeni[i].prog >= maxProg) {
      // uvecavamo broj odlicnih takmicara
      brojOdlicnih++;
      // azuriramo maksimalni broj osvojenih poena iz programiranja
      maxProg = poeni[i].prog;
    }
  }
  // vracamo ukupan broj odlicnih takmicara
  return brojOdlicnih;
}
```

```
int main(){
  int n;
  cout << "Unesite broj takmicara: ";
  cin >> n;

  vector<Poeni> poeni(n);
  cout << "Unesite za svakog takmicara prvo poene ";
  cout << "iz matematike pa iz programiranja" << endl;
  for(int i=0; i<n; i++)
    cin >> poeni[i].mat >> poeni[i].prog;

  cout << "Odlicnih takmicara ima: ";
  cout << odlicniTakmicari(poeni) << endl;
}
```

2.3 H-indeks

Problem: Rangiranje naučnika vrši se pomoću statistike koja se naziva Hiršov indeks ili kraće h-indeks. H-indeks je najveći broj h takav da naučnik ima bar h radova sa bar h citata. Napisati program koji na osnovu broja citata svih radova naučnika određuje njegov h-indeks.

Ako radove sortiramo u nerastući niz c po broju citata, *h-indeks* možemo izračunati kao najmanji broj h takav da je $c_h \leq h$. Zaista, ako je neki rad na poziciji h u sortiranom nizu i ako on ima c_h citata, znamo da postoji bar $h+1$ rad koji ima bar c_h citata (jer on ima tačno c_h citata i ispred njega postoji tačno h radova koji imaju bar onoliko citata koliko i on, jer je niz nerastući). Otuda vidimo da je dovoljno da u nerastuće sortiranom nizu proveravamo sve indekse h od 0 do $n-1$, sve dok ne naiđemo na prvi za koji važi da je $c_h \leq h$. Kada se to desi, znamo da postoji tačno h radova koji imaju bar h citata, jer je za prethodnu poziciju važilo da je $c_{h-1} > h-1$, tj. $c_{h-1} \geq h$, pa je postojalo tačno h radova sa bar c_h citata, a to je bar h citata. Takođe, ne postoji $h+1$ rad sa bar $h+1$ citatom, jer svi radovi od pozicije h do kraja niza imaju strogo manje od $h+1$ citata, a ispred njih postoji samo h radova.

Primer: ako naučnik ima 8 radova sa narednim brojem citata 3, 5, 12, 7, 4, 9, 0, 17, sortirani niz citata izgleda ovako:

```
17 12 9  7  5  4  3  0 --- sortirani niz
0  1  2  3  4  5  6  7 --- pozicije u nizu
```

https://petlja.org/biblioteka/r/problemi/zbirka-napredni-nivo/h_indeks

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

using namespace std;

int hIndeks(vector<int>& brojCitata) {
  // sortiramo radove na osnovu broju citata, nerastuce
  sort(brojCitata.begin(), brojCitata.end(), greater<int>());

  // Ako je h-indeks jednak h, tada h-ti po redu rad ima bar h
  // citata. Trazimo najveci broj koji zadovoljava taj uslov.
  int indeks = 0;
  while
    (indeks < brojCitata.size() && brojCitata[indeks] > indeks)
    indeks++;
  // vracamo rezultat
  return indeks;
}

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

  cout << "Unesite broj citata za svaki od radova: " << endl;
  for(int i=0; i<n; i++)
    cin >> brojCitata[i];

  cout << "H-indeks je jednak: " << hIndeks(brojCitata) << endl;
  return 1;
}
```

Drugi način da se reši ovaj problem je linearne složenosti. Za svaki broj h možemo da izračunamo tačan broj radova B_h koji imaju bar h citata.

Možemo primetiti da se broj radova koji imaju bar h citata može izračunati kao zbir broja radova koji imaju više od h citata tj. broja radova koji imaju bar $h+1$ citat i broja radova koji imaju tačno h citata, tj. $B_h = B_{h+1} + b_h$, gde je b_h broj radova koji imaju tačno h citata. Dakle, odgovara nam da pretragu organizujemo unatrag i da za svaki broj h od n pa unazad do nule određujemo broj radova koji imaju bar h citata, zaustavljajući se kada pronađemo prvu vrednost h tako da je $B_h \geq h$. Ostaje pitanje kako izračunati vrednosti b_h i kako izračunati početnu vrednost B_n. To možemo uraditi u jednom prolazu kroz niz citata c (zapravo, niz c ne moramo ni pamtiti, već prilikom učitavanja njegovih članova možemo samo izračunavati elemente b_h za $0 \leq h < n$, i ujedno izračunati vrednost B_n). Ove vrednosti možemo pamtiti u nizu brojača. Niz ima $n+1$ elemenat, pri čemu se na pozicijama od 0 do $n-1$ izračunavaju

vrednosti b_h, a na poziciji n se čuva vrednost B_n. Niz inicijalizujemo na nulu i svaki put kada učitamo broj citata nekog rada proveravamo da li je manji od n i ako jeste, uvećavamo brojač radova sa tim brojem citata (broj b_h), a u suprotnom uvećavamo broj radova sa bar n citata (broj B_n).

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

using namespace std;

int hIndeks(int n) {
  cout << "Unesite broj citata za svaki od radova: " << endl;

  // brojRadova[i] = broj radova koji imaju tacno i citata
  // brojRadova[n] = broj radova koji imaju >= n citata
  vector<int> brojRadova(n + 1, 0);
  for (int i = 0; i < n; i++) {
    // ucitavamo broj citata tekuceg rada
    int brojCitata;
    cin >> brojCitata;
    // azuriramo statistiku o broju radova
    brojRadova[brojCitata < n ? brojCitata : n]++;
  }

  cout << "Formirano" << endl;
  for(int i=0; i<n+1; i++)
    cout << brojRadova[i] << " ";
  cout << endl;

  // pokusavamo da uspostavimo sto veci h-indeks
  int indeks = n;
  // ukupnoRadova = broj radova koji imaju >= indeks citata
  int ukupnoRadova = brojRadova[n];
  while (ukupnoRadova < indeks) {
    // moramo da smanjimo h-indeks
    indeks--;
    // azuriramo broj radova sa >= indeks citata
    ukupnoRadova += brojRadova[indeks];
  }
  // vazi da je broj radova sa >= indeks citata >= indeks i
  // indeks je najveci broj za koji to vai, pa je on trazeni
  // rezultat
  return indeks;
}
```

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

  cout << "H-indeks je jednak: " << hIndeks(n) << endl;
  return 1;
}
```

2.4 Broj studenata iznad praga

Problem: Potrebno je odrediti prag poena za upis na fakultet. Komisija je stalno suočena sa pitanjima koliko bi se studenata upisalo, kada bi prag bio jednak datom broju. Napisati program koji učitava poene svih kandidata i efikasno odgovara na upite ovog tipa.

Ako se niz poena sortira nakon učitavanja, brojanje takmičara iznad praga se može ostvariti mnogo efikasnije. Kada se pronađe pozicija prvog takmičara čiji su poeni iznad praga, znamo da su svi takmičari od te pozicije pa do kraja niza primljeni (jer je niz sortiran, pa svi imaju više ili jednako poena od broja poena na toj poziciji) i da niko ispred te pozicije nije primljen (jer je to prva pozicija na kojoj je broj poena iznad praga). Prvu vrednost veću ili jednaku datoj u sortiranom nizu možemo jednostavno odrediti binarnom pretragom, pa je složenost ovog pristupa logaritamska.

U narednom programu se koristi funkcija `distance` koja računa broj elemenata koji se nalazi između dva iteratora.

`http://www.cplusplus.com/reference/iterator/distance/`

Naredni program se mora prevoditi sa opcijom `g++ -std=c++11`.

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

using namespace std;

int main(){
  // ucitavamo poene u niz
  int n;
  cout << "Unesite broj studenata: " << endl;
  cin >> n;
  vector<int> poeni(n);

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

  // sortiramo poene
  sort(begin(poeni), end(poeni));

  cout << "Unesite koliko pragova zelite da testirate: " << endl;
  // ucitavamo jedan po jedan prag
  int m;
  cin >> m;
  for(int i = 0; i < m; i++) {
    cout << "Unesite novi prag: " << endl;
    int prag;
    cin >> prag;
    // pronalazimo prvi element u nizu poena koji je veci ili
    // jednak pragu i raunamo njegovo rastojanje do kraja niza
    auto it = lower_bound(begin(poeni), end(poeni), prag);
    cout << "Broj studenata koji su iznad praga je: ";
    cout << distance(it, end(poeni)) << endl;
  }
  return 1;
}
```