Algoritmi i strukture podataka
6. Čas — Bibliotečke strukture podataka (II deo)
Implementacija struktura podataka

Sana Stojanović Đurđević

November 12, 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 Bibliotečke strukture podataka

1.1 Skupovi

U jeziku C++ skup je podržan kroz dve klase: `set<T>` (u zaglavlju `<set>`) i `unordered_set<T>` (u zaglavlju `<unordered_set>`), gde je T tip elemenata skupa. Implementacija je različita (prva je zadata na balansiranim binarnim drvetima, a druga na heš tablicama).

Set. Each value of an element must be unique. 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. Sets are typically implemented as binary search trees.

`http://www.cplusplus.com/reference/set/set/`

Svojstva: Elementima se pristupa prema njihovim vrednostima, a ne prema njihovoj poziciji. *Elementi u skupu su sve vreme uređeni prema funkciji poređenja (podrazumevano u rastućem poretku).* Ključ prema kojem se identifikuje element je jednak njegovoj vrednosti. Ključevi su jedinstveni, dva elementa ne mogu imati jednaku vrednost.

Properties:

Associative Elements in associative containers are referenced by their key and not by their absolute position in the container.

Ordered The elements in the container follow a strict order at all times. All inserted elements are given a position in this order.

Set The value of an element is also the key used to identify it.

Unique keys No two elements in the container can have equivalent keys.

Allocator-aware The container uses an allocator object to dynamically handle its storage needs.

Unordered set Containers that store unique elements in no particular order, and which allow for fast retrieval of individual elements based on their value.

In an `unordered_set`, the value of an element is at the same time its key, that identifies it uniquely. Keys are immutable, therefore, the elements in an `unordered_set` cannot be modified once in the container - they can be inserted and removed, though.

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).*

`unordered_set` containers are faster than `set` containers to access individual elements by their key, although they are generally less efficient for range iteration through a subset of their elements.

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

size. Returns the number of elements in the set container.

`http://www.cplusplus.com/reference/set/set/size/`

Najčešće korišćene osnovne operacije:

insert umeće novi element u skup (ako već postoji, operacija nema efekta). Extends the container by inserting new elements, effectively increasing the container size by the number of elements inserted. Because elements in a set are unique, the insertion operation checks whether each inserted element is equivalent to an element already in the container, and if so, the element is not inserted, returning an iterator to this existing element (if the function returns a value).

Return value When inserting the single element, function returns a pair, with its member `pair::first` set to an iterator pointing to either the newly inserted element or to the equivalent element already in the set. The `pair::second` element in the pair is set to `true` if a new element was inserted or `false` if an equivalent element already existed.

`http://www.cplusplus.com/reference/set/set/insert/`

find proverava da li skup sadrži dati element i vraća iterator koji pokazuje na njega ili `set::end` ako je odgovor negativan. Tako se provera pripadnosti elementa e skupu s može izvršiti sa `if (s.find(e) != s.end()) ...`

`http://www.cplusplus.com/reference/set/set/find/`

erase uklanja dati element iz skupa (ili prema njegovoj vrednosti ili prema iteratoru koji na njega pokazuje).

Return value Ako je element obrisan prema vrednosti onda funkcija vraća broj obrisanih elemenata (0 ako se element ne nalazi u skupu, odnosno 1 ako se element nalazi u skupu). Ako je element obrisan uz pomoć iteratora (ili opsega iteratora) funkcija vraća iterator koji pokazuje na element nakon poslednjeg uklonjenog elementa (ili `set::end`, ako je poslednji element bio obrisan).

`http://www.cplusplus.com/reference/set/set/erase/`

Funkcije `insert` i `erase` se mogu primeniti i na skup vrednosti. Kada se koristi `set` složenost ovih operacija je $O(\log k)$, gde je k broj elemenata u skupu, a kada se koristi `unordered_set`, složenost najgoreg slučaja je $O(k)$, dok je prosečna složenost $O(1)$, pri čemu je amortizovana složenost uzastopne primene operacija nad većim brojem elemenata takođe $O(1)$.[1] Naglasimo i da konstante kod složenosti $O(1)$ mogu biti relativno velike.

Problem: Napiši program koji određuje da li među učitanih n brojeva ima duplikata.

[1] Analiza ukupne dužine trajanja većeg broja operacija naziva se amortizovana analiza složenosti. Amortizovana cena izvršavanja n operacija podrazumeva količnik njihove ukupne dužine izvršavanja i broja n.

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

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

  bool duplikati = false;
  set<int> vidjeni;

  cout << "Unesite elemente: " << endl;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;

    if (vidjeni.find(x) != vidjeni.end()) {
      duplikati = true;
      break;
    }

    vidjeni.insert(x);
  }

  cout << (duplikati ? "da" : "ne") << endl;
  return 0;
}
```

Jezik C++ daje i funkcije koje izračunavaju presek, uniju i razliku dva skupa (nalaze se u zaglavlju `<algorithm>`):

presek `http://www.cplusplus.com/reference/algorithm/set_intersection/?kw=set_intersection`

unija `http://www.cplusplus.com/reference/algorithm/set_union/`

razlika `http://www.cplusplus.com/reference/algorithm/set_difference/`

U narednom kodu koristi se funkcija `inserter`:

inserter. Constructs an insert iterator that inserts new elements in successive locations starting at the position pointed by it.

An insert interator is a special type of output iterator designed to allow algorithms that usually overwrite elements (such as copy) to instead insert new elements automatically at a specific position in the container.

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

Problem: Učitavaju se šifre proizvoda koji se nalaze u dva magacina. Napisati program koji određuje one koji se nalaze u oba.

```
#include <iostream>
#include <set>
#include <algorithm>

using namespace std;

int main() {
  int n1;
  cout << "Unesite broj proizvoda koji se nalaze u I magacinu: ";
  cin >> n1;
  set<int> proizvodi1;
  cout << "Unesite proizvode: " << endl;
  for (int i = 0; i < n1; i++) {
    int x;
    cin >> x;
    proizvodi1.insert(x);
  }

  int n2;
  cout << "Unesite broj proizvoda koji se nalaze u II magacinu: ";
  cin >> n2;
  set<int> proizvodi2;
  cout << "Unesite proizvode: " << endl;
  for (int i = 0; i < n2; i++) {
    int x;
    cin >> x;
    proizvodi2.insert(x);
  }

  cout << "Proizvodi koji se nalaze u oba magacina: ";
  // za sve proizvode iz prvog magacina
  for (int x : proizvodi1)
    // proveravamo da li se nalaze u drugom i ispisujemo ih
    if (proizvodi2.find(x) != proizvodi2.end())
      cout << x << " ";
  cout << endl;

  cout << "Drugi nacin: ";
  set<int> presek;
  set_intersection(begin(proizvodi1), end(proizvodi1),
                   begin(proizvodi2), end(proizvodi2),
                   inserter(presek, begin(presek)));
  for (int x : presek)
    cout << x << " ";
  cout << endl;

  return 0;
}
```

U narednom primeru se može videti način korišćenja funkcija `lower_bound` i `upper_bound`:

lower_bound Returns an iterator pointing to the first element in the container which is not considered to go before val (i.e., either it is equivalent or goes after).

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

upper_bound Returns an iterator pointing to the first element in the container which is considered to go after val.

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

Ove dve funkcije imaju istu povratnu vrednost u slučaju da se element ne nalazi u skupu.
Ako se element nalazi u skupu, onda `lower_bound` vraća iterator koji pokazuje baš na taj element, dok `upper_bound` vraća iterator na element koji se nalazi iza njega.

```
// set::lower_bound/upper_bound
#include <iostream>
#include <set>

using namespace std;

int main ()
{
  set<int> myset;
  set<int>::iterator itlow,itup;

  for (int i=1; i<10; i++) myset.insert(i*10);

  itlow=myset.lower_bound (30);
  itup=myset.upper_bound (60);

  // 10 20 30 40 50 60 70 80 90
  //       ^
  //                   ^

  myset.erase(itlow,itup);                      // 10 20 70 80 90

  cout << "myset contains:";
  for (set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';

  return 0;
}
```

1.2 Mape

Mape su strukture podataka koje skup ključeva iz nekog konačnog domena preslikavaju u neki skup vrednosti.

U jeziku C++ mape su podržane klasama `map<K, V>` (u zaglavlju `<map>`) i `unordered_map<K, V>` (u zaglavlju `<unordered_map>`), gde je K tip ključeva, a V tip vrednosti. Slično kao u slučaju skupa, implementacija prve klase je zasnovana na balansiranim binarnim drvetima, a druga na heš tablicama.

Maps. Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a specific order.

In a map, the *key values* are generally used to sort and uniquely identify the elements, while the *mapped values* store the content associated to this key. The types of *key* and *mapped* value may differ, and are grouped together in member type `value_type`, which is a pair type combining both:

`typedef pair<const Key, T> value_type;`

`http://www.cplusplus.com/reference/map/map/`

Internally, the elements in a map are always sorted by its key*.* `map` containers are generally slower than `unordered_map` containers to access individual elements by their *key*, but they allow the direct iteration on subsets based on their order. The mapped values in a map can be accessed directly by their corresponding key using the bracket operator (`operator[]`).

Container properties:

Associative Elements in associative containers are referenced by their key and not by their absolute position in the container.

Ordered The elements in the container follow a strict order at all times. All inserted elements are given a position in this order.

Map Each element associates a key to a mapped value: Keys are meant to identify the elements whose main content is the mapped value.

Unique keys No two elements in the container can have equivalent keys.

Unordered maps. Unordered maps are associative containers that store elements formed by the combination of a key value and a mapped value, and which allows for fast retrieval of individual elements based on their keys.

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

`http://www.cplusplus.com/reference/unordered_map/unordered_map/`

Na raspolaganju su nam sledeće operacije:

operator indeksnog pristupa Na primer, ako mapa `ocene` preslikava imena učenika u njihove prosečne ocene, tada se ocena učenika `pera` može i pročitati i postaviti pomoću `ocene["pera"]`.

If the key that we are searching for matches the key of an element in the container, the function returns a reference to its mapped value.

If the key that we are searching for does not match the key of any element in the container, the function inserts a new element with that key and returns a reference to its mapped value. Notice that this always increases the container size by one, even if no mapped value is assigned to the element (the element is constructed using its default constructor).

`http://www.cplusplus.com/reference/map/map/operator[]/`

find vraća iterator koji ukazuje na slog sa datim ključem u mapi. Ako taj ključ ne postoji u mapi, vraća se iterator na kraj mape. Tako se provera pripadnosti ključa k mapi m može izvršiti sa `if (m.find(k) != m.end()) .....`

`http://www.cplusplus.com/reference/map/map/find/`

erase briše element iz mape. Argument može biti bilo vrednost ključa, bilo iterator koji pokazuje na element koji se uklanja. Moguće je i brisanje skupa elemenata odjednom. Ako je element obrisan prema vrednosti onda funkcija vraća broj obrisanih elemenata. Ako je element obrisan uz pomoć iteratora (ili opsega iteratora) funkcija vraća iterator koji pokazuje na element nakon poslednjeg uklonjenog elementa.

`http://www.cplusplus.com/reference/map/map/erase/`

Kada se koristi `map`, garantovana složenost ovih operacija je $O(\log k)$ gde je k broj trenutno pridruženih ključeva u mapi. Kada se koristi `unordered_map` prosečna složenost je $O(1)$, ali konstantni faktor može biti veliki, dok je složenost najgoreg slučaja $O(k)$.

Jezik dopušta i iteriranje kroz elemente mape (isto kao i za skupove). Na primer, kroz sve ocene se može proći sa sledećom petljom:

```
for (auto it : ocene)
  cout << it.first << " " << it.second << endl;
```

Primetimo da promenljiva `it` tokom iteracije uzima uređene parove (ključ, vrednost), pa se ključu pristupa preko polja `first`, a vrednosti preko polja `second`.

Problem: Organizovati strukturu podataka koja studentima pridružuje broj poena.

Ovaj način inicijalizacije mape je moguć tek od standarda 2011 — initializer list constructor (constructs a container with a copy of each of the elements in initializer list).

```
#include <iostream>
#include <map>
using namespace std;

int main() {
  map<string, int> poeni =
    {{"pera", 84}, {"ana", 92}, {"joca", 67}};

  cout << poeni["ana"] << endl;
  poeni["joca"] = 72;
  cout << poeni["joca"] << endl;
  poeni["ivana"] = 48;
  cout << poeni["ivana"] << endl;
  return 0;
}
```

Problem: Napisati program koji izračunava frekvenciju (broj pojavljivanja) svake od reči u tekstu.

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main() {
  map<string, int> frekvencije;
  string rec;
  cout << "Unesite reci: " << endl;

  while (cin >> rec)
    frekvencije[rec]++;

  for (auto it : frekvencije)
    cout << it.first << ": " << it.second << endl;
  return 0;
}
```

Problem: Napisati program koji izračunava frekvenciju (broj pojavljivanja) svakog slova u tekstu. Reči u tekstu se sastoje samo od malih slova. Ispisati samo ona slova koja se pojavljuju u tekstu.

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main() {
 int frekvencije_slova[26] = {0};
 string rec;
 cout << "Unesite reci koje se sastoje samo od malih slova:" << endl;

 while (cin >> rec)
   for(char c : rec)
     frekvencije_slova[c - 'a']++;


 for (int i=0; i < 26; i++)
   if (frekvencije_slova[i] > 0)
     cout << (char)('a' + i) << " " << frekvencije_slova[i] << endl;
 return 0;
}
```

Problem: Dat je niz celih brojeva. Odrediti ukupan broj segmenata čiji su svi elementi različiti.

Jedno prilično elegantno rešenje se zasniva na tome da *za svaku poziciju* analiziramo sve segmente kojima je *kraj* upravo na toj poziciji, a kojima su svi elementi različiti.

Ako su svi elementi nekog segmenta različiti, onda su i svim njegovim sufiksima svi elementi različiti. Ako neki segment sadrži duplikate, onda duplikate sadrže i svi segmenti kojima je on sufiks.

Zato je dovoljno da pronađemo najduži mogući segment koji se završava na poziciji *kraj* i kojem su svi elementi različiti i znaćemo da su svi segmenti sa različitim elementima koji se završavaju na poziciji *kraj* tačno svi njegovi sufiksi. Ako je to segment određen pozicijama iz intervala *[pocetak, kraj]* onda će takvi biti i segmenti određeni intervalima pozicija *[pocetak + 1, kraj], . . . , [kraj-1, kraj]*. Njih je ukupno *kraj-pocetak*.

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
int main() {
  int n;
  cout << "Unesite broj elemenata niza i elemente niza: ";
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  // ukupan broj segmenata niza ciji su svi elementi razliciti
  int broj = 0;

  // za svaki element u tekucem segmentu [pocetak, kraj] pamtimo
  // poziciju na kojoj se pojavljuje
  unordered_map<int, int> prethodno_pojavljivanje;

  int pocetak = 0;
  for (int kraj = 0; kraj < n; kraj++) {
    if (prethodno_pojavljivanje.find(a[kraj])
        != prethodno_pojavljivanje.end()) {

      // ako se element a[kraj] vec pojavljuje u nizu, najduzi
      // segment (sa svim razlicitim elementima) koji sadrzi element
      // na poziciji kraj pocinje nakon prethodnog pojavljivanja
      // elementa a[kraj]
      int novi_pocetak = prethodno_pojavljivanje[a[kraj]] + 1;

      // brisemo iz segmenta sve elemente od starog do ispred
      // novog pocetka i mapu uskladjujemo sa time
      for (int i = pocetak; i < novi_pocetak; i++)
        prethodno_pojavljivanje.erase(a[i]);

      // pomeramo pocetak
      pocetak = novi_pocetak;
    }

    // prosirujemo segment elementom a[kraj], pa pamtimo poziciju
    // njegovog pojavljivanja
    prethodno_pojavljivanje[a[kraj]] = kraj;

    // [pocetak, kraj] sadrzi sve razlicite elemente i on je najduzi
    // takav koji se zavrsava na poziciji kraj
    // sigurno su takvi i [pocetak+1, kraj], ..., [kraj-1, kraj]
    // njih ima (kraj - pocetak)
    broj += kraj - pocetak;
  }
  cout << broj << endl;
  return 0;
}
```

Problem: Broj segmenata datog zbira Napisati program koji za dati niz celih brojeva određuje koliko ima nepraznih segmenata uzastopnih elemenata niza čiji je zbir jednak datom broju z.

Elegantan i često primenjivan način da se dobiju zbirovi svih segmenata uzastopnih elemenata je da se izračunaju zbirovi prefiksa i da se zbir elemenata segmenta $[i, j]$ izrazi kao razlika zbira elemenata segmenta $[0, j]$ i zbira elemenata segmenta $[0, i-1]$. Dakle, u pomoćni niz b na svaku poziciju k možemo smestiti zbir prvih k elemenata niza (ovo opet možemo uraditi inkrementalno).

Niz b će imati jedan element više od niza a, i to je prvi element niza b čija je vrednost jednaka 0. Vrednosti prvih nekoliko elemenata niza b će biti: $\{0, a_0, a_0 + a_1, a_0 + a_1 + a_2, \ldots\}$. Odnosno, $b_j = a_0 + a_1 + \ldots a_{j-1}$. Tako se zbir elemenata niza a iz segmenta $[i, j]$ uvek određuje kao $b_{j+1} - b_i$.

Umesto čuvanja niza b, problem ćemo formulisati na drugi način: Za svaki element b_{j+1} koji odgovara zbiru prefiksa $[0, j]$ potrebno je da pronađemo da li postoji zbir b_i prefiksa $[0, i)$ za $i < j$ takav da je $b_{j+1} - b_i = z$, gde je z traženi zbir, tj. da se proveri da li se među zbirovima prethodnih prefiksa nalazi vrednost $b_i = b_{j+1} - z$.

U nekoj strukturi podataka koja omogućava efikasno pretraživanje čuvamo sve zbirove prefiksa za indekse $i < j$. Ako algoritam organizujemo tako da j uvećavamo od 0 do $n - 1$, tada se na kraju svakog koraka u tu strukturu može dodati i zbir tekućeg segmenta (b_{j+1}).

Struktura treba da realizuje pretragu po ključu, tako da je najbolje upotrebiti asocijativno preslikavanje, odnosno mapu tj. rečnik. Pošto se u zadatku traži samo određivanje broja segmenata sa datim zbirom, ključevi mogu biti zbirovi prefiksa, a vrednost pridružena svakom ključu može biti broj pojavljivanja prefiksa sa tim zbirom.

Ako računamo da će pretraga biti realizovana u $O(\log n)$ (što je najčešće slučaj ako se koriste strukture podataka zasnovane na binarnim stablima, kao što je u slučaju `map`), tada će ukupna složenost ove implementacije biti $O(n \log n)$.

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

using namespace std;

int main() {
  // ucitavamo dati niz brojeva
  int n, trazeniZbir;

  cout << "Unesite trazeni zbir: ";
  cin >> trazeniZbir;

  // zbir prefiksa
  int zbirPrefiksa = 0;

  // broj segmenata sa trazenim zbirom
  int broj = 0;

  // broj pojavljivanja svakog vidjenog zbira prefiksa
  map<int, int> zbiroviPrefiksa;

  // zbir pocetnog praznog prefiksa je 0 i on se za sada pojavio
  // jednom
  zbiroviPrefiksa[0] = 1;

  // ucitavamo elemente niza niz
  cout << "Unesite broj elemenata niza: ";
  cin >> n;

  cout << "Unesite elemente niza: ";
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;

    // uracunavamo element u zbir tekuceg prefiksa
    zbirPrefiksa += x;

    // trazimo broj pojavljivanja vrednosti zbirPrefiksa - trazeniZbir
    // i azuriramo broj pronadjenih segmenata
    auto it = zbiroviPrefiksa.find(zbirPrefiksa - trazeniZbir);

    // broj pojavljivanja tog zbira dodajemo na ukupan broj
    if (it != zbiroviPrefiksa.end())
      broj += it->second;

    // povecavamo broj pojavljivanja trenutnog zbira
    zbiroviPrefiksa[zbirPrefiksa]++;
  }


  cout << "Segmenata koji imaju trazeni zbir ima: ";
  cout << broj << endl;
  return 0;
}
```

Problem: Da li postoji segment datog zbira — uraditi za vežbu Napisati program koji za dati niz celih brojeva određuje da li postoji makar jedan neprazni segment uzastopnih elemenata čiji je zbir jednak datom broju z.

Zadatak se rešava slično kao prethodni, s tim što umesto preslikavanja (<zbir prefiksa,broj pojavljivanja>) možemo čuvati samo skup ranije viđenih vrednosti zbirova prefiksa.

Problem: Određivanje segmenata datog zbira — uraditi za vežbu Napisati program koji za dati niz celih brojeva određuje i ispisuje sve segmente uzastopnih elemenata čiji je zbir jednak datom broju z.

Zadatak se rešava slično kao problem "Broj segmenata datog zbira", s tim što se svaki ključ preslikava u niz vrednosti (vektor) i takvih da je b_i jednako tom ključu. Napomenimo da je u ovoj situaciji neophodno čuvati niz a u posebnoj memorijskoj lokaciji.

Izračunavanje parcijalnih suma U jeziku C++ parcijalne sume se mogu odrediti i bibliotečkom funkcijom `partial_sum` koja prima dva iteratora koji ograničavaju deo niza (ili vektora) čije se parcijalne sume izračunavaju i iterator koji ukazuje na početak dela niza (ili vektora) u koji se parcijalne sume upisuju.

Ako x predstavlja elemente prvog vektora, a y predstavlja elemente rezultujućeg vektora, onda važe naredne jednačine:

```
y0 = x0
y1 = x0 + x1
y2 = x0 + x1 + x2
y3 = x0 + x1 + x2 + x3
y4 = x0 + x1 + x2 + x3 + x4
... ... ...
```

`http://www.cplusplus.com/reference/numeric/partial_sum/`

```
// partial_sum example
#include <iostream>     // std::cout
#include <functional>   // std::multiplies
#include <numeric>      // std::partial_sum

int main () {
  int val[] = {1,2,3,4,5};
  int result[5];

  std::partial_sum (val, val+5, result);

  std::cout << "using default partial_sum: ";

  for (int i=0; i<5; i++)
    std::cout << result[i] << ' ';
  std::cout << endl;
}
```

1.3 Multiskupovi

Za razliku od skupova gde se svaki element pojavljuje najviše jednom. Multiskup je struktura podataka u kojoj se elementi mogu pojavljivati i više puta. Multiskup, dakle, odgovara mapi koja slika ključeve u njihov broj pojavljivanja (i tako se interno može implementirati).

U jeziku C++ multiskupovi se reprezentuju objektima klase `multiset<T>` i `unordered_multiset<T>` gde je `T` tip elemenata multiskupa. Implementacija je različita (prva je zadata na balansiranim binarnim drvetima, a druga na heš tablicama).

Ove klase nemaju svoja posebna zaglavlja već dele zaglavlja `set` i `<unordered_set>` sa skupovima, tj. neuređenim skupovima.

Multiset. Multisets are containers that store elements following a specific order, and where multiple elements can have equivalent values (internally, the elements in a multiset are always sorted following a specific strict weak ordering criterion).

`http://www.cplusplus.com/reference/set/multiset/multiset/`

`multiset` containers are generally slower than `unordered_multiset` containers to access individual elements by their key, but they allow the direct iteration on subsets based on their order.

Container properties:

Associative Elements in associative containers are referenced by their key and not by their absolute position in the container.

Ordered The elements in the container follow a strict order at all times. All inserted elements are given a position in this order.

Set The value of an element is also the key used to identify it.

Multiple equivalent keys Multiple elements in the container can have equivalent keys.

Allocator-aware The container uses an allocator object to dynamically handle its storage needs.

Unordered_multiset. Unordered multisets are containers that store elements in no particular order, allowing fast retrieval of individual elements based on their value, much like `unordered_set` containers, but allowing different elements to have equivalent values. Unordered containers organize their elements using hash tables that allow for fast access to elements by their key.

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

Internally, the elements in the `unordered_multiset` are not sorted in any particular, 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).

Elements with equivalent values are grouped together in the same bucket and in such a way that an iterator (see `equal_range`) can iterate through all of them.

Na raspolaganju su nam sledeće operacije:

insert umeće dati element u multiskup.

Extends the container by inserting new elements, effectively increasing the container size by the number of elements inserted.

Internally, multiset containers keep all their elements sorted following the criterion specified by its comparison object. The elements are always inserted in its respective position following this ordering.

If used for inserting just one element, the returning value is an iterator pointing to the newly inserted element in the multiset.

`http://www.cplusplus.com/reference/set/multiset/insert/`

erase briše element iz multiskupa. Ako je argument iterator, briše se element na koji taj iterator pokazuje, a ako je argument vrednost, brišu se sva pojavljivanja te vrednosti.

Ako je element obrisan prema vrednosti onda funkcija vraća broj obrisanih elemenata. Ako je element obrisan uz pomoć iteratora (ili opsega iteratora) funkcija vraća iterator koji pokazuje na element nakon poslednjeg uklonjenog elementa.

`http://www.cplusplus.com/reference/set/multiset/erase/`

find vraća iterator koji ukazuje na element koji se traži, ili iterator koji pokazuje na `multiset::end` ako se element ne nalazi u multiskupu. Notice that this

function returns an iterator to a single element (of the possibly multiple equivalent elements).

`http://www.cplusplus.com/reference/set/multiset/find/`

count izračunava broj pojavljivanja datog elementa u multiskupu.

`http://www.cplusplus.com/reference/set/multiset/count/`

equal_range vraća par iteratora koji ograničavaju deo multiskupa koji sadrži sve vrednosti pridružene datom ključu. Ako se traženi ključ ne nalazi u multimapi, par iteratora koji je vraćen kao povratna vrednost su jednaka i pokazuju na prvi element koji bi se nalazio iza traženog ključa (prema podrazumevanoj operaciji poređenja).

`http://www.cplusplus.com/reference/set/multiset/equal_range/`

Složenost:

U slučaju ubacivanja jedne vrednosti (`insert`), i u slučaju pretrage (`find`), složenost funkcije je $O(\log n)$ gde je n broj elemenata. U slučaju da se koristi `unordered_multiset` složenost ovih dveju funkcija je u proseku $O(1)$, pri čemu je složenost najgoreg slučaja $O(n)$.

U slučaju brisanja jedne vrednosti (`erase`) i u slučaju prebrojavanja (`count`), složenost je $O(\log n) + O(k)$ gde je n broj elemenata a k broj pojavljivanja tražene vrednosti. U slučaju da se koristi `unordered_multiset` složenost ovih dveju funkcija je u proseku $O(k)$, pri čemu je složenost najgoreg slučaja $O(n)$.

Problem: Sortirati sve reči koje se čitaju sa ulaza. Ako se neka reč pojavila više puta, prikazati je više puta.

```
#include <iostream>
#include <set>
#include <string>

using namespace std;

int main() {
  multiset<string> reci;
  string rec;
  cout << "Unesite reci: " << endl;

  while (cin >> rec)
    reci.insert(rec);

  cout << "Uneli ste reci (u sortiranom poretku): " << endl;
  for (auto rec : reci)
    cout << rec << endl;
}
```

Problem: Sa ulaza se unosi broj n i zatim n reči. Nakon toga se unosi broj k i k reči. Napisati program koji za svaku od tih k učitanih reči određuje koliko se puta pojavila među prvih n reči.

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

int main() {
  // ucitavamo reci i smestamo ih u multiskup

  multiset<string> reci;
  cout << "Unesite broj n i nakon toga n reci: ";
  int n;
  cin >> n;

  for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    reci.insert(s);
  }

  // za k reci ocitavamo i prijavljujemo broj pojavljivanja
  cout << "Unesite broj k i nakon toga k reci: ";
  int k;
  cin >> k;
  for (int i = 0; i < k; i++) {
    string s;
    cin >> s;
    cout << s << ": " << reci.count(s) << endl;
  }
  return 0;
}
```

1.4 Multimape

Multimape dopuštaju da se isti ključ preslikava u više od jedne vrednosti. Jedan način da se realizuju je preslikavanje ključeva u kolekciju (niz, vektor) vrednosti.

U jeziku C++ se multimape direktno predstavljaju objektima klase `multimap<K,V>` i `unordered_multimap<K,V>`, gde je `K` tip ključeva, a `V` tip vrednosti elemenata.

Ove klase nemaju svoja posebna zaglavlja već dele zaglavlja `map` i `<unordered_map>` sa mapama i neuređenim mapama.

Implementacija je različita (prva je zadata na balansiranim binarnim drvetima, a druga na heš tablicama).

Multimap. In a multimap, the key values are generally used to sort and uniquely identify the elements, while the mapped values store the content associated to this key. Internally, the elements in a multimap are always sorted by its key following a specific strict weak ordering criterion. `multimap` containers are generally slower than `unordered_multimap` containers to access individual elements by their key, but they allow the direct iteration on subsets based on their order.

Multimaps are typically implemented as binary search trees.

`http://www.cplusplus.com/reference/map/multimap/multimap/`

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

Elements with equivalent keys are grouped together in the same bucket and in such a way that an iterator (see `equal_range`) can iterate through all of them.

`http://www.cplusplus.com/reference/unordered_map/unordered_multimap/`
`unordered_multimap/`

Na raspolaganju imamo sledeće operacije:

insert pridružuje novu vrednost ključu. Ključ i vrednost se zadaju kao argument i to kao par (vrednost tipa `pair<K, V>`).

The relative ordering of elements with equivalent keys is preserved, and newly inserted elements follow those with equivalent keys already in the container.

In the versions returning a value, this is an iterator pointing to the newly inserted element in the multiset.

`http://www.cplusplus.com/reference/map/multimap/insert/`

find vraća iterator koji ukazuje na jednu vrednost pridruženu datom ključu ili `end`, ako ključu nije pridružena ni jedna vrednost.

`http://www.cplusplus.com/reference/map/multimap/find/`

equal_range vraća par iteratora koji ograničavaju deo multimape koji sadrži sve vrednosti pridružene datom ključu. Ako se traženi ključ ne nalazi u multimapi, par iteratora koji je vraćen kao povratna vrednost su jednaka i pokazuju na prvi element koji bi se nalazio iza traženog ključa (prema podrazumevanoj operaciji poređenja).

`http://www.cplusplus.com/reference/map/multimap/equal_range/`

Složenost:

U slučaju korišćenja `multimap` složenost ovih operacija je $O(\log k)$, gde je k broj trenutnih elemenata u multimapi. U slučaju korišćenja `unordered_multimap` prosečna složenost je konstantna, a složenost najgoreg slučaja je $O(k)$.

Problem: Napisati program koji učitava imena gradova i zemalja u kojima se nalaze. Napiši program koji ispisuje te podatke sortirane po zemljama, a zatim posebno ispisuje sve gradove iz Srbije.

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main() {
  multimap<string, string> gradovi;

  cout << "Unesite broj n i nakon toga n gradova i zemalja:";
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    string grad, zemlja;
    cin >> grad >> zemlja;
    gradovi.insert(make_pair(zemlja, grad));
  }

  cout << "Gradovi sortirani po zemljama: " << endl;
  for (auto it : gradovi)
    cout << it.first << " " << it.second << endl;

  cout << "Gradovi koji se nalaze u Srbiji: " << endl;
  auto gradovi_srbije = gradovi.equal_range("Srbija");
  for (auto it = gradovi_srbije.first; it != gradovi_srbije.second;
       it++)
    cout << it->second << endl;
}
```