

Asocijativni spremnici

Objektno programiranje - 3. vježbe (2. dio)

Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

22. ožujka 2023. godine



Asocijativni spremnici

- razlika u spremanju i dohvat u podataka:
 - sekvencijalni spremnici - prema **poziciji**
 - **asocijativni spremnici** - prema **ključu**
 - osnovni asocijativni spremnici:
 - **map** (preslikavanje) - sadrži parove **ključ-vrijednost**
 - indeks i vrijednost pridružena tom indeksu (tzv. asocijativno polje)
 - npr. imenik (ime osobe je indeks za dohvaćanje telefonskog broja)
 - **set** (skup) - samo ključevi
 - podržani efiksni upiti je li nešto unutra ili nije
 - ako ključevi ne moraju biti jedinstveni \Rightarrow **multi...** spremnici
 - ako ne čuvaju ključeve u sortiranom poretku \Rightarrow **unordered_...** spremnici (za organizaciju elemenata koriste **hash funkciju**)
- \Rightarrow ukupno imamo $2^3 = 8$ tipova asocijativnih spremnika

Tipovi asocijativnih spremnika

Naziv	Zaglavlje
<code>map</code> <code>multimap</code>	<code>map</code> <code>map</code>
<code>set</code> <code>multiset</code>	<code>set</code> <code>set</code>
<code>unordered_map</code> <code>unordered_multimap</code>	<code>unordered_map</code> <code>unordered_map</code>
<code>unordered_set</code> <code>unordered_multiset</code>	<code>unordered_set</code> <code>unordered_set</code>

- kao i ostali spremnici, to su predložci
- ⇒ navodimo tip ključa (a za preslikavanja i tip vrijednosti)

Primjer. (po defaultu su sljedeći spremnici prazni)

- `map<string, size_t> a;`
- `set<string> b;`

Primjer (upotreba preslikavanja)

- program čita riječi (do EOF) i ispisuje koliko se koja riječ pojavila

```
map<string, size_t> br_rijeci;
string rijec;
while (cin >> rijec)
    ++br_rijeci[rijec];
for (const auto &w : br_rijeci)
    cout << "Rijec " << w.first
        << " javlja se " << w.second
        << ((w.second > 1) ? " puta." : " put.")
        << endl;
```

- uočite: kako se dodaje u preslikavanje; element w preslikavanja je par; riječi se ispisuju u leksikografskom poretku



Primjer (upotreba skupa)

- kao u prethodnom primjeru ali ne brojimo neke veznike

```
map<string, size_t> br_rijeci;
set<string> veznik = {"i", "ili", "a"};
string rijec;
while (cin >> rijec)
    if(veznik.find(rijec) == veznik.end())
        ++br_rijeci[rijec];
```

- **find** vraća iterator na ključ koji se nalazi u skupu ili `end` iterator (ako traženog ključa nema u skupu)

- možemo navesti elemente, kopirati spremnik **istog tipa** ili navesti raspon (dokle god se vrijednosti mogu pretvoriti u tip vrijednosti u spremniku)

Primjer.

```
set<string> s1 = {"ab", "cd"};
set<const char *> s2 = {"e", "fg", "h"};
map<string, string> m1 = { {"abc", "d"},
                           {"ef", "ghi"},
                           {"j", "kl"} };

set<string> s3(s1);      ✓
set<string> s4(s2);      ✗
set<string> s4(s2.begin(), s2.end());  ✓
```

Primjer. (skup vs. multiskup) Odredite elemente spremnika:

```
vector<int> v = {3, 9, 1, 4, 2, 3, 1};  
set<int> s(v.cbegin(), v.cend());  
multiset<int> ms(v.cbegin(), v.cend());
```

- Što bi ispisao sljedeći kod za: (a) `[?]` = s; (b) `[?]` = ms?

```
cout << [?].size() << endl;  
for(auto br : [?])  
    cout << br << endl;
```

- za **uređene** spremnike, **ključevi se moraju moći uspoređivati** (po defaultu se koristi <)

Korištenje vlastite funkcije za uspoređivanje ključeva

- operacija mora definirati [strict weak ordering](#) na ključevima¹

```
bool usp(const string &a, const string &b){  
    return a.size() < b.size();  
}
```

- definiramo multiskup s dva tipa: (i) tip ključa (string)
(ii) tip usporedbe (tip pokazivača na funkciju kao što je `usp`)
- pri definiranju objekta tog tipa, dajemo pokazivač na `usp`
`set<string, decltype(usp)*> rijeci(usp);`
- pri upotrebi `decltype` za dobivanje pokazivača na funkciju dodali
* (za pokazivač na dani tip funkcije)
- u konstruktoru može `usp` umjesto `&usp` (automatski se pretvara
u pokazivač)

¹Strogi parcijalni uređaj na ključevima takav da je relacija „neusporedivosti” (s obzirom na taj uređaj) tranzitivna.

Primjer unosa i ispisa (za set i multiset)

Primjer.

```
set<string, decltype(usp)*> rijeci(usp);
string r;
while(cin >> r)
    rijeci.insert(r);
for(const auto &w : rijeci)
    cout << w << endl;
```

- promatramo sljedeći unos:

```
aasd ds sa dsa dsa d sd s ds d ads
```

- ispis za gornji kod je prikazan lijevo, a ako `set` zamijenimo s `multiset` dobivamo ispis prikazan desno

```
d
ds
dsa
aasd
```

```
d
s
d
ds
sa
sd
ds
dsa
dsa
ads
aasd
```

Napišite program koji učitava parove cijelih brojeva. Nakon učitavanja, potrebno ih je ispisati u uzlaznom poretku s obzirom na vrijednost najvećeg zajedničkog djelitelja.

Primjer. (ulaz i ispis)

40 25	0,0
10 50	7,3
-4 14	34,71
0 0	-4,14
7 3	40,25
-18 12	-18,12
34 71	10,50

Napomena. Za najveći zajednički djelitelj (*greatest common divisor*) dva broja možemo koristiti:

- standard C++14 - funkcija `__gcd` iz zaglavlja `algorithm`
- standard C++17 - funkcija `gcd` iz zaglavlja `numeric`

Ako su oba broja 0, funkcije vraćaju 0, a inače gcd zadanih brojeva.

Ako koristimo funkciju `gcd` kompajliramo korištenjem `-std=c++17`.



Iteratori za asocijativne spremnike

- dereferenciranjem iteratora za preslikavanje dobivamo par (ključ,vrijednost) (za skupove samo ključ)
- vrijednosti možemo mijenjati, ali ključeve ne

Primjer.

```
map<string, string> m = {"a", "b"}, {"c", "d"};
auto it = m.begin();
cout << it->first << " " << it->second << endl;
it->first = "e";      X
it->second += "f";    ✓
```

```
set<int> s = {0, 1, 2, 3};
set<int>::iterator its = s.begin();
if (its != s.end()) {
    *its = 42;      X
    cout << *its << endl;    ✓
}
```

Korištenje iteratora za prolazak spremnikom

Primjer. Što ispisuje donji kod?

```
set<int> s = {0,1,2,3};  
for(auto it = s.begin(); it != s.end(); ++it)  
    cout << *it << endl;  
map<int,double> m = {{4,6.2},{1,2},{7,0.1}};  
for(auto it = m.begin(); it != m.end(); ++it)  
    cout << "(" << it->first << ","  
        << it->second << ")" << endl;
```

Primjer (Dodavanje elemenata)

```
vector<int> v = {2, 4, 6, 8, 2, 4, 6, 8};  
set<int> s;  
map<string, size_t> m;  
string str("abc");
```

```
s.insert(v.cbegin(), v.cend());  
s.insert({1, 3, 5, 7, 1, 3, 5, 7});
```

```
m.insert({str, 1});  
m.insert(make_pair(str, 1));  
m.insert(pair<string, size_t>(str, 1));  
m.insert(map<string, size_t>::value_type(str, 1));
```

- (ne)uređeno preslikavanje i skup imaju jedinstvene ključeve \Rightarrow ubacivanje elementa koji je već prisutan nema učinka
- uočite kako možemo ubaciti par u preslikavanje

Povratna vrijednost od `insert/emplace`

- **za spremnike s jedinstvenim ključevima** vraća par
- prvi član para je iterator na element s danim ključem
- drugi član para je `bool` (je li element ubačen)

Primjer. Odredite tip od `r`, ispis, te sadržaj od `m` za unos:

jedan dva jedan tri dva pet jedan

```
map<string, size_t> m;
string s;
int br = 1;
while(cin >> s) {
    auto r = m.insert({s, br++});
    cout << "(" << r.first->first
         << "," << r.first->second << ")" << " "
         << r.second << endl;
}
```

Povratna vrijednost od `insert/emplace`

- ključevi u multi- spremnicima ne moraju biti jedinstveni
- ⇒ uvijek se ubacuje element
- ⇒ `insert` vraća samo iterator na ubačeni element

Primjer. Odredite tip od `r`, ispis, te sadržaj od `m` za unos:

jedan dva jedan tri dva pet jedan

```
multimap<string, size_t> m;
string s;
int br = 1;
while(cin >> s) {
    auto r = m.insert({s,br++});
    cout << "(" << r->first << ", "
         << r->second << ") " << endl;
}
```

Uklanjanje elemenata

erase ima tri verzije:

- (1.) brisanje elementa na koji pokazuje zadani iterator (vraća iterator na idući element ili `end`-iterator)
- (2.) brisanje raspona $[a, b)$ za dva iteratora `a` i `b` (vraća iterator na idući element ili `end`-iterator)
- (3.) za zadanu vrijednost ključa briše sve elemente s tim ključem i vraća broj uklonjenih elemenata

```
set<int> s = {2,4,3,1,5};  
multiset<int> ms = {1,4,1,2,3,4,5};  
cout << s.erase(6) << endl;  
cout << ms.erase(4) << endl;  
multimap<int,int> m = {{2,3},{2,3},  
                       {2,4},{1,5},{3,1},{3,2},{1,4}};  
cout << m.erase(m.begin())->second << endl;  
cout << m.erase(2) << endl;
```


Korištenje indeksa (**a[x]** i **a.at(x)**)

- **dohvaćanje vrijednosti pridružene ključu**
- ne podržavaju ih:
 - skupovi (nema vrijednosti - samo ključevi!)
 - multi... (više vrijednosti može biti pridruženo istom ključu!)
- za preslikavanja: ako nema tog ključa, stvara se novi element za taj ključ i ubacuje u preslikavanje (također, vraća lvalue)

Primjer.

```
set<int> s = {2, 4, 3, 1, 5};  
cout << s[2];      X  
unordered_multimap<int, int> m = {{1, 2}, {2, 3}};  
cout << m[2];      X  
unordered_map<int, int> m;  
m[1] = 2;      ✓ //m.at(1)=2; daje out_of_range  
m[1] = 5;      ✓ //m.at(1)=5; sad ok
```

Traženje elementa: **find** i **count**

- **find** - vraća iterator na (prvi) element sa zadanim ključem (ili end-iterator ako nema takvog elementa)
- **count** - vraća broj elementa sa zadanim ključem

Primjer. Za sljedeće spremnike:

(a) `set<int> s = {0, 1, 2, 1, 4, 3, 4};`

(b) `multiset<int> s = {0, 1, 2, 1, 4};`

(c) `map<int, string> s = {{1, "a"}, {2, "b"},
 {3, "a"}, {2, "d"}, {1, "c"}};`

(d) `multimap<int, string> s = {{1, "a"}, {2, "b"},
 {3, "a"}, {2, "d"}, {1, "c"}};`

odredite čemu bi bilo jednako:

```
s.find(1);
```

```
s.find(11);
```

```
s.count(1);
```

```
s.count(11);
```

Pronalaženje svih elemenata za dani ključ

- u **uređenim spremnicima** elementi s istim ključem su susjedni
⇒ zauzimaju raspon $[a, b)$
- **lower_bound** daje iterator na prvi element s ključem koji nije manji od zadanog, a **upper_bound** na prvi s većim ključem od zadanog
- ako nema takvih elemenata, oba vraćaju `end`-iterator

Primjer. Što radi sljedeći dio koda?

```
multimap<string, string> m = {"Smith", "John"},  
                             {"Garcia", "Maria"}, {"Smith", "Mary"};  
for (auto a = m.lower_bound("Smith"),  
     b = m.upper_bound("Smith"); a != b; ++a)  
    cout << a->second << endl;
```

equal_range funkcija

- umjesto prethodne dvije funkcije, ova vraća traženi **par iteratora** $[a,b)$ za zadani ključ
- ako nema takvih elemenata, oba iteratora iz para odnose se na poziciju gdje se element s tim ključem može ubaciti

Primjer. Kod s istom svrhom kao kod s prethodnog slajda:

```
multimap<string,string> m = {{"Smith","John"},  
                             {"Garcia","Maria"}, {"Smith","Mary"}};  
for (auto p = m.equal_range("Smith");  
     p.first != p.second; ++p.first)  
    cout << p.first->second << endl;
```

Napisati program koji od korisnika učitava naredbe, pri čemu je svaka naredba točno jednog od sljedeća tri oblika:

- `1 ime ocjena` (dodaje ocjenu studentu s imenom ime)
- `2 ime` (briše sve ocjene studenta s imenom ime)
- `3 ime` (ispisuje sve ocjene studenta s imenom ime ili poruku da taj student nema ocjena)

Primjer.

1 John 2

1 Mary 3

1 John 4

3 Mary

3 John

3 Alex

2 John

3 John

3 Mary

Mary - ocjene: 3

John - ocjene: 2 4

Alex nema ocjena.

John nema ocjena.

Mary - ocjene: 3

Neuređeni asocijativni spremnici

- koriste **hash funkciju** i operator `==` na ključevima
 - korisni ako nema očitog uređaja na ključevima ili ako je održavanje sortiranog poretka preskupo
 - organizirani kao kolekcija pretinaca - *hash* funkcija pridružuje elemente pretincima (ključ \mapsto pretinac)
- ⇒ performanse ovise o *hash* funkciji, te broju i veličini pretinaca

Primjer. Za često korištene tipove, standardna biblioteka definira svoje *hash* funkcije (klasa `std::hash`):

```
hash<const char *> h;  
cout << h("abc") << endl; //139988037536407  
cout << h("abd") << endl; //139988037536411
```

Problem

- standardna biblioteka ne implementira veziju `hash<pair<string, string>>` potrebnu u sljedećem kodu:

```
unordered_map<osoba,int> oib;
oib[osoba("John", "Doe")] = 123;
oib[osoba("Jane","Doe")] = 456;
for (auto i = oib.begin() ; i != oib.end() ; i++ )
    cout << i->first.first << " "
         << i->first.second << " : "
         << i->second << endl;
```

- podsjetnik: ovo zahtijeva `#include <unordered_map>`
- pritom je korišteno sljedeće:

```
typedef pair<string,string> osoba;
```

- kako se osoba sastoji od dva stringa, možemo koristiti činjenicu da standardna biblioteka zna kako *hashirati* stringove
- naša nova *hash* funkcija:

```
size_t ime_hash(const osoba &ime) {  
    return hash<string>() (ime.first) ^  
           hash<string>() (ime.second);  
}
```

- preostaje napraviti sljedeće:
 - konstruktoru preslikavanja dajemo dodamo **minimalan** broj pretinaca i pokazivač na *hash* funkciju
 - dodamo treći *template* parametar deklaraciji koji odgovara tipu funkcije koju smo prosljedili konstruktoru

```
unordered_map<osoba, int, decltype(ime_hash) *>  
    oib(100, ime_hash);
```


Razne operacije za pretince

- **broj pretinaca** i **najveći broj pretinaca** koje spremnik može imati:

```
unordered_set<int> s = {1, 3, 2, 5, 4};  
cout << s.bucket_count () << endl           //7  
      << s.max_bucket_count () << endl;    //115...
```

- u **kojem pretincu** tražiti element sa zadanim ključem i **koliko elemenata** ima u *i*-tom pretincu

```
auto i = s.bucket (3);  
cout << i << endl;  
cout << s.bucket_size (i) << endl;
```

- **prosječan** broj elemenata po pretincu

```
cout << s.load_factor () << endl;          //0.714286
```



Razne operacije za pretince

- **maksimalna prosječna veličina pretinca** (dodaju se pretinci tako da vrijedi prosječna veličina \leq maksimalna veličina)

```
cout << s.max_load_factor() << endl; //1
s.insert(6);
s.insert(7);
cout << s.bucket_count() << endl; //17
```

- **reorganizacija** pohrane tako da vrijedi:
 - broj pretinaca $\geq n$ (u donjem primjeru $n = 9$) i
 - broj pretinaca $>$ broj elemenata/maksimalna prosječna veličina

```
s.rehash(9);
cout << s.bucket_count() << endl; //11
```

- `s.reserve(9)` - slično - reorganizacija, ali tako da `s` može pohraniti $n = 9$ elemenata bez *rehashiranja*

Iteratori za prolažanje pretincem

- koristimo iteratore tipa `(const_)local_iterator`
- za dobivanje iterator na prvi i „jedan iza” zadnjeg elementa koristimo `(c)begin()` i `(c)end()`

```
unordered_set<int> s = {12,11,15,16};
for(int i = 0; i < s.bucket_count(); ++i){
    cout << "Elementi u pretincu " << i << ": ";
    unordered_set<int>::const_local_iterator it;
    for(it = s.cbegin(i); it != s.cend(i); ++it)
        cout << *it << " ";
    cout << endl;
}
```

```
Elementi u pretincu 0: 15  
Elementi u pretincu 1: 16 11  
Elementi u pretincu 2: 12  
Elementi u pretincu 3:  
Elementi u pretincu 4:
```

Zahtjev na tip ključeva neuređenih spremnika

- po defaultu, neuređeni spremnici koriste operator `==` na tipu ključeva za usporedbu elemenata
- u prethodnom problemu stoga nije dovoljno samo napisati `hash` funkciju, nego i funkciju koja će zamijeniti operator `==` ako je osoba sljedeći tip:

```
struct osoba {  
    string oib, ime, prezime;  
};
```

- funkcija koja će zamijeniti operator `==`:

```
bool jedn(const osoba &a, const osoba &b) {  
    return a.oib == b.oib;  
};
```

- tada ovako definiramo neuređeno preslikavanje `oib`:

```
unordered_map<osoba, int, decltype(ime_hash) *,  
    decltype(jedn) *> oib(100, ime_hash, jedn);
```

JMBAG se sastoji od 10 znamenki - prve četiri određuju ustanovu visoke naobrazbe koja je dodijelila taj JMBAG, a zadnjih šest određuju akademsku osobu unutar te ustanove.

Definirajte strukturu za pamćenje sljedećih podataka o studentima: jmbag, ime i prezime. Napišite program koji koristi neuređeno višestruko preslikavanje za spremanje ocjena pojedinog studenta - ulaz u program su linije oblika (pri čemu su ime i prezime po jedna riječ):

```
jmbag ime prezime ocjena
```

Pritom koristite *hash* funkciju koja za danog studenta određuje broj pretinca na temelju posljednjih 6 znamenki JMBAG-a (stavite da je barem 10 pretinaca). Za kontrolu ispišite sadržaje pojedinih pretinaca.