

# **Tema 14**

## **Algoritmi pretraživanja iz biblioteke šablona jezika C++**

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2



# Sadržaj

1. Uvod
2. Pretraživanja niza objekata metodima biblioteke STL
3. Particija niza objekata metodima biblioteke STL
4. Algoritmi binarnog pretraživanja biblioteke STL
5. Primer: Implementacija heš tabela



# 1. Uvod

- Problem pretraživanja
- Algoritmi pretraživanja biblioteke STL



# Problem pretraživanja

- Pretraživanje skupova objekata u memoriji prema sadržaju ili delu njihovog sadržaja opšti je problem programiranja
- **Asocijativni** kontejneri omogućavaju pronalaženje elemenata po sadržaju za vreme  $O(\log(n))$  ili  $O(1)$
- **Kontejneri sekvenci** su linearne strukture (nizovi), u kojima se načelno objekti čija pozicija u nizu nije unapred poznata pronalaze *pretraživanjem* za vreme  $O(n)$
- Pronalaženje sadržaja u *sortiranim* nizovima objekata može se izvršiti za vreme  $O(\log(n))$ 
  - osim toga, sortiranje znatno ubrzava *ažuriranje* nizova objekata
- U biblioteci STL postoje algoritmi pretraživanja sekvenci (nizova) objekata na različite načine

# Algoritmi pretraživanja biblioteke STL

- Biblioteka STL nudi tri algoritma za pronalaženje objekta u nizu definisanom pomoću prva dva argumenta (iteratora)
  - `find()` - pronalazi prvi objekt jednak trećem argumentu
  - `find_if()` - pronalazi prvi objekt za koji je *istinit predikat* naveden u trećem argumentu (vraća vrednost *true*); predikat ne sme da menja objekt koji ispituje i može se zadati kao *lambda izraz*: anonimna funkcija, koja se definiše na mestu njene upotrebe, npr.  
`[element](parametri) {return Logički zraz}`
  - `find_if_not()` - pronalazi prvi objekt za koji *nije istinit* predikat naveden u trećem argumentu (vraća vrednost *false*); predikat ne sme da menja objekt koji ispituje i može se zadati kao *lambda izraz*
- Svaki od algoritama vraća *iterator* koji pokazuje na pronađeni objekt ili na kraj niza, ako objekt nije pronađen

## 2. Pretraživanja niza objekata metodima biblioteke STL

1. Pronalaženje elementa u nizu objekata
2. Pronalaženje nekog od elemenata niza u nizu objekata
3. Pronalaženje ponavljanja elemenata drugog niza u nizu objekata



## 2.1 Pronalaženje elementa u nizu objekata

- Pronalaženje elementa u nizu može se izvršiti pomoću algoritma `find()`, npr.

```
std::vector<int> brojevi {5,46,-5,-6,23,17,5,9,6,5};  
int broj {23};  
auto iter = std::find(std::begin(brojevi),  
                      std::end(brojevi),  
                      broj);  
if (iter != std::end(brojevi))  
    std::cout << broj << " je pronađen\n";
```

- Element je pronađen ako iterator `end()` pokazuje na element niza, inače pokazuje na kraj, iza poslednjeg elementa niza
- Pronalaženje više instanci elementa u zadanom nizu vrši se ponavljanjem pretraživanja, od sledećeg elementa do kraja

# Primer: Pronalaženje svih pojava elementa niza

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

int main() {
    std::vector<int> brojevi { 5, 46, -5, -6, 23, 17, 5, 9, 6, 5 };
    size_t n{};
    int broj{ 5 };      // broj 5 se nalazi 3 puta u nizu brojevi
    auto poc_it = std::begin(brojevi);
    auto kraj_it = std::end(brojevi);
    while ((poc_it = std::find(poc_it, kraj_it, broj)) != kraj_it) {
        ++n;
        ++poc_it;
    }
    std::cout << "Element " << broj << " je pronadjen " << n << " puta"
              << std::endl;
    return 0;
}
```

Element 5 je pronadjen 3 puta



# Pronalaženje prvog većeg elementa u nizu

- Pronalaženje elementa u nizu koji je veći od neke vrednosti može se izvršiti algoritmom `find_if()`, npr.

```
std::vector<int>brojevi {5,46,-5,-6,23,17,5,9,6,5};  
int broj {5};    // prvi veći element niza je 46  
auto iter = std::find_if(std::begin(brojevi),  
                        std::end(brojevi),  
                        [broj](int n) { return n > broj; });  
if (iter != std::end(brojevi))  
    std::cout << "Broj " << *iter << " je veći od " << broj;
```

- Treći argument algoritma `find_if` je anonimni *lambda izraz* koji definiše *predikat*, koji se izračunava za svaki element niza
- Pronalaženje više svih elemenata većih od zadanog vrši se ponavljanjem pretraživanja, od sledećeg elementa do kraja

# Pronalaženje svih elemenata u nizu koji nisu veći od zadanog elementa

- Algoritam `find_if_not()` pogodan je za pronalaženje elementa u nizu koji *nisu veći* od zadanog, odnosno koji su *manji ili jednaki* zadanom, npr.

```
std::vector<int> brojevi {5,46,-5,-6,23,17,5,9,6,5};
size_t n {};
int broj {5};    // ima 5 elemenata koji nisu veći od 5
auto poc_iter = std::begin(brojevi);
auto kraj_iter = std::end(brojevi);
while ((poc_iter = std::find_if_not(poc_iter, kraj_iter,
    [broj](int n){return n>broj;}) )
    != kraj_iter){
    ++n;
    ++poc_iter;
}
std::cout << n << " elemenata nije veće od " << broj << std::endl;
```

## 2.2 Pronalaženje nekog od elemenata niza u nizu objekata

- Pronalaženje u zadanom nizu prve pojave nekog elementa iz drugog niza, može se izvršiti algoritmom `find_first_of()`
  - poređenje se vrši standardnim ili preklopljenim operatorom `==` (za korisničke klase)
- Npr. prvi samoglasnik u zadanom tekstu pronalazi segment


```
std::string tekst {"Tekstza pretrazivanje"}; // prvi je 'e'
std::string samoglasnici {"aeiou"};
auto iter = std::find_first_of(std::begin(tekst),
                               std::end(tekst),
                               std::begin(samoglasnici),
                               std::end(samoglasnici));

if (iter != std::end(tekst))
    std::cout << "Pronadjen samoglasnik '" << *iter << "'
               << std::endl;
```

# Primer: Pronalaženje svih elemenata nekog niza objekata u drugom nizu

```
#include <iostream>
#include <string>
#include <algorithm>

int main() {
    std::string tekst {"Tekst za pretrazivanje"};
    std::string samoglasnici {"aeiou"};
    std::string pronadjeni {}; // zapisuje sve pronađene znakove
    for(auto iter = std::begin(tekst);
        (iter = std::find_first_of(
            iter,
            std::end(tekst),
            std::begin(samoglasnici),
            std::end(samoglasnici))) != std::end(tekst); )
        pronadjeni += *(iter++);
    std::cout << "Pronadjeni samogl. '" << pronadjeni << "'" << std::endl;
    return 0;
}
```



# Pronalaženje u nizu objekata nekog od elemenata niza za koji važi zadani predikat

- Pronalaženje u zadanom nizu prve pojave nekog elementa iz drugog niza, za koji važi neki predikat, može se takođe izvršiti algoritmom `find_first_of()`
  - predikat može biti *lambda izraz*, koji se zadaje kao peti argument
  - mogu se porediti elementi različitog tipa, kad nije definisan operator `==`
- Npr. celi brojevi, koji su deljivi nekim od zadanih faktora

```
std::vector<long> brojevi {64L, 46L, -65L, -128L, 121L, 17L, 35L, 91L};  
int faktori[] {7, 11, 13};  
auto iter = std::find_first_of(  
    std::begin(brojevi), std::end(brojevi), // niz  
    std::begin(faktori), std::end(faktori), // traženi  
    [](long v, long d){return v%d == 0;}); // predikat  
if (iter != std::end(brojevi))  
    std::cout << "Prvi deljivi je " << *iter << std::endl;
```

```
Prvi deljivi je -65
```

## 2.3 Pronalaženje više elemenata drugog niza u nizu objekata

- U zadanom nizu mogu se pronaći *ponavljanja podnizova* (više elemenata drugog niza):
  - prvo *ponavljanje elemenata* algoritmom `adjacent_find()`
  - poslednje *ponavljanje podniza* elemenata algoritmom `find_end()`
  - prvo *ponavljanje podniza* elemenata algoritmom `search()`
  - zadani broj *n ponavljanja podniza* algoritmom `search_n()`



# 3. Particija niza objekata metodima biblioteke STL

- Particija (podela) elemenata nekog niza je *preuređenje niza*, tako da elementi za koje je neki predikat istinit prethode ostalim elementima niza
- Particija niza objekata može se izvršiti algoritmima
  - `partition()` - vrši podelu niza na dva dela, u skladu s zadanim predikatom, npr. na vrednosti manje od srednje vrednosti niza i ostale
  - `partition_copy()` - vrši podelu elemenata niza na dva dela, ali ne menja originalni niz, već particije kopira u dva posebna niza
  - `partition_point()` - vraća iterator kraja prve particije niza
- Algoritmi se koriste za ubrzavanje operacija nad neuređenim nizovima, kad se ne želi njihovo potpuno sortiranje



## 4. Algoritmi binarnog pretraživanja biblioteke STL

- **Binarno** pretraživanje pretpostavlja prethodno *sortirane* nizove objekata, jer se pretraživanje zasniva na proveru da li je tekući element manji ili veći od traženog
- Biblioteka šablona nudi sledeće algoritme binarnog pretraživanja niza objekata
  - `binary_search()` - pretražuje niz zadan pomoću dva iteratora i vraća logičku vrednost *true* ako je element pronađen, inače vraća *false*
  - `lower_bound()` - pretražuje niz zadan pomoću dva iteratora i pronalazi *prvi element* koji nije manji, odnosno veći je ili jednak zadanom elementu
  - `upper_bound()` - pretražuje niz zadan pomoću dva iteratora i pronalazi *prvi element* koji je strogo veći od zadanog
  - `equal_range()` - pronalazi sve elemente jednake zadanom



# Algoritam binary\_search()

- Algoritam samo ustanovljava da li se traženi element nalazi u zadanom nizu elemenata, npr. u kontejneru tipa dvostruko povezane liste

```
// Sortirana lista celih brojeva u rastućem poretku
std::list<int> lista {11,17,22,36,40,43,48,70,54,61,78,82,89,92,99};
int trazen_i {22}; // element koji se traži (nalazi se u listi)
if (std::binary_search(std::begin(lista),
                      std::end(lista),
                      trazen_i))
    std::cout << trazen_i << " se nalazi u listi" << std::endl;
else
    std::cout << trazen_i << " nije u listi" << std::endl;
```

- Algoritam omogućava upotrebu četvrtog argumenta, koji može biti lambda funkcija za poređenje elemenata, npr.

```
[](int a, int b){ return a > b; };
```

## lower\_bound() i upper\_bound()

- Algoritmi pretražuju sortirani niz elemenata i pronalaze prvi element koji je veći ili jednak ili je strogo veći od zadanog
  - elementu su uređeni preko operatora <

```
// Sortirana lista celih brojeva u rastućem poretku
std::list<int> lista {11,17,22,36,40,43,48,70,54,61,78,82,89,92,99};
int trazen_i {22}; // element koji se traži (nalazi se u listi)
std::cout << "Donja granica za " << trazen_i << " je "
    << *std::lower_bound(std::begin(lista),
        std::end(lista), trazen_i)
    << std::endl;
std::cout << "Gornja granica za " << trazen_i << " je "
    << *std::upper_bound(std::begin(lista),
        std::end(lista), trazen_i)
    << std::endl;
```

```
Donja granica za 22 je 22
Gornja granica za 22 je 36
```

# Algoritam equal\_range()

- Algoritam pronalazi objekte niza koji su ekvivalentni traženom
  - vraća objekt koji sadrži dva iteratora; prvi iterator pokazuje na element koji nije manji od traženog, a drugi na element koji je veći od traženog
- Efekt je kao da se jednim pozivom pokreću algoritmi `lower_bound()` i `upper_bound()`, pa se može napisati

```
// Sortirana lista celih brojeva u rastućem poretku
std::list<int> lista {11,17,22,36,40,43,48,70,54,61,78,82,89,92,99};
int trazen_i {22};    // element koji se traži (nalazi se u listi)
auto par = std::equal_range(std::begin(lista),
                           std::end(lista), trazen_i);
std::cout << "Donja granica za " << trazen_i << " je "
           << *par.first << std::endl;
std::cout << "Gornja granica za " << trazen_i << " je "
           << *par.second << std::endl;
```

## 5. Primer

- Implementacija heš tabela



# Implementacija heš tabela

- Heš tabele
- Kolizije u heš tabeli
- Heš funkcije



# Heš tabele

- **Heš tabele** predstavljaju strukture koja imaju najbolje teorijske i stvarne performanse u primenama gde su potrebne operacije pronalaženja, umetanja i brisanja elemenata
- Sve operacije se izvršavaju za konstantno srednje vreme  $O(1)$ , a u najgorem slučaju  $O(n)$ , kad je potrebna promena veličine heš tabele
- Tabela sadrži elemente koji se smeštaju u polje na poziciju koja se izračunava na osnovu dela sadržaja (ključa)
- Izračunata pozicija se naziva "heš kod", a računa se pomoću pogodno odabrane funkcije, koja treba da obezbedi ravnomernu distribuciju elemenata u polju

# Kolizije u heš tabeli

- Kada se za različite vrednosti ključa dobije isti rezultat, događa se "kolizija", koja se razrešava na različite načine, npr. upotrebom prve naredne slobodne pozicije u polju ili smeštanjem elementa u listu kolizija
- Složenost metoda je  $O(1+n/m)$  gde je  $n$  broj elemenata, a  $m$  dimenzija polja (tabele)
- Za  $n > m$ , kada se tabela popuni, tabelu je potrebno povećati, a elemente premestiti u novu tabelu, uz ponovno računanje heš indeksa (*rehashing*)
- Vremenska složenost ove operacije je  $O(n+m)$ , ali se ona događa retko, npr. kad se popuni tabela čija se veličina svaki put udvostručava

# Heš funkcije

- Osnovni element implementacije heš tabele je pogodna *heš funkcija*, koja uniformno raspoređuje objekte po polju tabele
- Npr. za ključeve tipa `string`, heš funkcija može imati oblik

```
int stringHash(const string& str, int modul) {  
    const int k = 997;  
    int v = 0;  
    for (char c : str) {  
        v = (v * k + c) % modul  
    }  
    return v;  
}
```

- Ključeve elemenata u heš tabeli ne treba menjati, već prvo ukloniti iz tabele, a zatim ponovo dodati, s novim ključem



# Perfektne heš funkcije

- **Perfektna** ili savršena heš funkcija idealno preslikava skup ključeva u potpuno različite elemente, *bez pojave kolizije*
  - to omogućava postizanje konstatnog vremena pristupa elementima heš tabele u praksi
- **Minimalna** perfektne heš funkcija preslikava  $n$  ključeva u  $n$  uzastopnih celih brojeva ( $0..n-1$  ili  $1..n$ )



# Praktična primena heš tabela

- Realizacija različitih algoritama visokih performansi, npr.
  - liste simbola prevodilaca
  - traženje anagrama
  - keširanje različitih sadržaja



# Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
3. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
4. Horton I., *Beginning C++*, Apress, 2014
5. Horton I., *Beginning Visual C++ 2013*, Wox/John Wiley&Sons, 2014
6. Horton I., *Using the C++ Standard Template Libraries*, Apress, 2015
7. Galowitz J., *C++ 17 STL Cookbook*, Packt, 2017
8. Veb izvori
  - <http://www.cplusplus.com/doc/tutorial/>
  - <http://www.learncpp.com/>
  - <http://www.stroustrup.com/>
9. Vikipedija [www.wikipedia.org](http://www.wikipedia.org)
10. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019

