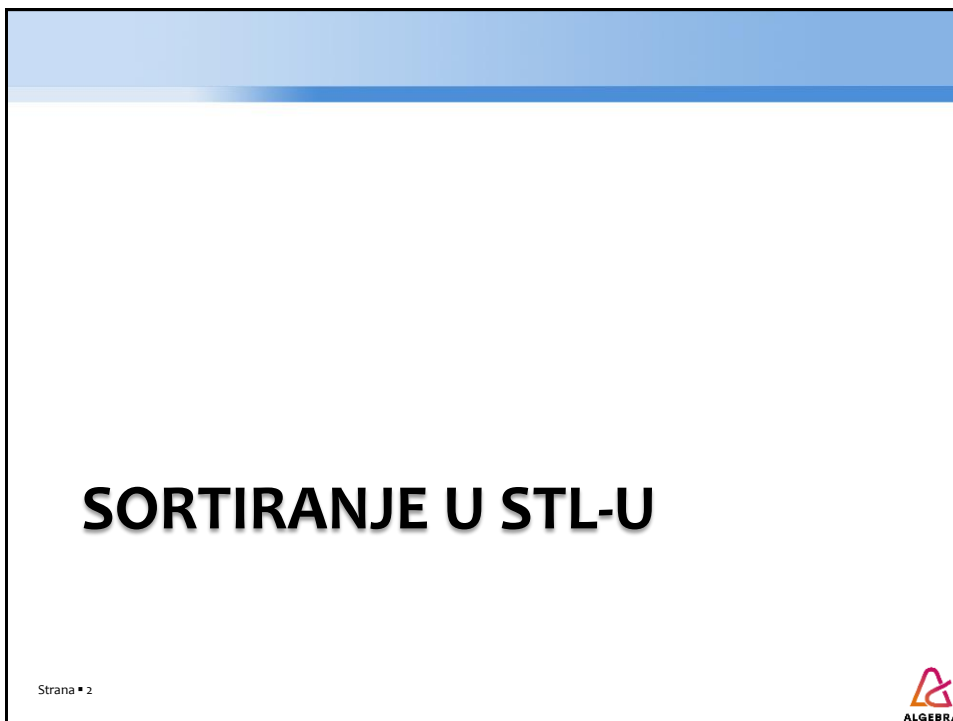

ALGEBRA

STRUKTURE PODATAKA I ALGORITMI

Predavanje 13


Ishod 5

1



SORTIRANJE U STL-U

Strana • 2



2

Uvod

- STL dolazi s dobrim algoritmima pomoću kojih možemo zadovoljiti sve potrebe za sortiranjima u našim aplikacijama
- Osnovna funkcija je `sort()` složenosti $O(n \log n)$ koja dolazi u dvije verzije:
 - `sort(begin, end)`
 - Rastuće sortira raspon `[begin, end)` koristeći operator manje od
 - `sort(begin, end, komparator)`
 - Sortira raspon `[begin, end)` koristeći zadani komparatorsku funkciju
 - Komparatorska funkcija je funkcija koja prima dva objekta i vraća treba li prvi doći prije drugog
 - Slično, ali ne jednako kao kod prioriternog reda (tamo je komparator bio struktura)

Strana * 3



3

Primjer rastućeg sortiranja

```
int brojevi[] = { 3, 1, 5, 2, 4};
sort(brojevi, brojevi + 5);
for (int i = 0; i < 5; i++) {
    cout << brojevi[i] << ' ';
}
cout << endl;

vector<string> osobe({ "Zeljka", "Anica", "Mirko",
                    "Ivana", "Branko" });
sort(osobe.begin(), osobe.end());
for (auto it = osobe.begin(); it != osobe.end(); ++it) {
    cout << *it << ' ';
}
cout << endl;
```

Strana * 4



4

Primjer padajućeg sortiranja i sortiranja objekata

```

struct pravokutnik {
    int a;
    int b;
    pravokutnik(int a, int b) {
        this->a = a;
        this->b = b;
    }
    int površina() {
        return this->a * this->b;
    }
};

bool padajuce(int a, int b) {
    return a > b;
}

bool asc_pravokutnici(pravokutnik a, pravokutnik b) {
    return a.površina() < b.površina();
}

```

Strana * 5



5

Primjer padajućeg sortiranja i sortiranja objekata

```

int main() {
    int brojevi[] = { 3, 1, 5, 2, 4};
    sort(brojevi, brojevi + 5, padajuce);
    for (int i = 0; i < 5; i++) {
        cout << brojevi[i] << ' ';
    }
    cout << endl;

    vector<pravokutnik> p({ pravokutnik(10, 10),
        pravokutnik(2, 2), pravokutnik(4, 4) });
    sort(p.begin(), p.end(), asc_pravokutnici);
    for (auto it = p.begin(); it != p.end(); ++it) {
        cout << it->površina() << ' ';
    }
    cout << endl;

    return 0;
}

```

Strana * 6



6

Zadatak

- Pripremimo vektor s razbacanim brojevima od 1 do 2 milijuna te ga sortirajmo rastuće. Ispišimo koliko traje sortiranje.

- Rješenje:

```
srand(unsigned(time(0)));
vector<int> v;
for (int i = 1; i <= 2000000; i++) { v.push_back(i); }
random_shuffle(v.begin(), v.end());

auto start = chrono::high_resolution_clock::now();
sort(v.begin(), v.end());
auto end = chrono::high_resolution_clock::now();

cout
  << "Vrijeme: "
  << chrono::duration_cast<chrono::milliseconds>(end- start).count()
  << " ms" << endl;
```

Strana * 7



7

Algoritmi ispod haube

- Visual Studio implementira funkciju sort pomoću Intro sorta i Insertion sorta
- Princip rada je sljedeći:
 - Ako se sortira < 32 elementa, koristi Insertion sort
 - Ako se sortira ≥ 32 elementa, koristi Intro sort
 - Započne s Quick sortom
 - Kao pivot koristi medijan od prvog, srednjeg i zadnjeg elementa
 - Ako Quick sort probije definiranu granicu dubine rekurzije, odustane od Quick sorta i prebaci se na Heap sort
 - Granica rekurzije je obično definirana kao $2 \cdot \log n$

Strana * 8



8

Stabilno sortiranje

- Ako imamo početne elemente i sortiramo ih po ključu:

12	49	12	3
Mireau Miric	Yuro Yurich	Anna Anicci	Schtepf Stefi

- Nestabilno sortiranje može proizvesti rezultat:

3	12	12	49
Schtepf Stefi	Anna Anicci	Mireau Miric	Yuro Yurich

- Dok stabilno sortiranje garantirano daje rezultat:

3	12	12	49
Schtepf Stefi	Mireau Miric	Anna Anicci	Yuro Yurich

Strana * 9



9

Primjer primjene stabilnog sortiranja

- Primjer:

- Imamo sortirane studente po prezimenu i imenu
- Želimo ih dodatno sortirati padajuće po broju bodova na ispitu
- Uz to, želimo da studenti koji imaju isti broj bodova zadrže svoj abecedni redoslijed

Student	Bod
Ana Anić	51
Branka Brankić	93
Bruno Brunić	43
Iva Ivić	55
...	
Željko Željkić	93

Student	Bod
Željko Željkić	93
Branka Brankić	93
Iva Ivić	55
Ana Anić	51
...	
Bruno Brunić	43

Student	Bod
Branka Brankić	93
Željko Željkić	93
Iva Ivić	55
Ana Anić	51
...	
Bruno Brunić	43

Strana * 10

Sortirani
abecedno

Nestabilno

Stabilno



10

Stabilno sortiranje

- U stabilnom sortiranju elementi zadržavaju svoj relativni položaj i nakon sortiranja
- Algoritam `sort()` nije stabilan
- Funkcija `stable_sort()` je stabilna verzija algoritma `sort()`
 - Koristi se jednako kao i `sort()`
 - Ima složenost $O(n \log^2 n)$
 - Ako postoji dovoljno memorije, ima složenost $O(n \log n)$
 - Koristi Merge sort u kombinaciji s Insertion sortom

Strana • 11



11

Sortiranje liste

- Neke kontejnere nema smisla sortirati (red, stog, mapa, ...)
- Metodom `sort()` možemo sortirati „sve” kontejnere osim listi
 - Lista nema iterator slučajnog pristupa `[]` niti `at()`
- `list<T>::sort()` je također stabilni sort temeljen na Merge sortu
 - Ima jednaku složenost

Strana • 12



12

Primjer sortiranja liste

```

bool desc_pravokutnici(pravokutnik a, pravokutnik b) {
    return a.povrsina() > b.povrsina();
}

int main() {
    list<pravokutnik> l({
        pravokutnik(10, 10),
        pravokutnik(2, 2),
        pravokutnik(4, 4) });

    l.sort(desc_pravokutnici);

    for (auto it = l.begin(); it != l.end(); ++it) {
        cout << it->povrsina() << ' ';
    }
    cout << endl;

    return 0;
}

```

Strana * 13



13

Sortiranje dijela raspona

- `partial_sort(begin, middle, end)`
 - Elementi ispred middle su najmanji elementi u cijelom rasponu i sortirani su
 - Nema garancije kako su poslagani svi ostali elementi
 - Koristi Heap sort, složenost je $O(n \log m)$, gdje je m udaljenost od begin do middle
 - Pitanje: u čemu je razlika u odnosu na `sort(begin, middle)`?
 - Isprobajte je li brže parcijalno sortiranje prvih 500k ili potpuno sortiranje svih razbacanih brojeva od 1 do 2 milijuna.

Strana * 14



14

Zadatak

- Ispišimo tri najmanja elementa u cijelom vektoru.

- Rješenje:

```
vector<int> v({ 8, 3, 1, 5, 2, 4, 6, 7, 9, 10, 13, 11, 15,
12, 14, 16, 17, 19, 20, 18 });
```

```
partial_sort(v.begin(), v.begin() + 3, v.end());
```

```
for (auto it = v.begin(); it != v.begin() + 3; ++it) {
    cout << *it << ' ';
}
cout << endl;
```

Strana • 15



15

Provjera sortiranosti

- `is_sorted(begin, end)`
 - Vraća je li raspon `[begin, end)` sortiran
- `is_sorted_until(begin, end)`
 - Vraća iterator na prvi element u rasponu `[begin, end)` koji nije sortiran
 - Ako vrati `end`, cijeli raspon je sortiran

Strana • 16



16

Primjer

```

srand(unsigned(time(0)));
vector<int> v;
for (int i = 1; i <= 35; i++) { v.push_back(i); }

random_shuffle(v.begin(), v.end());
for (auto it = v.begin(); it != v.end(); ++it) {
    cout << *it << ' ';
}
cout << endl;

cout << "Is sorted: " << is_sorted(v.begin(), v.end()) << endl;

cout << "Sortirani elementi: ";
auto first_unsorted = is_sorted_until(v.begin(), v.end());
for (auto it = v.begin(); it != first_unsorted; ++it) {
    cout << *it << ' ';
}
cout << endl;

```



17

Sortiranje n-tog elementa

- `nth_element(begin, nth, end)` u linearnoj složenosti modifikira raspon na način:
 - Na poziciji `nth` je vrijednost koja bi tu trebala biti da je polje sortirano (ta vrijednost je sad na svome mjestu)
 - Elementi ispred `nth` sigurno nisu veći od elementa na `nth`
 - Elementi iza `nth` sigurno nisu manji od elementa na `nth`
- Zadatak: izračunajmo koji broj je treći najmanji broj u rasponu brojeva od 1 do 2 milijuna. Usporedimo brzinu izvršavanja u usporedbi sa sortiranjem cijelog raspona.

Strana • 18



18

Primjer

```

srand(unsigned(time(0)));
vector<int> v;
for (int i = 1; i <= 2000000; i++) { v.push_back(i); }
random_shuffle(v.begin(), v.end());

auto start = chrono::high_resolution_clock::now();
nth_element(v.begin(), v.begin() + 2, v.end());
auto end = chrono::high_resolution_clock::now();

cout
  << "Vrijeme: "
  << chrono::duration_cast<chrono::milliseconds>(end-start).count()
  << " ms" << endl;

cout << *(v.begin() + 2) << endl;

```

Strana * 19



19

Sažetak

- Glavni alat
 - Funkcija `sort()` sortira zadani raspon prema zadanom kriteriju
- Želimo sortirati uz zadržavanje prethodnog međusobnog odnosa jednakih elemenata
 - Funkcija `stable_sort()`, ali performanse manje ili više lošije
- Želimo pronaći onu vrijednost koja bi bila na n -tom mjestu sortiranih elemenata
 - Funkcija `nth_element()`
- Želimo pronaći sve vrijednosti koja bi bile od početka do n -tog mjesta sortiranih elemenata
 - Funkcija `partial_sort()`

Strana * 20



20

PARTICIONIRANJE

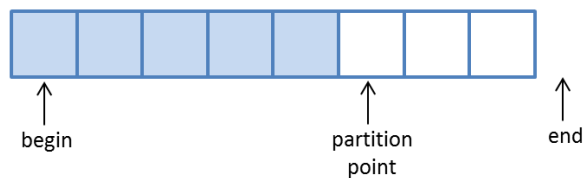
Strana • 21



21

Partitioniranje

- Ponekad imamo potrebu efikasno podijeliti elemente nekog kontejnera u one koji zadovoljavaju uvjet i one koji ga ne zadovoljavaju



- U primjeru su:
 - Plavo označeni elementi koji zadovoljavaju uvjet
 - Bijelo označeni oni koji ne zadovoljavaju
 - Točka particioniranja je prvi element koji ne zadovoljava uvjet

Strana • 22

Slika preuzeta s
fluentcpp.com

22

Funkcije za partitioniranje

- Osnovne funkcije su:
 - `partition(begin, end, predikat)`
 - Radi partitioniranje prema predikatu i vraća iterator na točku partitioniranja
 - Predikat je funkcija koja prima vrijednost i vraća `bool`
 - Složenost je $O(n)$
 - `is_partitioned(begin, end, predikat)`
 - Provjerava je li raspon partitioniran
 - `partition_point(begin, end, predikat)`
 - Vraća iterator na točku partitioniranja

Strana * 23



23

Zadatak s partitioniranjem

- Napravimo vektor sa slučajno poslaganim brojevima od 1 do 2 milijuna pa ga partitionirajmo tako da prvo dolaze parni, a zatim neparni. Ispišite prvi neparni broj.

- Rješenje:

```
bool paran(int n) {
    return n % 2 == 0;
}
int main() {
    srand(unsigned(time(0)));
    vector<int> v;
    for (int i = 1; i <= 2000000; i++) { v.push_back(i); }
    random_shuffle(v.begin(), v.end());
    auto start = chrono::high_resolution_clock::now();
    auto pp = partition(v.begin(), v.end(), paran);
    auto end = chrono::high_resolution_clock::now();
    cout << "Partition point: " << *pp << endl;
    return 0;
}
```

Strana * 24



24

BINARNO PRETRAŽIVANJE

Strana • 25



25

Binarno pretraživanje

- Binarno pretraživanje je postupak kojim tražimo neku vrijednost u sortiranom kontejneru i to u složenosti $O(\log n)$
- Dvije verzije:
 - `binary_search(begin, end, value)`
 - Vraća `true` ako tražena vrijednost postoji u zadanom rasponu
 - `binary_search(begin, end, value, komparator)`
 - Vraća `true` ako tražena vrijednost postoji u zadanom rasponu, ali koristeći jednaku komparatorsku funkciju kao prilikom sortiranja

Strana • 26



26

Zadatak

- Pripremite vektor a razbacanim vrijednostima od 1 do 2 milijuna. Potražite vrijednost 1.234.456 na sljedeće načine i usporedite trajanja:
 - Linearno pretraživanje na nesortiranom polju
 - Linearno pretraživanje na sortiranom polju
 - Sortiranje + binarno pretraživanje
 - Samo binarno pretraživanje