STRUKTURE PODATAKA I ALGORITMI

Predavanje 05

Ishod 2

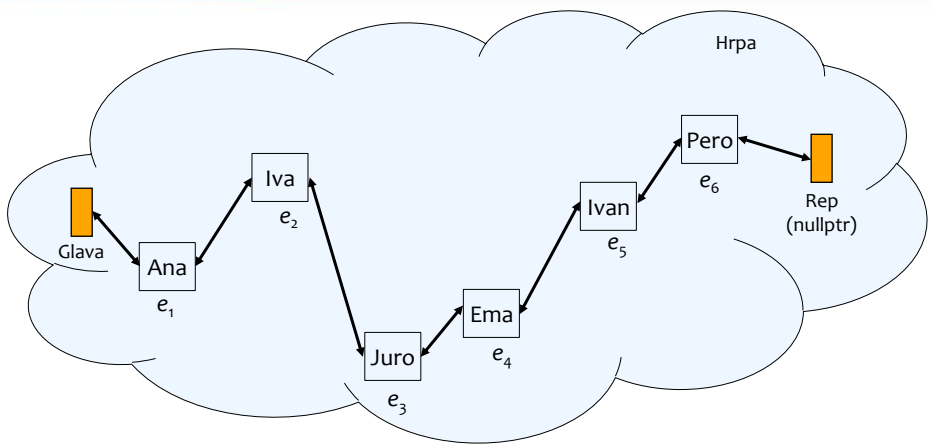
1

POVEZANA LISTA

ADT povezana lista

- ADT povezana lista je ADT lista s karakteristikama:
  - Element se obično naziva čvor (engl. *node*)
  - Čvorovi u memoriji nisu poslagani jedan iza drugoga
    - Ne smijemo koristiti aritmetiku s pokazivačima
    - Ne možemo direktno pristupiti *i*-tom elementu
  - Svaki čvor zna gdje se nalazi sljedeći (i možda prethodni) čvor
    - Najčešće se radi o jednom ili dva pokazivača
    - Ponekad se zbog kompaktnosti ispušta veza na prethodni čvor
  - Izmjena liste na bilo kojem dijelu je efikasna
    - Lista efikasno raste i smanjuje se za vrijeme izvođenja programa dodavanjem ili uklanjanjem čvorova

ADT povezana lista – grafički prikaz



- Svaki čvor se dinamički uzima s hrpe i vraća kad više nije potreban
- Svaki čvor osim podatka sadrži i jedan ili dva pokazivača

KONKRETNE POVEZANE LISTE

5

Konkretne povezane liste

- C++ dolazi s dvije implementacije:
  - Generička klasa `list<T>`
  - Generička klasa `forward_list<T>`
- Klase su vrlo slične, sa sljedećim glavnim razlikama:
  - `forward_list<T>` troši manje memorije po svakom čvoru jer sadrži samo pokazivač na sljedeći čvor
    - Važna stavka za okruženja s manjom količinom resursa (Arduino, …)
    - Omogućuje samo iteriranje od početka prema kraju
  - `list<T>` sadrži i pokazivač na prethodni element
    - Omogućuje i iteriranje od kraja prema početku
- Sve što u nastavku kažemo vrijedi za obje liste (razlike ćemo ono istaknuti)

6

Izrada i uništavanje povezane liste (1/2)

- Postoji šest osnovnih načina izrade povezane liste:
  - `list<int> jedan;`
    - Kreira praznu povezanu listu (*default*)
  - `list<int> dva(n);`
    - Kreira povezanu listu od *n* elemenata inicijaliziranih na *defaultnu* vrijednost (*fill*)
  - `list<int> tri(n, val);`
    - Kreira povezanu listu od *n* elemenata, svaki je kopija od *val* (*fill*)
  - `list<int> cetiri(iter1, iter2);`
    - Kreira povezanu listu kopiranjem elemenata iz zadanog raspona (*range*)

ALGEBRA BERNAYS SVEUČILIŠTE

7

Izrada i uništavanje povezane liste (2/2)

  - `list<int> pet(tri);`
    - Kreira povezanu listu na način da kopira sve elemente iz zadane povezane liste (*copy*)
  - `list<int> sest({ 11, 22, 33 });`
    - Kreira listu na način da kopira sve elemente iz inicijalizacijske liste (*initializer list*)
- Povezana lista se automatski uništava završetkom funkcije u kojoj je deklarirana
  - Ako povezana lista čuva objekte, na svakom se poziva destruktor
- `operator=` kopira sadržaj jedne povezane liste u drugu
  - Prethodni sadržaj druge povezane liste se uništava (prepisivanjem ili otpuštanjem)

ALGEBRA BERNAYS SVEUČILIŠTE

8

Pristup elementima povezane liste

- Povezana lista nema `operator[]` niti metodu `at()`
- Povezana lista nudi samo sljedeće načine pristupa elementima:
  - `l.front()` vraća referencu na prvi element
    - Ako je povezana lista prazna, ponašanje nije definirano
  - `l.back()` vraća referencu na zadnji element
    - Ako je povezana lista prazna, ponašanje nije definirano
    - Ne postoji na `forward_list<T>`
  - Korištenjem iteratora

Iteratori povezane liste (1/2)

- Najvažniji iteratori su:
  - `list<T>::iterator` je iterator čiji ++ pomiče prema kraju
  - `list<T>::reverse_iterator` je iterator čiji ++ pomiče prema početku
    - Ovaj iterator ne postoji na `forward_list<T>`

Iteratori povezane liste (2/2)

- Sljedeće metode vraćaju iteratore:
  - `l.begin()` – vraća iterator koji pokazuje na prvi element
  - `l.end()` – vraća iterator koji pokazuje na prvi element iza kraja
  - `l.rbegin()` – vraća reverzni iterator na zadnji element
    - Nema ga na `forward_list<T>`
  - `l.rend()` – vraća reverzni iterator na element ispred prvog
    - Nema ga na `forward_list<T>`
  - `l.before_begin()` – vraća iterator na element prije prvog
    - Nema ga na `list<T>`
    - Namijenjen za upotrebu u metodama `emplace_after`, `insert_after` i `erase_after`

ALGEBRA BERNAYS SVEUČILIŠTE

11

Još malo o iteratorima (1/2)

- Sljedeći kôd radi ispravno:

```
vector<int> v1({ 11, 22, 33, 44, 55 });
vector<int> v2(v1.begin() + 3, v1.end());
```

- Što je onda problem sa sljedećim kôdom:

```
list<int> l1({ 11, 22, 33, 44, 55 });
list<int> l2(l1.begin() + 3, l1.end());
```

- Odgovor: elementi liste nisu poslagan jedan iza drugoga u memoriji pa ne možemo pisati + 3
  - Sjetimo se: sve što nam iterator garantira su operacije `++it` i `*it`

ALGEBRA BERNAYS SVEUČILIŠTE

12

Još malo o iteratorima (2/2)

- Kako onda riješiti problem:

```
list<int> l1({ 11, 22, 33, 44, 55 });
auto it1 = l1.begin();
for (int i = 0; i < 3; i++) {
    ++it1;
}
list<int> l2(it1, l1.end());
```

- Možemo i malo jednostavnije:

```
list<int> l1({ 11, 22, 33, 44, 55 });
auto it1 = l1.begin();
advance(it1, 3);
list<int> l2(it1, l1.end());
```


Veličina povezane liste

- Povezana lista nema pojam kapaciteta
  - Memorija za element se alocira u trenutku kada je potrebna
- Povezana lista ima veličinu
  - `l.size()` vraća broj elemenata stavljenih u povezanu listu
    - Zbog performansi ne postoji na `forward_list<T>`
    - „Not providing size() is more consistent with the goal of zero overhead ... Maintaining a count doubles the size of a forward_list object (one word for the list head and one for the count), and it slows down every operation that changes the number of nodes.”

Preuzeto s: www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2543.htm

Ručna promjena veličine povezane liste

- Veličinu možemo i eksplicitno mijenjati:
  - `l.resize(n, val);`
    - Mijenja veličinu povezane liste na točno *n* elemenata
    - Ako je *n* manji od trenutne veličine, odbacuju se elementi s kraja
      - Uništavaju se
    - Ako je *n* veći od trenutne veličine, dodaju se elementi na kraj
      - Opcionalno, možemo reći koja je vrijednost `val` dodanih elemenata

Modifikatori povezane liste (1/4)

- Povezana lista nudi sljedeće modifikatore:
  - `l.assign()` je sličan konstruktorima

```
l1.assign(7, 100);
l2.assign(it1, it2);
l3.assign({ 11, 22, 33 });
```

    - Svi elementi prethodno sadržani u povezanoj listi se uništavaju
  - `l.push_front(val)`
    - Dodaje kopiju od *val* na početak
  - `l.emplace_front(arg1, arg2, …)`
    - Konstruira objekt na početku
  - `l.pop_front()`
    - Briše i uništava prvi element s početka

Modifikatori povezane liste (2/4)

- `l.clear()` kompletno prazni povezanu listu i uništava sve elemente
- `list<T>` dodatno nudi i sljedećih šest ekskluzivnih metoda:
  - `l.insert()` umeće jedan ili više elemenata na zadanu poziciju:
    - Prvi parametar je pozicija, ostali su slični parametrima u konstruktorima:

```
l.insert(it, 99);
l.insert(it, 10, 99);
l.insert(it, { 11, 22, 33 });
```

  - `l.emplace(it, arg1, arg2, …)`
    - Konstruira objekt na zadanoj poziciji
  - `l.erase()` uklanja jedan ili više elemenata:

```
l.erase(it);
l.erase(it1, it2);
```

ALGEBRA BERNAYS SVEUČILIŠTE

17

Modifikatori povezane liste (3/4)

- `l.push_back(val)`
  - Dodaje kopiju od *val* na kraj
- `l.emplace_back(arg1, arg2, …)`
  - Konstruira objekt na kraju
- `l.pop_back()`
  - Briše i uništava prvi element s kraja

ALGEBRA BERNAYS SVEUČILIŠTE

18

Modifikatori povezane liste (4/4)

- `forward_list<T>` dodatno nudi i sljedeće tri ekskluzivne metode:
  - `l.insert_after(it, val)`
    - Dodaje kopiju od *val* na poziciju iza *it*
  - `l.emplace_after(it, arg1, arg2, …)`
    - Konstruira objekt na poziciji iza *it*
  - `l.erase_after(it)`
    - Briše i uništava element iza *it*

ALGEBRA BERNAYS SVEUČILIŠTE

19

Primjer

```
list<int> l = { 11, 22, 33, 44, 55 };
auto it1 = l.begin();
advance(it1, 3);
l.insert(it1, 999);

// ispiši…

forward_list<int> fl = { 11, 22, 33, 44, 55 };
auto it2 = fl.begin();
advance(it2, 3);
fl.insert_after(it2, 999);

// ispiši…
```

ALGEBRA BERNAYS SVEUČILIŠTE

20

Ostale važnije metode

- `l.empty()` vraća je li povezana lista prazna ili ne
- `l.remove(val)` uklanja i uništava sve elemente koji imaju vrijednost jednaku *val*
- `l.remove_if(predikat)` uklanja i uništava sve elemente za koje funkcija `predikat` vrati `true`
  - Funkcija prima vrijednost iz liste, a vraća `true/false`
- `l.reverse()` preslaguje elemente od kraja prema početku

Zadatak

- Napišimo vlastitu implementaciju jednostruko povezane liste cijelih brojeva koja omogućuje:
  - Izradu prazne liste
  - Ubacivanje na početak liste
  - *izradu iteratora koji omogućuju pristup elementima

Što želimo

```
int main() {
    MojaLista l;
    l.push_front(11);
    l.push_front(22);
    l.push_front(33);

    for (MojaLista::iterator it = l.begin(); it != l.end(); ++it) {
        cout << *it << endl;
    }

    return 0;
}
```


23

MojaLista.h

```
struct Cvor {
    int podatak;
    Cvor* sljedeci;
};

class MojaLista {
private:
    Cvor* front;
public:
    MojaLista();
    ~MojaLista();
    void push_front(int broj);

    class iterator {
    private:
        Cvor* curr;
    public:
        iterator(Cvor* c);
        iterator& operator++();
        bool operator!=(const iterator& rhs) const;
        int& operator*() const;
    };

    iterator begin();
    iterator end();
```


24

MojaLista.cpp (1/2)

```
MojaLista::MojaLista() {
    front = nullptr;
}

MojaLista::~MojaLista() {
    Cvor* curr = front;
    while (curr != nullptr) {
        Cvor* tmp = curr->sljedeci;
        delete curr;
        curr = tmp;
    }
}

void MojaLista::push_front(int broj) {
    Cvor* c = new Cvor;
    c->podatak = broj;

    c->sljedeci = front;

    front = c;
}
```

ALGEBRA BERNAYS SVEUČILIŠTE

25

MojaLista.cpp (2/2)

```
MojaLista::iterator::iterator(Cvor* c) {
    curr = c;
}

MojaLista::iterator& MojaLista::iterator::operator++() {
    curr = curr->sljedeci;
    return *this;
}

int& MojaLista::iterator::operator*() const {
    return curr->podatak;
}

bool MojaLista::iterator::operator!=(const iterator& rhs) const {
    return curr != rhs.curr;
}

MojaLista::iterator MojaLista::begin() {
    return MojaLista::iterator(front);
}

MojaLista::iterator MojaLista::end() {
    return MojaLista::iterator(nullptr);
}
```

ALGEBRA BERNAYS SVEUČILIŠTE

26

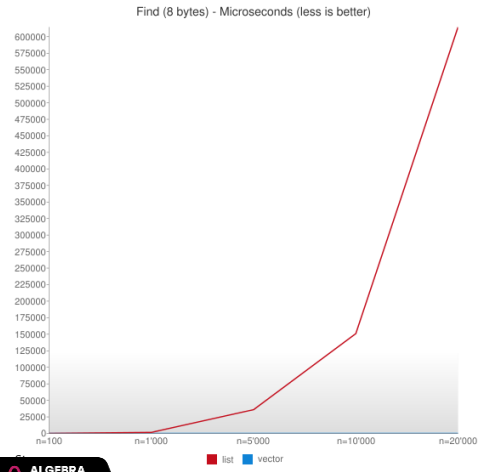
PERFORMANSE POVEZANE LISTE

Složenost nekih operacija

Metoda	Složenost	Metoda	Složenost
list<T> l;	O(1)	l.erase(iterator);	O(1)
list<T> l(it1, it2);	O(n)	l.erase(begin, end);	O(1)
l.size();	O(1)	l.remove(value);	O(n)
l.empty();	O(1)	l.remove_if(test);	O(n)
l.begin();	O(1)	l.reverse();	O(n)
l.end();	O(1)	l.sort();	O(n log n)
l.front();	O(1)	l.sort(comparison);	O(n log n)
l.back();	O(1)	l.merge(l2);	O(n)
l.push_front(value);	O(1)		
l.push_back(value);	O(1)		
l.insert(iterator, value);	O(1)		
l.pop_front();	O(1)		
l.pop_back();	O(1)		

Traženje slučajne vrijednosti

- Svaki kontejner sadrži nesortirane brojeve [0, N]
  - Nakon toga se svaki od brojeva linearno traži u kontejneru



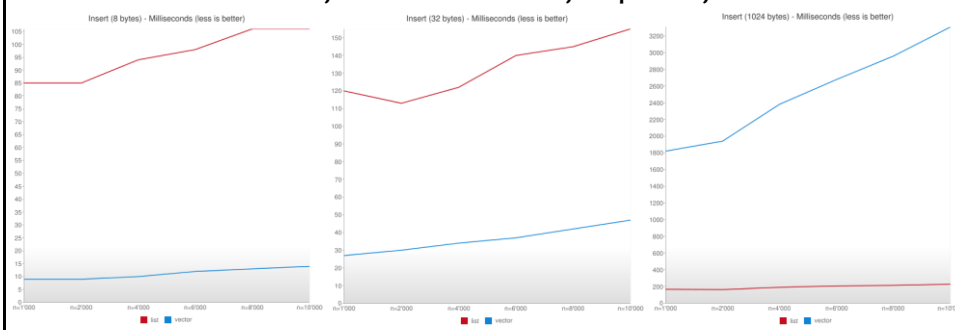
- Razlog: *cache*!
  - *Cache* je nekoliko redova veličine brži od RAM-a
  - Podaci u vektoru su jedan iza drugoga, dohvaćanjem prvog se odjednom dohvaća ogroman dio vektora u *cache*
  - Kod liste procesor većinu vremena čeka na prijenos RAM => *cache*

Preuzeto s: dzone.com/articles/c-benchmark-%E2%80%93-stdvector-vs

29

Umetanje na slučajno odabrano mjesto

- Umećemo 1000 vrijednosti na slučajne pozicije

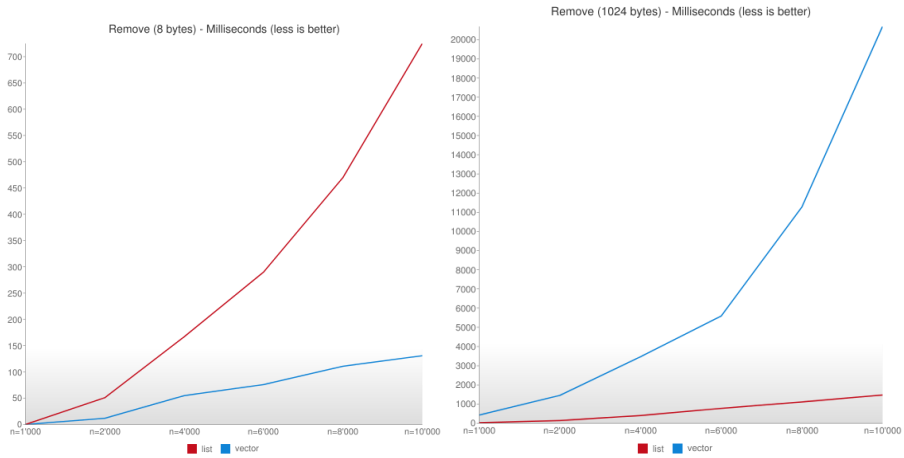


- Pronalazak pozicije u listi je sporiji od pomicanja puno malenih elemenata u desno (ako znamo poziciju, lista je brža)
- Povećanjem veličine elemenata performanse vektora drastično opadaju, a liste ostaju otprilike jednake

30

Brisanje sa slučajno odabranog mjesta

- U teoriji jednako umetanju



- Jednako objašnjenju kod umetanja

ALGEBRA BERNAYS SVEUČILIŠTE

31

Kada koristiti koji kontejner

- Teoretski savjeti:
  - Za linearno pretraživanje => vektor
  - Za umetanje/brisanje malih podataka => vektor
  - Za umetanje/brisanje velikih podataka pri kraju => vektor
  - Za umetanje/brisanje velikih podataka na početku => lista
- Praktični savjeti:
  - Krenite s vektorom
  - Testirajte performanse
  - Ako su nedovoljne, probajte s listom
- Bjarne Stroustrup: „Vectors are always better than lists”

ALGEBRA BERNAYS SVEUČILIŠTE

34

Dodatni materijali

- Dodatni materijali su dostupni na:
  - Linked lists
    - https://youtu.be/RicuM4AWT68
  - Implementing your own linked list
    - https://youtu.be/AgcXGKtKbwo