ALGEBRA

STRUKTURE PODATAKA I ALGORITMI

Predavanje 06

Ishod 2

1

Homework 01

- https://tinyurl.com/3875uzzy

ALGEBRA

2

ADT STOG

ALGEBRA

3

ADT stog

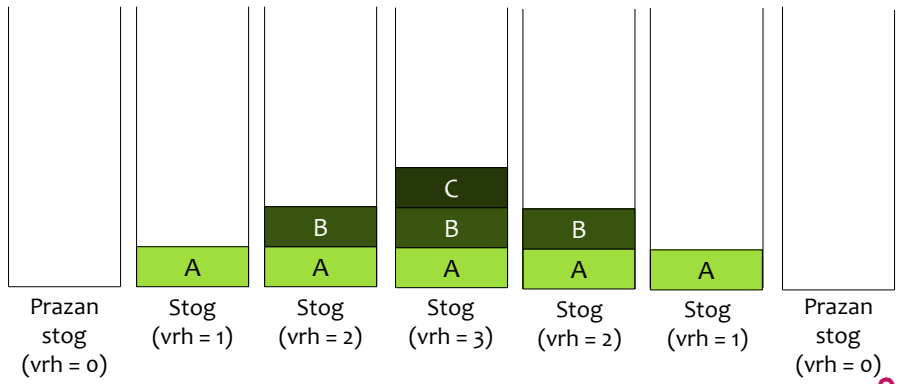
- ADT stog (engl. *stack*) je lista ili povezana lista uz ograničenje:
  - Sva umetanja i sva vađenja elemenata se rade na jednom kraju liste koji se onda naziv vrh (engl. *top*)
- Drugi nazivi su:
  - LIFO (engl. *Last In First Out*) lista
  - FILO (engl. *First In Last Out*) lista

ALGEBRA

4

LIFO/FILO princip

- Svaki stog ima vrh – to je posljednji element dodan na stog i prvi koji će biti skinut sa stoga



ALGEBRA

5

Operacije

- ADT stog obično definira sljedeće operacije:
  - Stavljanje novog elementa na vrh stoga (engl. *push*) u $O(1)$
    - Ako na stogu nema mjesta => *overflow*
  - Uklanjanje elementa s vrha stoga (engl. *pop*) u $O(1)$
    - Ako je stog prazan => *underflow*
  - Čitanje vrijednosti elementa s vrha stoga u $O(1)$
  - Dohvat broja elemenata na stogu u $O(1)$
  - Provjera je li stog prazan u $O(1)$

ALGEBRA

6

Osnovni primjeri primjene stoga

- Izvođenje programa
  - Svaki poziv funkcije stavlja *stack frame* na sistemski stog
  - Povratkom iz funkcije se uklanja *stack frame* sa sistemskog stoga
- Prepisivanje podataka iz datoteke u obrnutom redoslijedu
  - Čitamo podatke iz jedne datoteke i punimo ih na stog, zatim praznimo stog i pišemo podatke u drugu datoteku
- UNDO funkcionalnost
  - Svaka operacija se stavlja na stog
  - Zadavanjem naredbe UNDO uklanjamo zadnji element sa stoga

7

Napredniji primjeri primjene stoga

- Izračunavanje matematičkih izraza u reverznoj poljskoj notaciji (RPN)
- Parsiranje blokova kôda od strane kompajlera
- Pronalazak puta kroz labirint (primjena *backtrackinga*)
- Povijest posjećenih stranica u web pregledniku
- Provjera usklađenosti tagova u HTML-u i XML-u

8

ADAPTER

9

Uvod

- Adapterom nazivamo klasu A koja sadrži varijablu koja je po tipu neka druga klasa B
  - Kažemo da klasa A adaptira klasu B našim potrebama
  - Često se naziva i omotačem (engl. *wrapper*)
- Klasa A obično korisniku pruža korisnije/jednostavnije sučelje od klase B
- Klasa A u svojim metodama u stvari poziva metode klase B
- Više detalja o adapterima: NRAKO (Napredni razvoj aplikacija korištenjem obrazaca)

10

Primjer (1/4)

- Zamislimo da imamo klasu koja zna raditi s raznim Internet resursima:

```
class HttpUtilities {
public:
    string http_get(string url, int port, bool is_secure);
    string http_post(string url, int port, bool is_secure);
    string http_put(string url, int port, bool is_secure);
    string http_delete(string url, int port, bool is_secure);
    void set_credentials(string username, string password);
    void ftp_upload(char* file_data, int port);
    char* ftp_download(string url, int port);
    void make_tweet(string hashtag, string message);
    void make_facebook_post(string message);
    …
};
```


Primjer (2/4)

- Korištenje klase nije jednostavno i podložno je greškama:

```
HttpUtilities http;
http.set_credentials("sa", "SQL");
char* bytes = http.ftp_download("ftp://bla.com", 21);
string res = convert_bytes_to_text(bytes);
cout << res << endl;
```


Primjer (3/4)

- Međutim, sve što nama treba je dohvat tekstualnih datoteka s FTP-a
- Radimo novu klasu kojom adaptiramo prethodnu klasu našim potrebama:

```
class FtpTextDownloader {
private:
    HttpUtilities http;
public:
    string download_text(string username, string password,
                                        string url, int port) {
        http.set_credentials(username, password);
        char* bytes = http.ftp_download(url, port);
        string res = convert_bytes_to_text(bytes);
        return res;
    }
};
```


Primjer (4/4)

- Korištenje je znatno pojednostavljeno:

```
FtpTextDownloader x;
cout << x.download_text("sa", "SQL", "ftp://bla.com", 21);
```


STOG

15

Konkretni ADT stog

- ADT stog je u programskom jeziku C++ implementiran kao generička klasa `stack<T>`
- Klasa `stack<T>` je kontejnerski adapter, što znači sljedeće:
  - U sebi sadrži drugi kontejner koji mora implementirati barem sljedeće metode:
    - empty
    - size
    - back
    - push_back
    - pop_back
  - Klasa `stack<T>` pruža korisniku sučelje stoga, a u pozadini izvršava odgovarajuće operacije na sadržanom kontejneru
    - Klasa `stack<T>` adaptira drugi kontejner potrebama korisnika

16

Sadržani kontejneri u klasi `stack<T>`

- Da bi neki kontejner mogao biti sadržani kontejner u klasi `stack<T>`, mora implementirati barem spomenute metode
- U standardnom C++-u, sljedeće klase mogu biti sadržani kontejneri:
  - `vector<T>`
  - `list<T>`
  - `deque<T>`
  - Bilo koja naša klasa koja implementira tražene metode

Zadatak

- Implementirajte svoj stog cijelih brojeva tako da u pozadini koristi vektor (adaptiramo vektor našoj potrebi). Omogućite sljedeće:

```
int main() {
   MojStog s;
   s.push(11);
   s.push(22);
   s.push(33);

   while (!s.empty()) {
      cout << s.top() << endl;
      s.pop();
   }

   return 0;
}
```


MojStog.h

```
class MojStog {
private:
   vector<int> v;
public:
   MojStog();
   void push(int val);
   bool empty();
   int& top();
   void pop();
};
```


19

MojStog.cpp

```
MojStog::MojStog() {
}

void MojStog::push(int val) {
   v.push_back(val);
}

bool MojStog::empty() {
   return v.empty();
}

int& MojStog::top() {
   return v.back();
}

void MojStog::pop() {
   v.pop_back();
}
```


20

Izrada i uništavanje stoga

- Postoji tri osnovna načina izrade stoga:
  - `stack<int> jedan;`
    - Kreira prazni stog sa sadržanim kontejnerom tipa `deque<int>`
  - `stack<int, vector<int>> dva;`
    - Kreira prazni stog sa sadržanim kontejnerom tipa `vector<int>`
  -
```
vector<int> v({ 11, 22, 33 });
stack<int, vector<int>> tri(v);
```
    - Kreira prazni stog sa sadržanim kontejnerom tipa `vector<int>`, a uz to i kopira sve elemente iz vektora v
- U svim načinima izrade vektor možemo zamijeniti s `list<T>`, `deque<T>` ili nekom našom klasom koja sadrži tražene metode

Osnovne operacije sa stogom

- `s.push(val)` stavlja kopiju od `val` na vrh
- `s.emplace(arg1, arg2, …)` kreira novi objekt na vrhu
- `s.top()` vraća referencu na element na vrhu
- `s.pop()` uklanja i uništava element na vrhu
- `s.empty()` vraća je li stog prazan
- `s.size()` vraća broj elemenata na stogu
- Stog nema iteratora
  - Jedini način da dohvatimo sve elemente je da uništimo sadržaj stoga

ADT RED

ALGEBRA

23

ADT red

- ADT red (engl. *queue*) je lista ili povezana lista uz ograničenja:
  - Sva umetanja elemenata se rade na jednom kraju liste koji se onda naziv ulaz (engl. *rear*)
  - Sva vađenja elemenata se rade na drugom kraju liste koji se onda naziv izlaz (engl. *front*)
- Drugi nazivi su:
  - LILO (engl. *Last In Last Out*) lista
  - FIFO (engl. *First In First Out*) lista

ALGEBRA

24

Operacije

- ADT red obično definira sljedeće operacije:
  - Stavljanje novog elementa na ulaz (engl. *push*/*enqueue*) u $O(1)$
    - Ako u redu nema mjesta => *overflow*
  - Uklanjanje elementa s izlaza (engl. *pop*/*dequeue*) u $O(1)$
    - Ako je red prazan => *underflow*
  - Čitanje vrijednosti elementa koji se nalazi neposredno na izlazu u $O(1)$
  - Dohvat broja elemenata u redu u $O(1)$
  - Provjera je li red prazan u $O(1)$

ALGEBRA

Primjene reda

- Redovi se najčešće koriste u višedretvenom programiranju
  - Jedna ili nekoliko dretvi (proizvođači) ubacuju elemente na jednu stranu
  - Jedna ili više dretvi (konzumenti) vade elemente s druge strane
  - Prekompleksno za ovaj kolegij (OOP.NET, Java 2, IIS, …)
- Jednodretveno programiranje rjeđe koristi redove
  - Razlog: ista dretva je i na ulazu i na izlazu
  - Jedna primjena bi bila da program spremi neke vrijednosti, a onda ih kasnije koristi, uz garanciju da će ih uzeti onim redom kojim ih je spremio

ALGEBRA

RED

Konkretni ADT red

- ADT red je u programskom jeziku C++ implementiran kao generička klasa queue<T>
  - U Javi je to Queue<T>, u C# je to Queue<T>, ...
- I klasa queue<T> je kontejnerski adapter
  - Sadržani kontejner mora implementirati barem sljedeće metode:
    - empty
    - size
    - front
    - back
    - push_back
    - pop_front

Sadržani kontejneri u klasi queue<T>

- Da bi neki kontejner mogao biti sadržani kontejner u klasi `queue<T>`, mora implementirati barem spomenute metode
- U standardnom C++-u, sljedeće klase mogu biti sadržani kontejneri:
  - `list<T>`
  - `deque<T>`
  - Bilo koja naša klasa koja implementira tražene metode
- Zašto bi `vector<T>` bio loš izbor za sadržani kontejner?

Zadatak

- Ako imamo `queue<T>` koji kao sadržani kontejner koristi `list<T>`, opišite kako rade sljedeće metode definirane na `queue<T>`:
  - empty
  - size
  - front
  - back
  - push
  - pop

Izrada i uništavanje reda

- Postoji tri osnovna načina izrade reda:
  - `queue<int> jedan;`
    - Kreira prazni red s `deque<int>` sadržanim kontejnerom
  - `queue<int, list<int>> dva;`
    - Kreira prazni red sa `list<int>` sadržanim kontejnerom
  - ```
    list<int> l = { 11, 22, 33 };
    queue<int, list<int>> tri(l);
    ```
    - Kreira red s kopijom liste `l` kao sadržanim kontejnerom
- U svim načinima izrade listu možemo zamijeniti sa `deque<T>` ili našom klasom koja sadrži tražene metode

Osnovne operacije s redom

- `q.push(val)` dodaje kopiju od `val` na kraj
- `q.emplace(arg1, arg2, …)` kreira novi objekt na kraju
- `q.front()` vraća referencu na element na izlazu
- `q.pop()` uklanja i uništava element na izlazu
- `q.back()` vraća referencu na element na ulazu
- `q.empty()` vraća je li red prazan
- `q.size()` vraća broj elemenata u redu
- Red nema iteratora
  - Jedini način da dohvatimo sve elemente je da uništimo sadržaj reda


    ```

Vizualizacija algoritama

- Načine rada raznih struktura i algoritama možete naći na:
  - www.cs.usfca.edu/~galles/visualization/Algorithms.html

Zadatak

1. Korištenjem najprikladnije strukture prepišite sadržaj neke datoteke u drugu datoteku, ali obrnutim redoslijedom.

Rješenje – punjenje stoga

```
stack<string> s;

ifstream dat1("Source.cpp");
if (!dat1) {
    cout << "Greska pri otvaranju datoteke za citanje" << endl;
    return 1;
}

string line;
while (getline(dat1, line)) {
    s.push(line);
}

dat1.close();
```


Rješenje – pražnjenje stoga

```
ofstream dat2("Source_reversed.cpp");
if (!dat2) {
    cout << "Greska pri otvaranju datoteke za pisanje" << endl;
    return 1;
}

while (!s.empty()) {
    dat2 << s.top() << endl;
    s.pop();
}

dat2.close();
```


Zadatak

2. U prilogu su tri datoteke, dvije neispravne i jedna ispravna. Napišimo program koji će provjeravati je li neka C++ datoteka balansirana po pitanju zagrada ‘{‘, ‘[‘ i ‘(‘. Algoritam je sljedeći:
   a. Prođimo datoteku znak po znak.
   b. Ako je znak otvarajuća zagrada, gurnimo ga na stog.
   c. Ako je znak zatvarajuća zagrada, skinimo znak sa stoga i usporedimo jesu li jednaki. Ako nisu, greška i kraj.
   d. Ako na kraju ostane išta na stogu, greška i kraj.

Rješenje

```
stack<char> s;
string line;
int no = 1;
while (getline(dat, line)) {
    for (unsigned i = 0; i < line.length(); i++) {
        if (line[i] == '{' || line[i] == '[' || line[i] == '(') {
            s.push(line[i]);
        }
        else if (line[i] == '}') {
            if (s.empty() || s.top() != '{') {
                cout << "Greska u liniji " << no << ": " << line;
                return 0;
            }
            s.pop();
        }
        …
    }
}
if (s.empty()) cout << "Datoteka je ispravno balansirana" << endl;
else cout << "Datoteka ima viska i nije balansirana" << endl;
```


Za sljedeće predavanje

1. Napišite program koji može provjeriti je li matematički izraz ispravan po pitanju zagrada. Primjerice, sljedeći izraz je ispravan:

   (2 * (7 – 5)) / 3

   Dok ovaj nije:

   (2 * (7 – 5)) / 3)
2. Pročitati cplusplus.com/reference/stack/stack
3. Pročitati cplusplus.com/reference/queue/queue

ALGEBRA

Dodatni materijali

- Dodatni materijali su dostupni na:
  - Adapters
    - https://youtu.be/afOHWB1U9Ko
  - Stacks
    - https://youtu.be/q7EMKYvRhBg
  - Queues
    - https://youtu.be/GED57KdBMkY

ALGEBRA