Izvedba i primjene strukture podataka stog

Kačić, Filip

Sveučilišni prijediplomski studij Informatika

Filip Kačić

Izvedba i primjene strukture podataka stog

Završni rad

Mentor: Prof. dr. sc. Maja Matetić

Rijeka, veljača 2024. godine

Rijeka, rujan 2023. godine

Zadatak za završni rad

Pristupnik:Filip Kačić

Naziv završnog rada:Izvedba i primjene strukture podataka stog

Naziv završnog rada na engleskom jeziku:Implementation and application of stack data structure

Sadržaj zadatka:

Zadatak završnog rada je dati teorijski prikaz i definiciju strukture podataka stog, te detaljno opisati i objasniti njegovu izvedbu. Potrebno je navesti prednosti i nedostatke stoga,opisati kako se i kada se koristi, te ilustrirati primjere primjene stoga.

Mentor

Prof. dr. sc. Maja Matetić

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 12. travnja 2023.

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel:**+385(0)51584700**
E-mail:**ured@inf.uniri.hr**
OIB:**64218323816**
IBAN:**HR1524020061400006966**

Sažetak i ključne riječi

U ovom završnom radu obradit će se struktura podataka stog (eng. *stack data structure*), njegova izvedba i primjena. U uvodnom dijelu bit će riječ o tome što je informatika, što su strukture podataka, pa s time i što je stog i čemu služi te ukratko o njegovom podrijetlu. U razradi bit će detaljno definiran zajedno s operacijama koje se izvode nad njime, bit će navedeno kako se programira, gdje i kada se implementira i u kojim algoritmima mu je mjesto, a u zaključku će se ponoviti i zaokružiti cijela poanta rada.

Rad koristi slike iz stručnih knjiga i iz internetskih izvora radi lakšeg razumijevanja i vizualnog aspekta učenja. Također su navedeni primjeri te definirani jednostavni i složeni pojmovi. Napisan je kao završni rad preddiplomskog studija informatike, a može poslužiti kao uvod osobama kojima je ova tema nova ili kao podsjetnik za bitne osnove informatičarima i programerima svih razina.

KLJUČNE RIJEČI:

- struktura podataka
- stog
- programski jezik C++

1 Uvod

Cilj ovog završnog rada je objasniti strukturu podataka stog, ali prije definicije samog stoga i svega vezanog, bit će definirani neki općenitiji pojmovi koji vode do njega specifično. Takav način procesiranja, objašnjavanja informacija i slaganja znanja se zove dizajn od gore prema dolje (eng. *top-down design*) [1] i tako je osmišljen ovaj rad.

Informatika ima korijen u starofrancuskoj riječi za informaciju – *informacion*, koja znači savjet ili instrukcija, ili u latinskoj riječi *informatio* što znači obris, koncept ili ideja. Informacija je riječ koja se odnosila na više toga kroz povijest: na televizijske signale (1937. godine); na operativne sustave s bušenim karticama (1944. godine); na DNK (1953. godine). [2]

Informatika je znanost koja se bavi prikupljanjem, klasificiranjem, pohranjivanjem, pronalaženjem i širenjem zabilježenog znanja. [3] Danas je usko vezana uz proučavanje računalnih sustava.

Jedan od načina bilježenja znanja u računalu su strukture podataka. [4] Proučavanje struktura podataka služi kao temelj na kojemu se grade mnoga polja informatike. Nešto znanja o strukturama podataka je neophodno za osobe koji žele raditi na dizajnu, implementaciji, testiranju ili održavanju gotovo bilo kojeg softverskog sustava. [5] One su bitne da se podatcima u računalu može učinkovito pristupiti te da se mogu učinkovito mijenjati. [3]

Jedna od najjednostavnijih i najosnovnijih struktura podataka je upravo tema ovog rada – stog (eng. *stack*). Gotovo svaki značajan program će eksplicitno koristiti barem jednu od takvih struktura, a stog je uvijek implicitno korišten u programu, bez obzira na to je li deklariran ili ne. [4] Zato je bitno razumjeti stog.

Podrijetlo stoga kao strukture podataka u računalnim znanostima je nejasno jer su odgovarajući pojmovi već postojali u matematici i u poslovnim praksama s papirima prije uvođenja digitalnih računala. Ipak, D. E. Knuth, američki računalni znanstvenik i matematičar, citira A. M. Turing za razvoj stogova za povezivanje potprograma 1947. godine. [6]

2 Razrada

2.1 Apstraktni tip podataka (ADT) u C++ programskom jeziku

Stog je apstraktan tip podataka (eng. *abstract data type*) ili ATP (eng. *ADT*) što znači da je definiran kao skup objekata sa zajedničkim skupom operacija. ATP-ovi su apstrakcije, ali ne bilokakve, već matematičke, jer u njihovoj definiciji nije objašnjeno kako se njihov skup operacija implementira već samo da postoji.

Primjeri apstraktnih tipova podataka su programibilni objekti kao što su liste, setovi, grafovi (s njihovim operacijama), dok su *integer*-i, *real*-i te *boolean*-i samo, obični tipovi podataka. Ti, obični tipovi podataka imaju operacije koje su asocirane s njima, pa isto tako imaju i apstraktni tipovi podataka. ATP-ovi mogu imati operacije poput *add*, *remove*, *size*, i, *contains*.

Programski jezik C++ je objektno orijentirani potomak programskog jezika C-a. Raširen je u industriji i akademskoj zajednici kao izvrstan programski jezik jer je prirodan za uvođenje u strukture podataka i koristan na više razina, pa i na početničkoj. [5] Zato je opravdan kao programski jezik za primjere koda u razradi ovoga rada.

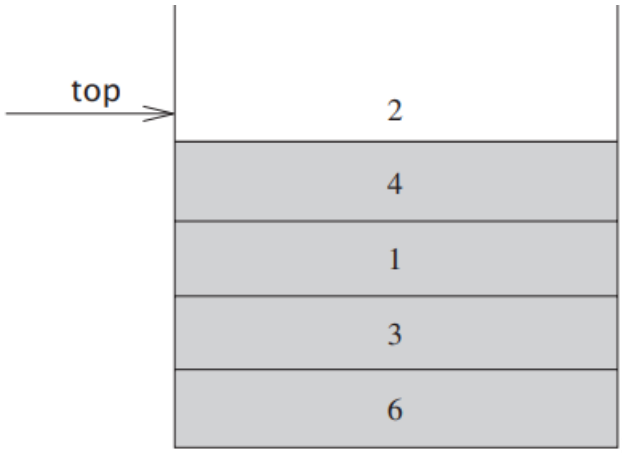
C++ u svojim klasama dozvoljava implementaciju apstraktnih tipova podataka sa skrivenim detaljima o tome kako se točno implementiraju. Bilo koji program pisanom u C++-u, u kojemu je potrebno napraviti operaciju nad ATP-om, može to napraviti jednostavno pozivajući prikladnu metodu.

Potrebno je napomenuti kako nema pravila koje operacije koji ATP mora podržavati, već je to dizajnerska odluka. [7]

2.2 Definicija

Stog je lista s ograničenjem da se operacije umetanja i brisanja mogu odvijati samo na jednoj poziciji zvanoj vrh (eng. *top*), koja je zapravo kraj liste (promatrajući od dolje prema gore).

Na slici 1 je primjer stoga nakon nekoliko operacije izvršenih nad njime. Generalni model je da je neki element na vrhu stoga i da je samo on vidljiv.



Slika 1. Primjer stoga

Stogovi su također poznati kao *LIFO* (eng. *Last In First Out*) ili *FILO* (eng. *First In Last Out)* liste. *LIFO* implicira da element koji je zadnji umetnut u listu izlazi prvi, a *FILO* implicira da element koji je prvi umetnut izlazi zadnji. [4]

Slično, u strukturi podataka red (eng. *array*), postoji redoslijed *FIFO* (eng. *First In Last Out*) prema kojemu je element koji se briše uvijek onaj koji je u listi najdulje. [6]

Postoje mnogi primjeri stoga u stvarnome svijetu, na primjer, tanjuri naslagani jedan na drugome u kantini. Tanjur koji je postavljen na vrhu je prvi koji se uklanja, a tanjur koji je postavljen na dno tamo najdulje ostaje. Redoslijed u kojemu su tanjuri postavljeni jedan na drugoga je obrnut od redoslijeda kojim će se micati (zato što se može utjecati samo na najgornji). Dakle, može se jasno vidjeti da primjer slijedi *LIFO/FILO* poredak, odnosno da je stog. [3]

Još jedan dobar primjer stoga u stvarnome svijetu je stalak za *CD*-ove za koji vrijedi sve isto kao i za tanjure u kantini.

2.3 Operacije stoga i vremenska složenost

Osnovne operacije na stogu su *push*, koja je ekvivalentna umetanju, i *pop*, koja briše najnedavnije umetnut element. Njihova imena su aluzije na fizičke stogove u stvarnom svijetu.

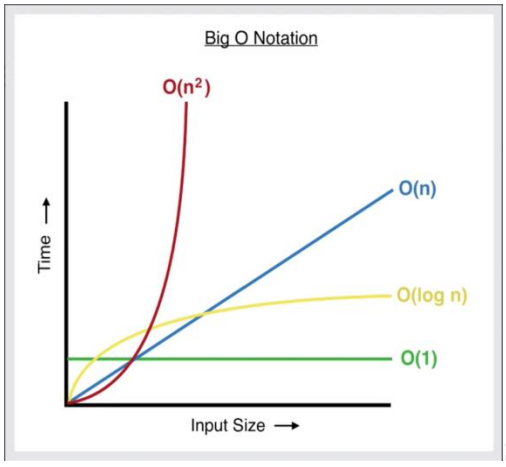
Najnedavnije umetnut element se može provjeriti, tj. vratiti, operacijom *top*. Ima li stog uopće elemenata se provjerava operacijom *isEmpty*, a koliko ima elemenata se provjeravaju operacijom *size*.

Dakle, najčešće operacije nad stogom su:

- ***push*** za umetanje elementa u stog
- ***pop*** za uklanjanje elementa iz stoga
- *top* vraća najgornji element stoga
- *isEmpty* vraća istinu (eng. *true*) ako je stog prazan, a u svakom drugom slučaju laž (eng. *false*)
- *size* vraća veličinu stoga

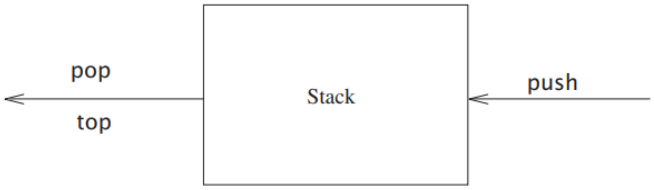
Vremenska složenost svih ti operacija je O(1).

Na slici 2 se vide funkcije velike O (eng. *Big O*) notacije koja opisuje gornju granicu složenosti funkcije, drugim riječima, najgori slučaj programa koji tako često zanima informatičare i programere. X-os predstavlja količinu ulaznih podataka, a y-os vrijeme. O(1) vremenska složenost je prikazana zelenom funkcijom. Vidi se da je O(1) odlična vremenska složenost za veliku količinu ulaznih podataka (eng. *Big Data*).



Slika 2. Veliko O notacija [8]

Model prikazan na slici 3 označava da su *push* operacije ulazne, a *pop* i *top* operacije izlazne. Sve najčešće operacije nad stogom su dio repertoara, ali, esencijalno, sve što se može sa stogom su operacije *push* i *pop*, one su osnovne i najbitnije.



Slika 3. Operacije *pop*, *top* i *push* [6]

2.4 Izvedba operacija nad stogom

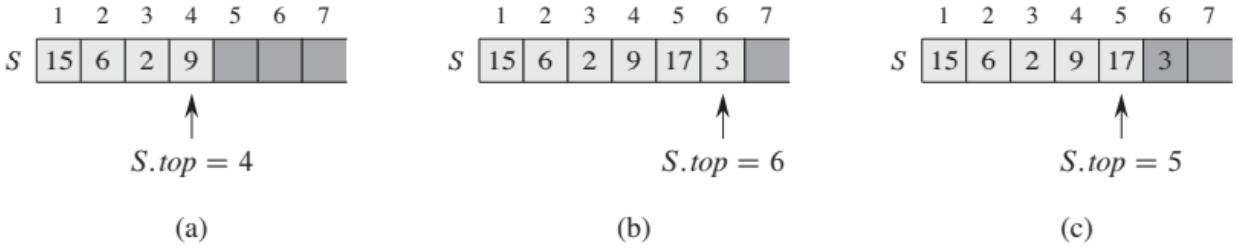
Dvije osnovne operacije na stogu se izvode pomoću nekoliko linija koda. Pseudokod je prikazan na slici 4.

```
PUSH(S, x)
1  S.top = S.top + 1
2  S[S.top] = x

POP(S)
1  if STACK-EMPTY(S)
2        error "underflow"
3  else S.top = S.top − 1
4        return S[S.top + 1]
```

Slika 4. Pseudokod *push* i *pop* operacija [6]

Slika 5 prikazuje implementaciju stoga S pomoću nekog niza. Elementi stoga se pojavljuju samo u blago osjenčanim pozicijama.

a) Stog S ima 4 elementa. Gornji element je 9.
b) Stog S nakon poziva PUSH(S, 17) i PUSH(S, 3).
c) Stog S nakon što je poziv POP(S) je vratio element 3, koji je najveći nedavno gurnuti element. Iako se element 3 i dalje pojavljuje u nizu, on više nije u stogu; na vrhu je element 17.



Slika 5. Stog implementiran pomoću niza (tri koraka)

Kada je S.top = 0, tada stog ne sadrži elemente, prazan je. Možemo testirati to s operacijom STACK-EMPTY() čiji je kod napisan na slici 6. Ako pokušamo izvesti operaciju *pop* na prazan stog, kaže se da stog *underflow*-a, što je inače greška. A ako S.top bude veći od broja elemenata u listi, onda stog *overflow*-a.

```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE
```

Slika 6. Pseudokod *isEmpty* operacije [6]

2.5 Implementacija stoga, prednosti i nedostatci

S obzirom na to da je stog jednostavna, linearna struktura podataka, zapravo lista, bilo koja implementacija liste bit će prikladna u velikoj većini situacija. Može se implementirati pomoću povezane liste, ali najčešće se implementira pomoću reda (eng. *array*) tako da se koriste vektori. [4] Povezana lista je struktura podataka u kojoj su objekti raspoređeni linearnim redoslijedom, ali za razliku od polja u kojima je linearni poredak određen indeksima polja, redoslijed u povezanom popisu je određen pokazivačima (eng. *pointers*). [6]

Upravo jedna od najvećih prednosti stoga je što ga je lako implementirati i što su njegove operaciju vrlo jednostavne za razumjeti kao što se moglo vidjeti u prošlom i pretprošlom poglavlju.

Stog koristi kontinuirani blok memorije što ga čini efikasnijim u korištenju memorije u usporedbi s drugim strukturama podataka, što može biti jedan od razloga zašto ga implementirati u programu.

Isto tako, brzo dodavanje i micanje elemenata s vrha stoga je jedna od njegovih prednosti.

Stog se koristi u dizajnu kompajlera za parsiranje (proces prevođenja izvornog programa) i općenito u sintaksnoj analizi. Također omogućuje *undo* i *redo* operacije u različitim aplikacijama kao što su uređivači tekstova i grafički alati. O primjeni stoga će se više govoriti u sljedećem poglavlju, ali je u ovom poglavlju bilo bitno naglasiti da se implementira u tako poznatim i čestim konceptima.

Kao i svaka struktura podataka, stog, uz svoje prednosti, ima i svoje nedostatke. Ograničen kapacitet koji može imati samo fiksan, određen broj elemenata je upravo jedan od njih. Ako stog postane pun - ako mu je kapacitet podataka koje može pohraniti popunjen, *push*-anje, tj. dodavanje, novih elemenata uzrokuje *overflow*, što vodi do gubitka podataka, a ako je pak previše elemenata *pop*-ano, tj. maknuto iz stoga, to može voditi do *underflow*-a.

(Po *overflow*-u stoga je dobila ime internetska stranica „*Stack Overflow*“ koja je bogata primjerima programskim kodova i rješenjima programskim problema.)

Također, učestalo dodavanje i micanje elemenata iz stoga može dovesti do toga da se memorija fragmentira što onemogućuje njeno efikasno korištenje.

Bitno je zapamtiti, zbog toga kako je stog definiran da se njegovim elementima pristupa isključivo s njegovog vrha, nasumičan pristup elementima nije moguć. Naime, da bi se pristupilo elementu u sredini stoga, svi elementi iznad njega bi se trebali maknuti, tako da stog nije dobar odabir za aplikacije kojima je potreban pristup nasumičnim elementima ili elementima u sredini liste (kao što su različiti algoritmi pretraživanja i sortiranja). [9]

Ako se stog implementira pomoću povezane liste, njegova stroga definicija se u nekim dijelovima krši i njegove prednosti i nedostatci se donekle mijenjaju. Naime, kod povezane liste veličina stoga je dinamična, što znači da se može mijenjati, a to se kosi sa strogom definicijom stoga da stog ima fiksnu veličinu. Time se memorija efikasnije koristi jer se alocira po potrebi i nestaje rizik *overflow*-a stoga (osim u slučaju kad samo računalo ostane bez memorijskog prostora). Negativna strana implementacije stoga pomoću povezane liste je potreba za više memorije jer se uz podatke na stogu, pohranjuju i pokazivači. Također, zbog pokazivača operacije nad stogom gube na brzini.

Dakle, svaka implementacija stoga ima svoje prednosti i nedostatke, a gdje koju koristiti ovisi o specifičnim potrebama programa. Ako je potreban stog fiksne veličine s bržim vremenom pristupa koji može podnijeti potencijalne greške, mogla bi biti prikladna implementacija stoga pomoći reda. S druge strane, ako je potreban stog promjenjive veličine s učinkovitim korištenjem memorije, mogla bi biti prikladna implementacija stoga pomoću povezane liste.

2.6 Primjene

Logično, ako se strukturi podataka ograniči broj mogućih operacija, tada se dobiva na brzini izvođenja operacija. Mali broj moćnih operacija je upravo ono zbog čega se stog tako često primjenjuje.

2.6.1 Balansiranje simbola

Stog se primjenjuje u kompajleru za analizu sintakse programskog jezika, za takozvano „balansiranje simbola“, kada kompajler pregledava kod programa za određenu vrstu sintaktičke greške. Naime, sekvenca '[()]' je balansirana, ispravna, ali '[[)]' nije. Pomoću jednostavnog algoritma i strukture podataka stog može se napisati kratak program za rješavanje tog problema. Program će provjeravati je li sve uravnoteženo, postoji li za svaku lijevu zagradu, njen desni par. Primjer takvog algoritma koji koristi stog, mogao bi biti napisan kao što je na slici 7 (u C++ jeziku) u oko 50 linija koda. [4]

Slika

```cpp
#include <iostream>
#include <fstream>
#include <stack>
using namespace std;

bool jeOtvoreniSimbol(char simbol) {
    return simbol == '(' || simbol == '[' || simbol == '{';
}
bool jeZatvoreniSimbol(char simbol) {
    return simbol == ')' || simbol == ']' || simbol == '}';
}
bool jeParSimbola(char otvoreni, char zatvoreni) {
    return (otvoreni == '(' && zatvoreni == ')') ||
           (otvoreni == '[' && zatvoreni == ']') ||
           (otvoreni == '{' && zatvoreni == '}');
}

int main() {
    stack<char> stogSimbola;
    char simbol;

    ifstream inputFile("input.txt");
    if (!inputFile) {
        cout << "Nije moguće otvoriti datoteku 'input.txt'\n";
    }
    while (inputFile.get(simbol)) {
        if (jeOtvoreniSimbol(simbol)) {
            stogSimbola.push(simbol);
        } else if (jeZatvoreniSimbol(simbol)) {
            if (stogSimbola.empty()) {
                cout << "Pogreška: Nepodudaranje zatvorenog simbola '" << simbol << "'\n";
            }
            char vrhSimbola = stogSimbola.top();
            stogSimbola.pop();
            if (!jeParSimbola(vrhSimbola, simbol)) {
                cout << "Pogreška: Neusklađeni par simbola '" << vrhSimbola << "' i '" << simbol << "'\n";
                return 1;
            }
        }
    }
    if (!stogSimbola.empty()) {
        cout << "Pogreška: Nepodudarajući otvoreni simbol(i)\n";
    }

    cout << "Podudaranje simbola je ispravno!\n";
    return 0;
}
```

Slika 7. C++ kod za program analize sintakse programskog jezika

Prvo, u program (sa slike 7) se uključuju sve potrebne biblioteke i definiraju se tri korisničke funkcije koje će provjeravati je li dani simbol otvoreni ('(', '[', '{') ili zatvoreni (')', ']', '}') te čine li ispravan par ('()' ili '[]' ili '{}').

Algoritam u glavnoj *main* funkciji započinje sa stvaranjem stoga. Stog se stvara pozivanjem već definirane funkcije *stack* iz istoimene biblioteke.

Zatim se otvara ulazna datoteka „*input.txt*“ za čitanje. Za svaki slučaj, ako datoteka nije uspješno otvorena, program ispisuje poruku o grešci i završava s kodom greške.

Onda slijedi jedna *while* petlja. U petlji se čitaju znakovi iz datoteke i za svaki se znak provjerava je li otvoreni ili zatvoreni simbol. Ako je znak otvoreni simbol, onda se dodaje na stog, a ako je zatvoreni, onda se provjerava njegova uparenost s posljednjim simbolom na stogu korištenjem funkcije „*jeParSimbola*“.

Nakon čitanja cijelog teksta, program provjerava je li stog prazan. Ako stog idalje sadrži otvorene simbole, to znači da postoje otvoreni simboli bez zatvorenih, što je greška.

Na kraju, ovisno o rezultatima provjere, program ispisuje odgovarajuće poruke o greškama ili, ako ih nema detektiranih, poruku „Podudaranje simbola je ispravno.“

Ovaj algoritam je efikasan i pouzdan način za provjeru ispravnosti zagrada u tekstu i može se prilagoditi za korištenje u raznim aplikacijama koje zahtijevaju ovu vrstu provjere sintakse .

2.6.2 *Undo* opcija

Još jedna zanimljiva primjena stoga je *undo* opcija.

Ideja je da se prethodna stanja programa (koja su ograničena na određeni broj) pohranjuju u memoriju takvim redoslijedom da se zadnje stanje pojavljuje prvo. To se ne može učiniti samo korištenjem nizova i upravu tu stog dobro dođe. [10]

Na slikama 8 i 9 se nalazi kod programa koji upravo tako primjenjuje stog te daje korisniku nekoliko mogućnosti za operiranje. Uklanjanje stanja se shvaća kao *undo*, jer se stanje tako vraća u prijašnje.

Slika

```cpp
#include <iostream>
using namespace std;
const int MAKS_BR_STANJA = 5; // definirajte maksimalnog broja stanja u stogu

class StogStanja {
private:
    int stog[MAKS_BR_STANJA];
    int vrh; // (eng. top) indeks vrha stoga
    int maksVelicina;
public:
    StogStanja() {
        vrh = -1; // inicijalizacija praznog stoga s ineksom -1
        maksVelicina = MAKS_BR_STANJA;
    }

    void dodajStanje(int stanje) { // metoda za dodavanje stanja na vrh stoga
        if (vrh < maksVelicina - 1) {
            for (int i = vrh; i >= 0; i--) { // pomicanje elemenata kako bi se napravilo mjesta za novo stanje
                stog[i + 1] = stog[i];
            } stog[0] = stanje; // dodavanje novog stanja na vrh stoga
            vrh++;
            cout << "Stanje " << stanje << " je dodano na vrh stoga.";
        } else {
            cout << "Stog je pun. Ne mozete dodati vise stanja.";
        } cout << endl;
    }

    void ukloniStanje() { // metoda za uklanjanje stanja s vrha stoga
        if (!jePrazan()) {
            int uklonjenoStanje = stog[0];
            for (int i = 0; i < vrh; i++) { // pomicanje elemenata kako bi se uklonilo stanje s vrha
                stog[i] = stog[i + 1];
            } vrh--;
            cout << "Stanje " << uklonjenoStanje << " je uklonjeno s vrha stoga.";
        } else {
            cout << "Stog je prazan. Ne mozete izvrsiti operaciju uklanjanja.";
        } cout << endl << endl;
    }

    // metoda za provjeru je li stog prazan (eng. isEmpty)
    bool jePrazan() {
        return vrh == -1;
    }

    // metoda za provjeru je li stog pun (eng. isFull)
    bool jePun() {
        return vrh == maksVelicina - 1;
    }

    // metoda za provjeru trenutnog stanja na vrhu stoga
    int trenutnoStanje() {
        if (!jePrazan()) {
            return stog[0];
        } else {
            cout << "Stog je prazan. Nema trenutnog stanja." << endl << endl;
            return -1; // vracanje zadanog stanja koje označava prazan stog
        }
    }
};
```

Slika 8. C++ kod za program s mogućnosti povratka u prijašnje stanje (prvi dio)

```
60
61 int main() {
62     StogStanja stogStanja1;
63     int odabir;
64     int novoStanje;
65     int trenutno = -1; // deklariranje varijable za trenutno stanje
66
67     while (true) {
68         cout << "// ODABIR OPERACIJE" << endl;
69         cout << "1: Provjera trenutnog stanja u stogu" << endl;
70         cout << "2: Dodavanje stanja na vrh" << endl;
71         cout << "3: Uklanjanje stanje s vrha" << endl;
72         cout << "4: Provjera praznoce stoga" << endl;
73         cout << "5: Provjera punoce stoga" << endl;
74         cout << "6: Izlaz iz programa" << endl;
75         cin >> odabir;
76
77         switch (odabir) {
78             case 1:
79                 trenutno = stogStanja1.trenutnoStanje();
80                 if (trenutno != -1) {
81                     cout << "Trenutno stanje na vrhu stoga: " << trenutno << endl << endl;
82                 } break;
83             case 2:
84                 cout << "Unesite novo stanje: ";
85                 cin >> novoStanje;
86                 stogStanja1.dodajStanje(novoStanje);
87                 cout << endl;
88                 break;
89             case 3:
90                 stogStanja1.ukloniStanje();
91                 break;
92             case 4:
93                 if (stogStanja1.jePrazan()) {
94                     cout << "Stog je prazan.";
95                 } else {
96                     cout << "Stog nije prazan.";
97                 } cout << endl << endl;
98                 break;
99             case 5:
100                if (stogStanja1.jePun()) {
101                    cout << "Stog je pun.";
102                } else {
103                    cout << "Stog nije pun.";
104                } cout << endl << endl;
105                break;
106            case 6:
107                cout << "Bok!" << endl;
108                return 0;
109            default:
110                cout << "Nevazeci odabir." << endl << endl;
111        }
112    }
113 }
```

Slika 9. C++ kod za program s mogućnosti povratka u prijašnje stanje (drugi dio)

Ovaj program (sa slika 8 i 9) s *undo* opcijom počinje definicijom konstante „MAKS_BR_STANJA“ koja, kako joj samo ime glasi, označava maksimalan broj stanja koji se može pohraniti na stog.

Nakon toga, definira se klasa „StogStanja“ koja sadrži tri privatne varijable, jedna za stog, jedna za indeks vrha stoga i jedna za maksimalnu veličnu stoga te javne metode „dodajStanje“, „ukloniStanje“, „jePrazan“, „jePun“ i „trenutnoStanje“. U ovom kodu, za razliku od prijašnjeg u ovome radu, svjesno, nije korištena biblioteka <*stack*> kako bi se transparentnije vidjelo kako točno stog može biti isprogramiran.

Metoda „dodajStanje“ (mogla se nazvati i „*pop*“) (u klasi „StogStanja“) omogućuje dodavanje novog stanja na vrh stoga. Provjerava je li stog pun i u slučaju da je - izvodi dodavanje. Prilikom dodavanja, postojeća stanja se pomiču kako bi se stvorilo mjesto za novo stanje, a novo stanje se dodaje na vrh. Program također korisniku ispisuje poruku o uspješnom dodavanju stanja ili poruku o grešci ako je stog pun.

Metoda „ukloniStanje“ je zapravo *undo* opcija i omogućuje uklanjanje stanja s vrha stoga. Provjerava je li stog prazan i u slučaju da nije - izvodi uklanjanje. Prilikom uklanjanja, preostala stanja se pomiču kako bi se uklonilo stanje s vrha. Program također ispisuje poruku o uspješnom uklanjanju stanja ili poruku o grešci ako je stog prazan.

Metode „jePrazan“ i „jePun“ provjeravaju je li stog prazan, odnosno pun, i vraćaju odgovarajuće *boolean* vrijednosti.

Metoda „trenutnoStanje“ provjera trenutno stanje na vrhu stoga. Ako stog nije prazan, vraća trenutno stanje, inače ispisuje poruku o praznom stogu.

U glavnoj funkciji „*main*“, stvara se objekt „stogStanja1“ klase „StogStanja“ za rad s tim stogom.

Unutar *while* petlje, korisniku se nudi izbor operacija: provjera trenutnog stanja, dodavanje stanja, uklanjanje stanja, provjera praznoće i punoće stoga te izlaz iz programa, pa se njegov odabir obrađuje u petlji, a odgovarajuće metode klase „StogStanja“ se pozivaju ovisno o odabiru.

Algoritam omogućuje interaktivnu uporabu programa s povratnim informacije o izvedenim operacijama i stanju stoga. Koristan je za različite primjene u kojima treba pratiti povijest stanja ili postupaka nekog programa.

2.6.3 Pohrana povijesti

Također, povijest pretraživača se pohranjuje u stogu. Na slici 10 je kod takve primjene.

Slika

```cpp
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int main() {
    stack<string> povijestPretrazivanja; // stog pohrane povijesti pretrazivanja
    string trenutnaPretraga;
    cout << "// PRETRAZIVAC" << endl << endl;

    while (true) {
        cout << "Korisnicki unos: ";
        getline(cin, trenutnaPretraga);

        if (trenutnaPretraga.empty()) { // ako je unos prazan, izlazi se iz programa
            cout << "Bok!" << endl << endl;
            break;
        }

        povijestPretrazivanja.push(trenutnaPretraga); // push-a, tj. dodaje trenutnu pretragu na vrh stoga povijesti pretrazivanja
        cout << endl << "Povijest pretrazivanja:" << endl;
        stack<string> tempStack = povijestPretrazivanja;

        while (!tempStack.empty()) {
            cout << tempStack.top() << endl;
            tempStack.pop();
        } cout << endl;
    } return 0;
}
```

Slika 10. C++ kod za program povijesti pretraživanja

Program na slici 10 prvo poziva biblioteku za stog i biblioteku za upravljanje *string* podatcima. *Main* funkcija započinje stvaranjem stoga „povijestPrestraživanja“ i varijable „trenutnaPretraga“ koja će pamtiti korisnikov unos.

Onda kreće *while* petlja. Korisnik unosi tekstualni podatak u varijablu, a ona se *push*-a na vrh stoga te se cijeli stog ispisuje od početka do kraja. Petlja traje dok korisnikov unos nije prazan (*string*).

2.6.4 Obrnuti *string*

Jedna vrlo jednostavna primjena stoga u programiranju se može vidjeti ako želimo obrnuti neku riječ ili *string*. Na slici 11 je kod takvog algoritma.

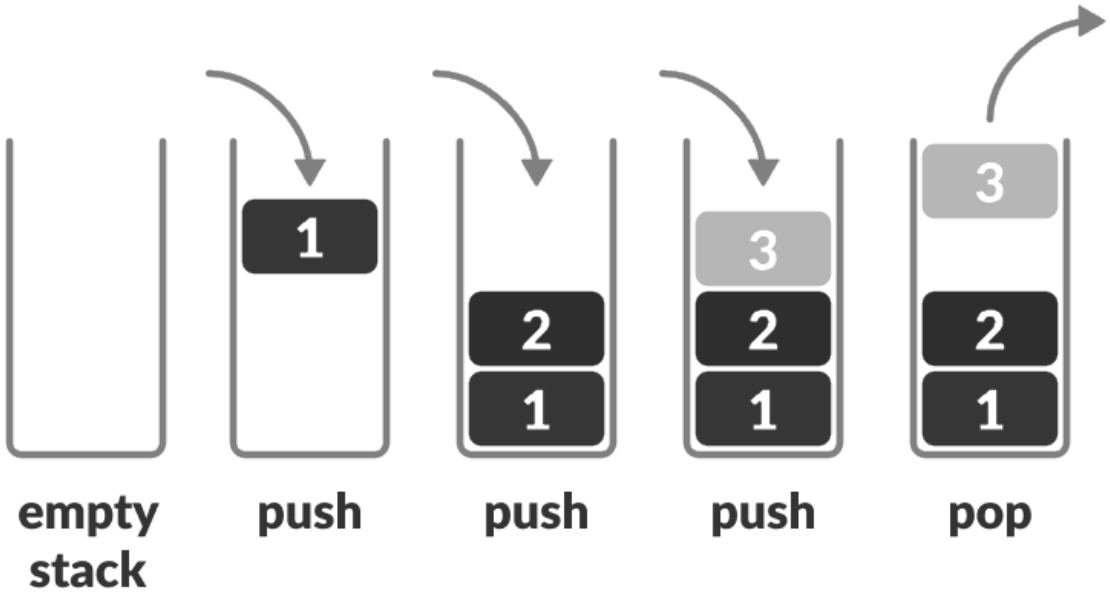
Slika

```cpp
#include <iostream>
#include <stack>
#include <string>
using namespace std;

string obrniString(string unos) {
    stack<char> stog;
    string obrnutiString = "";
    // dodavanje svakog znaka iz stringa na stog
    for (char c : unos) {
        stog.push(c);
    } // skidanje znakove sa stoga i njihovo dodavanje na obrnuti string
    while (!stog.empty()) {
        obrnutiString += stog.top();
        stog.pop();
    } return obrnutiString;
}

int main() {
    string unos;
    cout << "Unesite string: ";
    cin >> unos;
    string obrnut = obrniString(unos);
    cout << "Obrnuti string: " << obrnut << endl;
    return 0;
}
```

Slika 11. C++ kod za program koji obrne *string*

Ovaj program od korisnika traži da unese *string* i onda uz pomoć metode „obrniString“ obrne korisnikov *string* te ga ispiše *cout*-om. Metoda funkcionira tako što prvo svako slovo

push-a na stog, a zatim *pop*-a iz tog istog stoga. Zbog LIFO/FILO svojstva stoga, redoslijed slova bit će, prirodno, obrnut. Slika 12 prikazuje upravo to svojstvo stoga.



Slika 12. LIFO/FILO svojstvo stoga

3 Zaključak

Stog je ATP, zapravo jednostavna, ali moćna linearna struktura podataka koja slijedi LIFO/FILO poredak. Može se predočiti, vizualizirati na više načina, jedan od najčešćih primjera je hrpa pladnjeva u kantini.

Služi kao koncept za spremanje i dohvaćanje podataka u računalu.

C++ je prikladan programski jezik za njegovu izvedbu (s unaprijed definiranom klasom <*stack*> i metodama) u trenutnoj paradigmi dizajna i implementacije.

Stog u svojoj srži ima samo dvije, ali vrijedne, osnovne operacije koje se odnose samo na najgornji element liste zvan vrh - *push* za dodavanje elemenata na njega i *pop* za uklanjanje elemenata s njega, obje s odličnom vremenskom složenosti od O(1).

S obzirom na to da ima određen kapacitet za broj elemenata u sebi, nedostatak stoga je da će *over*, ili *under, flow*-ati, tj. da će se na njega pokušati dodati element dok je pun, ili da će se pokušati ukloniti element iz njega dok je prazan. To je potrebno spriječiti u kodu.

Stog je zanimljiv jer su mu najveće prednosti i najveći nedostatci. Njegov pristup podatcima samo s vrha mu pojednostavljuje i smanjuje kod, poboljšava vremensku složenost, ali i otežava nasumičan pristup elementima i pristup elementima koji nisu na vrhu. Zato je bitno znati kada ga koristit jer u slučaju da se korisit kad se ne bi trebao koristit može uzrokovati greške i mogu mu nedostajati funkcionalnosti koje su potrebne za neki određeni program.

Najbolji primjeri primjene stoga koji su obrađeni u ovome radu, primjeri kada ga koristit, su: analiza sintakse programskih jezika u kompajlerima, *undo* (i *redo*) operacija te bilježenje povijesti, npr. pretraživanja u pretraživačima. Provjera sintakse, vraćanje u prijašnje stanje i pamćenje povijesti su vrlo općeniti koncepti koji se mogu lako prilagoditi za brojne, različite aplikacije. Sve su to učestale primjene koje svjedoče korisnosti strukturi podataka stog.

Popis literature

[4] Mark Allen Weiss. Data Structures and Algorithm Analysis in C++ Fourth Edition. 3.6 The Stack ADT. Florida International University 2014

[5] Adam Drozdek. Data Structures and Algorithms in C++ Fourth Edition. Preface. Cengage Learning 2012

[6] Thomas H. Cormen, Charles R. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms Third Edition. 10 Elementary Data Structures. Massachusetts Institute of Technology 2009

[7] Mark Allen Weiss. Data Structures and Algorithm Analysis in C++ Fourth Edition. 3.1 Abstract Data Types (ADTs). Florida International University 2014

Popis izvora

[1] https://en.wikipedia.org/wiki/Bottom%E2%80%93up_and_top%E2%80%93down_design [Pristupljeno: rujan 2023. godine]

[2] https://www.etymonline.com/word/information [Pristupljeno: rujan 2023. godine]

[3] https://www.merriam-webster.com/dictionary/information%20science [Pristupljeno: rujan 2023. godine]

[8] https://miro.medium.com/v2/resize:fit:1400/1*FkQzWqqIMlAHZ_xNrEPKeA.png [Pristupljeno: rujan 2023. godine]

[9] https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-stack/ [Pristupljeno: rujan 2023. godine]

[10] https://www.freecodecamp.org/news/the-top-data-structures-you-should-know-for-your-next-coding-interview-36af0831f5e3/ [Pristupljeno: rujan 2023. godine]

Popis priloga

Sadržaj