UNIVERZITET U SARAJEVU

PRIRODNO-MATEMATIČKI FAKULTET

ODSJEK ZA MATEMATIKU

TEORIJSKA KOMPJUTERSKA NAUKA

BALANSIRAJUĆE STABLO

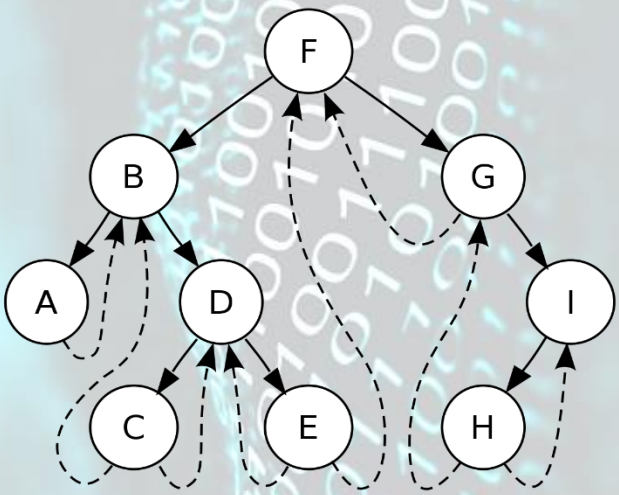
PREDMET: Strukture podataka i algoritmi

PREDMETNI PROFESOR: prof. dr. Esmir Pilav

PREDMETNI ASISTENT: Admir Beširević, MA

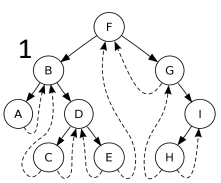
STUDENTICA: Berina Spirjan

BROJ INDEKSA: 5691/M



SARAJEVO, FEBRUAR 2021.

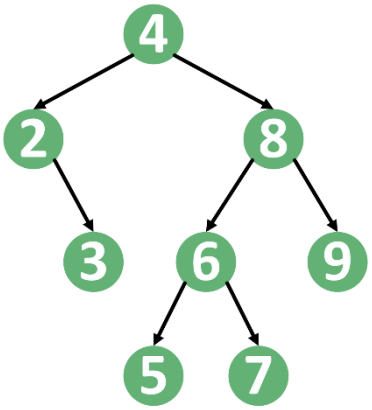
SADRŽAJ



OSNOVNE INFORMACIJE O PROJEKTU

Zadatak ovog projekta jeste da implementiramo klasu Balansirajuće stablo, koje ćemo kreirati koristeći sortirani niz koji će predstavljati vrijednosti čvorova koje umećemo u balansirajuće stablo.

Stablo predstavlja jako važnu ulogu u strukturama podataka, veoma pogodno je za modeliranje objekata koji predstavljaju hijerarhijsku organizaciju.

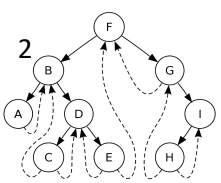


Definicija stabla: Stablo T je konačan, neprazan skup elemenata proizvoljnog tipa-čvorova takav da: postoji jedan poseban čvor koji se naziva korijen, a ostali čvorovi se mogu razdvojiti u n≥0 disjunktnih podskupova, koji su, također, podstabla. Ova podstabla se nazivaju podstablima korijena. Za predstavljanje stabala u memoriji se najčešće koristi ulančana reprezentacija. U tom načinu, čvor se predstavlja elementom koji, pored informacionog dijela (sadržaja), ima i pokazivače na djecu. Broj pokazivača može biti i promjenljiv, u zavisnosti od broja djece. Mnogo pogodnije je, zbog alokacije i dealokacije prostora, da broj pokazivača po čvoru bude fiksan i jednak stepenu stabla, da se iniciraju samo pokazivači na postojeća podstabla, dok su ostali prazni.

U kompjuterskim naukama, binarno stablo se rekurzivno definiše kao konačan skup čvorova koji je ili prazan ili se sastoji od korijena sa dva posebna podstabla, lijevim i desnim, koja su, također, binarna stabla. Ono je po svojoj definiciji poziciono stablo.

Za binarno stablo se kaže da je balansirano ako za svaki čvor važi da se broj čvorova u njegovom lijevom i desnom podstablu ne razlikuje za više od 1. Da bi se dobilo još i stablo minimalne visine za dati broj čvorova, jasno je da čvorove treba distribuirati ravnomjerno na lijevo i desno podstablo, maksimalno popunjavajući sve nivoe osim posljednjeg.

Cilj ovog projekta jeste da napravimo generičku strukturu podataka koja čuva elemente kao binarno stablo pretrage, gdje stablo održava balansiranost na sljedeći način: kada se na visini barem 3, desi da je broj elemenata u jednom podstablu duplo veći od broja elemenata u drugom podstablu, podstablo tog čvora se mijenja. To ćemo raditi tako što ćemo od elemenata podstabla napravi sortirani niz i onda na osnovu toga napraviti podstablo koje je idealno balansirano.



1.ORGANIZACIJA PROJEKTA

U ovom poglavlju naše dokumentacije obradit ćemo organizaciju projekta, raspored po klasama i strukturama, te public i private dijelove klase BalansirajuceStablo.

Implementaciju našeg projekta, sprovest ćemo pomoću tri fajla. Tačnije samu organizaciju projekta možemo sprovesti kroz dva foldera Sources i Headers. U folderu Headers kreiramo fajl pod nazivom balansirajuce. h, a u kojem ćemo definisati sve klase i strukture, te prototipe metoda i funkcija. U sklopu foldera Sources nalaze se dva fajla main. cpp i balansirajuce. cpp. U fajlu balansirajuce. cpp sprovodimo programsku implementaciju, tj. tijela funkcija i metoda. A main. cpp fajl nam služi za testiranje implementacije programa.

Kako bismo implementirali klasu BalansirajuceStablo, potrebno je da koristimo dodatnu strukturu Cvor. Budući da je neophodno da kreiramo generičku klasu za balansirajuće binarno stablo, neophodno je da prije same definicije klase deklarišemo generički tip. Strukturu Cvor deklarišemo u private dijelu funkcije.

1.1 struct Cvor

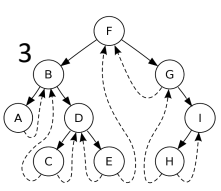
Struktura čvor posjeduje pet atributa koji su public dio strukture, jer nije drugačije navedeno. Ti atributi su : T vrijednost, unsigned int brojPotomaka, Cvor *roditelj, Cvor *lijevoDijete, Cvor *desnoDijete. Ova struktura je iznimno bitna za samu implementaciju klase BalansirajuceStablo, jer pomoću nje definišemo sve čvorove u stablu, te pomoću nje pristupamo vrijednostima čvorova, broju potomaka, da saznamo koji čvor je roditelj, koji lijevo, a koji desno dijete od proslijeđenog čvora. Atribut T vrijednost predstavlja generički tip koji se odnosi na vrijednost čvora. Drugi atribut strukture ukazuje na broj potomaka koje ima određeni čvor, dakle, sve čvorove koji se nalaze u njegovom podstablu. Cvor *roditelj je pokazivač na čvor koji je roditelj. Cvor *lijevoDijete je pokazivač na lijevo dijete od čvora, dok je Cvor *desnoDijete pokazivač na desno dijete od čvora. Ova struktura, također, posjeduje sljedeće konstruktore:

1.1.1 Cvor(T vrijednost, Cvor *roditelj, Cvor *lijevoDijete, Cvor *desnoDijete)

Ovaj konstruktor prima četiri parametra: T vrijednost, Cvor *roditelj, Cvor *lijevoDijete, Cvor *desnoDijete. Redom predstavljaju vrijednost čvora, pokazivač na roditelj za koji na početku postavljamo nullptr jer pretpostavljamo da ne postoji, pokazivač na lijevo dijete roditelja, te pokazivač za desno dijete roditelja, za koje pretpostavljamo da na početku ne postoje, pa ih postavljamo na nullptr.

1.1.2 Cvor (const Cvor &cvor)

Kopirajući konstruktor koji prima jedan parametar koji je const Cvor &cvor. Ovdje kopiramo vrijednost koja je sačuvana u čvoru.



1.1.3 Cvor (const Cvor &&cvor)

Pomjerajući konstruktor koji ima jedan parametar const Cvor &&Cvor, te nakon deklaracije koristimo operator noexcept, što predstavlja da funkcija neće baciti izuzetak za različite tipove.

1.2 class BalansirajuceStablo

Privatni dio klase (private):

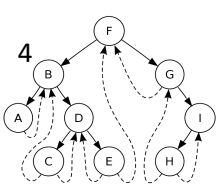
Klasa BalansirajuceStablo posjeduje privatnu strukturu Cvor, čiju smo implementaciju prethodno opisali, te dva atributa: Cvor *korijen, unsigned int brojElemenata, koji su respektivno, pokazivač na korijen stabla i broj elemenata koje to stablo posjeduje. Pored ova dva atributa klasa BalansirajuceStablo posjeduje, također, nekoliko privatnih metoda i funkcija. U nastavku slijedi pregled funkcija i metoda koje su navedene u private dijelu klase BalansirajuceStablo, a njihovu implementaciju ćemo razmatrati u idućem poglavlju:

- void izgradiPodstablo(Cvor *destinacija, Cvor *izvor);
- void preorderIspis(ostream &tok, Cvor *cvor);
- void postorderIspis(ostream &tok, Cvor *cvor);
- void azurirajPutanjuDoKorijena(Cvor *cvor, int povecanje);
- void balansiraj(Cvor *balanser);
- void popuniSortiraniNiz(vector<T> &sortiraniNiz, Cvor *cvor);
- void unistiStablo(Cvor *cvor);
- friend void inorderUTok(ostream &tok, typename BalansirajuceStablo<T1>::Cvor *cvor);
- Cvor* nadjiRoditelja(const T& element, typename BalansirajuceStablo<T>::Cvor *roditelj);
- Cvor* napraviStabloIzSortiranogNiza(const vector<T> &sortiraniNiz, int lijevaGranica, int desnaGranica);

Javni dio klase BalansirajuceStablo (public):

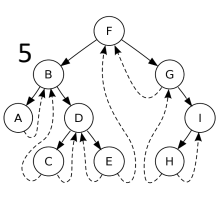
U ovom dijelu definišemo dio klase u kojem se nalaze konstruktori, operatori, destruktori, pomoćne funkcije i metode. Također, u nastavku se nalazi pregled public dijela klase, a nešto kasnije razmotriti dolje navedeni dio, pojedinačno.

- BalansirajuceStablo();
- BalansirajuceStablo(const BalansirajuceStablo<T> &stablo);
- BalansirajuceStablo(BalansirajuceStablo<T> &&stablo) noexcept ;
- ~BalansirajuceStablo();
- BalansirajuceStablo<T>& operator=(const BalansirajuceStablo<T> &stablo);
- BalansirajuceStablo<T>& operator=(BalansirajuceStablo<T> &&stablo) noexcept;
- void dodaj(const T& element);
- void izbrisi(const T &element);
- void ispisiUTokInorder(ostream &tok);
- void ispisiPreorder(ostream &tok);



- void ispisiPostorder(ostream &tok);
- Cvor *nadji(const T &x);
- friend ostream &operator<<(ostream &tok, const BalansirajuceStablo<T1> &stablo);

U sljedećem poglavlju ćemo predstaviti sve detalje gore navedenih funkcija, konstruktora, operatora u public i private dijelu klase BalansirajuceStablo. Radi jednostavnosti, pri samom opisu funkcija, vodit ćemo računa da, ukoliko pojedine funkcije koriste neke druge ili pomoćne funkcije, opis tih funkcija navodimo prije korištenja novokreirane funkcije.



2. DETALJNI PRIKAZ FUNKCIJA, OPERATORA I KONSTRUKTORA

void izgradiPodstablo(Cvor *destinacija, Cvor *izvor);

Ovu metodu koristimo kako bismo kreirali binarno stablo, a prima dva parametra: Cvor *destinacija, Cvor *izvor. Trebamo prvo ispitati da li postoji pokazivač na čvor izvor, te ukoliko postoji sljedeći korak jeste da provjerimo da li čvor izvor ima lijevo dijete, te ukoliko ima kreiramo novi čvor koji će pokazivati na lijevo dijete izvora. Potrebno je da potom dodijelimo lijevom djetetu od destinacije prethodno kreirani čvor lijevo dijete, a potom da postavimo roditelja od čvora lijevo dijete koji će predstavljati destinaciju. Te pozivamo rekurzivni poziv funkcije za lijevo dijete i lijevo dijete čvora izvor. Isti postupak ponavljamo i za desno dijete.

void preorderIspis(ostream &tok, Cvor *cvor);

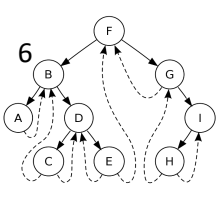
PreorderIspis metoda prima dva parametra: ostream &tok, Cvor *cvor. Ova metoda nam služi kako bismo stablo ispisali u preorder poretku. Preorder poredak kod stabla je poredak u kojem se prvo posjećuje roditelj, zatim lijevo, a potom desno dijete roditelja. Ovaj postupak se ponavlja sve dok se ne posjete svi čvorovi u stablu. Metoda je implementirana na način da prvo provjeravamo da li postoji cvor, te ukoliko postoji ispisujemo vrijednost tog čvora i pozivamo rekurziju za lijevo dijete ovog čvora, a potom i rekurzivni poziv za desno dijete ovog čvora. Rekurzija će se pozivati sve dok postoji pokazivač na čvor.

void postorderIspis(ostream &tok, Cvor *cvor);

Ova metoda prima dva parametra: ostream &tok, Cvor *cvor, a koristimo je kako bismo stablo ispisali u postorder poretku. Postorder poredak kod stabla je poredak u kojem se prvo posjećuje lijevo dijete, zatim desno dijete, a tek nakon toga roditelj ovih potomaka. Ukoliko razmatramo u globalnom slucaju, prvo se posjećuje lijevo podstablo, potom desno podstablo, a posljednji se posjećuje korijen stabla. Metoda je implementirana na način da ispitujemo postojanost čvora (tj. ukoliko posmatramo sa rekurzivnim pozivom ovaj dio koda će se izvršavati sve dok postoji pokazivač na čvor, tj. dok ne postane nullptr), te potom pozivamo rekurziju za lijevo dijete, za desno dijete, a potom ispisujemo vrijednosti čvorova u stablu.

void unistiStablo(Cvor *cvor);

Ova metoda prima jedan parametar koji predstavlja pokazivač na čvor koji želimo obrisati iz stabla. Metoda radi na principu postorder brisanja elemenata. Razlog toga jeste što je neophodno prvo obrisati svu djecu, a potom roditelje potomaka. Pozivamo prvo rekurziju za lijevo dijete, potom za desno i na kraju brišemo proslijeđeni parametar čvor, ovo radimo da bismo izbjegli curenje memorije, te postojanje pokazivača nakon njegove upotrebe.



void azurirajPutanjuDoKorijena(Cvor *cvor, int povecanje);

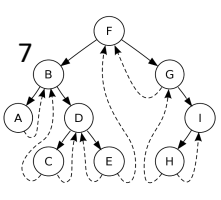
Ova metoda prima dva parametra: Cvor *cvor, int povećanje. Unutar ove metode vršimo prebrojavanje nivoa u stablu, a to ćemo koristiti da saznamo kada možemo da vršimo balansiranje, što u našem slučaju radimo kada se nađemo na nivou 3 ili više. Potom prolazimo while petljom obilazimo sve roditelje proslijeđenog čvora i ažuriramo broj potomaka. Ukoliko dodajemo element u stablo, tada imamo da je varijabla povećavanje 1. Ukoliko iz stabla brišemo neki element, tada imamo da je varijabla povećanje -1. Potrebno je da promijenimo broj potomaka za sve čvorove od korijena do mjesta brisanja, odnosno dodavanja shodno operaciji koju vršimo. Potom je neophodno da zapamtimo čvor koji je korijen podstabla za koji je potrebno izvršiti balansiranje. Potrebno je provjeriti da li čvor posjeduje dva djeteta, te ukoliko je to slučaj, kreiramo dva nova pokazivača na lijevo i desno dijete. Provjeravamo da li podstablo ima duplo više elemenata od drugog podstabla, ovisno od situacije, desno ili lijevo podstablo. Nakon toga prelazimo na nivo iznad ili ispod. Ukoliko pronađemo čvor koji zadovoljava uslove balansiranja, potrebno je da pozovemo funkciju balansiraj koja će izbalansirati to podstablo.

void balansiraj(Cvor *balanser);

Ova metoda je najbitnija metoda u projekatu, a koja koristi još tri tri pomoćne metode. Prima jedan parametar Cvor *balanser koji predstavlja pokazivač na čvor koji je potrebno balansirati. Prvi korak pri balansiranju je formiranje sortiranog niza od podstabla koje se balansira. Nakon toga kreiramo balansirano stablo iz sortiranog niza pozivom funkcije napraviStabloIzSortiranogNiza, a koja je prethodno opisana u ovom poglavlju dokumentacije. Potrebno je da provjerimo da li je čvor koji kreiramo prethodno korijen, ukoliko jeste, potrebno je da uništimo kompletno stablo, ali to se neće desiti zbog ograničenja da balansiramo tek na 3. nivou, ali je ipak omogućeno u ovom dijelu koda, ukoliko dođe do promjene za nivo balansiranosti. Prilikom umetanja u balansirano podstablo, trebamo korijen tog podstabla povezati sa roditeljima s kojim je prethodni korijen bio povezan. Neophodno je izvršiti provjeru da li je korijen tog podstabla bio lijevo dijete, te ukoliko jeste potrebno je povezati ga kao lijevo dijete, u suprotnom kao desno. Pošto smo kreirali novobalansirano podstablo, staro nebalansirano stablo uništavamo koristeći funkciju uništiStablo, deklarisano u dijelu iznad.

void popuniSortiraniNiz(vector<T> &sortiraniNiz, Cvor *cvor);

Ovu metodu koristimo za popunjavanje sortiranog niza na osnovu kojeg kreiramo balansirano stablo. Prima dva parametra: vector<T> &sortiraniNiz, Cvor *cvor. Respektivno, prvi parametar predstavlja sortirani niz generičkih vrijednosti, te pokazivač na čvor. Na samom početku je neophodno provjeriti da li postoji čvor, te imamo rekurzivni poziv funkcije za lijevo dijete. Potom je neophodno da roditelja dodamo u niz, tj. vrijednost čvora. Nakon toga, pozivamo rekurziju za desno dijete. Ova metoda ima istu strukturu kao inorder ispisivanje, ali ovdje vršimo inorder dodavanje članova u niz.



void unistiStablo(Cvor *cvor);

Ova metoda prima jedan parametar koji predstavlja pokazivač na čvor koji želimo obrisati iz stabla. Metoda radi na principu postorder brisanja elemenata. Razlog toga jeste što je neophodno prvo obrisati svu djecu, a potom roditelje potomaka. Pozivamo prvo rekurziju za lijevo dijete, potom za desno i na kraju brišemo proslijeđeni parametar čvor, ovo radimo da bismo izbjegli curenje memorije, te postojanje pokazivača nakon njegove upotrebe.

friend void inorderUTok(ostream &tok, typename BalansirajuceStablo<T1>::Cvor *cvor)

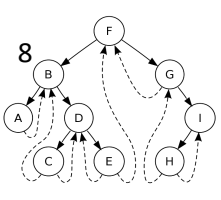
Ova metoda je prijateljska metoda koja prima dva parametra. Potrebno je da ispitamo postojanost pokazivača na čvor, te da ukoliko postoji izvršimo rekurzivni poziv funkcije za lijevo dijete čvora, potom ispišemo vrijednost čvora i nakon toga pozovemo rekurziju za desno dijete čvora. Dakle, ova prijateljska funkcija nam služi za ispis stabla u inorder poretku. Inorder poredak se zasniva na strategiji da se prvo posjeti lijevo dijete, potom roditelj, a nakon toga desno dijete tog čvora.

Cvor* nadjiRoditelja(const T& element, typename BalansirajuceStablo<T>::Cvor *roditelj);

Ova funkcija nam služi kako bismo za proslijeđeni element pronašli njegovog roditelja, te iz funkcije vraćamo pokazivač na tog roditelja. Funkcija prima dva parametra, element za koji je potrebno pronaći roditelja i pokazivač na roditelja. Na početku funkcije provjeravamo da li roditelj uopće postoji, te ukoliko postoji potrebno je da provjerimo da li je element manji od vrijednosti roditelja koji je proslijeđen kao parametar funkciji, ukoliko jeste potrebno je da provjerimo da li postoji lijevo dijete od tog roditelja, a ukoliko je ovaj rezultat zadovoljen iz funkcije vraćamo rekurzivni poziv za lijevo dijete proslijeđenog roditelja. Međutim, ukoliko je element jednak vrijednosti roditelja iz funkcije vraćamo pokazivač na roditelja. Treći uslov jeste ako je element veći od vrijednosti roditelja postupamo slično kao u prvom slučaju, samo ovog puta za desno dijete. Ukoliko pokazivač na roditelja ne postoji, iz funkcije vraćamo nullptr.

Cvor* napraviStabloIzSortiranogNiza(const vector<T> &sortiraniNiz, int lijevaGranica, int desnaGranica);

Ova funkcija prima tri parametra, sortirani niz, lijevu granicu niza, te desnu granicu niza, respektivno. Na početku je potrebno da provjerimo da li su granice saglasne, te ukoliko nisu iz funkcije vraćamo nullptr, što označava da smo taj dio niza završili. Ukoliko su lijeva i desna granice jednake, u tom slučaju je potrebno kreirati jedan čvor. Ovo je specijalan slučaj, jer se tada niz ne može više poloviti. Nakon toga, kao kod binarne pretrage, i ovdje polovimo niz na dva dijela i kreiramo čvor od srednjeg člana dijela niza. Potom, ažuriramo broj potomaka, jer znamo koliko elemenata ima okolo ovog člana niza. Sada imamo dvije polovice ovog dijela



niza i trebamo rekurzivno pozvati ovu metodu i primjeniti isti način nad tim dijelovima niza. Prvo uzmemo lijevi dio niza koji predstavlja elemente od početka do sredine dijela niza. Potrebno je da ispitamo postojanost lijevog djeteta od čvora, te da postavimo da je čvor sada roditelj od lijevog djeteta. Prije ovoga smo izvršili povezivanje novog čvora sa trenutnim, a kada se rekurzija odmota, nakon toga, radimo isto i za desno dijete. Iz funkcije vraćamo pokazivač na srednji element koji u prvom pozivu ove funkcije vraća korijen balansiranog podstabla koje stavljamo u naše stablo.

BalansirajuceStablo();

Konstruktor bez parametara postavlja vrijednost broja elemenata na nulu i pokazivač na korijen kao nullptr, jer pretpostavljamo da na početku ne postoji.

BalansirajuceStablo(const BalansirajuceStablo<T> &stablo);

Kopirajući konstruktor kopira broj elemenata i korijen proslijeđenog stabla. Potom ispituje da li postoji element u korijenu stabla, potom ukoliko postoji potrebno je prekopirati elemente stabla u novo stablo.

BalansirajuceStablo(BalansirajuceStablo<T> &&stablo) noexcept ;

Pomjerajući konstruktor, kojem smo na kraju definisanja proslijedili operator noexcept kako bismo osigurali da funkcija neće baciti izuzetak za bilo koji proslijeđeni tip. Dakle, ovdje modifikujemo parametre stabla. Te nakon modifikacije, postavlja korijen proslijeđenog stabla na nullptr i broj elemenata tog stabla na 0.

~BalansirajuceStablo();

Destruktor koji je potrebno implementirati kako bismo spriječili curenje memorije. Potrebno je da pozovemo funkciju unistiStablo za korijen, te da nakon brisanja stabla postavimo broj elemenata na nulu i pokazivač na korijen kao nullptr.

BalansirajuceStablo<T>& operator=(const BalansirajuceStablo<T> &stablo);

Operator dodjele koji prima jedan parametar koji predstavlja stablo za koje želimo da izvršimo dodjelu tj. kopiranje.