BalansiranoStablo

Robert Marić - 5966

Dodao sam u strukturu Cvor varijable broj_elemenata_l i broj_elemenata_d koje nam govore koliko svaki čvor ima elemenata u lijevom podstablu i u desnom podstablu.

```
template <typename Tip>
pair<typenameBalansiranoStablo<Tip>::Cvor*,bool> BalansiranoStablo<Tip>::Insert(const Tip vrijednost){}
```

Počet ćemo od funkcije Insert koja je morala biti modificirana, jer se disbalans stabla dešava prilikom unosa elemenata. Prvo, provjeravam da li je ta vrijednost koju želimo unijeti već unutar stabla pomoću funkcije Find (koja vraća čvor tog elementa ako postoji). To provjeravam jer prilikom prolaska kroz stablo mijenjam broj_elemenata_l I broj_elemenata_d, a ne bih htio da promijenim te vrijednosti I na kraju element bude u stablu. Prilikom prolaska provjeravam svaki čvor da vidim da li dodavanjem tog novog elementa pravi disbalans stabla. To radim pomoću funkcija:

```
template <typename Tip>
bool BalansiranoStablo<Tip>::DaLiJeVisinaMin3(Cvor *cvor,int el,int lijevoILIdesno)const{}
template <typename Tip>
bool BalansiranoStablo<Tip>::DaLiJeBalansirano(Cvor *cvor, int a, int b) const{}
```

Ako su uslovi ispunjeni, sačuvam trenutni čvor u pomoćnu varijablu “pomocnitrenutni” , jer ću izgubiti trenutnu varijablu. Ako su uslovi ispunjeni tek nakon umetanja novog elementa vršim balansiranje pomoću funkcije

```
template <typename Tip>
void BalansiranoStablo<Tip>::Balansiraj(Cvor *trenutni){}
```

u kojoj pomoću funkcije

```
template <typename Tip>
void BalansiranoStablo<Tip>::InOrderCvora(Cvor *cvor, vector<Tip>& v){}
```

smještam sve elemente tog podstabla kojeg treba balansirati, u rastućem poretku, u vektor. Nakon toga brišem sve elemente koje se nalaze u tom vektoru iz stabla pomoću

```
template <typename Tip>
void BalansiranoStablo<Tip>::EraseZaBalansiranje(const Tip vrijednost){}
```

te ponovo unosim te iste elemente pomoću

```
template <typename Tip>
void BalansiranoStablo<Tip>::BalansirajRekurzivno(int pocetak, int kraj,vector<Tip> v){}
```

koja ubacuje elemente na način tako što ubaci sredinu pa lijevu sredinu pa od te lijevo sredine lijevu sredinu i tako dok sve elemente ne unese. Za unos ne koristim

```
template <typename Tip>
pair<typenameBalansiranoStablo<Tip>::Cvor*,bool> BalansiranoStablo<Tip>::Insert(const Tip vrijednost){}
```

nego

```
template <typename Tip>
void BalansiranoStablo<Tip>::InsertBalansirano(const Tip vrijednost){}
```

jer ne želim da opet provjerava balansiranost jer znam kad unesem opet sve elemente da će biti balansirana.

```
template <typename Tip>
void BalansiranoStablo<Tip>::Erase(const Tip vrijednost){}
```

Funkcionira tako što nakon brisanja elementa "vrijednost" provjerava da li je stablo balansirano I ako jeste vrši balansiranje.

Imamo dvije "slične" funkcije Erase() I EraseZaBalansiranje(). Razlika je u tome što Erase može korisnik koristit dok EraseZaBalansiranje samo program. Ovo sam uradio zato što prilikom brisanja cijelog podstabla kod balansiranja ne moram provjeravati balansiranost prilikom brisanja zato što znam da će biti balansirano kad sve elemente ponovno vratim. Dok, kad korisnik radi brisanje nekog elementa moram provjeriti da li je nakon brisanja stablo opet balansirano pa ako nije opet uradit balansiranje.

Funkcija

```
template <typename Tip>
void BalansiranoStablo<Tip>::PromjenaBrojaElemenata(Cvor *cvor){}
```

služi da smanjimo broj elemenata lijevo I desno do čvora kojeg brišemo.

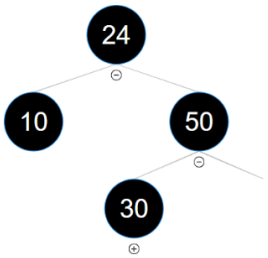
Isto tako imamo dvije funkcije SljedbenikZaBalansiranje() i Sljedbenik() koje su opet "slične", ali razlika je opet što Sljedbenik može korisnik koristit dok SljedbenikZaBrisanje koristi program prilikom brisanja.

Pomoću

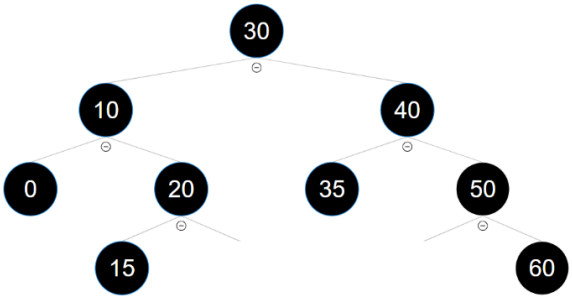
```
template <typename Tip>

typenameBalansiranoStablo<Tip>::Cvor*
BalansiranoStablo<Tip>::SljedbenikZaBalansiranje(BalansiranoStablo<Tip>::Cvor* cvor) const
```

mijenjam brojeve lijevih i desnih elemenata, jer prilikom brisanja određeni čvorovi se premještaju, to jeste dodaju na druge pa trebam to uzeti u obzir prilikom brisanja. Ovo je još jedan razlog zašto sam napravio SljedbenikZaBalansiranje() i Sljedbenik(). Jer, ne bi htio da korisnik prilikom korištenja Sljedbenik() funkcije promijeni brojeve elemenata lijevo i desno nekog čvora.

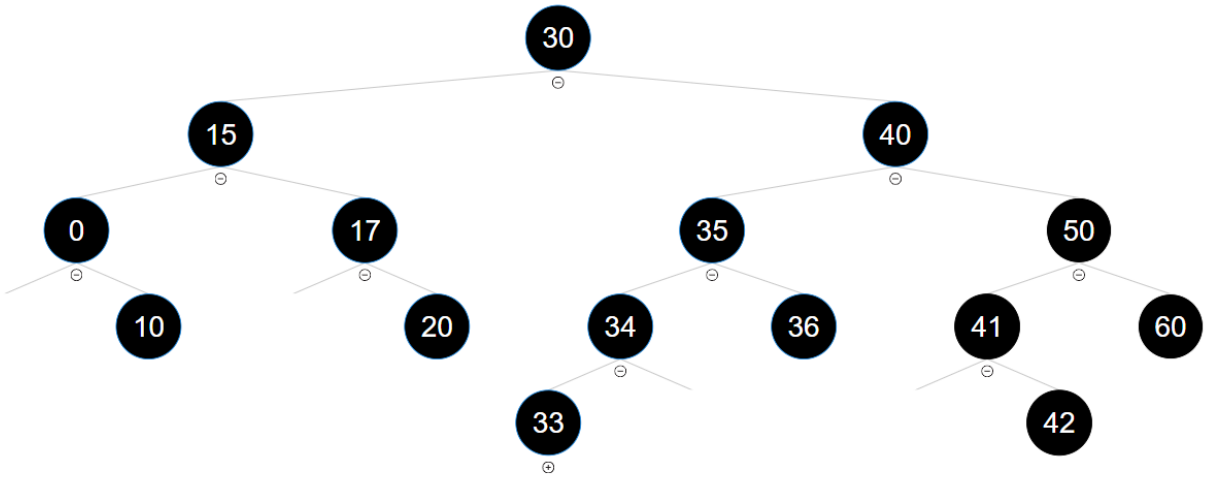
Ovako izgleda proces prilikom unosa elemenata : 20, 50, 10, 30, 40, 0, 60, 35, 15, 17, 36, 34, 41, 42, 33, 32, 7, 16.



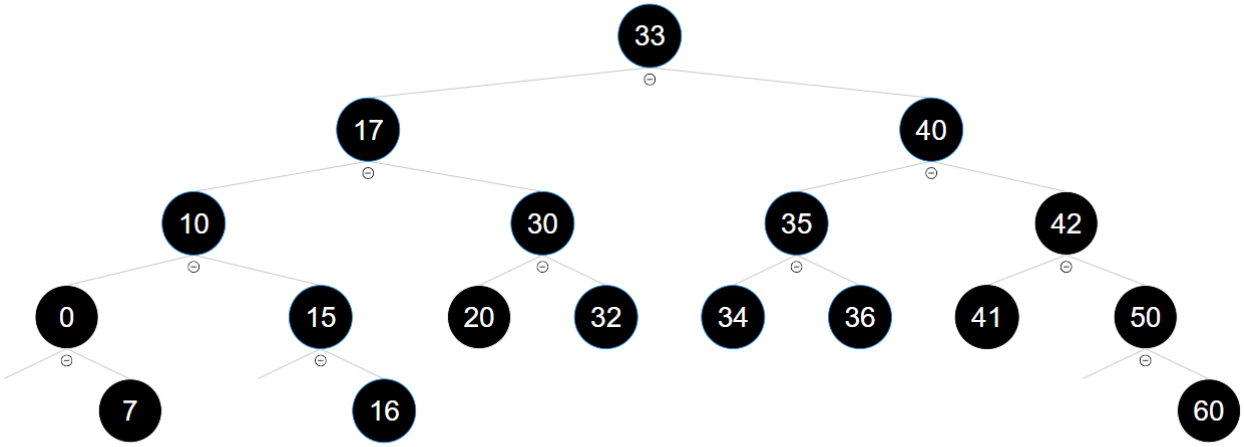
Problem pravi broj 40 prilikom unosa pa moramo balansirati:



Problem pravi broj 17 prilikom unosa pa moramo balansirati:

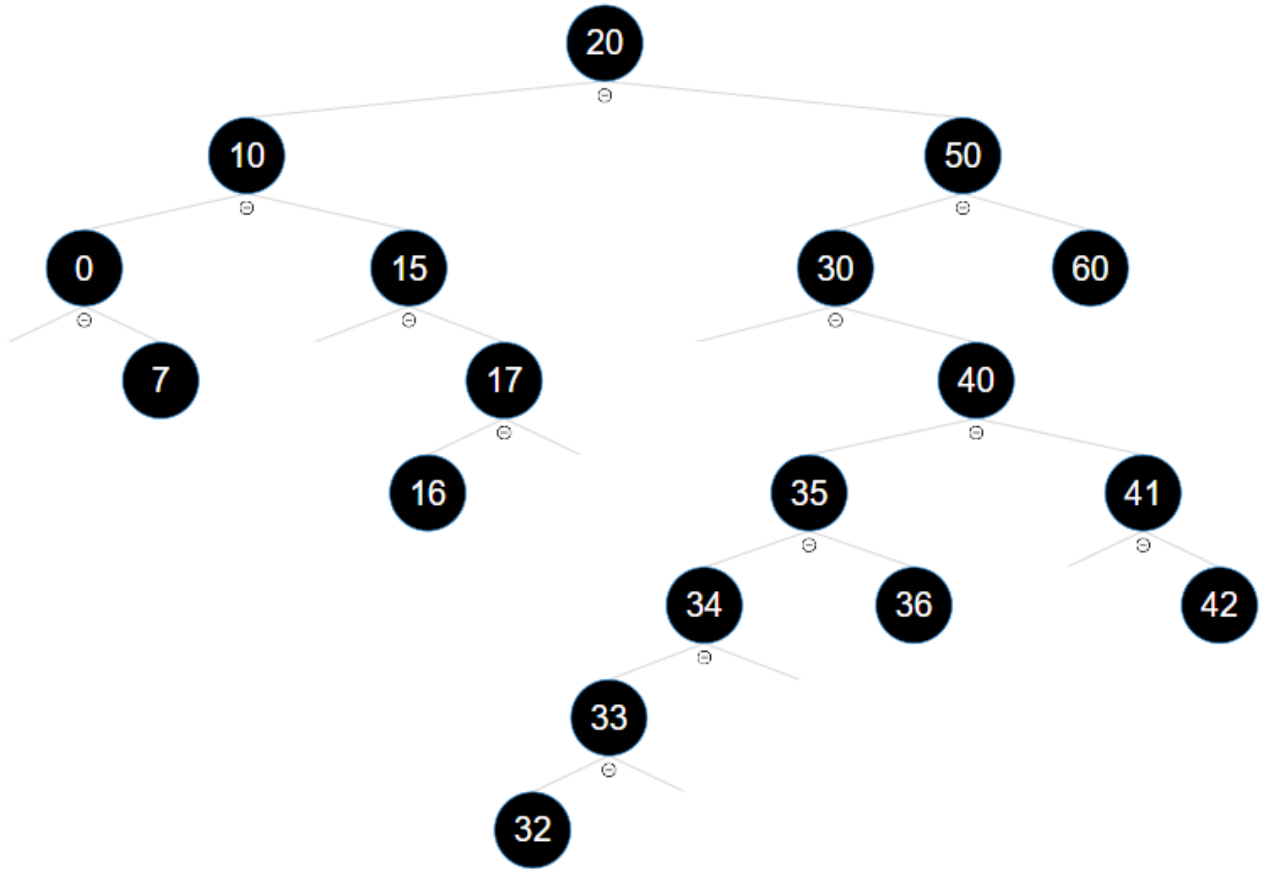


Problem pravi broj 32 prilikom unosa pa moramo balansirati:



Ovo je finalno Balansirano stablo.

Ovako bi izgledalo bez balansiranja.



Ovo je isto stablo nakon što smo izbrisali nekoliko elemenata. Vidimo da je opet balansirano. Elementi koje smo izbrisali su redom: 30, 10, 40, 17, 35, 15, 0, 60, 36, 33.