Algoritmi i strukture podataka
5. čas, AVL stablo

Aleksandar Veljković

2017/2018

1 Uvod

AVL stablo je specijalna vrsta binarnog uredjenog stabla (*BST - Binary Search Tree*) koje ima osobinu samobalansiranja. Operacije pretraživanja, dodavanja i uklanjanja čvorova kod običnog BST stabla imaju vremensku složenost $O(h)$, gde h predstablja visinu stabla. Degenerisano stablo sa n čvorova, kod koga svaki unutrašnji čvor ima samo jednog sina, ima visinu n pa su sve prethodno navedene operacije nad stablom složenosti $O(n)$. Da bi se povecala efikasnost, smanjila vremenska složenost operacija, potrebno je održavati visinu stabla što manjom mogućom. AVL stablo ima uslov da se visine levog i desnog podstabla **svakog čvora u stablu** razlikuju najviše za 1. Drugim rečima, ako je visina levog podstabla nekog čvora x jednaka h_1 a visina desnog podstabla čvora x jednaka h_2, neophodno je da važi da je $|h_1 - h_2| \leq 1$.

Proverimo da li ovakav uslov obezbedjuje visinu stabla sa n čvorova nižu od $O(m)$. Krenimo malo drugačijim putem, neka je N_h broj čvorova u stablu visine h i neka se visina jednog od dva podstabla uvek razlikuje za 1 od drugog. Broj čvorova u takvom stablu se može predstaviti rekurentnom jednačinom:

$$N_h = N_{h-1} + N_{h-2} + 1$$

Broj čvorova u stablu visine h jednak je broju čvorova u levom i desnom podstablu njegovog korena +1 za koren. Imajući u vidu da je jedno od podstabala uvek kraće od drugog, ovo predstavlja najgori slučaj za prethodno navedeni uslova AVL stabla.

Kako važi da je:

$$N_h = N_{h-1} + N_{h-2} + 1 > N_{h-1} + N_{h-2} \geq N_{h-2} + N_{h-2} = 2 \cdot N_{h-2}$$

Za početni uslov $N_0 = 1$ (za potrebe definicije stabla, list stabla ima visinu 0 pa je broj čvorova u stablu visine 0 jednak 1).

$$N_h > 2 \cdot N_{h-2} = 2^{\frac{h}{2}}$$

$$N_h > 2^{\frac{h}{2}}$$

Primenom logaritma za osnovu 2 na obe strane nejednakosti dobija se:

$$\log_2 N_h > \log_2 (2^{\frac{h}{2}})$$

$$\log_2 N_h > \frac{h}{2} \cdot \log_2 2$$

$$\log_2 N_h > \frac{h}{2}$$

$$2 \log_2 N_h > h$$

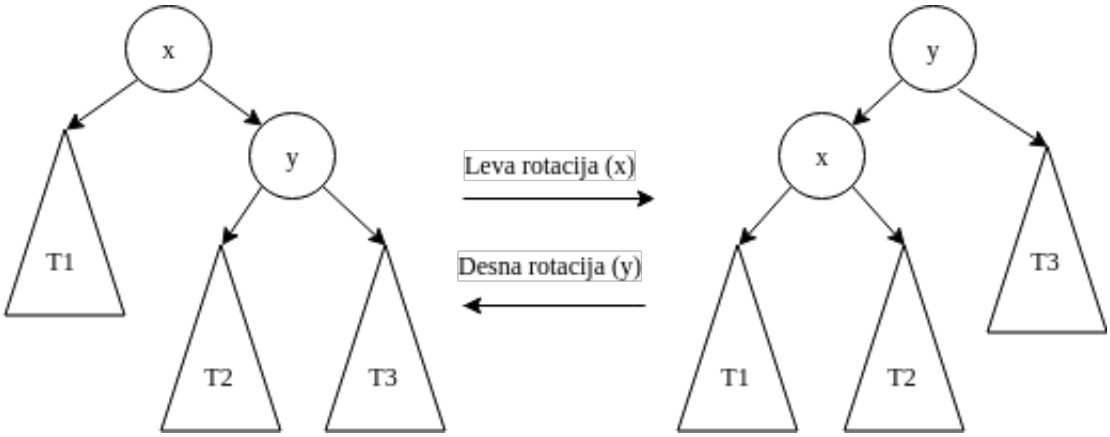
Tj.

$$h = O(\log n)$$

Ovime je pokazano da uslov AVL stabla, da se visina levog i desnog podstabla svakog čvora razlikuje najviše za 1, garantuje visinu stabla $O(\log n)$, gde je n broj čvorova u stablu, pa je time i vremenska složenost operacija nad AVL stablom (pretraživanje, dodavanje i uklanjanje čvorova) ograničena sa $O(\log n)$.

2 Rotacije

Da bi se obezbedio uslov AVL stabla, potrebno je prethodno definisati operacije rotacija čvorova stabla. Leva i desna rotacija prikazane su na Slici 1.



Slika 1. Leva i desna rotacija čvorova x i y

Operacije rotacija čvorova stabla su vremenske složenosti $O(1)$ jer zahtevaju samo nekoliko razmena pokazivača.

3 Balansiranje

Dodavanje i uklanjanje čvorova AVL stabla se vrše isto kao u slučaju običnog BST stabla. Nakon dodavanja ili brisanja čvora stabla, moguća je pojava nebalansiranih čvorova, kod kojih se visine levog i desnog podstabla razlikuju za više od jedan. Tada je potrebno izvršiti potrebne popravke nad stablom kako bi ponovo svi čvorovi postali balansirani ali i BST uslovi ostali ispunjeni. Popravljanje se vrši počevši od najnižeg nebalansiranog čvora a zatim se popravljaju svi ostali nebalansirani čvorovi krećući se ka korenu stabla. Popravke predstavaljaju rotacije odgovarajućih čvorova stabla, kako se rotacije izvršavaju u vremenu $O(1)$ a nebalansiranih čvorova od najnižeg nebalansiranog čvora pa sve do korena ima najviše $O(\log n)$, vremenska složenost dodavanja i uklanjanja čvorova AVL stabla, uz dodatne popravke, ostaje $O(\log n)$. Odgovarajuće rotacije se primenjuju u zavisnosti od oblika nebalansiranosti čvora.

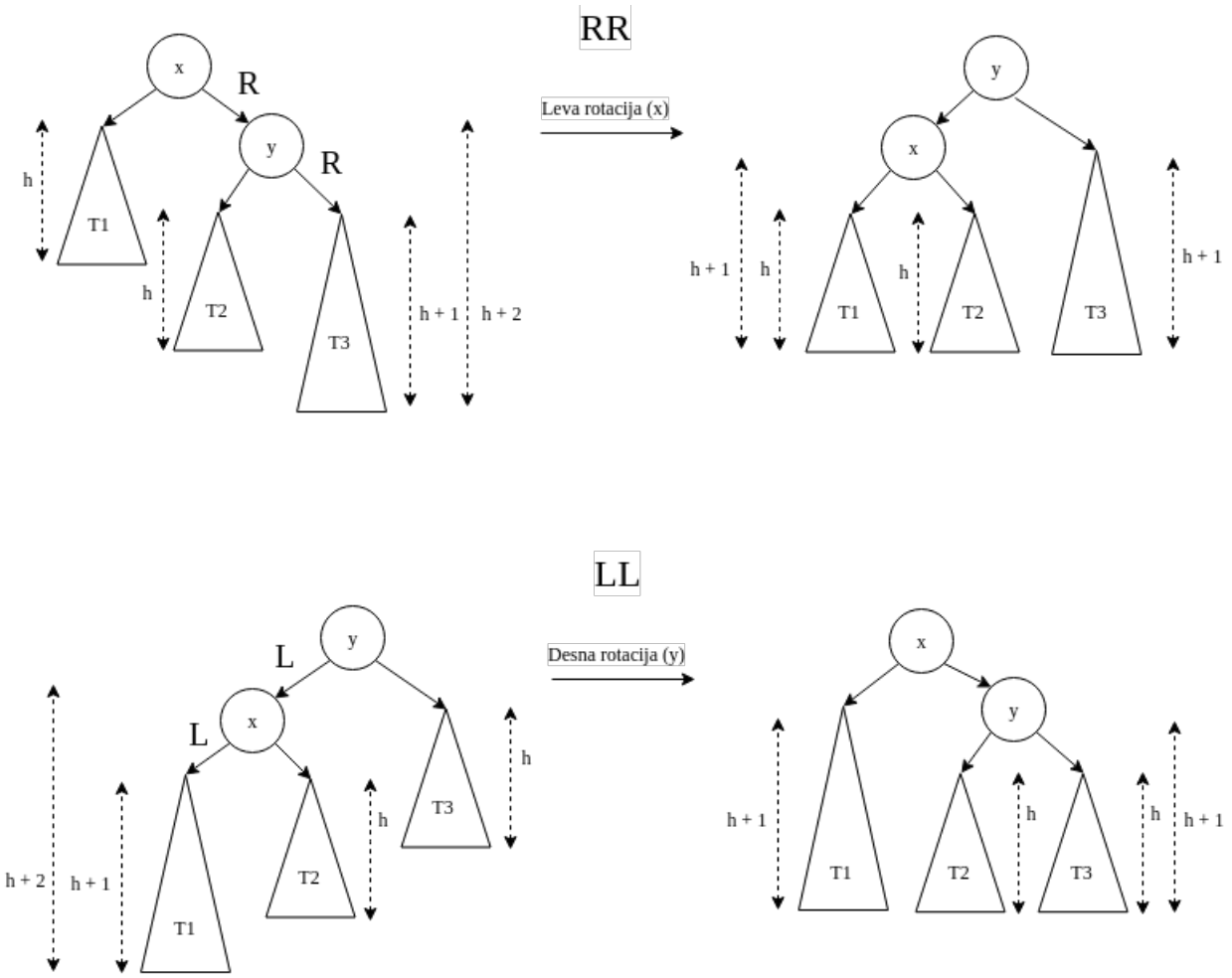
Ako je čvor nebalansiran tako da je njegovo desno podstablo više od levog postoje dve moguće situacije:

- Ako je desno podstablo desnog sina nebalansiranog čvora više od levog podstabla desnog sina nebalansiranog čvora taj slučaj se označava sa **RR** i popravljanje se ogleda u levoj rotaciji primenjenoj na nebalansiranom čvoru.
- U suprotnom, slučaj se označava sa **RL** i popravka podrazumeva desnu rotaciju nad levim sinom nebalansiranog čvora a nakon toga levu rotaciju nebalansiranog čvora.

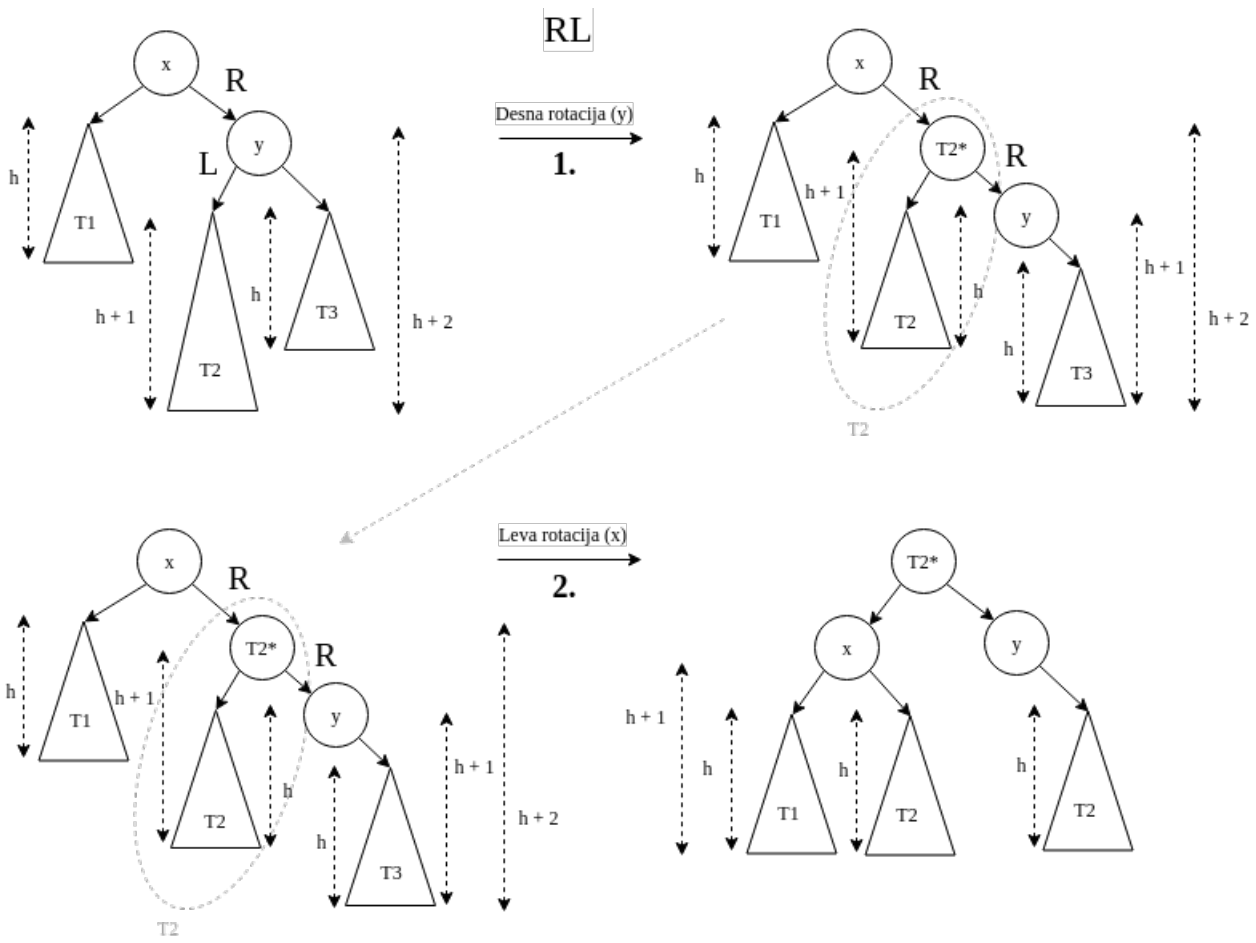
Ako je čvor nebalansiran tako da je njegovo levo podstablo više od desnog i tada postoje dve moguće situacije:

- Ako je levo podstablo levog sina nebalansiranog čvora više od desnog podstabla levog sina nebalansiranog čvora taj slučaj se označava sa **LL** i popravljanje se vrši desnom rotacijom nebalansiranog čvora.
- U suprotnom, slučaj se označava sa **LR** i popravka podrazumeva levu rotaciju nad desnim sinom nebalansiranog čvora a nakon toga desnu rotaciju nebalansiranog čvora.

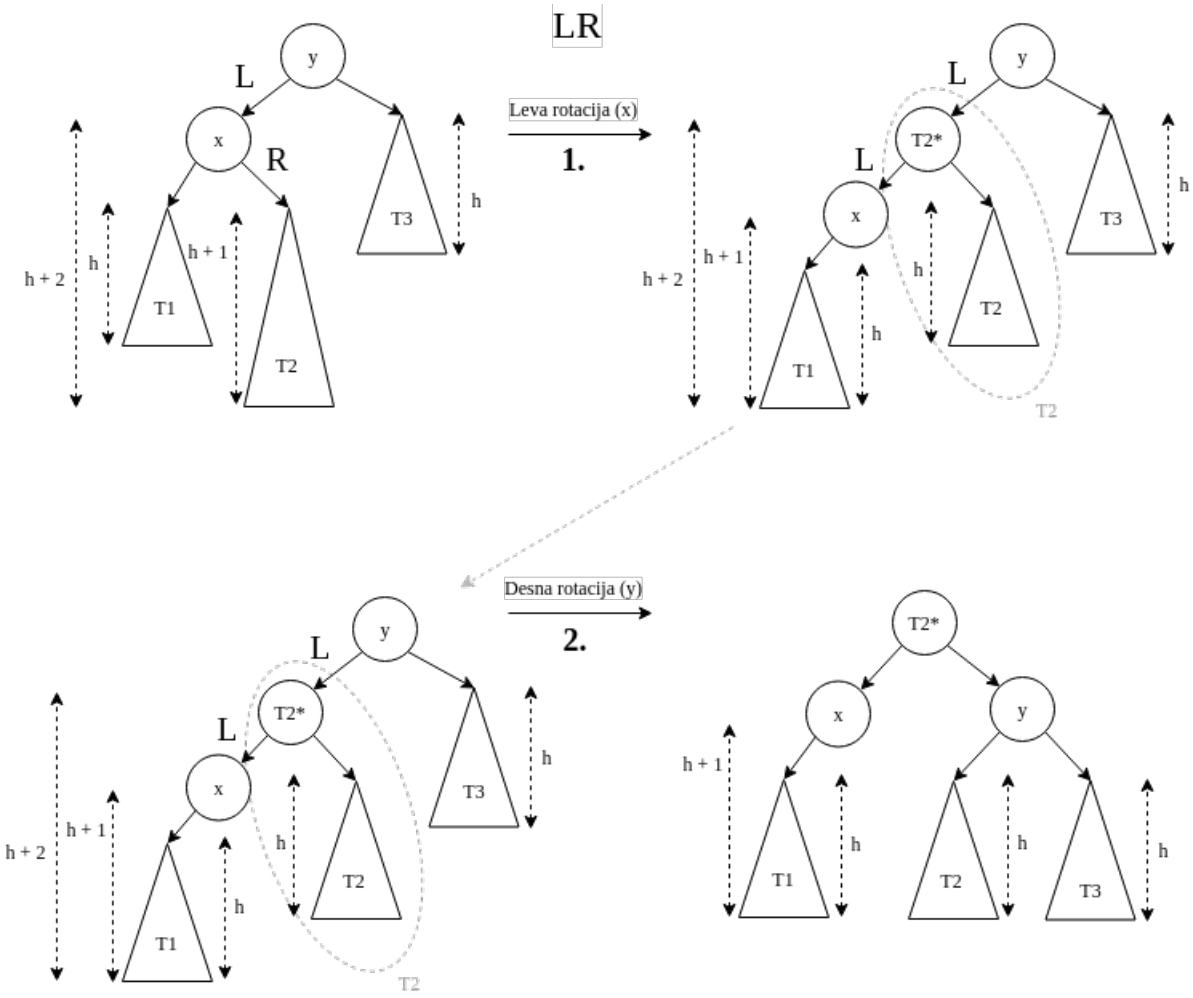
Primeri rotacija prikazani su na slikama 2, 3 i 4.



Slika 2. Popravke RR i LL slučajeva



Slika 3. Popravka RL slučaja



Slika 4. Popravka LR slučaja

4 Struktura set

Struktura **set** definisana je u biblioteci <set>. Implementirana je pomoću samobalansirajućeg stabla i predstavlja apstrakciju skupa elemenata bez ponavljanja. Elementi su smešteni u samobalansirajuće stablo i iteracijom kroz strukturu vrši se LKD obilazak stabla. Struktura **set** podržava dodavanje (**insert()**), uklanjanje (**erase**), pretraživanje (**find()**) i obilazak elemenata u skupu.

Primer:

```
set<int> skup; // Skup ce sadrzati elemente tipa int

skup.insert(5);
skup.insert(4);
skup.insert(1);
skup.insert(2);
skup.insert(2); // Duplikat se nece sacuvati dva puta u skupu
skup.insert(3);

skup.erase(5); // Uklanjanje elementa 5
skup.erase(skup.begin()) // Uklanjanje prvog elementa u skupu


for(auto it = skup.begin(); it != skup.end(); it++)
```

```
    cout << *it << " ";

// Ispis: 2 3 4

auto it = skup.find(3); // Funkcija find vraca iterator na trazeni element,
                        // ako je vrednost pronadjena, inace vraca skup.end() iterator
```

5 Struktura map

Struktura **map** definisana je u biblioteci <map>. Kao i set, implementirana je pomoću samobalansiraju' ceg stabla, ali su vrednosti koje se čuvaju u strukturi nešto drugačije. Elementi koji se čuvaju u strukturi su podeljeni na dva dela, ključ koji se smešta u stablo i služi za pretraživanje i vrednost koja je pridružena ključu. Podržane su operacije dodavanja (**insert()**), uklanjanja (**erase**), pretraživanje (**find()**) i obilazak elemenata u u strukturi. Obilazak se vrši LKD redosledom u odnosu na ključeve smeštene u stablo. Iterator na element ove sturkture predstavlja uredjeni par, ciji je prvi element (**first**) ključ a drugi (**second**) vrednost pridružena ključu. Dodavanje novih elemenata u strukturu **map** i pristup elementima moguć je i korišćenjem operatora [ ].

Primer:

```
map<int, string> mapa; // Kljucevi mape ce biti tipa int dok ce vrednosti
                       // pridruzene kljucevima biti tipa string

mapa[1] = "Prvi element";
mapa.insert ( pair<int,string>(2,"Drugi element") );
mapa[3] = "Treci element";
mapa[4] = "Cetvrti element";
mapa[5] = "Peti element";

mapa.erase(3); // Uklanjanje elementa sa kljucem 3
mapa.erase(mapa.begin()) // Uklanjanje prvog elementa u mapi, u ovom slucaju elementa
                         // sa kljucem 1

for(auto it = mapa.begin(); it != mapa.end(); it++)
    cout << it->first << " => " << it->sedond << endl;

// Ispis: 2 => Drugi element
//        4 => Cetvrti element
//        5 => Peti element

cout << mapa[2]; // Ispis: Drugi element

mapa.find(2); // Funkcija find vraca iterator na trazeni element zadatim kljucem,
              // ako vrednost nije pronadjena funkcija vraca iterator mapa.end()
```