Implementacija AVL stabla

Seminarski rad u okviru kursa
Konstrukcija i analiza algoritama 2
Matematički fakultet

Nevena Nikolić, 1021/2018
Aleksandar Anžel, 1025/2018

21. decembar 2019

Sažetak

U ovom radu će biti detaljnije opisani implementirani algoritmi za rad sa AVL stablima, kao i analiza njihove složenosti.

Sadržaj

1 Algoritmi

AVL stablo je binarno stablo pretrage za koje važi sledeće: za svaki čvor, apsolutna vrednost razlike visina njegovog levog i desnog podstabla je najviše 1. Visina takvog stabla je proporcionalna logaritmu broja čvorova, pa su i operacije pretrage, umetanja i brisanja nad takvim stablom logaritamske složenosti.

1.1 Struktura

```
typedef struct cvor
{
        int kljuc;
        unsigned visina;
        unsigned br_potomaka;
        struct cvor *levo;
        struct cvor *desno;
} Cvor;
```

Čvor AVL stabla predstavljamo strukturom podataka čija su polja ceo broj *kljuc* i pokazivači *levo* i *desno*, koji pokazuju na njegovog levog i desnog potomka. Uz ova osnovna polja, čuvamo još dva podatka: *visina* (visina podstabla čiji je koren dati čvor) i *br_potomaka* (broj čvorova u podstablu čiji je koren dati čvor, uključujući i njega). Inicjalne vrednosti ovih polja za novokreirani čvor jesu: $NULL$ za pokazivače *levo* i *desno*, 1 za visinu i broj potomaka tog čvora, dok se $kljuc$ postavlja na vrednost prosleđenu funkciji *novi_cvor*, u kojoj i vršimo navedene inicijalizacije.

1.2 Rotacije

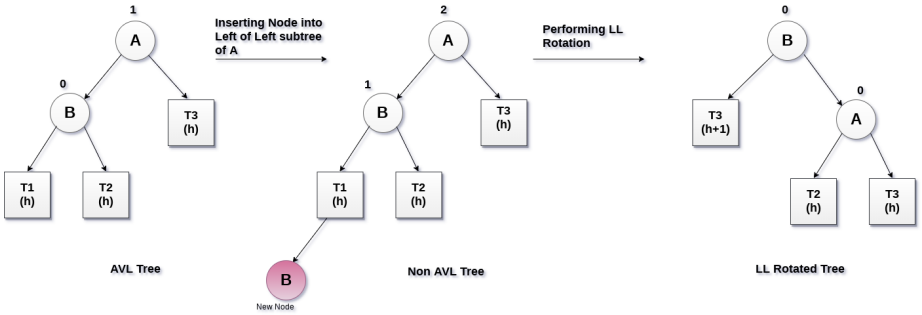
Prilikom umetanja i brisanja čvora iz AVL stabla, može se poremetiti uravnoteženost, pa je potrebno izvršiti određene modifikacije kako bi se stablo balansiralo, uz očuvanje složenosti operacija. Ove modifikacije zovemo rotacijama i razlikujemo 4 tipa:

- LL rotacija
- RR rotacija
- LR rotacija
- RL rotacija

Svaka rotacija je konstantne složenosti.

LL rotaciju potrebno je izvršiti kada se novi čvor dodaje u levo podstablo levog podstabla kritičnog čvora (slika 1).

Kod u nastavku predstavlja implementaciju gore navedenog postupka. Potrebno je ažurirati brojeve potomaka i visine, a sama rotacija se sastoji od dva preusmeravanja pokazivača.



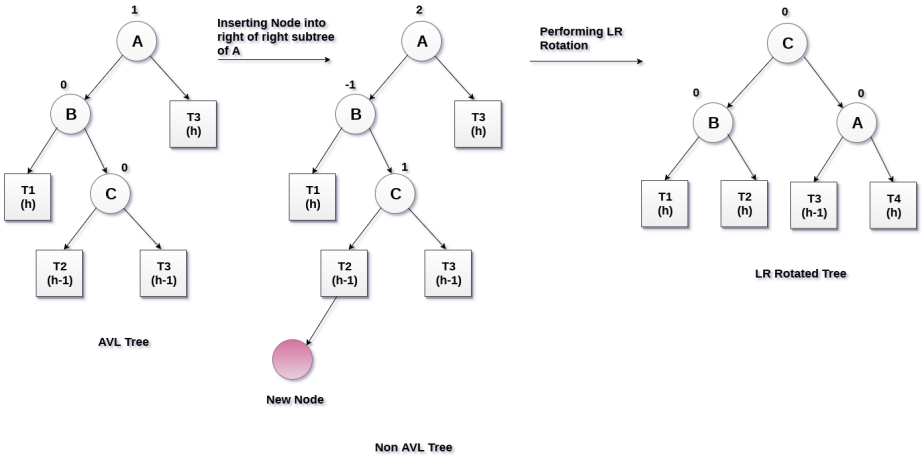
Slika 1: LL rotacija.

```
Cvor *LL_rotacija (Cvor *r)
{
        Cvor *x = r->levo;
        Cvor *tmp = x->desno;

        unsigned x_levo_br = 0;
        unsigned r_desno_br = 0;

        if(x->levo != NULL)
                x_levo_br = x->levo->br_potomaka;

        if(r->desno != NULL)
                r_desno_br = r->desno->br_potomaka;

        /* Azuriranje broja potomaka */
        r->br_potomaka -= 1 + x_levo_br;
        x->br_potomaka += r_desno_br + 1;

        /* Rotiranje */
        x->desno = r;
        r->levo = tmp;

        /* Azuriranje visina */
        r->visina = max(visina_stabla(r->levo),
            visina_stabla(r->desno))+1;
        x->visina = max(visina_stabla(x->levo),
            visina_stabla(x->desno))+1;

        /* Vracanje novog korena */
        return x;
}
```

RR rotaciju potrebno je izvršiti kada se novi čvor dodaje u desno podstablo desnog podstabla kritičnog čvora. Postupak je sličan LL rotaciji, s tim što sve posmatramo simetrično (kao u ogledalu).

Prethodno opisane rotacije spadaju u tzv. jednostruke rotacije. U nastavku opisujemo dvostruke rotacije.

LR rotaciju potrebno je izvršiti kada se novi čvor dodaje u desno podstablo levog podstabla kritičnog čvora (slika 2).



Slika 2: LR rotacija.

Primetimo da se svaka dvostruka rotacija može izvršiti uzastopnom primenom dve jednostruke rotacije.

Specijalno, LR rotaciju izvršavamo tako što najpre izvršimo RR (levu) rotaciju nad levim sinom kritičnog čvora, a potom LL (desnu) rotaciju nad samim kritičnim čvorom (kod u nastavku).

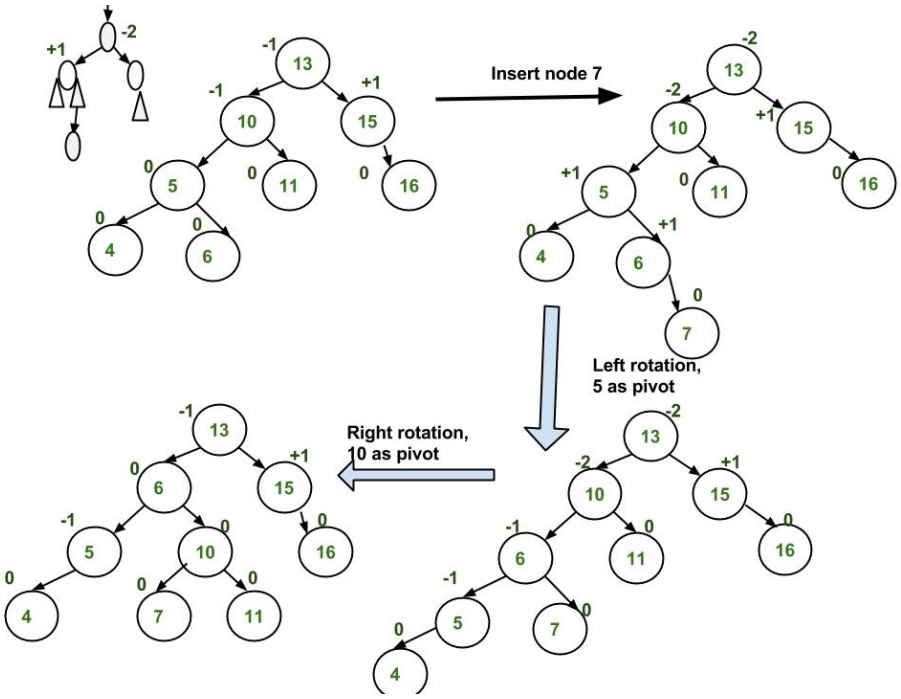
```
Cvor *LR_rotacija (Cvor *r)
{
        r->levo = RR_rotacija(r->levo);
        return LL_rotacija(r);
}
```

Pokažimo korektnost ovog postupka na jednom primeru (slika 3).

RL rotaciju potrebno je izvršiti kada se novi čvor dodaje u levo podstablo desnog podstabla kritičnog čvora. Opet, postupak je sličan LR rotaciji, s tim što sve posmatramo simetrično (kao u ogledalu).

1.3 Pretraga, umetanje i brisanje

Operacije pretrage umetanja i brisanja se izvode isto kao i kod svakog binarnog stabla pretrage s tim što se prilikom umetanja i brisanja izvode i dodatne operacije balansiranja stabla, opisane u prethodnom poglavlju.



Slika 3: Analiza korektnosti postupka rotiranja.

1.4 Naredni (x,k)

Pretpostavimo da želimo da nađemo k-ti najmanji ključ među onima koji su veći od datog ključa x a da pri tome zadržimo logaritamsku složenost od broja čvorova. Za ove potrebe kao dodatnu informaciju čuvali smo i broj potomaka za svaki čvor. Uvedimo pojam **ranga**: rang čvora v predstavlja broj čvorova čije su vrednosti ključeva manji ili jednaki od vrednosti ključa čvora v. Rang nekog čvora se može sračunati na osnovu ranga oca i informacija o broju potomaka. Pokažimo ovo induktivno.

- **Baza indukcije**: Primetimo da je rang korena ili jednak 1 (ako koren nema levog potomka) ili jednak $koren \rightarrow levo \rightarrow br\_potomaka + 1$.
- **Induktivna pretpostavka**: Pretpostavimo da znamo rangove krenuvši od korena do nekog čvora w.
- **Induktivni korak**: Sračunajmo rangove levog (w_l) i desnog (w_d) sina čvora w. Označimo desnog sina čvora w_l sa w_{ld}, a levog sina čvora w_d sa w_{dl}.
  Rang čvora w_l je jednak $rang(w) - 1 - (w_{ld} \rightarrow br\_potomaka)$, dok je rang čvora w_d jednak $rang(w) + 1 + (w_{dl} \rightarrow br\_potomaka)$.

Implementacija opisanog postupka je data u nastavku.

```
unsigned vrati_rang (Cvor *r, int kljuc)
{
  unsigned rang = 1;  /* Broj cvorova sa kljucem
      manjim ili jednakim kljuca korena */

  /* Ako koren ima levog sina... */
  if (r->levo != NULL)
    rang += r->levo->br_potomaka;

  while (r->kljuc != kljuc)
  {
    if(r->kljuc > kljuc) /* trazeni element je u
        levom podstablu, racunamo rang levog sina */
    {
      if (r->levo->desno != NULL)
        rang -= 1 + r->levo->desno->br_potomaka;
      else
        rang -=1;

      r = r->levo;
    }

    else if (r->kljuc < kljuc)
    {
      if (r->desno->levo != NULL)
        rang += 1 + r->desno->levo->br_potomaka;
      else
        rang += 1;

      r = r->desno;
    }
  }

  return rang;
}
```

Primetimo da je AVL stablo, osim po ključevima, sortirano po rangovima. Ova činjenica nam omogućava efikasnu pretragu stabla i po rangu a ne samo po vrednostima ključeva. Čuvali smo informaciju o broju potomaka svakog čvora a ne o rangu jer ažuriranje rangova pri umetanju i brisanju remeti traženu složenost.

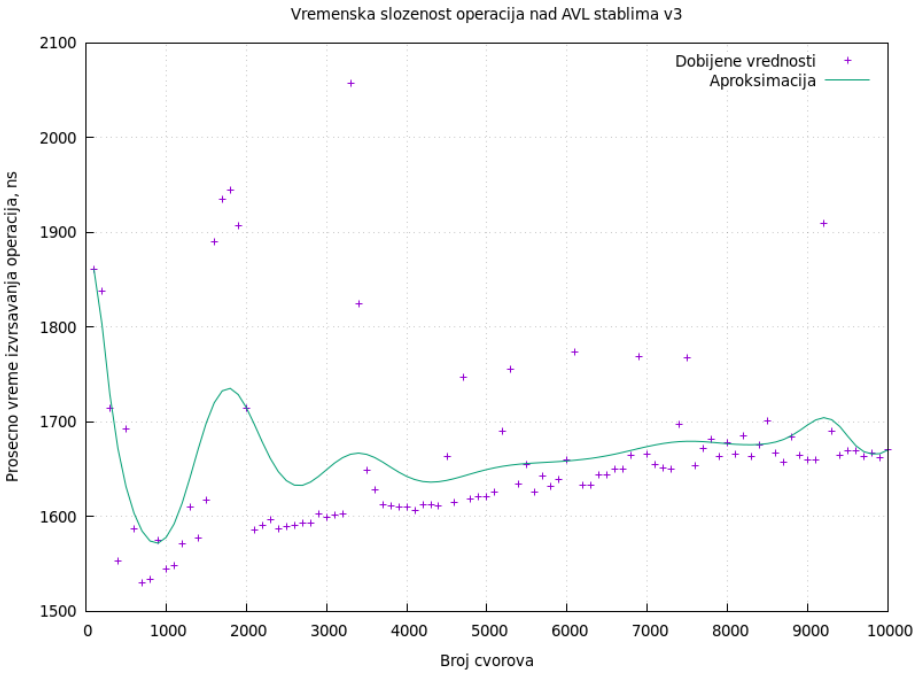
Za implementaciju funkcije *Naredni (x,k)* najpre treba pronaći čvor sa vrednošću ključa x u stablu i odrediti njegov rang, prethodno opisanim postupkom. Napomenimo da rangove ne čuvamo za svaki čvor na putu od korena stabla do čvora x, već ih samo koristimo za potrebna izračunavanja dok ne dođemo do čvora x, čiji rang treba zapamtiti. Traženi čvor je onaj čvor čiji je rang jednak $rang(x) + k$. Sada treba ponovo pretražiti stablo počevši od korena, ovaj put po rangu (kod ispod).

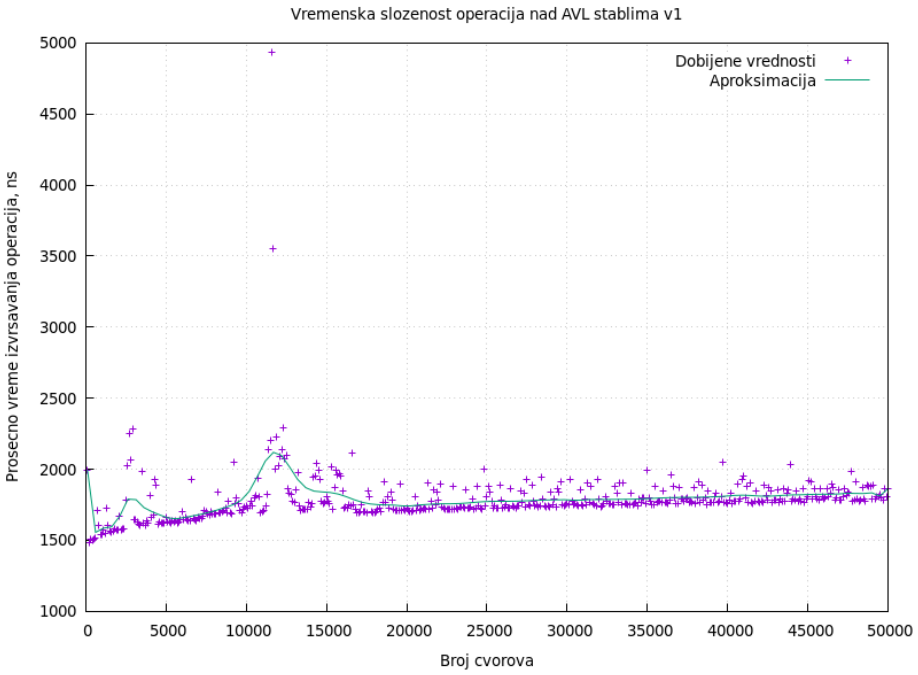
```
Cvor *pronadji_x_k (Cvor *r, int x, unsigned k)
{
  unsigned trazeni_rang = vrati_rang(r,x) + k;

  unsigned rang = 1;  /* Broj cvorova sa kljucem
      manjim ili jednakim kljuca korena */

  /* Ako koren ima levog sina... */
  if (r->levo != NULL)
    rang += r->levo->br_potomaka;

  while (trazeni_rang != rang)
  {
    if(trazeni_rang < rang)
    {
      if (r->levo == NULL)
        return NULL;

      if (r->levo->desno != NULL)
          rang -= 1 + r->levo->desno->br_potomaka;
      else
        rang -=1;

      r = r->levo;
    }

    else
    {
      if (r->desno == NULL)
        return NULL;

      if (r->desno->levo != NULL)
        rang += 1 + r->desno->levo->br_potomaka;
      else
        rang += 1;

      r = r->desno;
    }
  }

  return r;
}
```

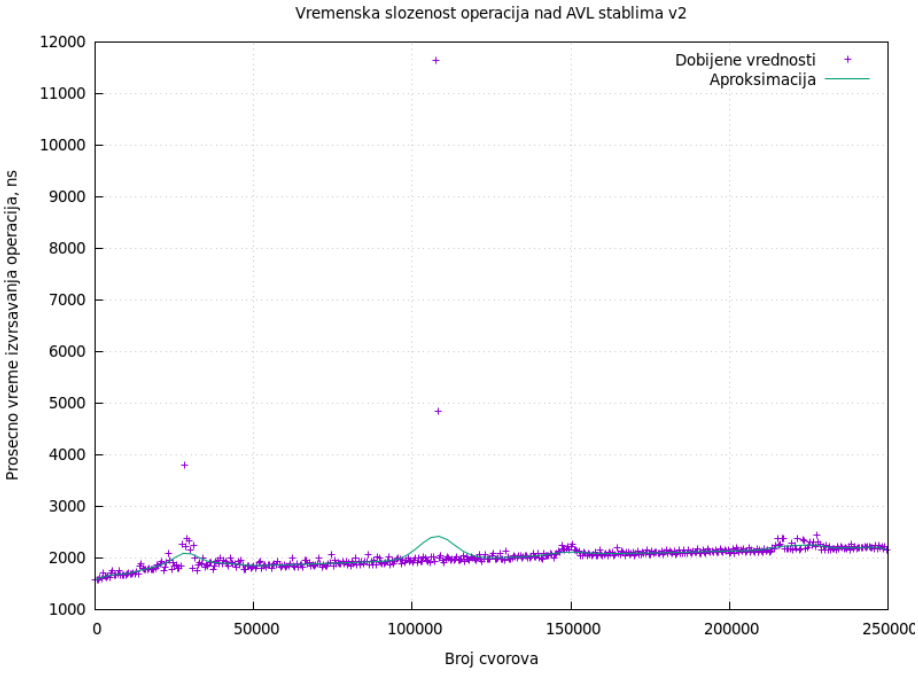
Može se desiti da čvor sa vrednošću ključa x ne postoji u stablu ili da traženi rang prevazilazi maksimalan rang stabla. Tada ispisujemo odgovarajuću poruku, kao što je prikazano u nastavku.

```
void naredni_x_k (Cvor *r, int x, unsigned k)
{
  if (pretrazi(r,x) == 0)
  {
    /* printf("Cvor sa kljucem %d nije pronadjen u
        stablu.\n",x); */
    return;
  }

  r = pronadji_x_k (r, x, k);

  if (r == NULL)
  {
    /* printf("Trazeni rang je van opsega.\n"); */
    return;
  }

  else
  {
    /* printf("Trazeni kljuc je %d.\n",r->kljuc); */
    return;
  }
}
```

2 Složenost




Vremenska slozenost operacija nad AVL stablima v1
Dobijene vrednosti
Aproksimacija
Prosecno vreme izvrsavanja operacija, ns
Broj cvorova
5000
4500
4000
3500
3000
2500
2000
1500
1000
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000


Vremenska slozenost operacija nad AVL stablima v2
Dobijene vrednosti
Aproksimacija
Prosecno vreme izvrsavanja operacija, ns
Broj cvorova
12000
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
50000
100000
150000
200000
250000