PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET METROPOLITAN BEOGRAD

KI202 - JAVA 5: STRUKTURE PODATAKA I ALGORITMI – DEO B

AVL Stabla

Lekcija 03

PRIRUČNIK ZA STUDENTE

KI202 - JAVA 5: STRUKTURE PODATAKA I ALGORITMI – DEO B

Lekcija 03

AVL STABLA

- AVL Stabla
- Poglavlje 1: AVL stabla
- Poglavlje 2: Dizajniranje klasa za AVL stablo
- Poglavlje 3: Brisanje elementa iz AVL stabla
- Poglavlje 4: Klasa AVLTree
- Poglavlje 5: Testiranje klase AVLTree
- Poglavlje 6: Vremenska analiza klase AVLTree
- Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

Uvod

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

U okviru ove lekcije studenti se upoznaju sa:

- Balansiranim stablima i postupcima balansiranja binarnih stabala pretraživanja.
- AVL stablima
- Postupcima dodavanja i brisanja elemenata pri radu sa AVL stablima.

U prethodnoj lekciji smo se upoynali sa binarnim stablima pretraživanja. Vreme za pretraživanje, ubacivanje i brisanje elemenata zavisi od visine stabla. U najgorem slučaju, visina stabla je O(n). Ali, ako je stablo idealno izbalansirano, tj. istovremeno je i kompetno binarno stablo, onda je visina stabla log n. Pitanje je da li stablo možemo da održavamo u formi idealno balansiranog? Da, ali će to zahtevati dodatne operacije koje mogu da budu skupe. Kompromis je da se stablo održavamo u formi dobro blansiranog stabla, tj. da su visine levog i desnog podstabla svakog čvora približno iste. Ovo ćemo moći da ostvarimo primenom AVL stabala.

AVL stabla su dobro izbalansirana stabla. AVL stabla su izumljena 1962. godine od strane dva ruska informatičara,, G. M. Adelson-Velsky i E. M. Landis (pa stoga naziv AVL). U AVL stablu, razlika visina podstabala nekog čvora je nula ili 1, i može se dokazati da je maksimalna visina AVL stabla reda O(log n).

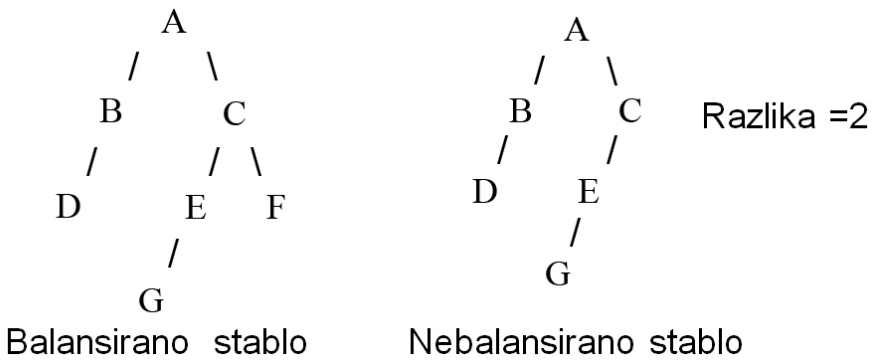
Poglavlje 1

AVL stabla

BALANSIRANJE STABLA

Stablo je balansirano ako se visine levog i desnog podstabla razlikuju maksimalno za 1 i kada su njegova leva i desna podstabla takođe balansirana

Definicija balansiranog stabla: za stablo se kaže da je balansirano (uravnoteženo), ako za svaki čvor važi da se broj nivoa u svakom njegovom podstablu ne razlikuje za više od jedan.

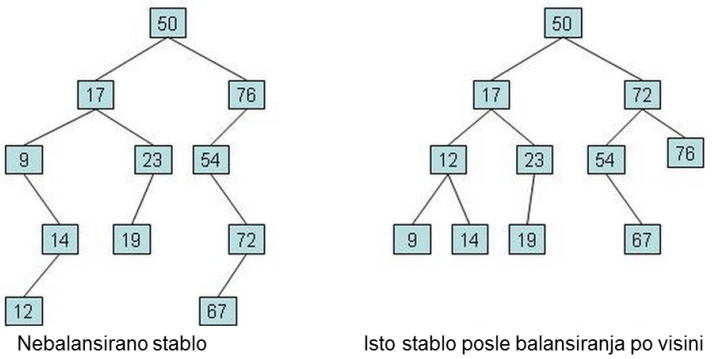


Slika 1.1 Balansirano i nebalansirano stablo

Na Slici-1 su primeri balansiranog i nebalansiranog stabla. Takođe može se reći da je stablo balansirano:

- Kada se visine levog i desnog podstabla razlikuju maksimalno za 1 i kada je:
- levo podstablo balansirano
- desno podstablo balansirano

Slika-2 prikazuje primer balansiranja nekog stabla.



Slika 1.2 Balansiranje stabla a) stablo pre balansiranja b) stablo usled balansiranja

Samo-balansirajuće binarno stablo pretraživanja (engl. self-balancing binary search tree) je bilo koje binarno stablo pretraživanja koje automatski održava malu vrednost svoje visine (maksimalan broj nivoa ispod korena) uprkos proizvoljnom umetanju i brisanju elemenata.

OSNOVI AVL STABLA

AVL stablo je BST koje ima sledeće karakteristike: a) podstablo svakog čvora se razlikuje u visini za tačno 1, b) svako podstablo AVL stabla je AVL stablo

Nedostatak opisanih binarnih stabala iz prethodne lekcije je što se brisanjem ili dodavanjem elemenata ona mogu deformisati i pretvoriti u listu (Slika-3). Stoga je ideja je da se prilikom svakog novog umetanja ili brisanja elementa iz stabla ono izbalansira (Slika-3).

AVL stablo (Georgy Adelson-Velsky i Landis) je samobalansirajuće binarno stablo pretraživanja. Osnovne karakteristike su:

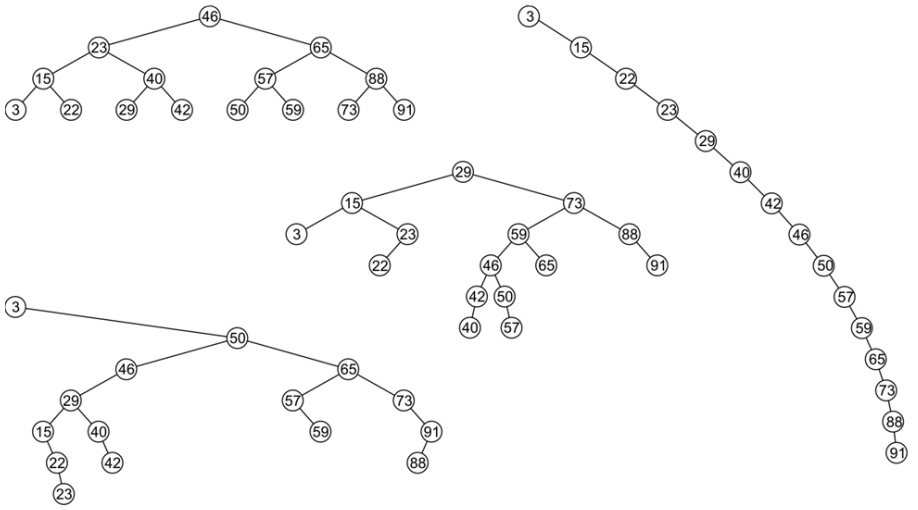
- U AVL stablu, visine dva podstabla deteta bilo kojeg čvora se mogu razlikovati samo za 1.
- Ako se u bilo kojem trenutku razlikuju za više od 1, vrši se rebalansiranje da se povrati tražena karakteristika AVL stabla.
- Svako podstablo AVL stabla je AVL stablo.

AVL stablo garantuje da su operacije umetanja, pretrage i brisanja elemenata iz stabla i u najgorem slučaju logaritamske složenosti, tj. O (log n).

Međutim, kod uobičajenog nebalansiranog binarnog stabla može da dođe do disbalansa, i onda pretraga koja ide kroz tu granu nema logaritamsku već linearnu složenost.

Balansiranost *AVL* stabla se održava tako što se nakon svakog umetanja ili brisanja proverava da li je došlo do gubitka balansiranosti na nekom delu stabla.

Ukoliko je došlo do disbalansa, identifikuje se kritični čvor, to jest koren najmanjeg podstabla koje ne zadovoljava definiciju *AVL* stabla. Ovo podstablo se zatim balansira vršenjem odgovarajućih rotacija elemenata, tako da se povrati uravnoteženost podstabla.



Slika 1.3 Primeri istog stabla, i slučaj kada se stablo pretvorilo u listu

VISINA ČVORA I PRIMERI AVL STABLA

Za svaki čvor AVL stabla visina levog i desnog podstabla se može razlikovati za ne više od 1. Razlika visina levog i desnog podstabla se naziva faktor uravnoteženosti (balance factor) stabla

Kod AVL stabala je uglavnom potrebno izračunati faktor uravnoteženosti stabla (engl. balance factor).

Faktor uravnoteženosti (balansa) nekog čvora T se računa kao:

```
BalanceFactor(T) = height(T.left) - height(T.right)
```

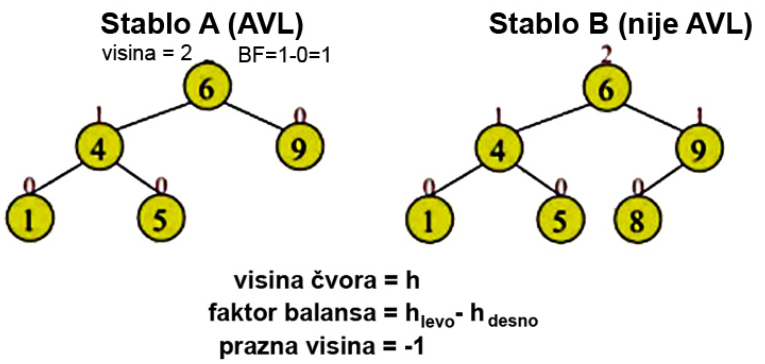
AVL stabla računaju *BalanceFactor* za svaki čvor. Za svaki čvor visina levog i desnog podstabla se može razlikovati za ne više od 1, tj.

$|BalanceFactor(T)| \leq 1$.

Stoga treba sačuvati visinu svakog čvora, tj. visina će biti dodatni podatak koji će se čuvati u čvoru stabla.

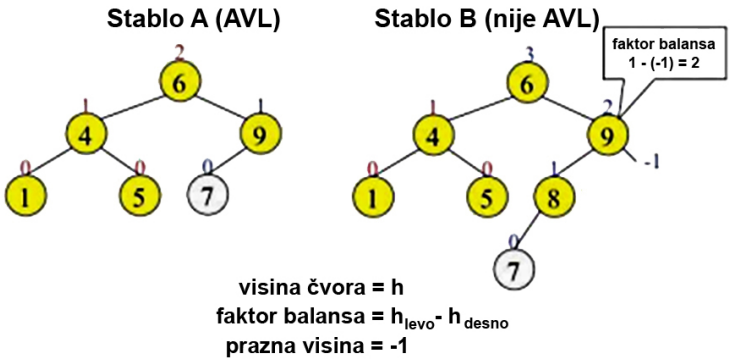
Visina čvora u stablu se može odrediti na sledeći način (Slika-4):

- Visina čvora koji nema decu je 0
- Visina čvora sa jednim potomkom je 1
- Visina čvora koje ima više od jednog potomka je jednaka 1 + visina najvišeg podstabla.



Slika 1.4 Definisanje visine čvora

Nakon umetanja novog čvora, visina se menja, i može doći do neuravnoteženosti stabla, Slika-5.



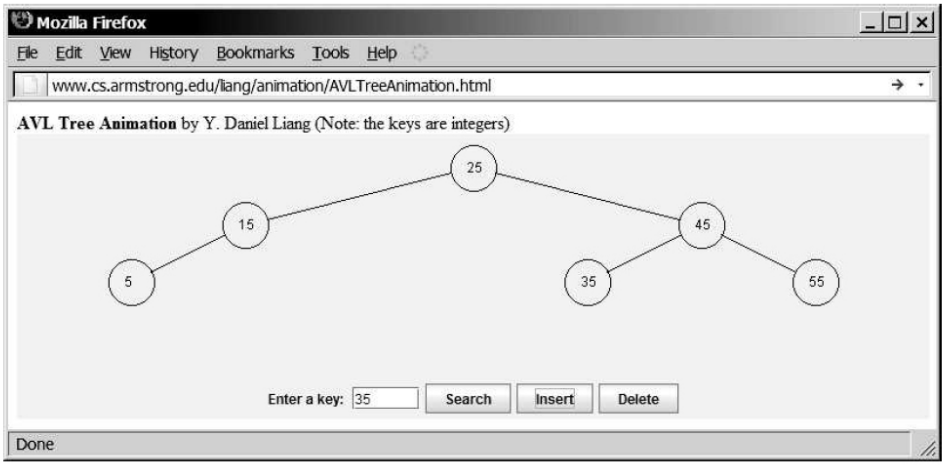
Slika 1.5 Visina čvora nakon umetanja

ANIMACIJA AVL STABLA

Uvid u interaktivnu aplikaciju koja ilustruje rad sa AVL stablom

Proces ubacivanja elementa u AVL stablo je isti kao i ubacivanje u obično binarno stablo pretrage osim što je nekad potrebno izvršiti rebalansiranje stabla. Razlika između levog i desnog podstabla se naziva **faktor uravnoteženosti.** Čvor je **uravnotežen** ako je njegov faktor uravnoteženosti -1, 0, or 1. Čvor se naziva levo-teškim ako je njegov faktor uravnoteženosti -1, a desno-teškim ako je +1.

Na sledećem linku se nalazi interaktivna aplikacija koja ilustruje rad sa AVL stablom www.cs.armstrong.edu/liang/animation/web/AVLTree.html , kao što je i prikazano na slici ispod.



Slika 1.6 Interaktivna aplikacija koja omogućava pretragu, ubacivanje i brisanje elementa AVL stabla

DESNA (LL) I LEVA (RR) ROTACIJA AVL STABLA

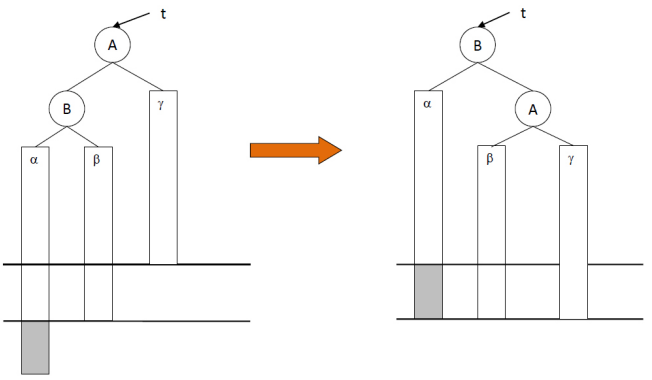
LL neuravnoteženost nastaje u čvoru A, gde čvor ima faktor balansa manji od -1 a njegovo levo dete B ima faktor balansa -1 ili 0. RR neuravnoteženost nastaje u obrnutom slučaju

Rotacije kod AVL stabla

Ukoliko čvor nije neuravnotežen, nakon operacije dodavanja ili brisanja, neophodno ga je izbalansirati. Proces balansiranja čvora se naziva rotacija (rotation). Postoji četiri različite rotacije: LL (desna rotacija), RR (leva rotacija), LR (dvostruka desna rotacija), i RL (dvostruka leva rotacija).

LL rotacija

LL neuravnoteženost nastaje u čvoru *A*, gde čvor ima *faktor balansa* manji od -1(npr. **-2)** a njegovo levo dete *B* ima faktor balansa **-1** ili **0** (Slika-7a). Ovaj tip neuravnoteženosti može biti korigovan korišćenjem jednostavne desne rotacije u čvoru *A*, kao što je prikazano na Slici-7b.



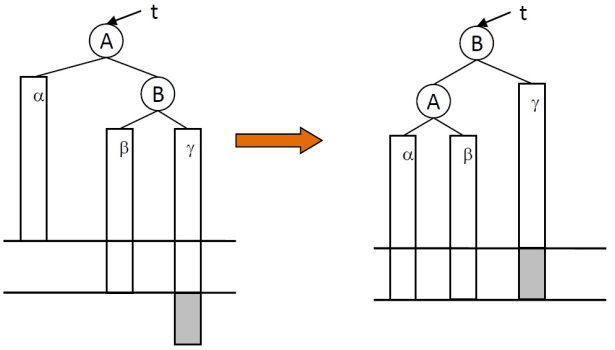
Slika 1.7 Desna rotacija (LL)

Desna rotacija AVL stabla se vrši kada je ispunjen uslov:

```
(t.balans < -1) &amp;&amp; (t.levi.balans < 0)
```

RR rotacija

RR neuravnoteženost nastaje u čvoru *A*, kada čvor ***A*** ima faktor balansa veći od 1 (+2) a njegovo desno dete ***B*** ima faktor balansa **+1** ili **0** (Slika-8a). Ovaj tip neuravnoteženosti može biti korigovan korišćenjem jednostavne leve rotacije u čvoru *A*, kao što je prikazano na Slici-8b.



Slika 1.8 Leva rotacija

Leva rotacija AVL stabla se vrši kada je ispunjen uslov:

```
(t.balans > 1) &amp;&amp; (t.desni.balans > 0)
```

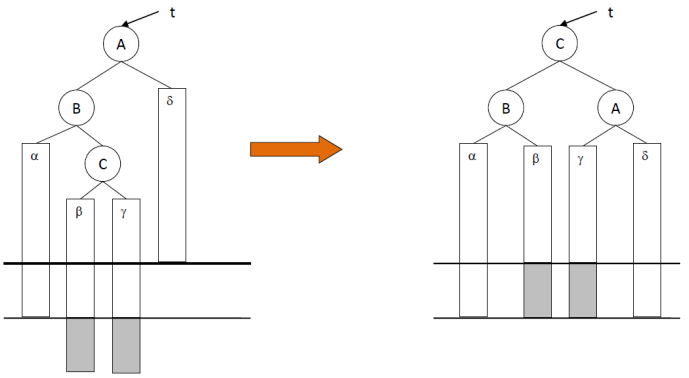
DVOSTRUKA DESNA (LR) I DVOSTRUKA LEVA ROTACIJA (RL)

LR neuravnoteženost nastaje u čvoru A, kada čvor A ima faktor balansa -2, a njegovo levo dete B ima faktor balansa +1. RL neuravnoteženost nastaje u obrnutom slučaju

LR rotacija

LR neuravnoteženost nastaje u čvoru *A*, kada čvor *A* ima faktor balansa -2, a njegovo levo dete *B* ima faktor balansa +1 (Slika-9a).

Pretpostavimo da je desni potomak čvora *B* čvor *C*. Ovaj tip neuravnoteženosti može biti korigovan korišćenjem dvostruke rotacije u čvoru A (prvo se vrši leva rotacija u čvoru B a zatim desna rotacija u čvoru A), kao što je prikazano na Slici-9b.



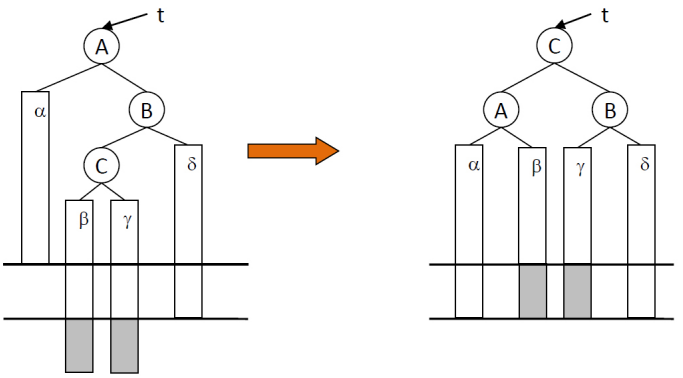
Slika 1.9 Dvostruka desna rotacija

Dvostruka desna rotacija AVL stabla se vrši kada je ispunjen uslov:

```
(t.balans < -1) &amp;&amp; !(t.levi.balans < 0)
```

RL rotacija

RL neuravnoteženost nastaje u čvoru *A*, kada čvor *A* ima faktor balansa +2 a njegovo desno dete *B* ima faktor balansa -1 (Slika-10a). Pretpostavimo da je levi potomak čvora *B* čvor *C*. Ovaj tip neuravnoteženosti može biti korigovan korišćenjem dvostruke leve rotacije u čvoru *A* (prvo se vrši desna rotacija u čvoru *B* a zatim leva rotacija u čvoru *A*), kao što je prikazano na Slici-10b.



Dvostruka leva rotacija AVL stabla se vršu u slučaju da je:

```
(t.balans > 1) &amp;&amp; !(t.desni.balans > 0)
```

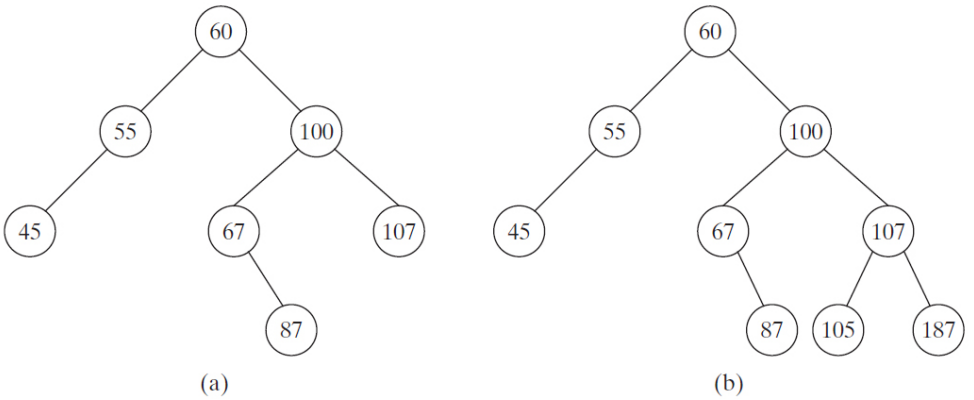
ZADACI ZA SAMOSTALNI RAD

Uraditi samostalno sledeće zadatke

Zadatak 1. Šta je AVL stablo? Opisati sledeće termine: faktor uravnoteženosti, levo-težak i desno-težak čvor.

Zadatak 2. Prikazati faktore uravnoteženosti za svaki čvor na slici **Slika-11.**

Zadatak 3. Opisati LL rotaciju, RR rotaciju, LR rotaciju i RL rotaciju za AVL stablo.



Slika 1.10 Faktor uravnoteženosti određuje da li je čvor uravnotežen

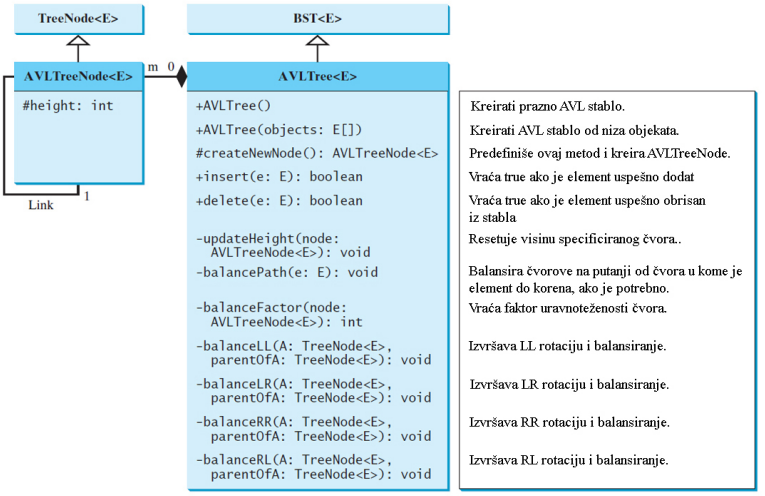
Poglavlje 2

Dizajniranje klasa za AVL stablo

HIJERARHIJA INTERFEJSA I KLASA

Pošto je AVL stablo ustvari binarno stablo pretraživanja, klasa AVLTree može biti dizajnirana kao podklasa koja nasleđuje osobine klase BST

AVL stablo je binarno stablo, i stoga možemo definisati klasu *AVLTree* koja proširuje klasu *BST*, kao što je prikazano na Slici-1. Klase *BST* i *TreeNode* smo definisali u prethodnoj lekciji o binarnim stablima pretraživanja (*L03 – Stabla I*).



Slika 2.1 Klasa AVLTree priširuje klasu BST sa novim implementacijama metoda insert i delete.

KLASE AVLTREE I AVLTREENODE

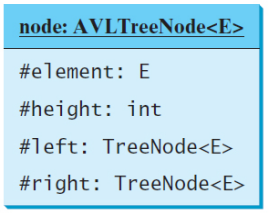
Klasa AVLTreeNode ima kao dodatak visinu čvora u odnosu na klasu TreeNode. Klasa AVLTree priširuje klasu BST sa novim implementacijama metoda insert i delete

U cilju balansirana stabla, neophodno je poznavati visinu svakog čvora.

Stoga, radi lakšeg računanja, sačuvati visine svakog čvora u okviru klase *AVLTreeNode* i definisati da *AVLTreeNode* bude podklasa klase *BST.TreeNode*.

Primetimo da je klasa *TreeNode* definisana kao statička unutrašnja klasa u okviru klase *BST*. Klasa *AVLTreeNode* će biti definisana kao statička unutrašnja klasa klase *AVLTree*. *TreeNode*

sadrži polja *element*, *left*, i *right*, koja su nasleđena i u *AVLTreeNode*. Stoga, *AVLTreeNode* sadrži četiri polja, kao što je prikazano na Slici-2.



Slika 2.2 Klasa AVLTreeNode sadrži zaštićene (protected) podatke članove element, height, left, i right.

U klasi *BST*, metod *createNewNode()* kreira objekat tipa *TreeNode*. Ovaj metod je nadglasan (predefinisan) u klasi *AVLTree* u cilju kreiranja objekta tipa *AVLTreeNode*.

Pretraživanje elementa u klasi *AVLTree* se odvija na isti način kao i u uobičajenom binarnom stablu, pa stoga metod *search* definisan unutar klase *BST* može biti korišćen i kod klase *AVLTree*.

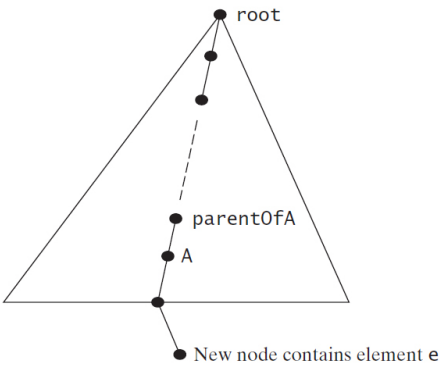
Metodi *insert* i *delete* su nadglasani (overridden) kako bi se prilikom brisanja i ubacivanja elementa izvršile operacija rebalansiranja, kako bi obezbedili da na kraju prethodnih operacija ukupno stablo bude uravnoteženo.

UBACIVANJE ELEMENTA U AVL STABLO

Ubacivanje elementa u AVL stablo se vrši na isti način kao i dodavanje elementa u BST stablo, s tim što na kraju, ukoliko je neophodno, treba izvršiti i uravnotežavanje

Novi element se u stablo uvek dodaje kao list stabla. Kao rezultat dodavanja novog čvora, može se povećati visina nekog od predaka novog čvora. Stoga, nakon dodavanja novog čvora u stablo, neopodno je proveriti sve čvorove u putanji od novog čvora do korena, i izvršiti odgovarajuće rotacije ako je došlo do poremećaja ravnoteže AVL stabla.

Algoritam koji vrši balansiranje putanje od novog čvora do korena je dat u nastavku.



Slika 2.3 Čvorovi na putanji od novog lista do korena mogu da postanu disbalansirani

```
balancePath(E e) {
 Odrediti cvorove na putanji od elementa e do korena stabla root;
 for each cvor A u putanji od e do root {
  Azuriraj visinu cvora A;
  Neka je parentOfA roditelj cvora A,
   koji je sledeci cvor u putanji, ili null ako je A koren (root);

  switch (balanceFactor(A)) {
   case -2:
   if balanceFactor(A.left) == -1 ili 0
     Izvrsiti LL rotaciju;
    else
     Izvrsiti LR rotaciju;
    break;
   case +2:
    if balanceFactor(A.right) == +1 ili 0
     Izvrsiti RR rotation;
    else
     Izvrsiti RL rotaciju;
  } // End of switch
 } // End of for
} // End of method
```

Algoritam proverava sve čvorove na putanji od novo ubačenog čvora do korena. Pritom se vrši ažuriranje visina čvorova na putanji. Ukoliko je čvor u ravnoteži nikakva se akcija ne preduzima. Ukoliko čvor nije u ravnoteži vrši se odgovarajuća rotacija.

IMLEMENTACIJA ROTACIJA

Prilikom izvršenja rotacije, koja se vrši nad čvorovima A i B, visina ovih čvorova može biti promenljena, ali se visina ostalih čvorova u stablu ne menja

U sekciji *Balansiranje AVL stabla*, ilustrovali smo moguće rotacije nad neuravnoteženim čvorovima stabla. U nastavku je dat algoritam za Desnu (LL) rotaciju AVL stabla.

```
balanceLL(TreeNode A, TreeNode parentOfA) {
 Neka je B levi potomak cvora A.

 if (A je koren)
  Neka B postane new root (koren)
 else {
  if (A je levi potomak cvora parentOfA)
   Neka B postane  levi potomak od parentOfA;
  else
   Neka B postane desni potomak od parentOfA;
 }

 Postaviti da desni potomak čvora B postane levi potomak cvora A, dodeljujući
```

```
pokazivaču A.left vrednost B.right;
 Neka A postane desni potomak čvora B, dodeljujući referenci B.right vrednost A;
 Azurirati visine cvorova A i B;
} // End of method
```

Primetimo da visina čvorova *A* i *B* može biti promenljena, ali visina ostalih čvorova u stablu se ne menja.

Na sličan način je moguće implementirati RR, LR, i RL rotaciju stabla. Kompletna implementacija ovih algoritama je ostavljena za vežbe.

ZADACI ZA SAMOPROVERU

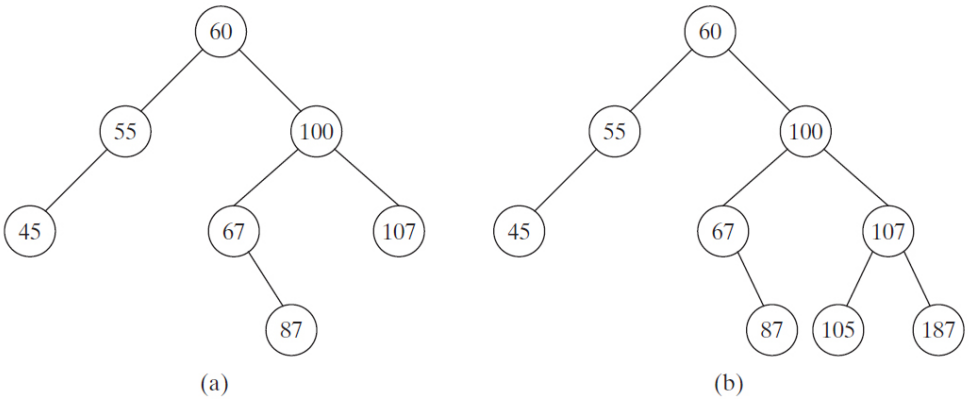
Uraditi samostalno sledeće zadatke

Zadatak 1. Koji su atributi klase *AVLTreeNode* ?

Zadatak 2. Da li je *AVLTreeNode* potklasa klase *TreeNode*?

Zadatak 3. Da li je *AVLTree* potklasa klase *BST*.

Zadatak 4. Za stablo sa **Slika 4a**, prikazati novo AVL stablo posle ubacivanja elementa 40. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?
Zadatak 5. Za stablo sa **Slika 4a**, prikazati novo stablo posle ubacivanja elementa 50. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?



Slika 2.4 Faktor uravnoteženosti određuje da li je čvor uravnotežen

Zadatak 6. Za stablo sa **Slika 4b,** prikazati novo stablo posle ubacivanja elementa 80. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?
Zadatak 7. Za stablo sa **Slika 4b,** prikazati novo stablo posle ubacivanja elementa 89. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?

Poglavlje 3

Brisanje elementa iz AVL stabla

OSNOVI O BRISANJU ELEMENATA IZ AVL STABLA

Brisanje iz AVL stabla se vrši kao i brisanje elemenata iz BST stabla. Ono što dodatno treba uraditi jeste provera ravnoteže stabla i eventualno balansiranje dela stabla

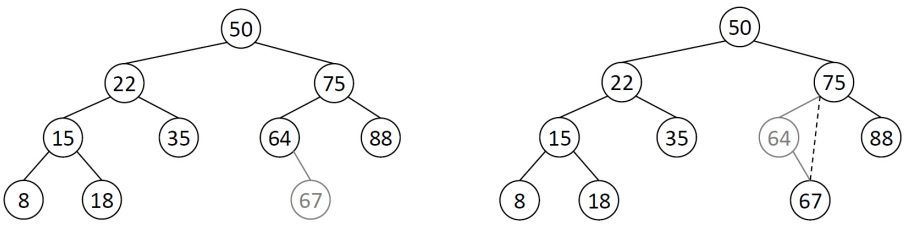
Kao što smo opisali u prethodnoj lekciji o binarnim stablima, da bi obrisali element iz stabla neophodno je locirati njegovog roditelja. Tamo je sa *current* označen element koji se briše a *parent* pokazuje na roditelja čvora *current*.

Čvor *current* može biti levi ili desni potomak čvora *parent*. Imali smo dva slučaja kod brisanja elementa.

- **Slučaj 1:** Čvor *current* nema levog potomka. Da bi obrisali čvor *current* jednostavno povezati čvor parent čvor sa desnim potomkom čvora *current*.
- ***Slučaj 2:*** Čvor *current* ima levog potomka. Neka *rightMost* pokazuje na čvor koji sadrži najveći element levog podstabla čvora *current* i neka *parentOfRightMost* pokazuje na roditelja čvora *rightMost*. Čvor *rightMost* ne može da ima desnog potomka ali može da ima levog potomka. Zameniti vrednost čvora *current* vrednošću koja se nalazi u čvoru *rightMost*, povezati čvor *parentOfRightMost* sa levim potomkom čvora *rightMost*, i na kraju obrisati čvor *rightMost*.

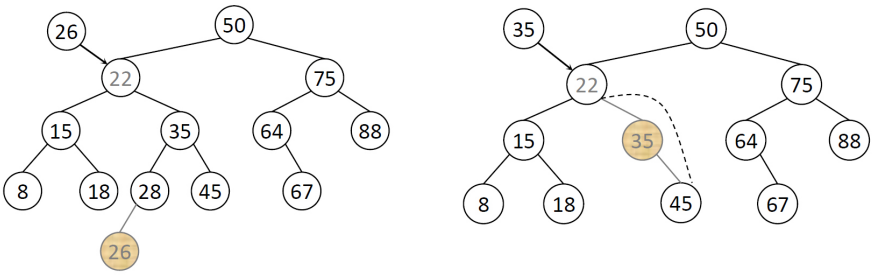
Ono što treba dodatno uraditi nakon brisanja čvora jesta provera ravnoteže stabla. U nastavku su primeri brisanja elementa iz *AVL* stabla a zatim i balansiranja u slučaju poremećaja ravnoteže.

Na Slici-1a je primer brisanja elementa *67* koji nema nijednog naslednika, dok je na Slici-1b primer brisanja elementa 64 koji ima jednog naslednika.



Slika 3.1 a) Čvor bez naslednika: b) Čvor sa jednim naslednikom

Na slikama 2a i 2b su različiti primeri brisanja čvora 22 koji ima oba deteta.



Slika 3.2 Čvor sa dva naslednika

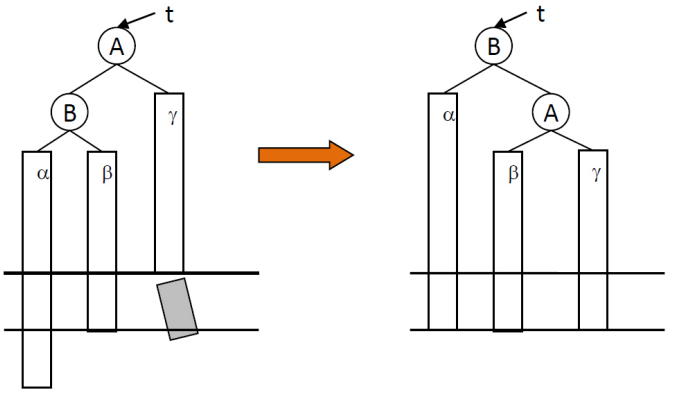
DESNA (LL) I DVOSTRUKA DESNA (LR) ROTACIJA

Desna rotacija (LL) se vrši kada je (t->balans < -1) && (t->levi->balans < 0)

Desna rotacija pri brisanju čvora iz AVL stabla se radi ako je ispunjen uslov:

```
(t.balans < -1) &amp;amp;&amp;amp; (t.levi.balans < 0)
```

Primer desne rotacije je dat na Slici-3.

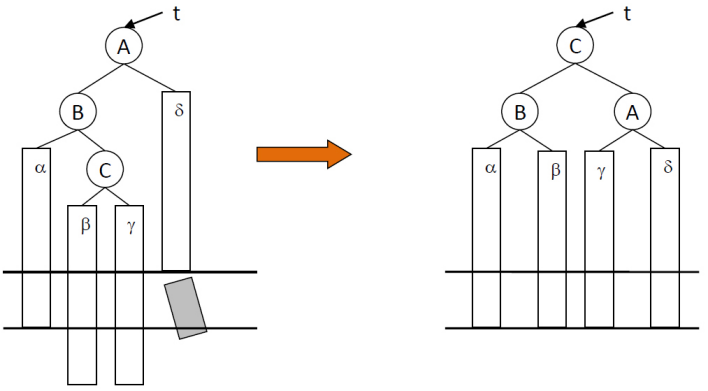


Slika 3.3 Desna rotacija pri brisanju elementa iz AVL stabla

Dvostruka desna rotacija se vrši za slučaj ako je:

```
(t.balans < -1) &amp;amp;&amp;amp; !(t.levi.balans < 0)
```

Primer dvostruke desne rotacije je dat na Slici-4.



Slika 3.4 Dvostruka desna rotacija pri brisanju elementa iz AVL stabla

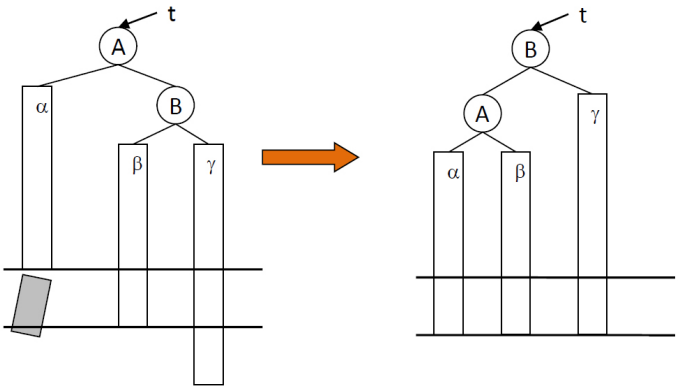
LEVA (RR) I DVOSTRUKA LEVA (RL) ROTACIJA

Leva rotacija (LL) se vrši kada je (t.balans > 1) && (t.desni.balans > 0)

Leva rotacija usled brisanja čvora iz AVL stabla se vrši ako je ispunjen uslov:

```
(t.balans > 1) &amp;amp;&amp;amp; (t.desni.balans > 0)
```

Primer leve rotacije je dat na Slici-5.

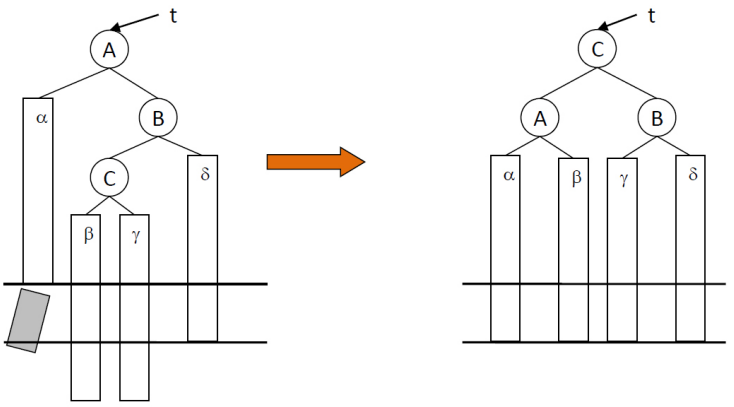


Slika 3.5 Leva rotacija pri brisanju elementa iz AVL stabla

Dvostruka leva rotacija nakon brisanja elementa iz AVL stabla se vrši ako je ispunjen sledeći uslov:

```
(t.balans > 1) &amp;amp;&amp;amp; !(t.desni.balans > 0)
```

Primer dvostruke leve rotacije je dat na Slici-6.



Slika 3.6 Dvostruka leva rotacija pri brisanju elementa iz AVL stabla

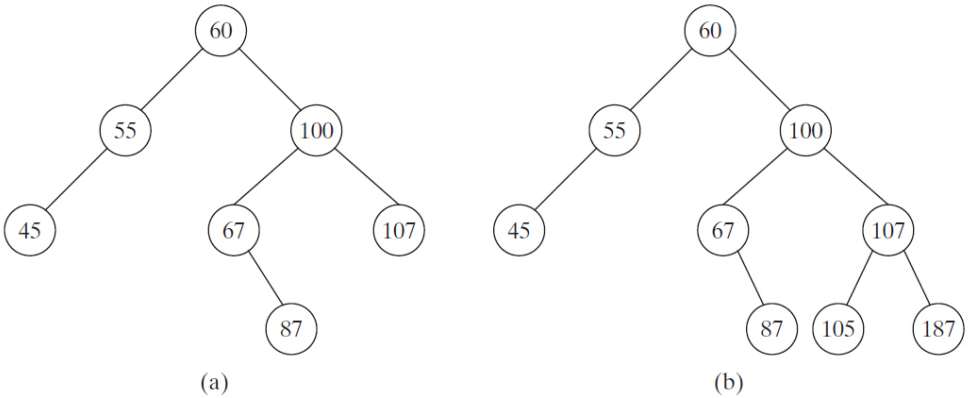
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADACI ZA SAMOPROVERU

Uraditi samostalno sledeće zadatke

Zadatak 1. Za stablo sa **Slika 7a,** prikazati novo stablo posle brisanja elementa 107. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?
Zadatak 2. Za stablo sa **Slika 7a**, prikazati novo stablo posle brisanja elementa 60. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?

Zadatak 3. Za stablo sa **Slika 7a**, prikazati novo stablo posle brisanja elementa 55. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?



Slika 3.7 Faktor uravnoteženosti određuje da li je čvor uravnotežen.

Zadatak 4. Za stablo sa **Slika 7b,** prikazati novo stablo posle brisanja elemenata 67 i 87. Koja rotacija se primenjuje kako bi se stablo uravnotežilo? Koji čvor nije bio uravnotežen?
?

Poglavlje 4

Klasa AVLTree

IMPLEMENTACIJA KLASE AVLTREE

Cilj zadatka je kreirane kompletne klase za AVL stablo u Javi koje uključuje opcije dodavanja i brisanja elemenata iz stabla, i balansiranja dela stabla u slučaju gubitka ravnoteže

U nastavku je kompletan kod klase *AVLTree* koja je obrađena na predavanjima. Testiranje ove klase je moguće izvršiti pomoću test programa obrađenog na predavanjima u okviru sekcije *Testiranje klase AVLTree*.

```
public class AVLTree<E extends Comparable<E>> extends BST<E> {
  /** Create a default AVL tree */
  public AVLTree() {
  }

  /** Create an AVL tree from an array of objects */
  public AVLTree(E[] objects) {
    super(objects);
  }

  @Override /** Override createNewNode to create an AVLTreeNode */
  protected AVLTreeNode<E> createNewNode(E e) {
    return new AVLTreeNode<E>(e);
  }

  @Override /** Insert an element and rebalance if necessary */
  public boolean insert(E e) {
    boolean successful = super.insert(e);
    if (!successful)
    return false; // e is already in the tree
    else {
    balancePath(e); // Balance from e to the root if necessary
    }

    return true; // e is inserted
  }

  /** Update the height of a specified node */
  private void updateHeight(AVLTreeNode<E> node) {
    if (node.left == null &amp;amp;&amp;amp; node.right == null) // node is a leaf
    node.height = 0;
    else if (node.left == null) // node has no left subtree
```

```
  node.height = 1 + ((AVLTreeNode<E>)(node.right)).height;
  else if (node.right == null) // node has no right subtree
  node.height = 1 + ((AVLTreeNode<E>)(node.left)).height;
  else
  node.height = 1 +
      Math.max(((AVLTreeNode<E>)(node.right)).height,
      ((AVLTreeNode<E>)(node.left)).height);
}

/** Balance the nodes in the path from the specified
 * node to the root if necessary
 */
private void balancePath(E e) {
  java.util.ArrayList<TreeNode<E>> path = path(e);
  for (int i = path.size() - 1; i >= 0; i--) {
  AVLTreeNode<E> A = (AVLTreeNode<E>)(path.get(i));
  updateHeight(A);
  AVLTreeNode<E> parentOfA = (A == root) ? null :
      (AVLTreeNode<E>)(path.get(i - 1));

  switch (balanceFactor(A)) {
      case -2:
      if (balanceFactor((AVLTreeNode<E>)A.left) <= 0) {
          balanceLL(A, parentOfA); // Perform LL rotation
      }
      else {
          balanceLR(A, parentOfA); // Perform LR rotation
      }
      break;
      case +2:
      if (balanceFactor((AVLTreeNode<E>)A.right) >= 0) {
          balanceRR(A, parentOfA); // Perform RR rotation
      }
      else {
          balanceRL(A, parentOfA); // Perform RL rotation
      }
  }
  }
}

/** Return the balance factor of the node */
private int balanceFactor(AVLTreeNode<E> node) {
  if (node.right == null) // node has no right subtree
  return -node.height;
  else if (node.left == null) // node has no left subtree
  return +node.height;
  else
  return ((AVLTreeNode<E>)node.right).height -
      ((AVLTreeNode<E>)node.left).height;
}

/** Balance LL */
private void balanceLL(TreeNode<E> A, TreeNode<E> parentOfA) {
```

```
  TreeNode<E> B = A.left; // A is left-heavy and B is left-heavy

  if (A == root) {
  root = B;
  }
  else {
  if (parentOfA.left == A) {
      parentOfA.left = B;
  }
  else {
      parentOfA.right = B;
  }
  }

  A.left = B.right; // Make T2 the left subtree of A
  B.right = A; // Make A the left child of B
  updateHeight((AVLTreeNode<E>)A);
  updateHeight((AVLTreeNode<E>)B);
}

/** Balance LR  */
private void balanceLR(TreeNode<E> A, TreeNode<E> parentOfA) {
  TreeNode<E> B = A.left; // A is left-heavy
  TreeNode<E> C = B.right; // B is right-heavy

  if (A == root) {
  root = C;
  }
  else {
  if (parentOfA.left == A) {
      parentOfA.left = C;
  }
  else {
      parentOfA.right = C;
  }
  }

  A.left = C.right; // Make T3 the left subtree of A
  B.right = C.left; // Make T2 the right subtree of B
  C.left = B;
  C.right = A;

  // Adjust heights
  updateHeight((AVLTreeNode<E>)A);
  updateHeight((AVLTreeNode<E>)B);
  updateHeight((AVLTreeNode<E>)C);
}

/** Balance RR  */
private void balanceRR(TreeNode<E> A, TreeNode<E> parentOfA) {
  TreeNode<E> B = A.right; // A is right-heavy and B is right-heavy

  if (A == root) {
```

```
  root = B;
  }
  else {
  if (parentOfA.left == A) {
      parentOfA.left = B;
  }
  else {
      parentOfA.right = B;
  }
  }

  A.right = B.left; // Make T2 the right subtree of A
  B.left = A;
  updateHeight((AVLTreeNode<E>)A);
  updateHeight((AVLTreeNode<E>)B);
}

/** Balance RL  */
private void balanceRL(TreeNode<E> A, TreeNode<E> parentOfA) {
  TreeNode<E> B = A.right; // A is right-heavy
  TreeNode<E> C = B.left; // B is left-heavy

  if (A == root) {
  root = C;
  }
  else {
  if (parentOfA.left == A) {
      parentOfA.left = C;
  }
  else {
      parentOfA.right = C;
  }
  }

  A.right = C.left; // Make T2 the right subtree of A
  B.left = C.right; // Make T3 the left subtree of B
  C.left = A;
  C.right = B;

  // Adjust heights
  updateHeight((AVLTreeNode<E>)A);
  updateHeight((AVLTreeNode<E>)B);
  updateHeight((AVLTreeNode<E>)C);
}

@Override /** Delete an element from the binary tree.
 * Return true if the element is deleted successfully
 * Return false if the element is not in the tree */
public boolean delete(E element) {
  if (root == null)
  return false; // Element is not in the tree

  // Locate the node to be deleted and also locate its parent node
```

```
    TreeNode<E> parent = null;
    TreeNode<E> current = root;
    while (current != null) {
    if (element.compareTo(current.element) < 0) {
        parent = current;
        current = current.left;
    }
    else if (element.compareTo(current.element) > 0) {
        parent = current;
        current = current.right;
    }
    else
        break; // Element is in the tree pointed by current
    }

    if (current == null)
    return false; // Element is not in the tree

    // Case 1: current has no left children
    if (current.left == null) {
    // Connect the parent with the right child of the current node
    if (parent == null) {
        root = current.right;
    }
    else {
        if (element.compareTo(parent.element) < 0)
        parent.left = current.right;
        else
        parent.right = current.right;

        // Balance the tree if necessary
        balancePath(parent.element);
    }
    }
    else {
    // Case 2: The current node has a left child
    // Locate the rightmost node in the left subtree of
    // the current node and also its parent
    TreeNode<E> parentOfRightMost = current;
    TreeNode<E> rightMost = current.left;

    while (rightMost.right != null) {
        parentOfRightMost = rightMost;
        rightMost = rightMost.right; // Keep going to the right
    }

    // Replace the element in current by the element in rightMost
    current.element = rightMost.element;

    // Eliminate rightmost node
    if (parentOfRightMost.right == rightMost)
        parentOfRightMost.right = rightMost.left;
    else
```

```
        // Special case: parentOfRightMost is current
        parentOfRightMost.left = rightMost.left;

    // Balance the tree if necessary
    balancePath(parentOfRightMost.element);
    }

    size--;
    return true; // Element inserted
  }

  /** AVLTreeNode is TreeNode plus height */
  protected static class AVLTreeNode<E extends Comparable<E>>
    extends BST.TreeNode<E> {
    protected int height = 0; // New data field

    public AVLTreeNode(E o) {
    super(o);
    }
  }
}
```

OPIS METODA KLASE AVLTREE

Navodimo opis klase AVLTree i njenih osnovnih metoda

AVLTree klasa nasleđuje *BST*. Kao i BST klasa, AVLTree ima prazan konstruktor koji pravi jedno prazno AVL stablo (lines 3–4), kao i konstruktor koji pravi inicijalno AVL stablo na osnovu niza elemenata (lines 7–9).

Metod **createNewNode()** pravi novi čvor, objekat klase *TreeNode*. Ovaj metod je overrajdovan kako bi vratio objekat klase *AVLTreeNode* (lines 12–14).

Linije 17–26 overrajduju metod za ubacivanje novog elementa u stablo. Prvo se poziva obično ubacivanje u BST, a zatim **balancePath(e)** (line 22) kako bi se stablo uravnotežilo.

Metod **balancePath** prvo nalazi putanju od čvora koji sadrži element **e** do korena (line 46).Za svaki čvor u putanji, ažurira se njegova visina(line 49), proverava se njegov faktor uravnoteženosti (line 53), i izvršavaju se odgovarajuće rotacije ako je to potrebno (lines 53–69).

Metode za izvršavanje rotacija prikazane su linijama 85–182. Svaki metod prima dva argumenta tipa *TreeNode*, A i ParentOfA, a zatim primenjuje odgovarajuću rotaciju nad elementom A. Posle rotacije, visine čvorova A, B i C se ažuriraju. (lines 102, 129, 152, 179).

Metod za brisanje je overrajdovan linijama 187–264. Metod je skoro isti kao onaj u BST klasi, osim što je potrebno izvršiti uravnoteženje stabla u dva slučaja (lines 224, 249).

ZADACI ZA SAMOSTALNI RAD

Uraditi samostalno sledeće zadatke

Zadatak 1. Zašto je **createNewNode** metod definisan kao **protected**?

Zadatak 2. Kada se poziva **updateHeight** metod, kada **balanceFactor**, a kada **balancePath** ?

Zadatak 3. Koja polja sa podacima čuva *AVLTree* klasa?

Zadatak 4. Prilikom ubacivanja i brisanja elemenata, izvršavaju se rotacije uravnoteživanja stabla, da li je moguće da posle izvršene rotacije i dalje neki čvorovi ostanu neuravnoteženi?

Zadatak 5. Napisati program koji slučajo generiše 500,000 brojeva i ubacuje ih u *BST* stablo, izmeša 500,000 brojva i izvrši pretragu, a zatim ponovo izmeša brojeve pre brisanja svih brojeva. Napisati isti program koji testira *AVLTree*. Porediti izvršavanje programa. Šta je očekivano, koji tip stabla radi efikasnije u ovom slučaju?

Zadatak 6. Napisati program koji animira ubacivanje, brisanje i pretragu AVL stabla kao na **AVL stabla-Slika6**.

Poglavlje 5

Testiranje klase AVLTree

TESTIRANJE OPCIJA KLASE AVLTREE

U ovoj sekciji je detaljan opis primera korišćenja klase AVLTree

U narednom listingu je program koji testira opcije ubacivanja i brisanja u AVL stablu.

```
public class TestAVLTree {
  public static void main(String[] args) {
    // Create an AVL tree
    AVLTree<Integer> tree = new AVLTree<Integer>(new Integer[]{25,
    20, 5});
    System.out.print("After inserting 25, 20, 5:");
    printTree(tree);

    tree.insert(34);
    tree.insert(50);
    System.out.print("\nAfter inserting 34, 50:");
    printTree(tree);

    tree.insert(30);
    System.out.print("\nAfter inserting 30");
    printTree(tree);

    tree.insert(10);
    System.out.print("\nAfter inserting 10");
    printTree(tree);

    tree.delete(34);
    tree.delete(30);
    tree.delete(50);
    System.out.print("\nAfter removing 34, 30, 50:");
    printTree(tree);

    tree.delete(5);
    System.out.print("\nAfter removing 5:");
    printTree(tree);

    System.out.print("\nTraverse the elements in the tree: ");
    for (int e: tree) {
    System.out.print(e + " ");
    }
  }
```

```
  public static void printTree(BST tree) {
    // Traverse tree
    System.out.print("\nInorder (sorted): ");
    tree.inorder();
    System.out.print("\nPostorder: ");
    tree.postorder();
    System.out.print("\nPreorder: ");
    tree.preorder();
    System.out.print("\nThe number of nodes is " + tree.getSize());
    System.out.println();
  }
}
```

Rezultat programa biće:

```
After inserting 25, 20, 5:
Inorder (sorted): 5 20 25
Postorder: 5 25 20
Preorder: 20 5 25
The number of nodes is 3

After inserting 34, 50:
Inorder (sorted): 5 20 25 34 50
Postorder: 5 25 50 34 20
Preorder: 20 5 34 25 50
The number of nodes is 5

After inserting 30
Inorder (sorted): 5 20 25 30 34 50
Postorder: 5 20 30 50 34 25
Preorder: 25 20 5 34 30 50
The number of nodes is 6

After inserting 10
Inorder (sorted): 5 10 20 25 30 34 50
Postorder: 5 20 10 30 50 34 25
Preorder: 25 10 5 20 34 30 50
The number of nodes is 7

After removing 34, 30, 50:
Inorder (sorted): 5 10 20 25
Postorder: 5 20 25 10
Preorder: 10 5 25 20
The number of nodes is 4

After removing 5:
Inorder (sorted): 10 20 25
Postorder: 10 25 20
Preorder: 20 10 25
The number of nodes is 3
Traverse the elements in the tree: 10 20 25
```

ANALIZA DODAVANJA ELEMENATA U STABLO

Cilj primera je detaljan uvid u postupak dodavanja elemenata u AVL stablo i eventualno balansiranja usled neuravnoteženosti u delu stabla

Slika-1 prikazuje kako se AVL stablo razvija u toku ubacivanja novih elemenata u njega.

U stablo se prvo dodaju elementi *25* i *20* (Slika-1a). Zatim se *5* dodaje kao levi potomak čvora *20* (Slika-1b).

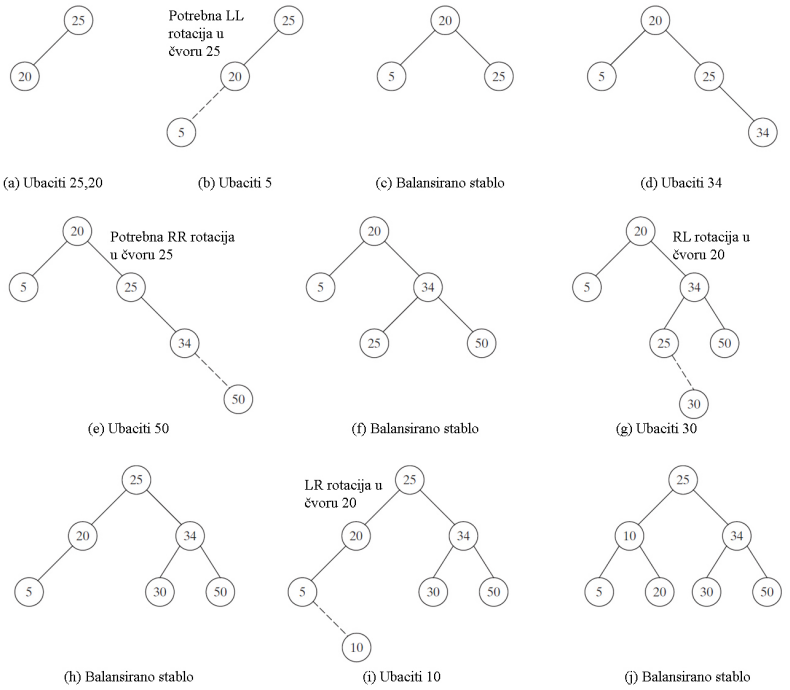
Stablo sada nije uravnoteženo jer je preteška leva strana čvora *25*. Treba izvršiti LL rotaciju u čvoru *25* i stablo dobija oblik kao na Slici-1c.

Nakon dodavanja elementa *34*, stablo izgleda kao na Slici-1d, pri čemu ravnoteža nije narušena.

Nakon ubacivanja elementa *50* stablo izgleda kao na Slici-1e. Ovo stablo nije uravnoteženo jer je otežana desna strana čvora *25*. Stoga treba izvršiti levu (RR) rotaciju u čvoru *25*, što će rezultirati u stablu sa Slike-1f.

Nakon dodavanja elementa *30* stablo dobija oblik sa Slike-1g. Vidimo da stablo sada nije uravnoteženo. Da bi ga uravnotežili izvršićemo dvostruku levu (RL) rotaciju (desna rotacija u čvoru *34* pa leva rotacija u čvoru *20*), što će rezultirati u AVL stablu sa Slike-1h.

Nakon dodavanja elementa *10* u stablo, ono će izgledati kao na Slici-1i.



Slika 5.1 Razvijanje stabla nakon ubacivanja elemenata

Sada imamo nauravnoteženost u čvoru *20*, pa stoga u njemu treba izvršiti dvostruku desnu (*LR*) rotaciju (leva rotacija u čvoru *5* pa desna rotacija u čvoru *20*) što će rezultirati u uravnoteženom AVL stablu sa Slike-1j.

ANALIZA BRISANJA ELEMENATA IZ STABLA

Cilj primera je detaljan uvid u postupak brisanja elemenata u AVL stablo i eventualno balansiranja usled nastanka neuravnoteženosti u delu stabla

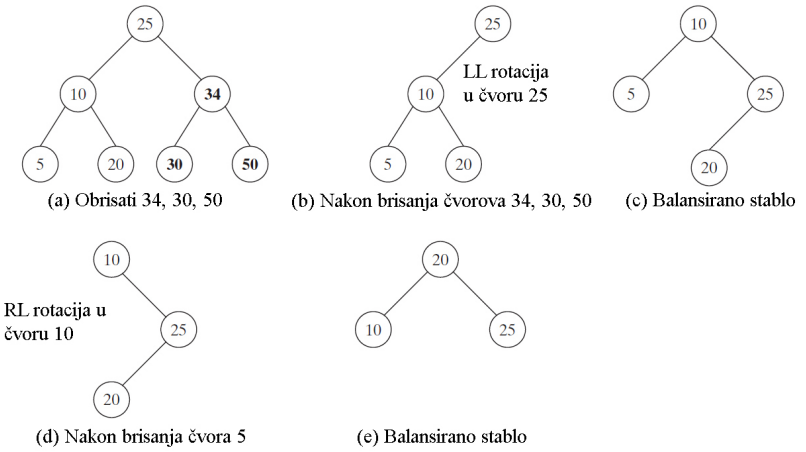
Slika-2 prikazuje kako se stablo razvija u toku brisanja elemenata iz njega.

Usled brisanja elemenata *34*, *30*, i *50*, stablo će izgledati kao na Slici-2b.

Vidimo da stablo sada nije uravnoteženo. Stoga treba Izvršiti LL rotaciju, što će rezultirati u AVL stablu sa Slike-2c.

Usled brisanja elementa *5*, stablo će izgledati kao na Slici-2d. Stablo ponovo nije uravnoteženo.

Stoga, sada treba izvršiti RL rotaciju kako bi dobili AVL stablo prikazano na Slici-2e.



Slika 5.2 Razvijanje stabla usled brisanja elemenata iz njega.

ZADACI ZA SAMOSTALNO VEŽBANJE

Na osnovu materijala sa predavanja i vežbi uraditi samostalno sledeće zadatke:

Zadatak 1. Prikazati promenu AVL stabla korak po korak posle ubacivanja 1, 2, 3, 4, 10, 9, 7, 5, 8, 6 u stablo tim redom.

Zadatak 2. Za stablo kreirano u prethodnom zadatku, prikazati promene posle brisanja 1, 2, 3, 4, 10, 9, 7, 5, 8, 6 tim redosledom.

Zadatak 3. Kako biste obišli sve elemente AVL stabla upotrebom foreach petlje?

Zadatak 4. Pretpostavimo da klasa TreeNode, definisana u okviru klase BST, sadrži referencu na svog roditelja. Izmeniti AVLTree klasu tako da podrži ovu izmenu. Napisati test program koji dodaje brojeve 1, 2, . . . , 100 u stablo i prikazuje sve putanje do listova stabla.

Zadatak 5. Napisati program koji prikazuje AVL stablo gde se za svaki čvor prikazuje faktor uravnoteženosti.

Zadatak 6. Napisati program koji će testirati sve metode klasa *AVLTree*.

Poglavlje 6

Vremenska analiza klase AVLTree

ANALIZA METODA SEARCH, INSERT I DELETE KLASE AVLTREE

Pošto je visina AVL stabla O(log n), vremenska složenost metoda search, insert, i delete u klasi AVLTree je reda O(log n)

Vremenska složenost metoda *search*, *insert* i *delete* klase *AVLTree* zavisi od visine stabla. Možemo dokazati da je visina AVL stabla *O(log n)*. Neka sa *G(h)* označimo minimalni broj čvorova AVL stabla visine *h*. Očigledno, *G(1)* je 1 a *G(2)* je 2. Minimalni broj čvorova u AVL stablu sa visinom *h >= 3* mora da ima dva minimalna stabla: jedno sa visinom *h – 1* i drugo visine *h - 2*. Stoga važi da je:

$$G(h) = G(h - 1) + G(h - 2) + 1$$

Prisetimo se da Fibončijev broj sa indeksom i može biti opisan pomoću rekurentne relacije $F(i) = F(i - 1) + F(i - 2)$. Stoga, funkcija *G(h)* je u osnovi ista kao *F(i)*. Može biti dokazano da važi:

$$h < 1.4405 \log(n + 2) - 1.3277$$

gde je *n* broj čvorova u stablu. Stoga, visina AVL stabla je *O(log n)*.

Metode za pretraživanje, ubacivanje i brisanje čvora, tj. *search*, *insert*, i *delete* prolaze samo kroz čvorove na jednoj putanji stabla. Metodi *updateHeight* i *balanceFactor* se izvršavaju za konstantno vreme za svaki čvor na putanji. Metod *balancePath* se izvršava konstantno vreme za čvor u putanji. Stoga je vremenska složenost metoda *search*, *insert* i *delete* reda *O(log n)*.

ISPITIVANJE PERFORMANSI AVL STABLA

Cilj zadatka je kreiranje test programa u cilju ispitivanja performansi AVL, 2-4 i RB stabla

U nastavku je program koji ispituje performanse klase *AVLTree*:

```
public class TreePerformanceTest {
  public static void main(String[] args) {
    final int TEST_SIZE = 500000; // Tree size used in the test

    // Create an AVL tree
    Tree<Integer> tree1 = new AVLTree<Integer>();
    System.out.println("AVL tree time: " +
```

```
    getTime(tree1, TEST_SIZE) + " milliseconds");
  }

  public static long getTime(Tree<Integer> tree, int testSize) {
    long startTime = System.currentTimeMillis(); // Start time

    // Create a list to store distinct integers
    java.util.List<Integer> list = new java.util.ArrayList<Integer>();
    for (int i = 0; i < testSize; i++)
    list.add(i);

    java.util.Collections.shuffle(list); // Shuffle the list

    // Insert elements in the list to the tree
    for (int i = 0; i < testSize; i++)
    tree.insert(list.get(i));

    java.util.Collections.shuffle(list); // Shuffle the list

    // Delete elements in the list from the tree
    for (int i = 0; i < testSize; i++)
    tree.delete(list.get(i));

    // Return elapse time
    return System.currentTimeMillis() - startTime;
  }
}
```

Rezultat programa biće:

```
AVL tree time: 7609 milliseconds
```

Metod *getTestTime* kreira listu razlčitih broja u intervalu od *0* do *testSize – 1*, meša (shuffle) listu brojeva, i dodaje elemente liste u stablo, ponovo meša (shuffle), briše elemente iz stabla, i konačno vraća vrednost vremena izvršavanja programa.

Program kreira AVL, i dređuje vreme izvršavanja operacija dodavanja i brisanja 500000 elemenata.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADACI ZA SAMOSTALNI RAD

Uraditi samostalno sledeće zadatke

Zadatak 1. Koja je minimalna, a koja maksimalna visina za AVL stablo of 3 čvora, 5 čvorova i 7 čvorova?

Zadatak 2. Ako je AVL stablo visine 3, koliko maksimalno čvorova može da ima? Koliko minimalno čvorova može da ima?

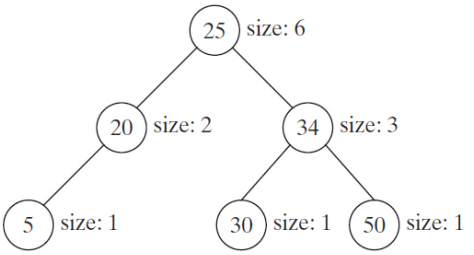
Zadatak 3. Ako je AVL stablo visine 4, koliko maksimalno čvorova može da ima? Koliko minimalno čvorova može da ima?

Zadatak 4. K-ti najmanji element u BST stablu se može naći u O(n) vremenu pomoću inorder iteratora. U slučaju AVL stabla to vreme je O(log n). Kako bismo postigli ovo, treba dodati novo polje u *AVLTreeNode* kako bi se čuvao broj čvorova njegovog podstabla. Veličina čvora je za jedan veća od zbira veličina čvorova koji su njegova deca. Primer stabla sa ovim podacima je dat na **Slika-1**.

Dodati u *AVLTree* klasu metod koji će naći k-ti najmanji element.

```
public E find(int k)
```

Metod vraća **null** ako je k < 1 ili k > veličina stabla. Ovaj metod može biti implementiran koristeći rekurzivni metod **find(k, root)**, koji vraća **k-ti** najmanji element u stablu sa korenom **root**.



Slika 6.1 Podatak o veličini čvora AVLTreeNode koji čuva broj čvorova u podstablu.

Neka su A i B levo i desno dete korena. Ako podrazumevamo da stablo nije prazno i da važi k <= root.size, **find(k, root)** se može definisati kao na slici ispod:

$$find(k, root) = \begin{cases} root.element, \text{ if } A \text{ is null and } k \text{ is } 1; \\ B.element, \text{ if } A \text{ is null and } k \text{ is } 2; \\ find(k, A), \text{ if } k <= A.size; \\ root.element, \text{ if } k = A.size + 1; \\ find(k - A.size - 1, B), \text{ if } k > A.size + 1; \end{cases}$$

Slika 6.2 Opis funkcije za traženje k-tog najmanjeg elementa

Modifikovati ubacivanje i brisanje elemenata u AVL stablo kako bi se ažurirale vrednosti veličine čvora. Ove dve operacije i dalje ostaju u vremenu O(log n). Metod **find(k)** se onda može implementirati u O(log n) vremenu.Što znači da se k-ti element može naći u AVL stablu za **O(log n)** vreme.

Zaključak

REZIME

Na osnovu svega obrađenog možemo zaključiti sledeće:

Stablo je balansirano ako se visine levog i desnog podstabla razlikuju maksimalno za 1 i kada su njegova leva i desna podstabla takođe balansirana. Samo-balansirajuće BST stablo je bilo koje binarno stablo pretraživanja koje automatski održava najmanju moguću vrednost svoje visine uprkos proizvoljnim umetanjima i brisanjima elemenata. Najpoznatija samobalansirajuća binarna stabla pretraživanja su: AVL stabla, B-stabla, 2-4 stabla i Crveno-crna (red-black – RB) stabla.

AVL stablo je BST koje ima sledeće karakteristike: a) podstablo svakog čvora se razlikuje u visini za tačno 1, b) svako podstablo AVL stabla je AVL stablo. Nakon što se element doda ili obriše iz AVL stabla, ukoliko ono postane neuravnoteženo, neophodno je izvršiti odgovarajuće rotacije stabla kako bi se ono izbalansiralo. Postoji četiri različite rotacije: LL (desna rotacija), RR (leva rotacija), LR (dvostruka desna rotacija), i RL (dvostruka leva rotacija). Brisanje iz AVL stabla se vrši kao i brisanje elemenata iz BST stabla. Ono što dodatno treba uraditi jeste provera ravnoteže stabla i eventualno balansiranje dela stabla.

Pošto je AVL stablo ustvari binarno stablo pretraživanja, klasa *AVLTree* može biti dizajnirana kao podklasa koja nasleđuje osobine klase *BST*. Klasa *AVLTreeNode* ima kao dodatak visinu čvora u odnosu na klasu *TreeNode*. Klasa *AVLTree* priširuje klasu *BST* sa novim implementacijama metoda *insert* i *delete*. Pošto je visina *AVL* stabla *O(log n)*, vremenska složenost metoda *search*, *insert*, i *delete* u klasi *AVLTree* je reda *O(log n)*.

REFERENCE

Korišćena literatura

[1] D. Liang, Introduction to Java Programming, Comprehensive Version, deseto izdanje, Prentice Hall, 2014.

[2] M.A. Weiss, Data Structures and Problem Solving Using Java, treće izdanje, Addison Wesley, 2005.

[3] R. Sedgewick, K.Wayne, Algorithms, 4. Izdanje, Pearson Education, 2011.

[4] https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm