Diplomska naloga

UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO
Matematika – praktična matematika (VSŠ)

Petra Tome

URAVNOTEŽENA DREVESA

Diplomska naloga

Fakulteta za matematiko in fiziko

Ljubljana, 2005

Diplomska naloga

Zahvaljujem se vsem, ki so kakorkoli pripomogli k nastanku moje diplomske naloge. Zahvala mentorju Matiji Lokarju, za svetovanje in koristne napotke pri nastajanju diplomske naloge. Zahvala vsem prijateljem za moralno in prijateljsko podporo, še posebna zahvala gre mojemu dragemu Marku za vsestransko pomoč in podporo.

Diplomsko delo posvečam svojim staršem.

Fakulteta za matematiko in fiziko

KAZALO

POVZETEK **5**

1. UVOD **7**

2. DREVO **8**
2.1. DEFINICIJA 8
2.2. TERMINOLOGIJA 8

3. DVOJIŠKO DREVO **12**
3.1. DEFINICIJA 12
3.2. PREDSTAVITEV DVOJIŠKEGA DREVESA 13
3.2.1. Predstavitev s tabelo 13
3.2.2. Predstavitev s kazalci 14
3.2.3. Predstavitev s kazalci v jeziku java 14
3.3. LASTNOSTI DVOJIŠKIH DREVES 17
3.3.1. Število listov 17
3.3.2. Število vozlišč 17
3.3.3. Višina dreves 18
3.4. ČASOVNA ZAHTEVNOST ALGORITMOV NAD DVOJIŠKIMI DREVESI 20

4. ISKALNO DVOJIŠKO DREVO **21**
4.1. DEFINICIJA 21
4.2. ZGRADBA ISKALNEGA DVOJIŠKEGA DREVESA 21
4.3. OPERACIJE NAD DVOJIŠKIMI DREVESI 22
4.3.1. Išči 22
4.3.2. Minimum in maksimum 23
4.3.3. Vstavljanje 26
4.3.4. Brisanje 27
4.4. O ČASOVNI ZAHTEVNOSTI OPISANIH POSTOPKOV 30

5. URAVNOTEŽENA DVOJIŠKA DREVESA **31**
5.1. DEFINICIJA URAVNOTEŽENEGA DREVESA 31
5.2. AVL DREVO 35
5.2.1. Vstavljanje in brisanje v AVL drevesu 35
5.3. RDEČE-ČRNO DREVO 58
5.3.1. Vstavljanje in brisanje v rdeče-črnem drevesu 60
5.4. 2-3 DREVESA 87
5.4.1. Predstavitev 2-3 drevesa 87
5.4.2. Iskanje v 2-3 drevesu 88
5.4.3. Vstavljanje in brisanje v 2-3 drevesu 89

6. ZAKLJUČEK **100**

7. LITERATURA **101**

Diplomska naloga
Fakulteta za matematiko in fiziko

iplomska nalog

PROGRAM DIPLOMSKEGA DELA

V diplomski nalogi predstavite uravnotežena iskalna drevesa. Podrobneje obravnavajte AVL drevesa, rdeče-črna drevesa in 2-3 drevesa.

Osnovna literatura:

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to algorithms*, second edition, MIT Press, 2001

R. Sedgewick, *Algorithms in Java (Parts 1 - 4)*, Addison-Wesley, 2003

J. Kozak, *Podatkovne strukture in algoritmi*, Društvo matematikov, fizikov in astronomov, 1986, Ljubljana.

Mentor:
mag. Matija Lokar

Fakulteta za matematiko in fiziko

Diplomska naloga

POVZETEK

V diplomskem delu obravnavam uravnotežena drevesa kot obliko iskalnih dvojiških dreves. Navadno iskalno dvojiško drevo se namreč lahko pri izvajanju operacij *vstavi* in *briši* izrodi – postane podobno linearnemu seznamu. V tem primeru je zahtevnost algoritmov bistveno slabša od optimalnega primera, ki ga dobimo pri poravnanih iskalnih dvojiških drevesih. Zaradi tega drevesa uravnotežimo.

Proučila sem tri tipe uravnoteženih dreves: AVL drevesa, rdeče-črna drevesa in 2-3 drevesa. Pri tem sem se osredotočila predvsem na pravila, ki jih pri posameznem tipu dreves uporabimo za odpravljanje neuravnoteženosti, ki je posledica vstavljanja ali brisanja podatkov. Pri vseh treh tipih uravnoteženih dreves je poudarek na operacijah *vstavi* in *briši*, saj ti dve operaciji povzročata spremembe v sami strukturi in zato zahtevata ustrezno preurejanje vozlišč.

Pri AVL drevesu je ključnega pomena faktor ravnotežja, ki uravnava višino dreves tako, da se višini poddreves ne razlikujeta za več kot ena. Pri tem preurejamo drevesa z ustreznimi rotacijami.

Drugi tip uravnoteženega drevesa so rdeče - črna drevesa. Pravila o uravnoteženosti so dokaj preprosta, so pa algoritmi nad njimi nekoliko bolj zapleteni. Uravnoteženost uravnavamo z »barvanjem« vozlišč in preureditvijo v obliki rotacij. Rdeče - črna drevesa so v splošnem nekoliko bolj učinkovita od AVL dreves.

Zadnji tip uravnoteženih dreves, ki ga obravnavamo v diplomskem delu, so 2-3 drevesa. To so popolnoma poravnana drevesa z dvema ali tremi sinovi. Podatki so shranjeni samo v listih dreves. Učinkovitost operacij (nizko višino drevesa) dosežemo s prizadevanjem, da ima kar se da veliko vozlišč stopnjo tri.

Preden se lotimo uravnoteženih dreves, so v drugem poglavju predstavljeni splošni pojmi v povezavi z drevesi. V tretjem poglavju je podrobneje predstavljeno dvojiško drevo in nekaj njegovih lastnosti, kot so višina drevesa, število vozlišč v drevesu in podobno. Te lastnosti potrebujemo pri izpeljavi določenih značilnosti iskalnih dvojiških dreves, ki jim je posvečeno četrto poglavje. Peto poglavje, osrednji del diplomske naloge, podrobno obravnava tri tipe uravnoteženih dreves: AVL drevesa, rdeče-črna drevesa in 2-3 drevesa. V zaključku je navedena kratka primerjava značilnosti vseh treh predstavljenih tipov uravnoteženih dvojiških dreves.

Fakulteta za matematiko in fiziko

Diplomska naloga

SUMMARY

My diploma is concentrated on the balanced trees as a form of search binary trees. When carrying out the operations *insert* and *delete*, the usual search binary tree can degenerate – it becomes similar to the linear list. In this case the difficulty of algorithms is substantially worse from the optimal example of the levelled search binary trees. Due to this reason we balance the trees.

I studied three types of balanced trees: AVL trees, red-black trees and 2-3 trees. I focused especially on the rules that we use for individual forms of trees in order to prevent unbalance which is the consequence of insertion or deletion of the data. With all these three types of balanced trees the stress is on the operations *insert* and *delete*, since these two operations cause changes in the very structure and therefore demand a suitable reorganization of nodes.

The balance factor, which regulates the height of trees in a way that the heights of sub-trees do not differ for more than one, is of a key importance for the AVL trees. We rearrange trees with suitable rotations.

The second type of balanced trees is the red-black trees. The rules on balance are quite simple but the algorithms above them are a bit more complicated. The balance is regulated with the »colouring« of nodes and the rearrangement in the form of rotations. In general, red–black trees are a bit more efficient than the AVL trees.

The last type of balanced trees that are dealt with in my diploma is the 2–3 trees. These are completely levelled trees with two or three sons. The data are kept only in the tree leaves. The efficiency of operations (the low height of a tree) is gained with effort to have as many level three nodes as possible.

In the second chapter, before passing on to balanced trees, I introduced the basic notions connected with the trees. In the third chapter there is a more detailed description of a binary tree and some of its features like the height of a tree, the number of tree nodes and similar. These features are needed for the execution of certain features of search binary trees, which I described in the fourth chapter. The fifth chapter, the central part of my diploma, discusses in details the three types of balanced trees: AVL trees, red–black trees and 2–3 trees. In the conclusion there is a short comparison of features of all three introduced types of balanced binary trees.

Fakulteta za matematiko in fiziko

Diplomska naloga

1. UVOD

Obstajajo številne podatkovne strukture. Med njimi je tudi podatkovna struktura drevo, ki se zelo pogosto uporablja. Podatke hranimo v vozliščih dreves, med katerimi je eno posebno, imenovano koren. Iz njega izhajajo vsa ostala vozlišča. S pomočjo dreves lahko predstavimo tudi druge podatkovne strukture, kot sta na primer vrsta s prednostjo ali slovar.

Drevo je nelinearna podatkovna struktura. V obliki iskalnega drevesa je zelo primerna za shranjevanje podatkov na urejen način, še posebej, kadar je teh podatkov veliko. Algoritmi za delo nad drevesi, še posebej dvojiškimi, so ob uporabi rekurzije kratki in pregledni. S primerno izpeljavo operacij časovna zahtevnost algoritmov narašča sorazmerno počasi glede na število podatkov, shranjenih v drevesu. Časovna zahtevnost teh algoritmov je povezana z višino dreves. Zato si pri operacijah, ki lahko povzročijo spremembo v višini drevesa, kot sta vstavljanje in brisanje elementov, prizadevamo, da je iskalno drevo ves čas čim bolj uravnoteženo. To pomeni, da poskušamo doseči, da je število vozlišč v poddrevesih vseh vozlišč čim bolj enako. Za uravnoteženost drevesa skrbimo sproti ob vstavljanju in ob brisanju podatkov iz drevesa. V ta namen uporabljamo posebno tehniko rotacij in pravil, ki skrbijo za tako uravnoteževanje.

V nadaljevanju si bomo najprej ogledali osnovne pojme o drevesih, in še posebej o dvojiških drevesih. Glavna nit, ki bo vodila skozi vsa poglavja, bodo iskalna dvojiška drevesa. Ogledali si bomo osnovne postopke in na kratko analizirali njihovo časovno zahtevnost. Videli bomo, da se časovna zahtevnost lahko v najslabših primerih iz zahtevnosti reda $O(\log n)$ spremeni v zahtevnost reda $O(n)$, kjer je n število podatkov. Ta pojav lahko preprečimo s pomočjo uravnoteženih dreves. Ogledali si bomo nekaj tipov teh dreves.

Fakulteta za matematiko in fiziko

2. DREVO

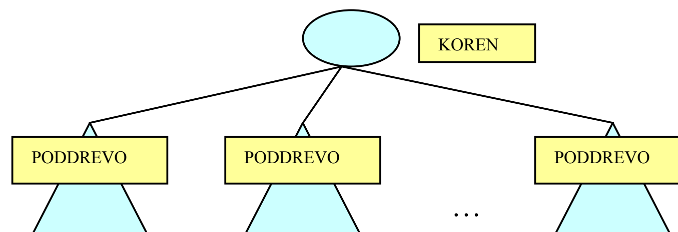
Kot smo že omenili v uvodu, je drevo ena od podatkovnih struktur, ki se zelo pogosto uporablja. Podatke hranimo v vozliščih dreves. Med njimi je eno posebno, imenovano koren. Iz njega izhajajo vsa ostala vozlišča.

2.1. DEFINICIJA

Obstaja veliko različnih definicij pojma drevo. Tako je v terminologiji teorije grafov drevo povezan graf brez ciklov, poznamo pa tudi drugačne definicije. Zelo uporabna je rekurzivna definicija drevesa. Ta pravi:

> Drevo je končna množica vozlišč, od katerih je eno posebej označeno (imenujemo ga koren), ostala vozlišča pa razpadejo na $n > 0$ disjunktnih množic $T_1,...,T_n$. Te množice (imenujemo jih poddrevesa) so tudi drevesa.

Shematsko si torej drevo predstavljamo takole:

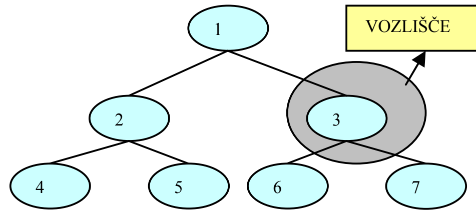


2.2. TERMINOLOGIJA

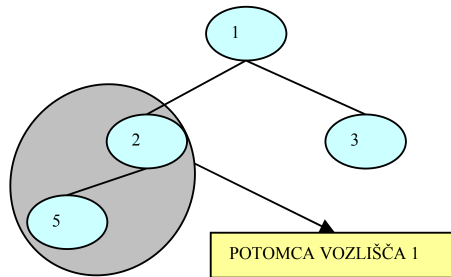
Oglejmo si nekaj pojmov in definicij povezanih s podatkovno strukturo drevo. Našteli bomo le tiste, ki jih bomo kasneje potrebovali.

Diplomska naloga
Fakulteta za matematiko in fiziko

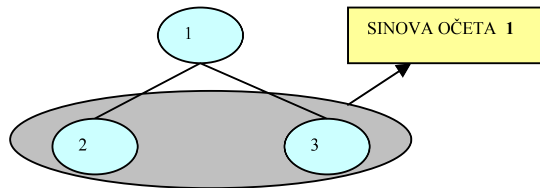
vozlišče: je osnovni element drevesa. Vsebuje podatek (ali več podatkov) in informacijo o iz njega izhajajočih poddrevesih. Ima lahko več potomcev, vendar največ enega prednika;



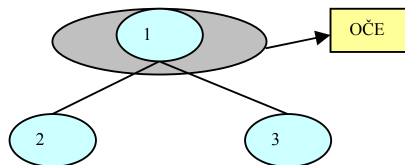
potomec vozlišča: poljubno vozlišče v poddrevesu, ki izhajajo iz danega vozlišča;



sin ali naslednik: koren poddrevesa, ki izhajaja iz tega vozlišča;

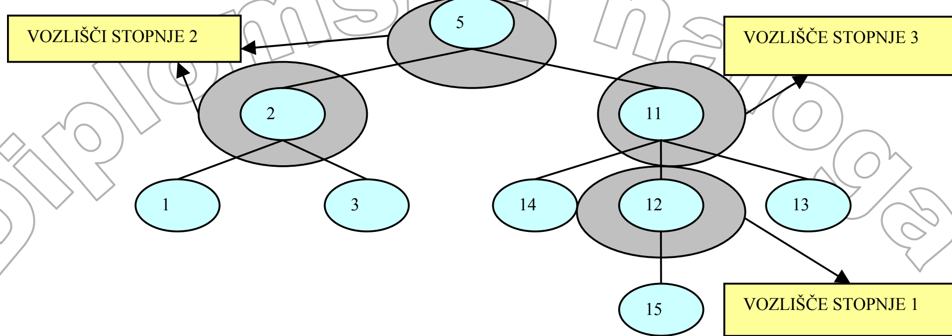


oče ali prednik: vozlišče 1 je oče svojima sinovoma 2 in 3. Koren celotnega drevesa je edino vozlišče brez očeta;



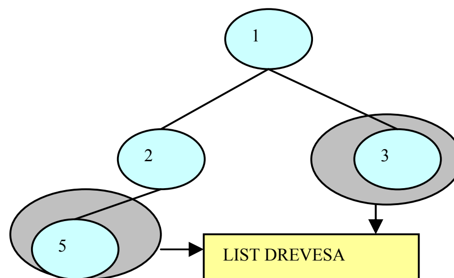
stopnja vozlišča: število poddreves, ki izhajajo iz vozlišča;

Diplomska naloga
Fakulteta za matematiko in fiziko

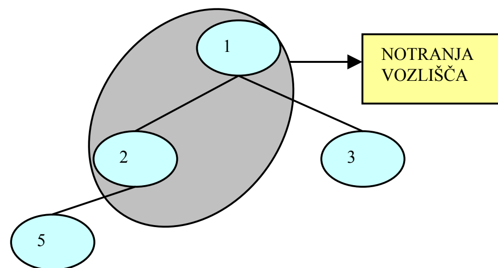


stopnja drevesa: maksimalna stopnja vozlišč. Drevo na prejšnji sliki ima stopnjo 3;

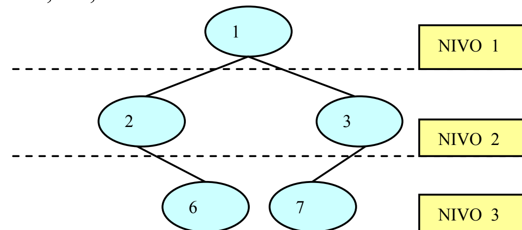
list ali končno vozlišče: vozlišče s stopnjo 0, oziroma vozlišče, ki nima sinov;



notranja vozlišča: vsa vozlišča, razen listov;

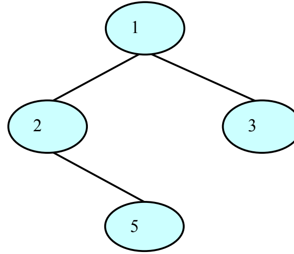


nivo vozlišča: koren ima nivo 1. Če ima oče nivo n, ima sin nivo n +1. Koren drevesa ima torej nivo 1, njegovi sinovi nivo 2, itd;

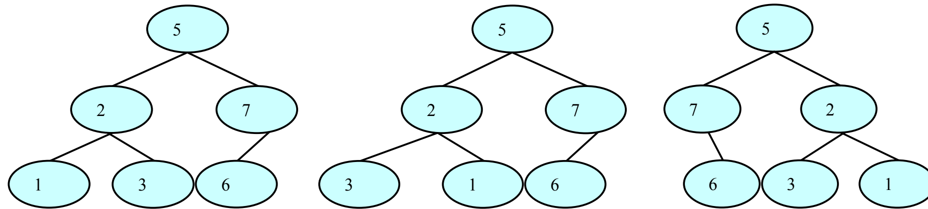


Diplomska naloga
Fakulteta za matematiko in fiziko

višina drevesa: je enaka največjemu nivoju vozlišča v drevesu. Na sliki vidimo drevo višine 3.

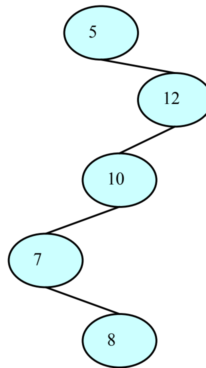


urejeno drevo: drevo, ki ima določen vrstni red poddreves v vsakem vozlišču. Govorimo o prvem, drugem, tretjem, ... poddrevesu. Na sliki vidimo primer treh dreves z enakimi podatki. Če jih obravnavamo kot neurejena drevesa, so to enaka drevesa, kot urejena drevesa pa so različna.

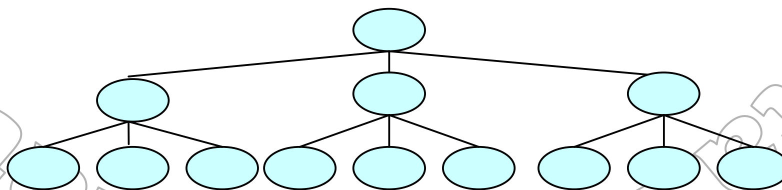


Pri urejenosti nas vsebina podatkov ne zanima. Urejenost se nanaša izključno na to, ali je pomembno v kakšnem vrstnem redu so poddrevesa.

izrojeno drevo: drevo, kjer ima vsako vozlišče (razen lista) le enega sina.



polno drevo stopnje k: drevo, kjer ima vsako vozlišče (z izjemo listov) natanko k sinov. Vsi listi so na istem nivoju. Na sliki vidimo polno drevo stopnje 3;



prazno drevo: drevo brez vozlišč;

Diplomska naloga
Fakulteta za matematiko in fiziko

3. DVOJIŠKO DREVO

Med vsemi drevesi verjetno najpogosteje uporabljamo dvojiška drevesa. To so urejena drevesa stopnje 2. Vsa vozlišča imajo torej dva, enega ali pa nobenega naslednika (sina). Ker je dvojiško drevo urejeno, je vrstni red poddreves pomemben. Govorimo o levem in desnem poddrevesu. Korena teh poddreves imenujemo levi oziroma desni sin.

3.1. DEFINICIJA

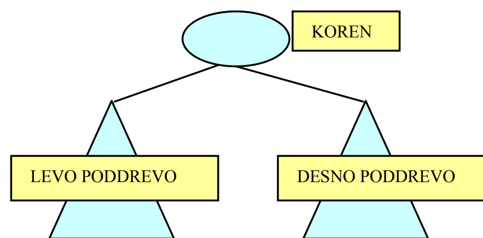
Ker v splošni definiciji pojma drevo ne govorimo o urejenosti, navedimo rekurzivno definicijo še za dvojiška drevesa. Tu upoštevamo stopnjo vozlišč in urejenost sinov.

> Dvojiško drevo je bodisi prazno, bodisi ga sestavlja končna množica vozlišč, od katerih je eno posebej odlikovano in ga imenujemo koren, druga pa razpadejo v dve disjunktni množici, levo in desno poddrevo, ki sta prav tako dvojiški drevesi.

Ali povedano malo drugače:

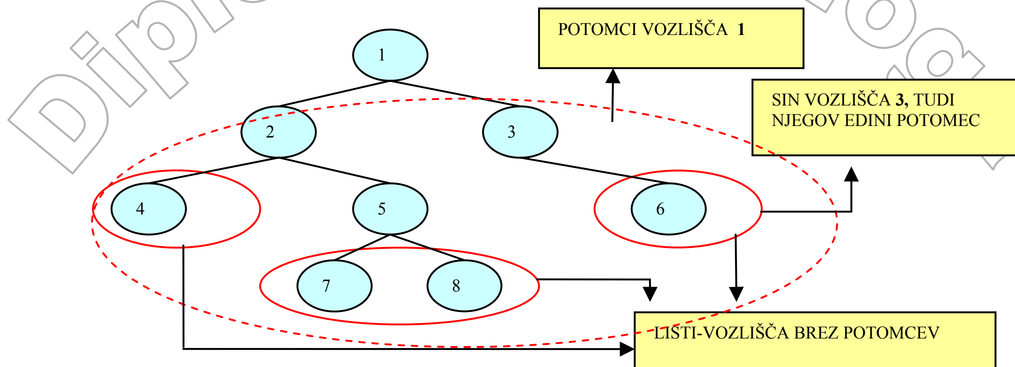
> Prazno drevo je dvojiško drevo. Neprazno drevo sestavlja posebno vozlišče, koren, ter levo in desno poddrevo, ki sta prav tako dvojiški drevesi.

Shematsko si neprazno dvojiško drevo lahko predstavljamo takole:



Diplomska naloga
Fakulteta za matematiko in fiziko

Na sliki si oglejmo še nekaj pojmov, ki smo si jih prej ogledali pri splošnih drevesih.



Vozlišče 1 je koren in vsa ostala vozlišča so njegovi potomci. Vozlišče 1 ima dva naslednika ali sinova (vozlišči 2 in 3). Vozlišče 3 ima samo enega potomca (vozlišče 6), vozlišča 4, 7, 8, 6 pa nimajo potomcev in so listi drevesa.

3.2. PREDSTAVITEV DVOJIŠKEGA DREVESA

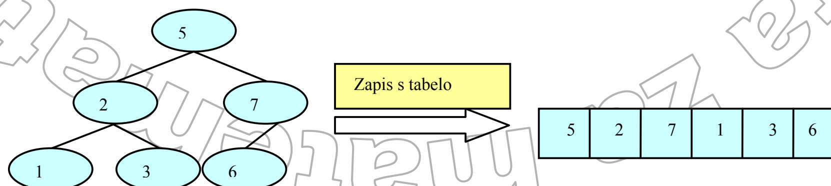
Pri predstavitvi dvojiških dreves se odločimo, na kakšen način bomo drevo hranili oziroma ga predstavili v izbranem programskem jeziku. Na izbiro imamo več načinov predstavitve dvojiškega drevesa.

3.2.1. PREDSTAVITEV S TABELO

Podatke hranimo v tabeli. Samo strukturo (levi, desni sin, oče) določajo indeksi elementov. Prevedbo naredimo tako, da je oče vozlišča, ki ga hranimo v tabeli na i-tem mestu, v tabeli na mestu $i/2$. Levi sin (če obstaja) ima v tabeli mesto $2*i$ in desni sin je na mestu $2*i+1$. Koren drevesa damo v tabelo mesto 1. Če v programskem jeziku tabelo numeriramo drugače (npr. od 0 naprej), indeks 0 enostavno zanemarimo.

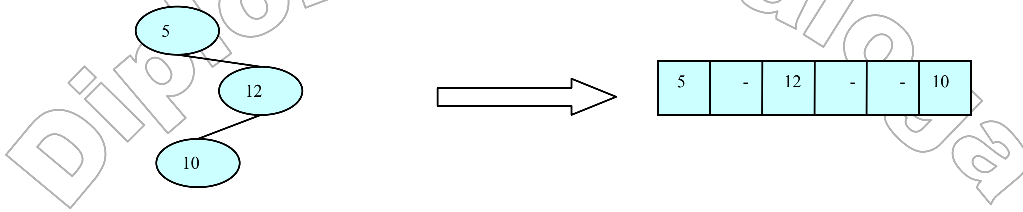
Če je torej vozlišče na i-tem mestu v tabeli, veljajo naslednje formule za njegovega očeta in oba sinova:

- oče(i) = $i/2$
- levi sin(i) $= 2*i$
- desni sin(i) = $2*i+1$



Diplomska naloga Fakulteta za matematiko in fiziko

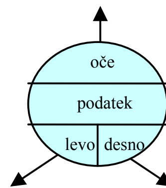
Ta predstavitev je učinkovita, če je drevo polno, oziroma podobno polnemu. V nasprotnem primeru imamo v tabeli veliko praznih mest, kjer bi sicer bili ustrezni sinovi vozlišč.



Diplomska naloga

3.2.2. PREDSTAVITEV S KAZALCI

Pogosto za predstavitev dreves uporabimo kazalčno predstavitev. Tu vozlišča drevesa predstavimo s strukturo, kjer poleg podatka hranimo še kazalce (reference) na vozlišča, kjer hranimo očeta in sinova.



3.2.3. PREDSTAVITEV S KAZALCI V JEZIKU JAVA

Oglejmo si, kako dvojiško drevo predstavimo v programskem jeziku Java.V ta namen bomo sestavili razred `Drevo`. V njem bomo napisali vse osnovne metode za delo z dvojiškimi drevesi, kot jih zahteva formalna predstavitev abstraktne podatkovne strukture dvojiško drevo. Zaradi enostavnosti bomo predpostavili, da v drevesu kot podatke hranimo cela števila. Najprej bomo sestavili razred `Vozel` s komponentami podatek (podatek, ki ga hranimo v vozlišču), levi, desni (referenci na naslednika) in oče (referenca na očeta). Nato bomo pripravili razred `Drevo`, ki je predstavljeno s podatkom, kje je korensko vozlišče. Razredi vsebujejo osnovne metode, potrebne za sleditev besedilu diplome, ostale metode si lahko natančneje pogledate v prilogi diplome.

RAZRED Vozel:

```
// Razred Vozel s komponentami:
//   * podatek(podatek, ki hranimo v vozlišču);
//   * levi, desni (referenci na naslednika);
//   * oče (referenca na očeta);
// Vsebuje ustrezne konstruktorje in metode
// s kateri upravljamo z vozliščem

public class Vozel{
   //podatek, ki ga hranimo v vozlišču
   private int podatek;
   private Vozel levi, desni;  // kazalci na levega in desnega naslednika
   private Vozel oce;          //  in očeta

   // konstruktor, ki ustvari prazen vozel
   public Vozel(){
      this.podatek = 0;
      this.levi = null;
      this.desni = null;
```

Fakulteta za matematiko in fiziko

```
        this.oce = null;
    }

    //vozel s podatkom p
    public Vozel(int p){
        this.podatek = p;
        this.levi = null;
        this.desni = null;
        this.oce = null;
    }

    //vozel s podatkom p in kazalcema na levega (l) in desnega (d) naslednika
    public Vozel(int p, Vozel l, Vozel d){
        this.podatek = p;
        this.levi = l;
        this.desni = d;
        this.oce = null;  // očeta nastavljamo sproti
    }

    //metoda, ki nastavi podatek v vozlu
    public void nastaviPodatek(int x){
        this.podatek = x;
    }

    //metoda, ki vrne podatek v vozlu
    public int vrniPodatek(){
        return this.podatek;
    }

    //metoda, ki nastavi levega naslednika vozlišču v
    public void nastaviLevi(Vozel v){
        this.levi = v;
    }

    //metoda, ki vrne levega naslednika vozlišča
    public Vozel vrniLevi(){
        return this.levi;
    }

    //metoda, ki vozlišču v nastavi desnega naslednika
    public void nastaviDesni(Vozel v){
        this.desni = v;
    }

    //metoda, ki vrne desnega naslednika vozlišča
    public Vozel vrniDesni(){
        return this.desni;
    }

    //metoda, ki nastavi očeta
    public void nastaviOce(Vozel v){
        this.oce = v;
    }

    //metoda, ki vrne očeta
    public Vozel vrniOce(){
        return this.oce;
    }

}
```

Diplomska naloga
Fakulteta za matematiko in fiziko

RAZRED Drevo:

// Razred Drevo je predstavljen s podatkom kje je korensko vozl*išče*

```
public class Drevo{
    //referenca na korensko vozlišče
    private Vozel koren = null;

  //KONSTRUKTORJI
    //konstruktor, ki ustvari prazno drevo
    public Drevo(){
        this.koren = null;
    }

    //konstruktor, ki ustvari drevo z vozlom v
    public Drevo(Vozel v){
        this.koren = v;
    }

  //METODE
   //metoda, ki ugotovi ali je drevo prazno
    public boolean prazno(){
        //vrne true ali false
        return this.koren == null;
    }

    //metoda vrne drevo
    public int vrni() throws Exception{
        if(!this.prazno()){ //če ni prazno ga vrne
            return this.koren.vrniPodatek();
        }
        throw new Exception("Drevo je prazno! ");  //drugače vrže izjemo
    }

    //metoda, ki vrne levo poddrevo
    public Drevo leviSin(){
        Drevo novo = new Drevo();  // ustvarimo novo prazno drevo
        if(!this.prazno()){  // če ni prazno
            novo.koren = this.koren.vrniLevi() ;//koren levega postane koren
                                                //novega
        }
        return novo;
    }

    //metoda, ki vrne desno poddrevo
    public Drevo desniSin(){
        Drevo novo = new Drevo();  // ustvarimo novo prazno drevo
        if(!this.prazno()){  // če ni prazno
            novo.koren = this.koren.vrniDesni(); //koren desnega postane koren
                                                  //novega
        }
        return novo;
    }

    //metoda, ki vrne očeta
    public Drevo oce(){
        Drevo novo = new Drevo();  //novo prazno drevo
        if(!this.prazno()){ // če ni prazno
            novo.koren = this.koren.vrniOce();//koren očeta postane koren novega
        }
        return novo;
    }
}
```

Diplomska naloga
Fakulteta za matematiko in fiziko

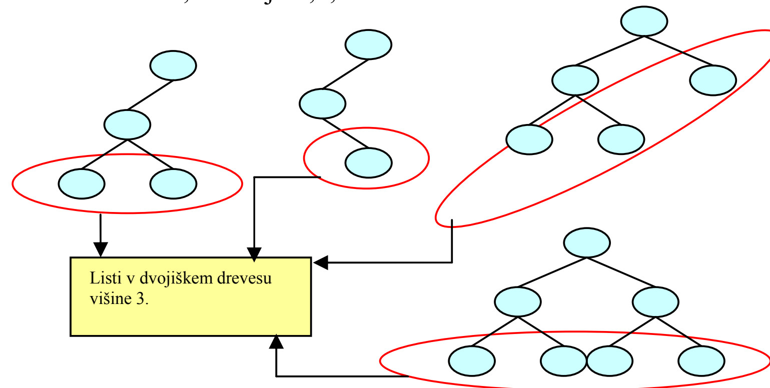
Diplomska naloga
Fakulteta za matematiko in fiziko

3.3. LASTNOSTI DVOJIŠKIH DREVES

V dvojiškem drevesu ima vsako vozlišče kvečjemu dva sinova. Iz tega dejstva lahko izpeljemo nekaj lastnosti dvojiški dreves.

3.3.1. ŠTEVILO LISTOV

Dvojiško drevo višine k ima največ 2^{k-1} listov. Točno število listov je odvisno oblike dvojiškega drevesa in se giblje od 1 (izrojeno drevo) do 2^{k-1} (polno drevo). Na sliki vidimo primer štirih dreves višine 3, ki imajo 1,2,3 in 4 liste.



3.3.2. ŠTEVILO VOZLIŠČ

Največje število vozlišč dvojiškega drevesa višine k je

$$n_{\max} = \sum_{k=1}^{i} 2^{k-1} = 2^{k} - 1$$

DOKAZ:
Dokažimo najprej da ima drevo na k-tem nivoju največ $2^{(k-1)}$ vozlišč. To dokažemo s popolno indukcijo.
Naj bo $k = 1$. Na prvem nivoju imamo samo koren, torej trditev velja.
Naj sedaj trditev velja za nivo k. Na nivoju k imamo torej $2^{(k-1)}$ vozlišč. Ker ima vsako vozlišče največ dva sinova, je torej na $k+1$-vem nivoju največ $2 * 2^{(k-1)} = 2^k$ vozlišč.
Trditev torej velja tudi za nivo $k+1$ in trditev velja v splošnem.

Sedaj ni več težko dokazati, da je največje število vozlišč drevesa višine k enako $2^k - 1$. Seštejemo maksimalno število vozlišč na vsakem nivoju, upoštevamo formulo za vsoto geometrijske vrste in dobimo

$$n_{\max} = \sum_{k=1}^{i} 2^{k-1} = 2^k - 1$$

Polno dvojiško drevo višine k ima natančno $n = 2^k - 1$ vozlišč.

Dopolnimo naš razred `Drevo` z metodo, ki prešteje število vozlišč v drevesu.

Metoda število_vozlišč:
Algoritem rekurzivno prešteje vozlišča v levem poddrevesu in desnem poddrevesu, rezultata sešteje in prišteje 1 (koren).

```
public int stevilo_vozlisc(){
// metoda, ki vrne število vozlišč v dvojiškem drevesu
    if(prazno()){ //v praznem jih ni
        return 0;
    }
    // številu vozlišč v levem poddrevesu prištejemo število vozlišč v desnem
    // na koncu prištejemo še koren, zato + 1
    else return leviSin().stevilo_vozlisc() + desniSin().stevilo_vozlisc() + 1;
}
```

3.3.3. VIŠINA DREVESA

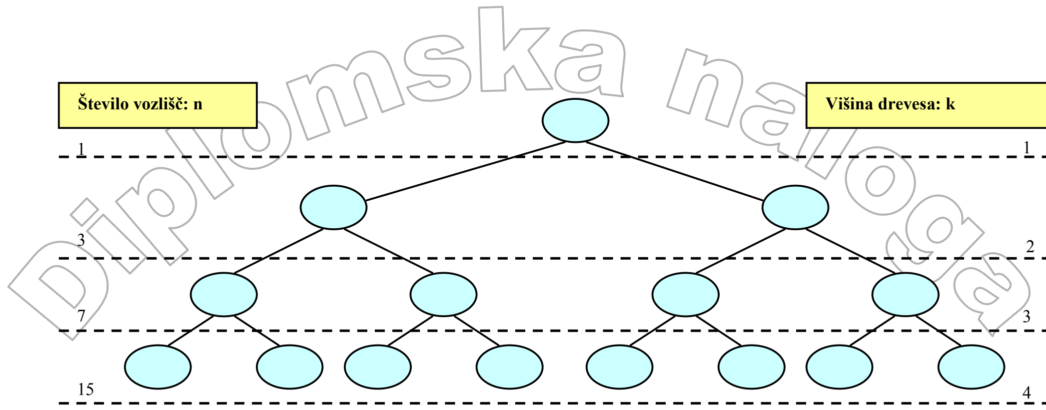
Višina drevesa je enaka največjemu nivoju vozlišča v drevesu. Lahko jo definiramo tudi kot:

- višina praznega drevesa je enaka 0
- višina nepraznega drevesa je enaka 1 + max (višina levega poddrevesa, višina desnega poddrevesa)

Višina drevesa z enim samim vozliščem je torej enaka 1.

Poglejmo v kakšnem odnosu sta višina drevesa in število vozlišč dvojiškega drevesa. Med vsemi drevesi z n vozlišči ima polno dvojiško drevo minimalno višino. Zato si najprej oglejmo kakšno je razmerje med številom vozlišč in višino polnega drevesa.

Diplomska naloga
Fakulteta za matematiko in fiziko



V prejšnjem razdelku smo pokazali, da ima drevo višine k največ $2^k - 1$ vozlišč. Pri polnem drevesu to mejo tudi dosežemo. Drevo ima na nivoju k največ 2^{k-1} vozlišč.

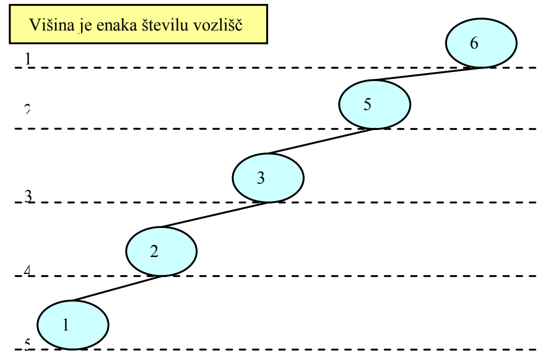
Če je v drevesu torej n vozlišč, nam kratek račun

$$n = 2^k - 1$$
$$n + 1 = 2^k$$
$$k = \log_2(n+1)$$

pokaže, da je najmanjša višina drevesa z n vozlišči

$$k = \log_2(n+1)$$

Največjo višino drevesa z n vozlišči dobimo v primeru izrojenega drevesa. Višina je kar enaka številu vozlišč drevesa.



Zgoraj predstavljeni primer, primer levega izrojenega drevesa, je le en izmed primerov izrojenega drevesa.

Drevo z n vozlišči ima torej višino med $\log_2(n+1)$ in n.

$$\log_2(n+1) \leq \text{višina(d, n)} \leq n$$

Dopolnimo razred `Drevo` z metodo, ki določi višino dvojiškega drevesa.

Metoda višina:

// metoda višina, ki določi višino dvojiškega drevesa

```
public int visina(){
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
    //vrne višino dvojiškega drevesa
    if(prazno()){   //če je prazno, je višina nič
       return 0;
    }
    // k maksimumu obeh višin  prištejemo še koren
    return  Math.max(leviSin().visina(), desniSin().visina()) + 1;
}
```

Diplomska naloga

3.4. ČASOVNA ZAHTEVNOST ALGORITMOV NAD DVOJIŠKIMI DREVESI

Veliko algoritmov nad dvojiškimi drevesi lahko shematsko predstavimo takole:

```
algoritem nekaj(Drevo d){
   if d.prazno() {
      naredi1;
      vrni rezultat;
   }
   rezultat_levo  = nekaj(d.levopoddrevo());
   rezultat_desno = nekaj(d.desnopoddrevo());
   // kombiniramo rezultat iz levega, desnega poddrevesa in koren
   združi_rešitve (rezultat_levo, rezultat_desno, d.vrni());

   vrni rezultat;
}
```

Če analiziramo časovno zahtevnost takega algoritma v odvisnosti od števila vozlišč drevesa, vidimo, da je odvisna od strukture drevesa, oziroma od njegove višine. Časovna zahtevnost bo torej v ugodnem primeru sorazmerna z logaritmom števila podatkov. To bo veljalo tako pri polnem drevesu, kot pri drevesih, kjer za vsa poddrevesa velja, da je v levem poddrevesu približno enako število podatkov kot v desnem poddrevesu.

Fakulteta za matematiko in fiziko

Diplomska naloga

4. ISKALNO DVOJIŠKO DREVO

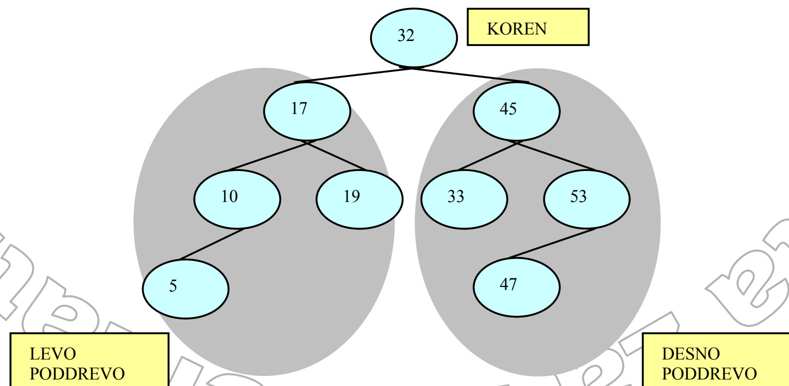
Če podatke hranimo v dvojiškem drevesu in pogosto uporabljamo postopek, s katerim iščemo nek element v tej množici podatkov, je smiselno, da za hranjenje te množice uporabimo posebno obliko dvojiškega drevesa – iskalno dvojiško drevo. To vsebuje podatke, urejene po nekem ključu. V samem drevesu ni podvojenih elementov, vsi podatki so torej različni. Iskalno dvojiško drevo pogosto uporabljamo tudi za predstavitev slovarja ali vrste s prednostjo

4.1. DEFINICIJA

Iskalno dvojiško drevo je dvojiško drevo, za katerega velja, da so vsi elementi v levem poddrevesu manjši od korena in v desnem poddrevesu večji od korena. Tudi levo in desno poddrevo sta iskalni dvojiški drevesi.

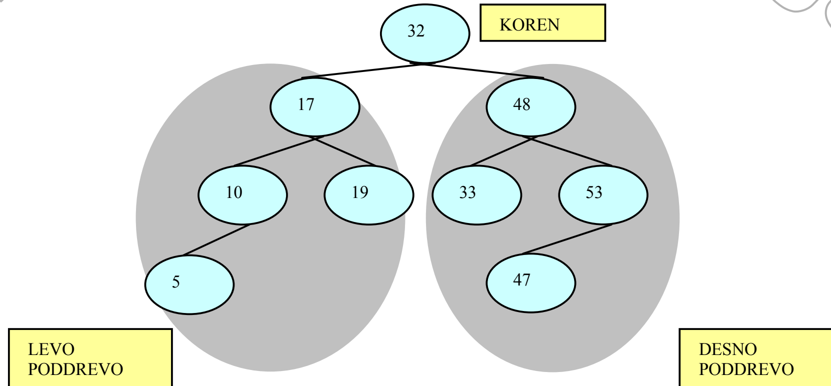
4.2. ZGRADBA ISKALNEGA DVOJIŠKEGA DREVESA

Iskalno dvojiško drevo je torej zgrajeno enako kot dvojiško drevo, s to razliko, da so podatki v drevesu urejeni po nekem ključu. Za predstavitev iskalnega dvojiškega drevesa bomo uporabili standardno predstavitev iskalnega dvojiškega drevesa; vozlišče s podatkom, ter referencami na levo in desno poddrevo, ter referenco na očeta.



Fakulteta za matematiko in fiziko

Pri sami definiciji iskalnega dvojiškega drevesa moramo biti pozorni na to, da morata biti obe poddrevesi tudi iskalni poddrevesi. Za primer na spodnji sliki sicer velja, da so vsi podatki v levem poddrevesu manjši od korena, in podatki v desnem poddrevesu večji od korena, a ker desno poddrevo ni iskalno drevo (njegov koren je večji od enega od elementov v njegovem desnem poddrevesu), celotno drevo ni iskalno drevo.



4.3. OPERACIJE NAD ISKALNIMI DVOJIŠKIMI DREVESI

Poleg operacij, ki jih poznamo že iz splošnih dvojiških dreves, je osnovna operacija nad vsemi iskalnimi dvojiškimi drevesi iskanje podatka, shranjenega v drevesu. Poleg te operacije *išči*, v iskalnem dvojiškem drevesu pogosto izvedemo tudi operacije *najmanjši*, *največji*, *vstavi*, *briši*,…Vse te naštete operacije nad iskalnimi dvojiškimi drevesi imajo časovno zahtevnost $O(h)$, kjer je h višina iskalnega drevesa.

4.3.1. IŠČI

Oglejmo si najprej, kako v splošnem dvojiškem drevesu poiščemo, če je v njem dani element. Metoda naj vrne `true`, če element je v drevesu in `false` sicer.
Izhajamo iz splošne definicije dvojiškega drevesa. V praznem drevesu vemo, da podatka ni. Če drevo ni prazno, je podatek lahko v korenu. Če je, smo s postopkom zaključili. V nasprotnem primeru bomo element iskali v levem in zatem še v desnem poddrevesu.

Metoda *jeVDrevesu*:

```
// metoda s katero poiščemo element v splošnem dvojiškem drevesu
public static boolean jeVDrevesu(Drevo drevo, int elt){
    //vrne true če je v drevesu element elt in false sicer
    if(drevo.prazno()) return false;   // v praznem drevesu elementa ni
    if(drevo.vrni() == elt) return true; // našli
    return(jeVDrevesu(drevo.leviSin(), elt) ||   //iščemo levo in desno
           jeVDrevesu(drevo.desniSin(),elt));
}
```

Vidimo, da moramo v splošnem pregledati praktično vse elemente dvojiškega drevesa. Prav vsa vozlišča pa pregledamo vedno, ko iščemo podatek, ki ni v drevesu.

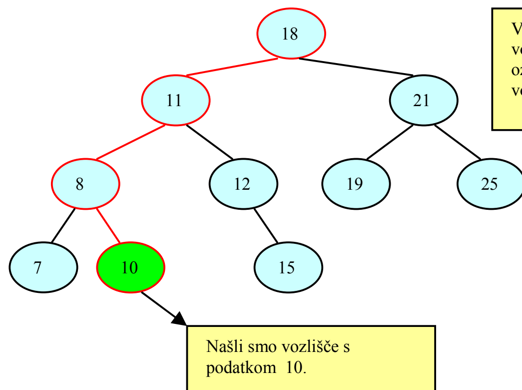
Diplomska naloga Fakultete za matematiko in fiziko

V iskalnem dvojiškem drevesu pri iskanju elementa uporabimo dejstvo, da so podatki urejeni. Če podatka ne najdemo v korenu, se vprašamo, ali je vrednost, ki jo iščemo, večja ali manjša od tiste v korenu. Če je manjša, vemo, da podatka zagotovo ni v desnem poddrevesu in ga iščemo levo. Če pa je iskani podatek večji od korena, ga iščemo v desnem poddrevesu. Ko se odločimo za levo ali desno pot, v ugodnem primeru (če je v levem poddrevesu toliko vozlišč kot v desnem) razpolovimo število vozlišč, med katerimi iščemo naš podatek.
V vsakem primeru bomo na tekočem koraku bodisi našli odgovor, bodisi prešli iz i-tega na i +1-vi nivo. Časovna zahtevnost je torej res $O(h)$, kjer je h višina iskalnega drevesa.

Metoda `jeVIskalnemDrevesu`:

// metoda s katero poiščemo element v iskalnem dvojiškem drevesu

```
public static boolean jeVIskalnemDrevesu(Drevo drevo, int elt){
        // ali v iskalnem drevesu drevo obstaja element elt
        if (drevo.prazno()) return false; // v praznem drevesu ni podatka
        if (drevo.vrni() == elt) return true; // ali je iskani element koren
        if (elt < drevo.vrni()) return jeVIskalnemDrevesu(drevo.leviSin(),
        elt);
        // iščemo levo
        return jeVIskalnemDrevesu(drevo.desniSin(),elt); // iščemo desno
}
```



4.3.2. MINIMUM IN MAKSIMUM

Denimo, da v iskalnem drevesu iščemo najmanjši element. V splošnem drevesu moramo pregledati vsa vozlišča in med njimi poiskati najmanjšega, v iskalnem dvojiškem drevesu pa lahko izrabimo urejenost podatkov. Ker so vsi podatki v levem poddrevesu manjši od korena, bomo najmanjši element iskali v levem poddrevesu. Ker je levo poddrevo spet iskalno drevo, uporabimo kar isti postopek (torej rekurzivni klic). Izjema je le, če drevo levega poddrevesa nima. Ker so vsi podatki v desnem poddrevesu (pri praznem desnem poddrevesu je pogoj na prazno izpolnjen) večji od korena, je v tem primeru najmanjši element kar koren drevesa.

Metoda NAJMANJŠI:

// metoda, ki v iskalnem dvojiškem drevesu poišče najmanjši element. Če je drevo prazno vrže
// izjemo

```
//vrne najmanjši element v iskalnem dvojiškem drevesu
public static int najmanjsi(Drevo drevo) throws Exception{
        if (drevo.prazno())
```

Diplomska naloga
Fakulteta za matematiko in fiziko

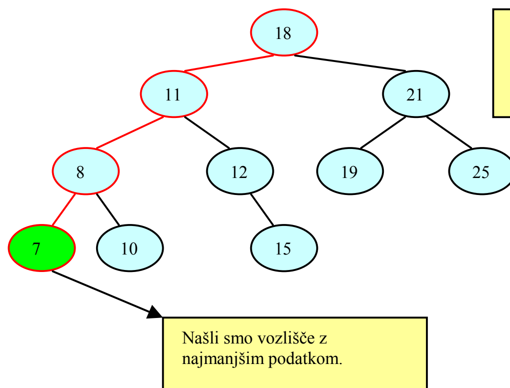
```
            throw new Exception(" Drevo je prazno"); // v drevesu ni najmanjšega
        if (drevo.leviSin().prazno()) {
            return drevo.vrni(); // če levega poddrevesa ni, je koren najmanjši
        }
        return najmanjsi(drevo.leviSin()); // najmanjši je v levem poddrevesu

}
```

Če premislimo, kako postopek deluje, vidimo, da je najmanjši element v iskalnem dvojiškem drevesu kar najbolj levo vozlišče. To lahko izrabimo za to, da napišemo nerekurzivni postopek.

Metoda MIN:

// nerekurzivna metoda za iskanje najmanjšega elementa

```
public static int min(Drevo drevo) throws Exception{
        if(drevo.prazno()){
            throw new Exception(»Drevo je prazno.«); // v praznem drevesu ga ni
        }
        else{
            int min;
            while(!drevo.prazno()){  // dokler drevo ni prazno
                min = drevo.vrni();  // podatek v korenu je trenutno najmanjši element
                drevo = drevo.leviSin(); // premaknemo se v levo
            }
            return min; // vrnemo min
        }
}
```



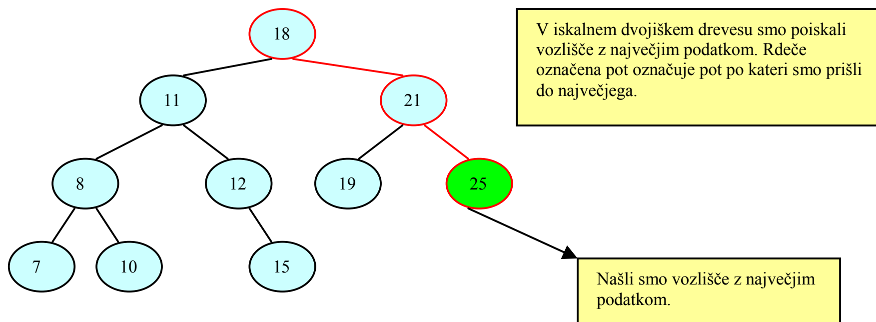
Metoda. ki poišče največji element v iskalnem drevesu, je simetrična metodi iskanja najmanjšega elementa. Ker so vsi podatki v levem poddrevesu manjši od korena, bomo največji element iskali v desnem poddrevesu. In ker je desno poddrevo spet iskalno drevo, uporabimo kar isti postopek (torej rekurzivni klic). Izjema je le, če drevo desnega poddrevesa nima. Ker so vsi podatki v levem poddrevesu (pri praznem levem poddrevesu je pogoj na prazno izpolnjen) manjši od korena, je v tem primeru največji element kar koren drevesa.

Diplomska naloga Fakulteta za matematiko in fiziko

Diplomska naloga

Metoda NAJVEČJI:
// metoda, ki v iskalnem dvojiškem drevesu poišče največji element. Če je drevo prazno, vrže izjemo

```
public static int najvecji(Drevo drevo) throws Exception {
// vrne največji element v iskalnem dvojiškem drevesu. Če je drevo prazno vrže izjemo
        if(drevo.prazno())
            throw new Exception(" Drevo je prazno"); // v drevesu ni največjega
        if(drevo.desniSin().prazno()) {
            return drevo.vrni();   // če desnega poddrevesa ni, je koren največji
        }
         return najvecji(drevo.desniSin());   // največji je v desnem poddrevesu
}
```



Podobno kot pri iskanju minimuma, tudi tu vidimo, da je največji element v iskalnem dvojiškem drevesu kar najbolj desno vozlišče. Napišimo še nerekurzivni postopek.

Metoda MAX:
// nerekurzivna metoda za iskanje največjega elementa

```
public static int max(Drevo drevo) throws Exception{
        if(drevo.prazno()){
            throw new Exception(»Drevo je prazno.«); // v praznem drevesu ga ni
        }
        else{
            int max = drevo.vrni();
            while(!drevo.prazno()){   // dokler drevo ni prazno
                max = drevo.vrni();   // podatek v korenu je trenutno največji element
                drevo = drevo.desniSin(); // premaknemo se po drevesu navzdol
            }
            return max; // vrnemo max
        }
}
```

Obe metodi iskanja minimuma in maksimuma imata v najslabšem primeru časovno zahtevnost $O(h)$, kjer je h višina drevesa.

Fakulteta za matematiko in fiziko

Diplomska naloga

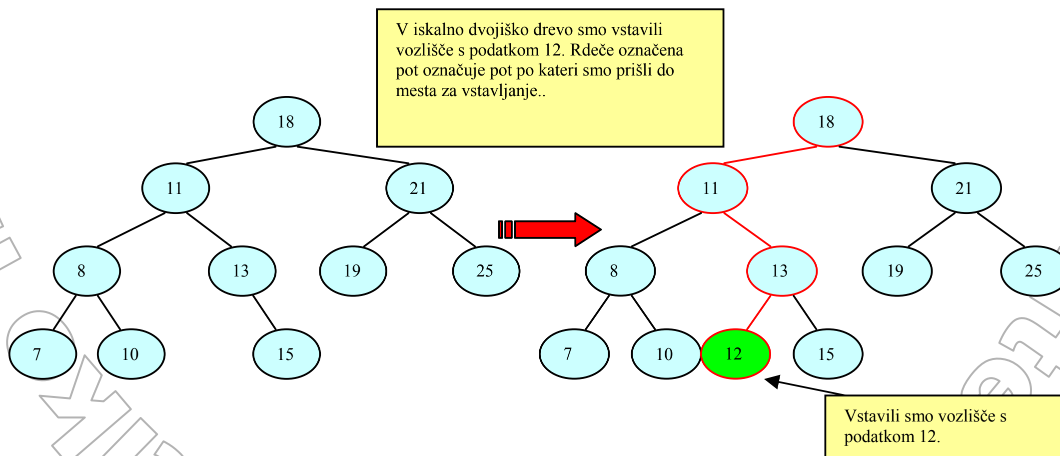
4.3.3. VSTAVLJANJE

Za vstavljanju podatka v iskalno dvojiško drevo sestavimo metodo *vstavi*. Ker vstavljamo podatek v urejeno drevo, s primerjavo podatka s korenom takoj ugotovimo, kam moramo vstaviti podatek. Če je le ta manjši od korena, postopek rekurzivno nadaljujemo v levem poddrevesu, drugače pa v desnem poddrevesu. Če je podatek, ki ga vstavljamo enak korenu, ne naredimo nič, saj smo rekli, da v iskalnem drevesu nimamo podvojenih elementov.

Časovna zahtevnost vstavljanja podatka v iskalno dvojiško drevo je zahtevnosti $O(h)$, kjer je h višina drevesa..

Metoda VSTAVI:

```
// metoda vstavi, ki v iskalno dvojiško drevo vstavi nov element
public void vstaviVDrevo(int nPodatek){
        // vstavi element v drevo
        koren = vstaviVDrevo(koren, nPodatek);
}
public Vozel vstaviVDrevo(Vozel koren, int nPodatek){
        if(koren == null){ // vstavljamo v prazno drevo
           koren = new Vozel(nPodatek);
        }
        else{
           if(nPodatek < koren.vrniPodatek()){   // nov vozel v levo poddrevo
              koren.nastaviLevi(vstaviVDrevo(koren.vrniLevi(), nPodatek));
           }
           else{   // nov vozel v desno poddrevo
              koren.nastaviDesni(vstaviVDrevo(koren.vrniDesni(),nPodatek));
           }
        }
        return koren;
}
```

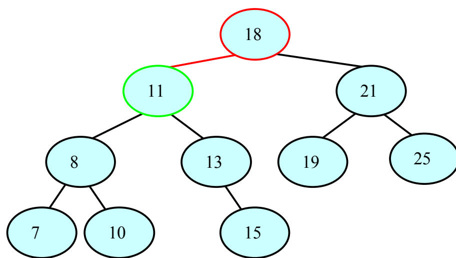


Fakulteta za matematiko in fiziko

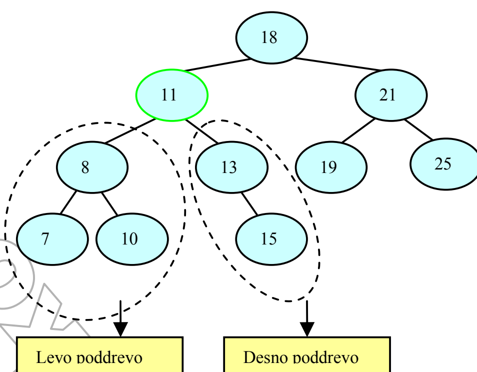
4.3.4. BRISANJE

Za brisanje podatka v iskalnem dvojiškem drevesu sestavimo metodo *briši*. Pri odstranjevanju vozlišča v iskalnem dvojiškem drevesu imamo več možnosti. Brišemo lahko vozlišče, ki je list, brišemo vozlišče z enim ali pa brišemo vozlišče z dvema sinovoma. Prvi dve možnosti sta enostavni. Če brišemo vozlišče, ki je list, ga enostavno zbrišemo (v njegovem očetu referenco nastavimo na null). Če ima brisano vozlišče enega naslednika, ga nadomestimo z njim (le popravimo ustrezno referenco njegovega očeta). Kadar ima vozlišče, ki ga želimo odstraniti, oba naslednika, vozlišča ne moremo preprosto odstraniti. Nadomestiti ga moramo z najbližjim po velikosti. Kandidata za to sta največji med od njega manjšimi podatki (najbolj desno vozlišče v levem poddrevesu) ali najmanjši podatek med od njega večjimi (najbolj levo vozlišče v desnem poddrevesu). Naša metoda *briši* za naslednika izbrisanega vozlišča vzame najbolj levo vozlišče v desnem poddrevesu. Vozlišče, ki ga želimo odstraniti, najprej poiščemo tako, kot smo to storili pri metodi *išči.* Ko ga najdemo, nad njim izvedemo metodo *izbrisiVozel.* Pri tej z rekurzivnim postopkom poiščemo vozlišče, s katerim nadomestimo zbrisano vozlišče. To je najbolj levo vozlišče v desnem poddrevesu. Če koren desnega poddrevesa nima levega poddrevesa, potem je najbolj levo vozlišče kar koren.
Časovna zahtevnost brisanja podatka v iskalnem dvojiškem drevesu je zahtevnosti $O(h)$, kjer je h višina drevesa.

Na primeru si poglejmo, kako deluje naš algoritem za brisanje elementa v dvojiškem drevesu:

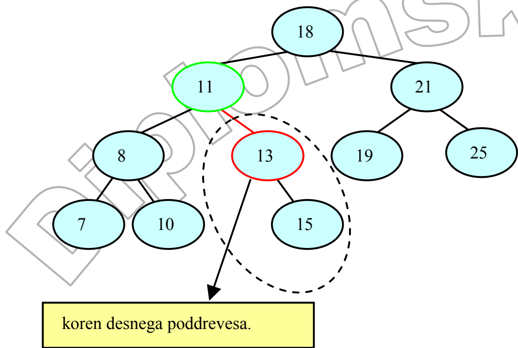


V drevesu bi želeli zbrisati podatek 11. Vozlišče s podatkom, ki ga brišemo, najprej poiščemo po principu metode išči.

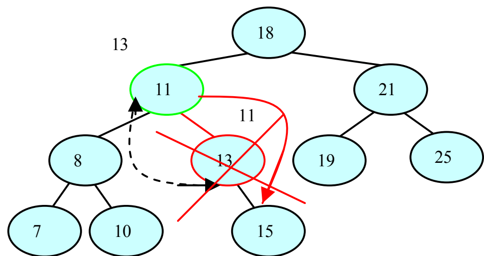


Ko smo vozlišče 11 našli, smo pripravljeni za brisanje. Ker ima vozlišče obe poddrevesi, poiščemo skrajno levo vozlišče v desnem poddrevesu, da z njim nadomestimo brisano vozlišče.

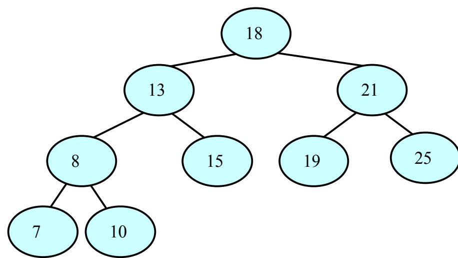
Diplomska naloga
Fakulteta za matematiko in fiziko



Skrajno levo vozlišče torej iščemo v desnem poddrevesu. Ker koren desnega poddrevesa nima levega poddrevesa, potem je kar koren skrajno levo vozlišče. V našem primeru je to vozlišče 13.
Sedaj, ko smo našli vozlišče s katerim bomo nadomestili brisano vozlišče, moramo popraviti kazalce.

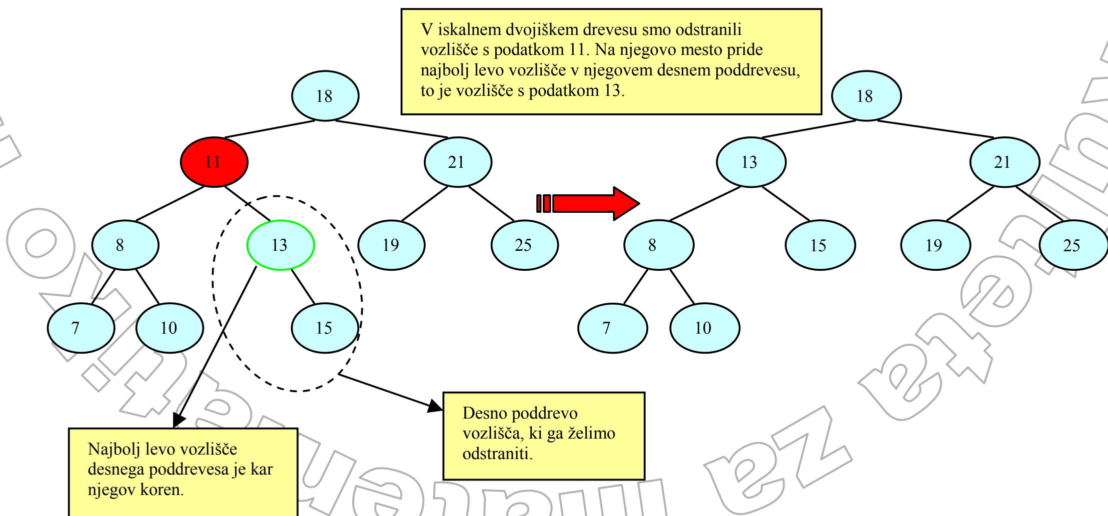


Najprej zamenjamo podatke v vozlišču ki ga želimo zbrisati, s podatkom vozlišča, ki pride na njegovo mesto. Kazalec, ki kaže na desno poddrevo brisanega vozlišča, sedaj pokaže na koren desnega poddrevesa nadomestnega vozlišča. To vozlišče namreč nima levega poddrevesa. Tako nam preostane samo še da zbrišemo vozlišče.



Po končanem algoritmu je naše drevo tako.

Če vse skupaj povzamemo z eno sliko:



Diplomska naloga
Fakulteta za matematiko in fiziko

Metoda BRIŠI:

```
//metoda briši,ki iz iskalnega dvojiškega drevesa odstrani  element
public static void brisi(int nPodatek) throws Exception{
     koren = brisi(koren, nPodatek);
}

public Vozel brisi (Vozel koren, int nPodatek) throws Exception{
    if(koren == null){
        throw new Exception(" Drevo je prazno"); // če je drevo prazno ne naredimo nič
    }
    else{
        if(nPodatek == koren.vrniPodatek()){ // našli smo podatek
            return izbrisiVozel(koren);    // in ga zbrišemo
        }
        else if(nPodatek < koren.vrniPodatek()){ / /podatek manjši od korena
            // gremo rekurzivno v levo in odstranimo podatek
            koren.nastaviLevi(brisi(koren.vrniLevi(), nPodatek));
        }
        else{
          // drugače gremo rekurzivno  v desno in tam odstranimo podatek
            koren.nastaviDesni(brisi(koren.vrniDesni(), nPodatek));
        }
    }
    //vrnemo koren drevesa
    return koren;
}


//  iz drevesa odstranimo vozel
private static Vozel izbrisiVozel(Vozel koren) {
    if(koren.vrniLevi() == null){  // levega poddrevesa ni
    koren = koren.vrniDesni();  // nastavimo koren desnega poddrevesa
    }
    else if(koren. vrniDesni() == null){  // desnega poddrevesa ni
        koren = koren.vrniLevi();  // nastavimo koren levega poddrevesa
    }
    else{  // imamo desno in levo poddrevo
        // poiščemo skrajno levi vozel v desnem poddrevesu,
        Vozel zadnji = poisciSkrajniLevi(koren.vrniDesni());
        // zamenjamo podatke
        koren.nastaviPodatek(zadnji.vrniPodatek());
        // zbrišemo skrajno levega
        koren.nastaviDesni(zbrisiSkrajniLevi(koren.vrniDesni()));
    }
    return koren;
}

// poiščemo skrajni levi vozel
private static Vozel poisciSkrajniLevi(Vozel koren) {
    if(koren.vrniLevi() != null) {  // levo poddrevo ni prazno
        // vrne skrajno levega
        return poisciSkrajniLevi(koren.vrniLevi());
    }
    else{
        // najmanjši je kar koren
        return koren;
    }
}

// izbrišemo skrajni levi vozel
private static Vozel zbrisiSkrajniLevi(Vozel koren) {
    if(koren.vrniLevi() != null) {  // levo poddrevo ni prazno
    // gremo v levo in ga zbrišemo
        koren.nastaviLevi(zbrisiSkrajniLevi(koren.vrniLevi()));
        return koren;
    }
    else{
        return koren.vrniDesni();  // vrne koren desnega poddrevesa
```

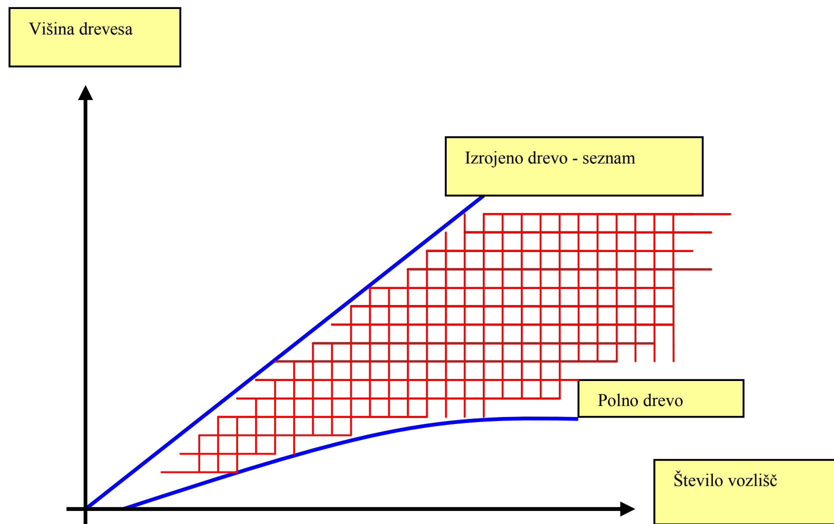
Diplomska naloga Fakultete za matematiko in fiziko

```
    }
}
```

Diplomska naloga

4.4. O ČASOVNI ZAHTEVNOSTI OPISANIH POSTOPKOV

Pri vseh omenjenih postopkih lahko pokažemo, da imajo v najslabšem primeru enako časovno zahtevnost. Ta je za iskalno dvojiško drevo višine h enaka $O(h)$. Ker vemo, da se višina dvojiškega drevesa z n vozlišči giblje med n in $\log(n)$, je zahtevnost teh postopkov bodisi logaritemska, bodisi linearna. Slednji bi se seveda radi izognili.
Težava je v tem, da lahko iskalno dvojiško drevo kaj hitro dobi veje zelo neenakomernih dolžin. Če na primer z metodo *vstavi* v iskalno dvojiško drevo zaporedoma vstavljamo urejeno zaporedje podatkov, dobimo izrojeno drevo (seznam). V tem primeru so opisani postopki linearne in ne logaritemske časovne zahtevnosti. Torej v primerjavi s splošnim dvojiškim drevesom nismo prihranili prav nič.
Če podatke, ki jih bomo hranili v iskalnem drevesu, poznamo že od prej, seveda lahko poskrbimo, da bo zgrajeno drevo kar se da »idealno« (torej čim bolj podobno polnemu). Vendar ko nad takim drevesom izvajamo operacije *vstavi* in *briši*, ti dve lahko povzročita, da se struktura iskalnega dvojiškega drevesa spremeni. V skrajnem primeru spet dobimo izrojeno ali izrojenemu podobno drevo. Zato bi radi imeli tako različico iskalnega dvojiškega drevesa, kjer bi bile vse operacije izvedene tako, da se »idealna« struktura iskalnega dvojiškega drevesa obdrži – torej da je njegova višina kar se da blizu logaritma števila podatkov.



Graf ponazarja razmerje med številom vozlišč in višino drevesa. Skrajni meji sta izrojeno drevo - linearni seznam in polno drevo. Dejanska višina iskalnega drevesa je nekje v črtkanem območju. Želimo, da bi bila čim bližje spodnji krivulji. To nam omogočajo uravnotežena drevesa.

Fakulteta za matematiko in fiziko

Diplomska naloga

5. URAVNOTEŽENA DVOJIŠKA DREVESA

V prejšnjem razdelku smo povedali, da je časovna zahtevnost vseh osnovnih operacij nad iskalnim dvojiškim drevesom (*vstavi*, *briši*, *išči*) odvisna od višine drevesa. Zato želimo pri danem številu vozlišč višino drevesa kar se da omejiti. Preprečiti torej želimo, da bi se drevo izrodilo – torej postalo podobno linearnemu seznamu. Najlažje to storimo tako, da ob vsakem vstavljanju in brisanju preverimo njegovo zgradbo. To pomeni, da sproti preverjamo razliko med višinama levega in desnega poddrevesa. Če se drevo razvija enostransko (razlika med višinama raste), ga je potrebno preurediti oziroma uravnotežiti.

5.1. DEFINICIJA URAVNOTEŽENEGA DREVESA

Drevo je uravnoteženo natanko tedaj, kadar se v vsakem vozlišču višini njegovih poddreves razlikujeta kvečjemu za ena.

Med uravnotežena drevesa sodijo:

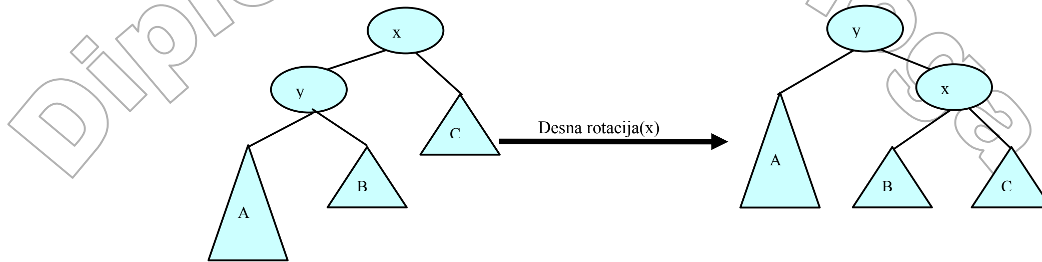
- AVL drevesa
- rdeče – črna drevesa
- 2 – 3 drevesa
- lomljena drevesa
- …

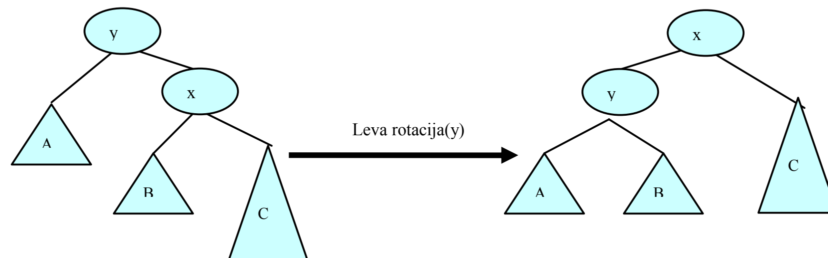
V nadaljevanju si bomo ogledali AVL drevesa, rdeče – črna drevesa in 2 – 3 drevesa.

Preden začnemo z obravnavo posameznih vrst dreves, si bomo pogledali primere rotacij, ki jih bomo uporabljali. Poznamo levo rotacijo, desno rotacijo, levo-levo rotacijo, desno-desno rotacijo, levo-desno rotacijo in desno-levo rotacijo. Pri desni rotaciji gre le za simetrijo leve rotacije, zato imamo pravzaprav le tri bistveno različne tipe rotacij.

Fakulteta za matematiko in fiziko

Slika prikazuje osnovni vrsti rotacije. Poddrevesa A, B in C naj ne bodo prazna. Desno rotacijo okoli vozlišča x izvedemo tako, da vse skupaj »zavrtimo« v desno.





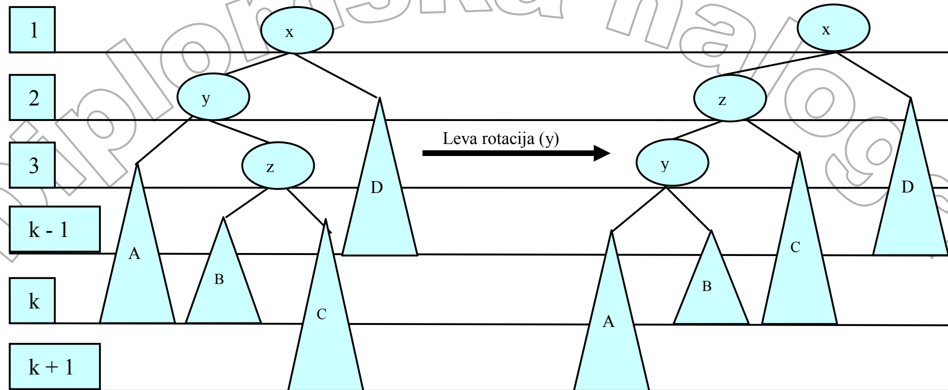
Leva rotacija je simetrična desni. Vozlišče x postane novi koren poddrevesa, vozlišče y pa njegov levi sin. Vozlišče y ima sedaj kot desnega sina bivšega levega sina vozlišča x, torej koren drevesa B.

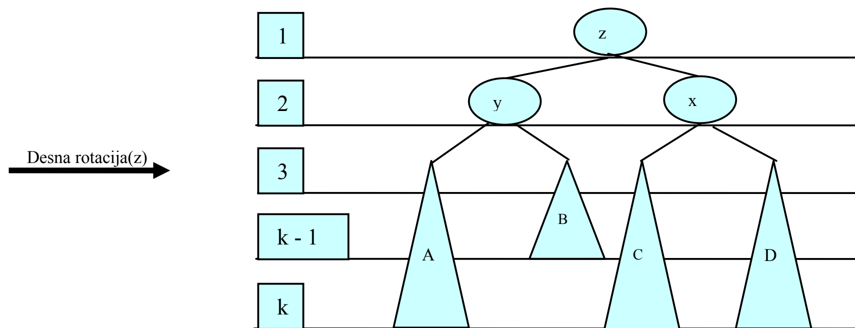
Obe osnovni rotaciji (desna in leva) predelata drevo iz ene oblike v drugo s spremembo konstantnega števila kazalcev. Spremeniti moramo po en kazalec v vozlišču x in y, skupaj torej dva kazalca.

Levo – desna rotacija se izvede, kadar je levo poddrevo višje od desnega poddrevesa (D), desno poddrevo vozlišča y je za 1 višje od poddrevesa A poddrevo C (desno poddrevo vozlišča z) za 1 višje od poddrevesa B. Pri iskanju ustreznega mesta za vstavljanje podatka smo šli od neuravnoteženega vozlišča torej dvakrat v levo. Pri levo – desni rotaciji najprej izvedemo rotacijo v levo okrog vozlišč y in z. Potem pa se izvede še desna rotacija, okoli vozlišč x in z.

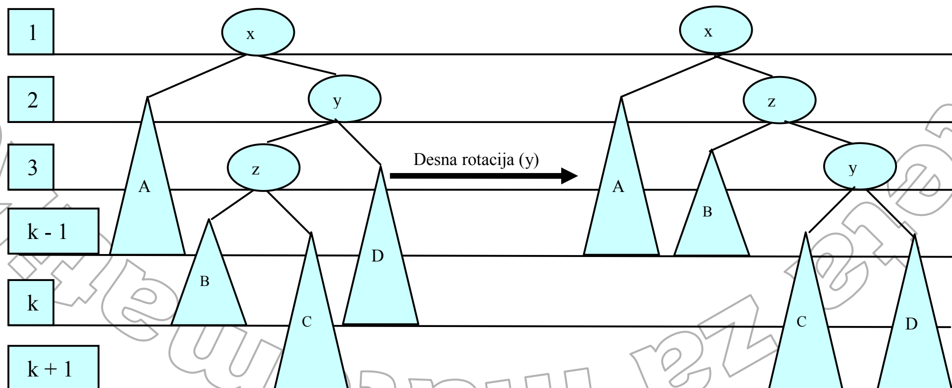
Pri levi rotaciji vozlišče z postane novi koren poddrevesa, vozlišče y pa njegov levi sin. Vozlišče y ima sedaj kot desnega sina bivšega levega sina vozlišča z, torej koren drevesa B. Pri desni rotaciji vozlišče z postane novi koren drevesa, vozlišče y ostane njegov levi sin, vozlišče x pa postane njegov desni sin. Vozlišče x dobi novega levega sina in sicer bivšega desnega sina vozlišča z.

Diplomska naloga
Fakulteta za matematiko in fiziko

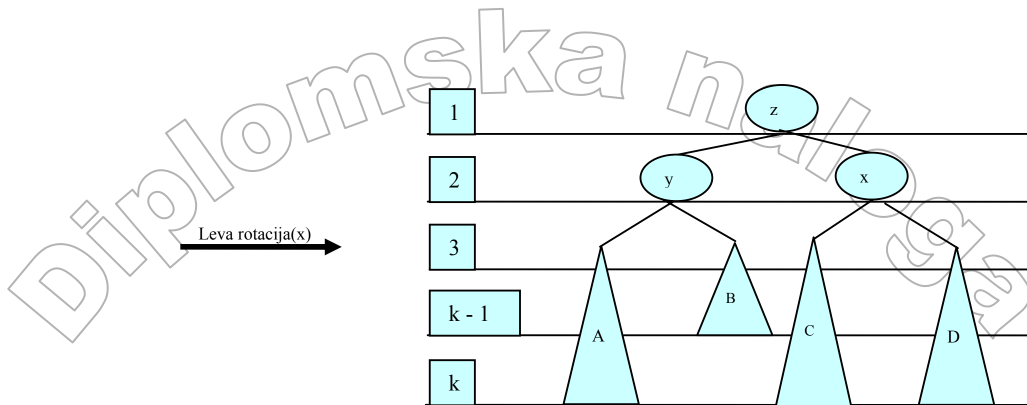




Desno – leva rotacija se izvede, kadar je višina desnega poddrevesa za 2 višja od levega poddrevesa. Levo poddrevo vozlišča y je za 1 višje od desnega poddrevesa vozlišča y. Desno poddrevo vozlišča z je za 1 višje od levega poddrevesa vozlišča z. Pri desno – levi rotaciji se najprej izvede desna rotacija okrog vozlišč y in z, nato pa se izvede še leva rotacija okoli vozlišč x in z.
Pri desni rotaciji vozlišče z postane novi koren poddrevesa, vozlišče y pa njegov desni sin. Vozlišče y ima sedaj kot levega sina bivšega desnega sina vozlišča z, torej koren drevesa C. Pri levi rotaciji vozlišče z postane novi koren drevesa, vozlišče x pa njegov levi sin, vozlišče y pa ostane njegov desni sin. Vozlišče x dobi novega desnega sina in sicer bivšega levega sina vozlišča z.



Diplomska naloga
Fakulteta za matematiko in fiziko



Levo – leva in desno – desna rotacija sta podobni levo – desni oziroma desno – levi rotaciji in ju ne bomo posebej opisovali.

DEFINICIJA RAVNOTEŽNEGA FAKTORJA:

Naj višina(D) označuje višino drevesa D, višina(D_l) in višina(D_d) pa višini levega in desnega poddrevesa iskalnega dvojiškega drevesa D. Vpeljemo absolutni faktor ravnotežja:

$$fA(D) := \left| višina(D_l) - višina(D_d) \right|$$

Za uravnoteženo iskalno dvojiško drevo velja za vsako poddrevo:

$$fA(D) \leq 1$$

Ker bo za vsako vozlišče potrebno poznati ne le gornjo razliko, ampak tudi njen predznak, vpeljemo še faktor ravnotežja:

$$f(D) := višina(D_l) - višina(D_d)$$

Diplomska naloga
Fakulteta za matematiko in fiziko

5.2. AVL DREVO

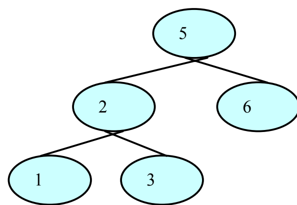
AVL drevo je dvojiško iskalno drevo za katerega velja:

- prazno drevo je AVL drevo
- višina desnega in levega poddrevesa se lahko razlikuje za največ 1 ($fA(D) \leq 1$)
- levo in desno poddrevo sta tudi AVL drevesi

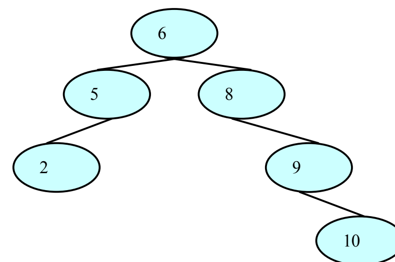
V nobenem poddrevesu AVL drevesa se torej višini levega in desnega poddrevesa ne razlikujeta več kot za 1.

Koren AVL drevesa je:

1. levo-višinski, če je levo poddrevo višje od desnega poddrevesa ($f(D) > 0$)
2. desno-višinski, če je desno poddrevo višje od levega ($f(D) < 0$)
3. uravnotežen, če sta višini enaki ($f(D) = 0$)



AVL drevo



NI AVL drevo

Adelson-Velskii in Landis (začetnice njunih priimkov so torej dale poimenovanje AVL drevesom) sta dokazala, da je red hitrosti naraščanja višine uravnoteženega drevesa glede na število vozlišč drevesa $O(\log(n))$, kar je enako kot pri polnem drevesu.

Pri AVL drevesu v vozlišču poleg podatka hranimo še njegov ravnotežnostni faktor.

5.2.1. VSTAVLJANJE IN BRISANJE V AVL DREVESU

Ko v AVL drevo vstavimo nov podatek, ali izbrišemo obstoječega, se ravnotežje lahko poruši. Pride do kršitve AVL pravil:

- levo poddrevo je za več kot 2 višje od desnega ($f(D) \leq 2$)
- desno poddrevo je za več kot 2 višje od levega ($f(D) \geq -2$)

AVL ravnotežje se lahko podre kjerkoli, ni nujno da ravno pri korenu drevesa.
Za uravnoteževanje uporabimo vrsto enojnih ali dvojnih rotacij. Vsaka od njih ima svojo levo in desno različico.

Sestavimo razred `AVLDrevo`. Najprej definirajmo razred `AVLVozel`. Podroben zapis metod in konstruktorjev si lahko ogledate v prilogi diplome, tu pa bomo navedli le imena metod. Zaradi enostavnosti ne bomo uporabljali dedovanja. Navedli bomo le kodo ključnih metod, kot so rotacije in podobno. Metode namenoma niso napisane »optimalno«. Večkrat so po

Diplomska naloga
Fakulteta za matematiko in fiziko

nepotrebnem zelo časovno potratne, vendar je njihov namen le osnovna ilustracija postopkov in ne primer, kako bi bilo potrebno sprogramirati AVL drevo.

Razred AVLVozel:

```
// Razred AVLVozel :
//   * levi, desni (referenci na AVL naslednika);
//   * oce (referenca na očeta);
//   * podatek
//   * faktor ( faktor ravnotežja v vozlišču)
// Vsebuje ustrezne konstruktorje in metode s kateri upravljamo z AVL vozliščem

public class AVLVozel{

   private int podatek;       // podatek, ki ga hranimo v vozlišču
   private int faktor = 0;    // faktor ravnotežja v vozlišču
   private AVLVozel levi;     // referenca na levi AVLVozel
   private AVLVozel desni;    // referenca na desni AVLVozel
   private AVLVozel oce;      // referenca na očeta

   public AVLVozel(); // AVL vozrl s podatkom 0

   public AVLVozel(int p) // AVL vozel s podatkom p

   public AVLVozel(int p,AVLVozel o) // AVL vozel s podatkom p in očetom o

   public AVLVozel(int p, AVLVozel l, AVLVozel d)     // AVL vozel s podatkom p in
   // kazalcema na levega (l) in desnega (d) naslednika

   public void nastaviPodatek(int x) // metoda, ki nastavi podatel v vozlu

   public int vrniPodatek()   // metoda, ki vrne podatek v vozlu

   public void nastaviLevi(AVLVozel v) // metoda, ki nastavi levega naslednika

   public AVLVozel vrnilevi() //metoda, ki vrne levega naslednika

   public void nastaviDesni(AVLVozel v) // metoda, ki nastavi desnega naslednika

   public AVLVozel vrnidesni() // metoda, ki vrne desnega naslednika

   public void nastaviOce(AVLVozel v) // metoda, ki nastavi očeta

   public AVLVozel vrnioce() // metoda, ki vrne očeta

   public void nastaviFaktor(int x){ // metoda, ki nastavi faktor ravnotežja v vozlišču

   public int vrniFaktor()  // metoda, ki vrne faktor ravnotežja v vozlišču

   // metoda višina, ki nastavi višino vozlišča
   public int visina(){
      if(this.vrnidesni() != null || this.vrnilevi() != null ){
         int l = (this.vrnilevi()!= null)?(1 + this.vrnilevi().visina()):0;
         int d = (this.vrnidesni()!= null)?(1 + this.vrnidesni().visina()):0;
         return  Math.max(l, d);
      }
      else
```

Diplomska naloga Fakulteta za matematiko in fiziko

```
            return 0;
        }
    }
```

Razred AVLDrevo:

```
public class AVLDrevo{

    private AVLVozel koren;

    public AVLDrevo(){// konstruktor, ki ustvari prazno AVL drevo

    public AVLDrevo(AVLVozel v) // konstruktor, ki ustvari drevo z vozlom v

    public boolean prazno() // metoda, ki ugotovi ali je drevo prazno

    public AVLVozel koren()

    public int vrni()throws Exception  // metoda vrne drevo

    public AVLDrevo leviSin() // metoda, ki vrne levega sina (levo poddrevo)

    public AVLDrevo desniSin() // metoda, ki vrne desnega sina (desno poddrevo)

    public AVLDrevo oce() // metoda, ki vrne očeta

    // desna rotacija
    public AVLVozel desnaRotacija(AVLVozel x) {
            // nastavimo trenutna vozlišča
            AVLVozel tmp = x.vrnioce();
            AVLVozel tmp1 = x;
            AVLVozel tmp2 = x.vrnilevi();
            // če je oče obstaja
            if(tmp != null)
                // če je x levi sin
                if(tmp.vrnilevi() == tmp1)
                    // potem očetu nastavimo tmp2 za levega sina
                    tmp.nastaviLevi(tmp2);
                else
                    // drugače mu nastavimo tmp2 za desnega sina
                    tmp.nastaviDesni(tmp2);
            // naredimo prevezavo vozlišč
            tmp2.nastaviOce(tmp);
            tmp1.nastaviLevi(tmp2.vrnidesni());
            tmp2.nastaviDesni(tmp1);
            tmp1.nastaviOce(tmp2);
            // dokler tmp1 kaže na konkreten vozel
            while(tmp1 != null){
                // nastavimo faktor ravnotežja v vozlišču
                tmp1.nastaviFaktor(ravnotezje(tmp1));
                // pomaknemo se navzgor proti korenu
                tmp1 = tmp1.vrnioce();
            }
            // kot rezultat vrnemo tmp2
            return tmp2;
    }

    // leva rotacija
    public AVLVozel levaRotacija(AVLVozel x) {

            // nastavimo trenutna vozlišča
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
            AVLVozel tmp = x.vrnioce();
            AVLVozel tmp1 = x;
            AVLVozel tmp2 = x.vrnidesni();
            // če je oče obstaja
            if(tmp != null)
                  // če je x levi sin
                  if(tmp.vrnilevi() == tmp1)
                        // potem očetu nastavimo tmp2 za levega sina
                        tmp.nastaviLevi(tmp2);
                  else
                        // drugače mu nastavimo tmp2 za desnega sina
                        tmp.nastaviDesni(tmp2);
            // naredimo prevezavo vozlišč
            tmp2.nastaviOce(tmp);
            tmp1.nastaviDesni(tmp2.vrnilevi());
            tmp2.nastaviLevi(tmp1);
            tmp1.nastaviOce(tmp2);
            while(tmp1 != null){
                  // nastavimo faktor ravnotežja v vozlišču
                  tmp1.nastaviFaktor(ravnotezje(tmp1));
                  // pomaknemo se navzgor proti korenu
                  tmp1 = tmp1.vrnioce();
            }
            // kot rezultat vrnemo tmp2
            return tmp2;
      }

      // metoda, ki izračuna faktor ravnotežja
      public int ravnotezje(AVLVozel v){
        // l in d sta na začetku 0
        int l = 0;
        int d = 0;
        // če je levo vozlišče različno od nič, potem je l višina levega + 1
        if(v.vrnilevi() != null) l = v.vrnilevi().visina() + 1;
           // če je desno vozlišče različno od nič, potem je d višina desnega + 1
           if(v.vrnidesni() != null) d = v.vrnidesni().visina() + 1;
              // kot rezultat vrnemo njuno razliko
              return (l - d);
      }
```

VSTAVLJANJE:

Standardna metoda za vstavljanje v iskalno drevo ni več primerna, saj nam ne ohranja ravnotežnostnih faktorjev. Zato jo nadomestimo z novo metodo, ki bo v osnovi podobna prejšnji. Najprej enako kot prej določimo mesto vstavljanja. Ko vstavimo podatek, popravimo faktorje ravnotežja. Ob vrnitvi nazaj proti korenu drevesa na vsakem nivoju preverimo ravnotežnostni faktor in po potrebi izvedemo ustrezne rotacije. Prehod nazaj ni težaven, saj imamo referenco tudi na očeta.

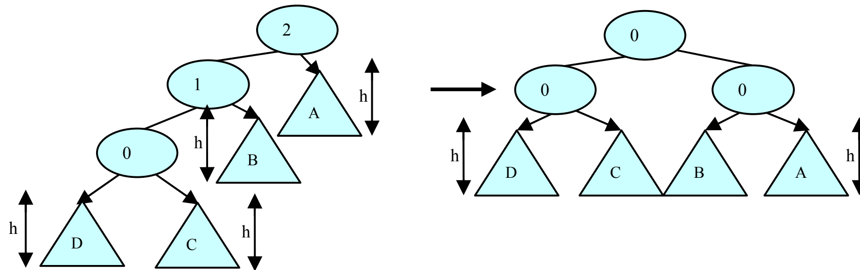
Vstavljanje v prazno drevo ali drevo z enim samim elementom je preprosto. V prvem primeru dobimo drevo, ki ima le koren drevesa (z ravnotežnostnim faktorjem 0), v drugem pa drevo s korenom in enim sinom ($fA(D) = 1$). V splošnem pa vstavljanje poteka takole:

1. Na običajni način (rekurzija) iščemo mesto za vstavljanje podatka. Ko pridemo do lista, vstavimo element. Ker je to list, ima faktor ravnotežja seveda 0. Ker smo s tem pot v globino rekurzije zaključili, se (avtomatično) vrnemo na predhodnji klic.

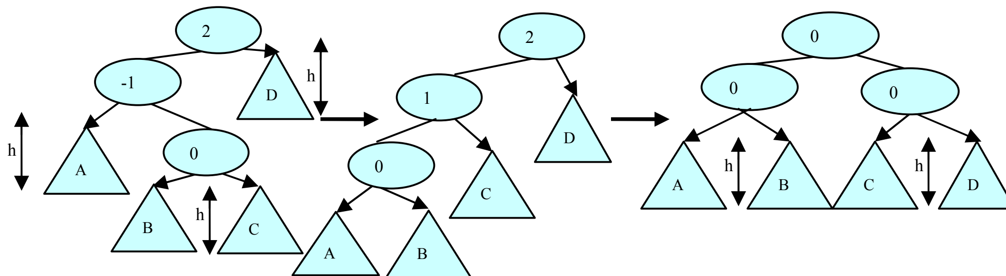
Diplomska naloga Fakulteta za matematiko in fiziko

2. Če z vstavljanjem v AVL drevo nismo porušili ravnotežja v vozlišču tega drevesa ($-1 < fA(D) < 1$), je postopek končan in spet nas rekurzija vrne nivo višje.
3. Če z vstavljanjem v AVL drevo v tem vozlišču porušimo ravnotežje, moramo izvesti ustrezne rotacije. Naj bo to vozlišče x. Imamo štiri možnosti:

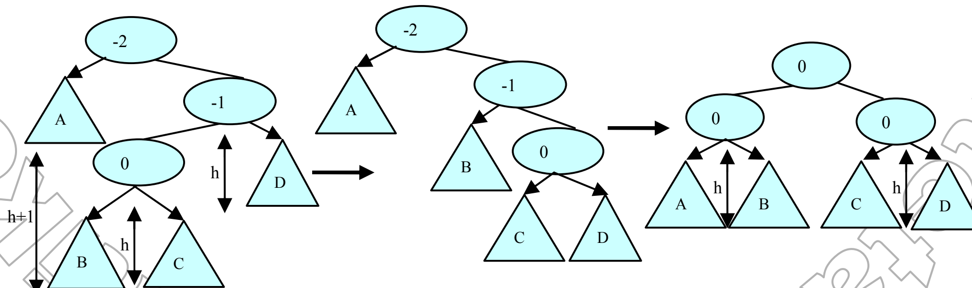
   a. od vozlišča s prevelikim faktorjem ravnotežja, vozlišča x, smo šli pri iskanju mesta za vstavljanje dvakrat v levo. Zato to vozlišče rotiramo v desno. Izvedemo desno rotacijo. Po rotaciji je postopek je končan, saj se je vzpostavilo ravnotežje. Na sliki smo v vozlišča napisali njihove ravnotežnostne faktorje.



   b. od vozlišča x smo pri vstavljanju šli najprej v levo nato v desno, zato mora sedaj vozlišče s prevelikim faktorjem ravnotežja v desno poddrevo. Kandidat, ki pride na njegovo mesto, je koren desnega poddrevesa njegovega levega poddrevesa. Njegovo levo poddrevo sodi levo od korena in desno od korena desnega poddrevesa. Njegovo desno poddrevo, če obstaja, pa mora ostati desno. Gre za primer L – D rotacije.

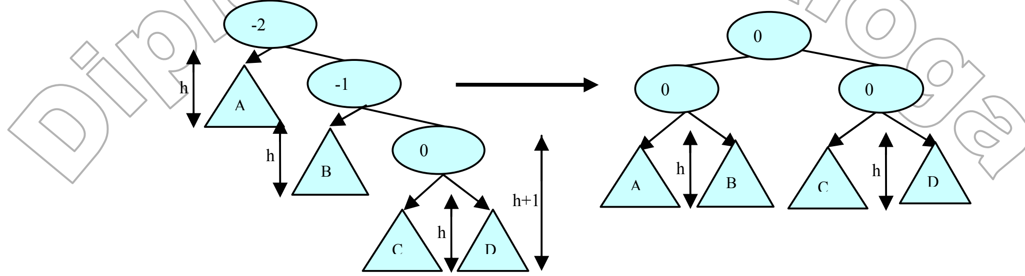


   c. Od vozlišča x smo šli pri vstavljanju desno in nato v levo. Položaj je zrcalna slika točke 2.b. Izvedemo rotacijo D – L.



Diplomska naloga
Fakulteta za matematiko in fiziko

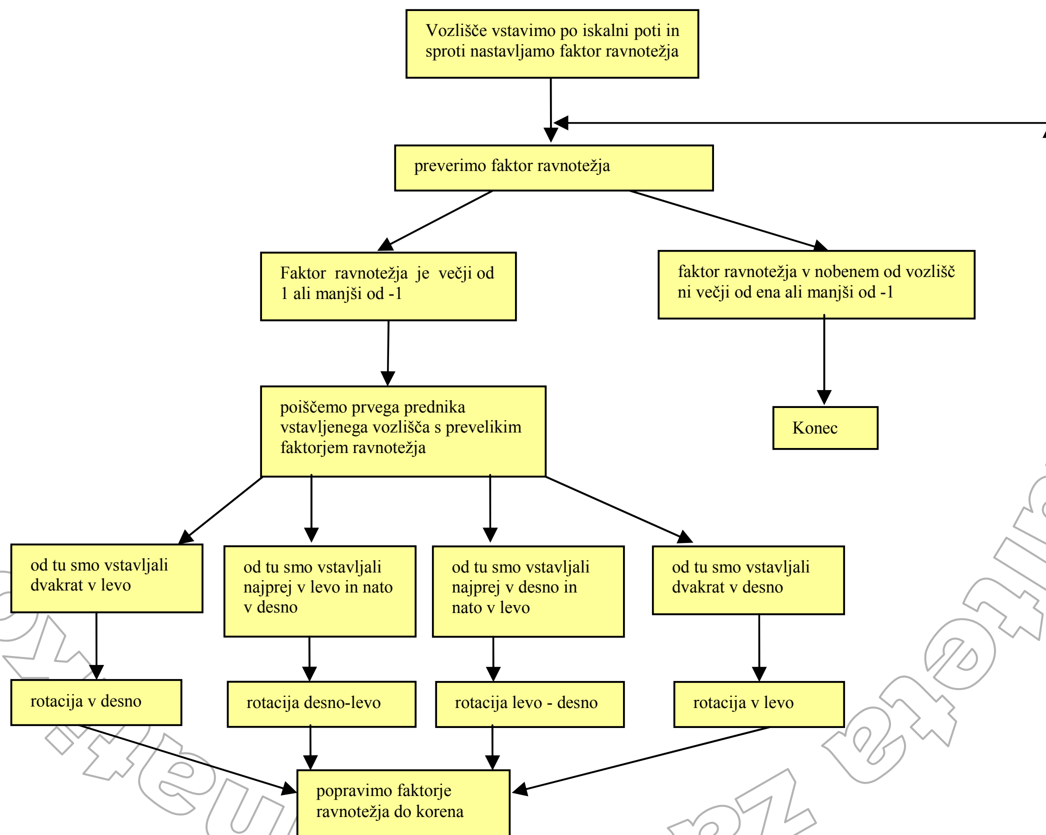
d. od najbližjega vozlišča s prevelikim faktorjem ravnotežja smo vstavljali dvakrat v desno, zato to vozlišče rotiramo v levo, torej primer leve rotacije. Po rotaciji je ravnotežje vzpostavljeno in postopek je končan.



Po izvedeni rotaciji popravimo ravnotežnostne faktorje na celotni poti do korena drevesa.

Ko vstavljamo v AVL drevo, sledimo iskalni poti, dokler ne pridemo do lista. Vstavimo novo vozlišče in postavimo ravnotežnostni faktor novo vstavljenega vozlišča na nič. Nato preverimo, če so se spremenili ravnotežnostni faktorji na poti proti korenu. V vozlišču, v katerem se nahajamo, preverimo razliko v višini levega in desnega poddrevesa, nato se pomaknemo nivo višje.

SHEMA VSTAVLJANJA ELEMENTA V AVL DREVO:

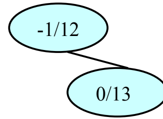


Diplomska naloga
Fakulteta za matematiko in fiziko

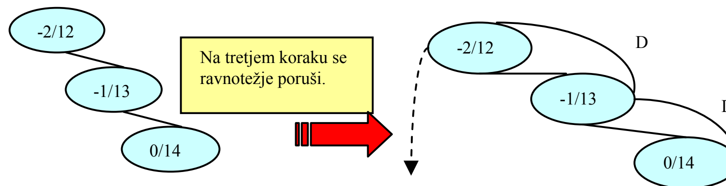
Diplomska naloga

Poglejmo si konkretni primer sestavljanja AVL drevesa. V na začetku prazno drevo zaporedoma vstavljamo naslednje podatke: 12, 13, 14, 5, 5, 3, 11, 6, 10 in 8.

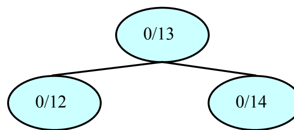
Prva dva koraka nam ne delata težav. Dobimo drevo



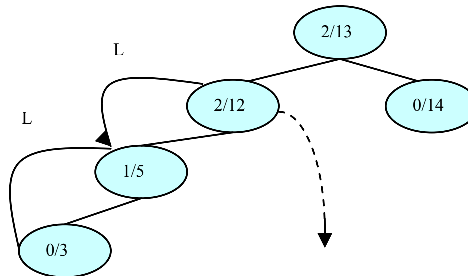
Prvo vozlišče, ki poruši ravnotežje, je vozlišče 14.



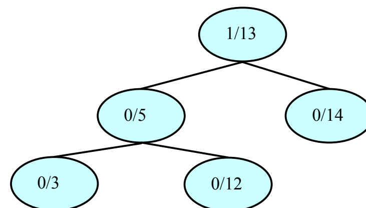
Njegovo najbližje vozlišče s prevelikim faktorjem je koren 12. Pri iskanju mesta za novo vozlišče smo šli dvakrat v desno. Neuravnoteženost popravimo tako, da del drevesa zavrtimo v nasprotni smeri vstavljanja, torej izvedemo levo rotacijo. Novo drevo je uravnoteženo.



Vstavljamo naprej. Vstavljanje podatka 5 ne pokvari ravnotežja, podatek 3 pa ga.

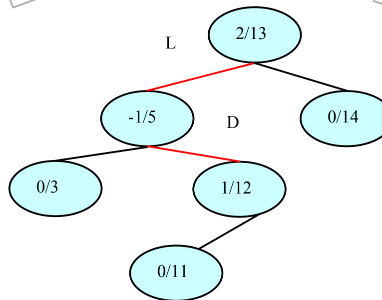


Poiščemo prvo vozlišče z napačnim ravnotežnostnim faktorjem. To je vozlišče s podatkom 12. Ravnotežje smo porušili z vstavljanjem dvakrat v levo poddrevo. Pri vozlišču 12 uporabimo desno rotacijo in dobljeno drevo je:

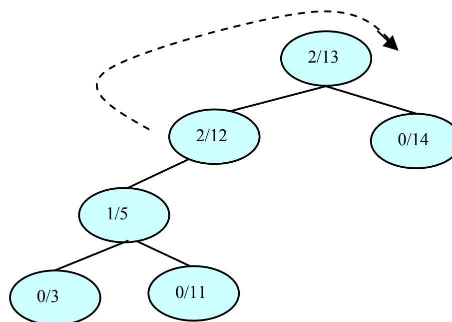


Fakulteta za matematiko in fiziko

Vstavimo še podatek 11 in dobimo:

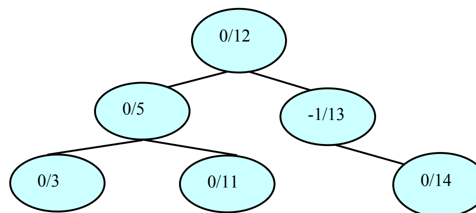


Z vstavljanjem vozlišča 11 smo zopet podrli ravnotežje, zato ga bomo popravili z rotacijo. Poiščemo prvega prednika vstavljenega vozlišča, ki ima prevelik absolutni faktor ravnotežja. V našem primeru je to koren drevesa s podatkom 13. Rotacija, ki jo izvedemo tu, je nekoliko bolj zapletena. V tem primeru gre za dvojno rotacijo levo – desno. Glede na predhodni opis te rotacije že vemo, da gre vozlišče 13 s položaja korena v desno poddrevo. Na njegovo mesto pride koren desnega poddrevesa njegovega levega poddrevesa (najbolj desno vozlišče njegovega levega poddrevesa), torej vozlišče 12.
Kot smo pri opisu rotacije (L – D) že povedali, najprej izpeljemo levo rotacijo. V našem primeru rotiramo okoli vozlišča 5.



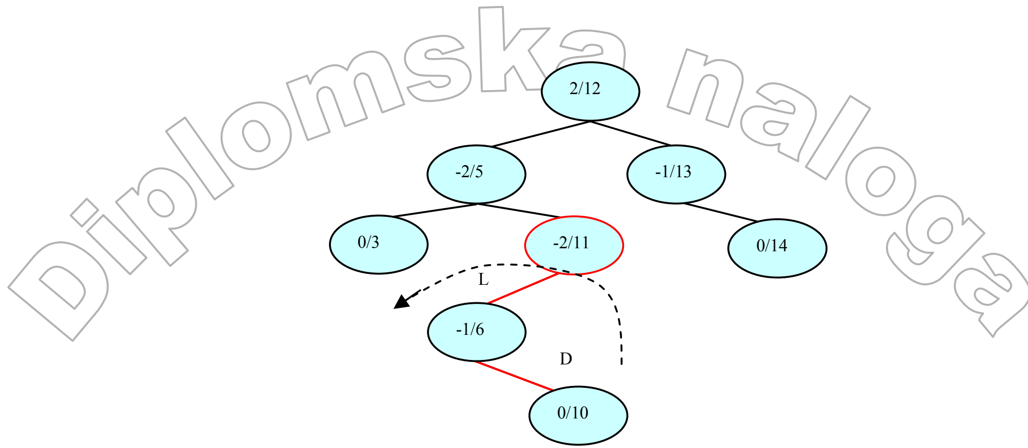
Nato pa izvedemo še desno rotacijo okoli vozlišča 13.

AVL drevo po šestem koraku:

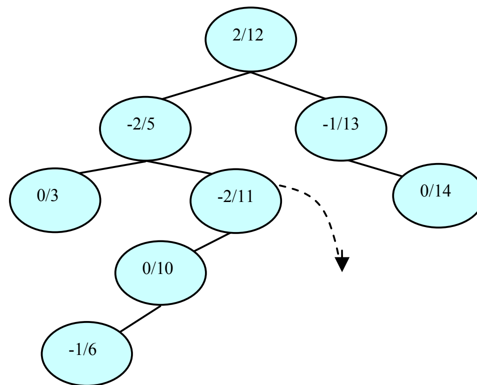


Sledi vstavljanje 6 in nato 10:

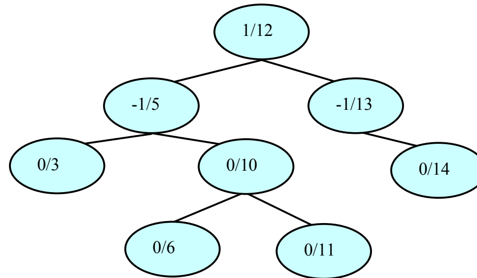
Diplomska naloga
Fakulteta za matematiko in fiziko



Zopet popravimo drevo. Poiščemo najbližjega prednika vstavljenega vozlišča 10 s prevelikim faktorjem ravnotežja vozlišča. To je vozlišče 11. Od vozlišča 11 pridemo do vozlišča 10 po poti najprej v levo nato v desno. Gre za dvojno rotacijo D – L. Najprej izvedemo rotacijo v desno okoli vozlišča 6.

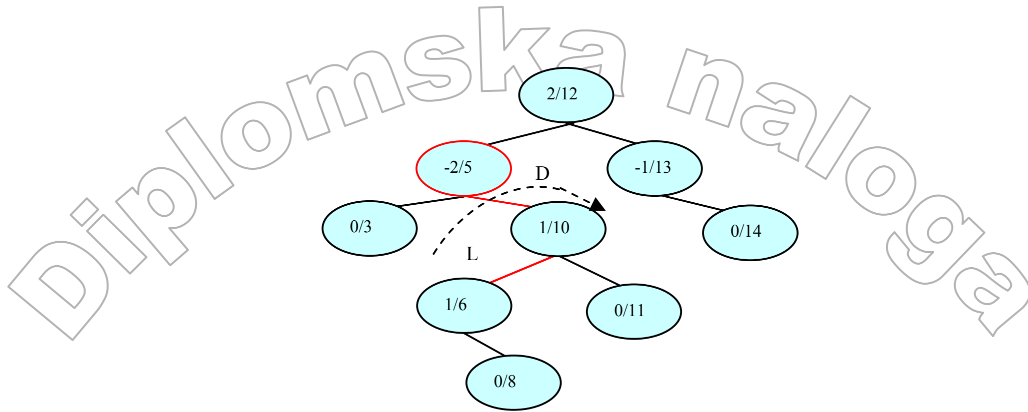


nato pa še levo rotacijo okoli vozlišča 11:

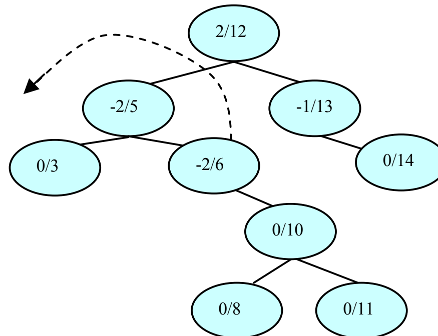


Vstavimo še 8 in dobimo:

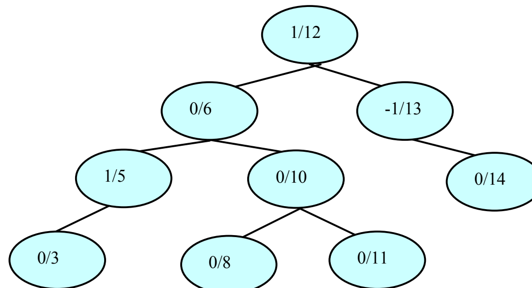
Diplomska naloga
Fakulteta za matematiko in fiziko



Ukrepamo podobno kot prej. Najbližji prednik vstavljenega vozlišča 8, ki prevelik faktor ravnotežja, je vozlišče 5. Do vozlišča 8 pridemo, če gremo od vozlišča 5 najprej v desno in nato v levo. Ponovno imamo primer dvojne rotacije, primer D – L. Najprej izvedemo rotacijo v desno okrog vozlišča 10.



Potem izvedemo še rotacijo v levo okoli vozlišča 5. Naše AVL drevo je na koncu torej tako.

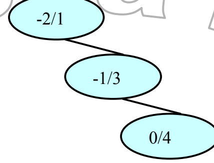


Tako AVL drevo torej dobimo, če v prazno drevo zaporedoma vstavimo podatke 12, 13, 14, 5, 3, 11, 6, 10 in 8.

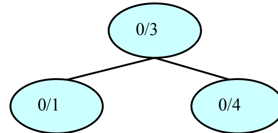
Poglejmo si še primer, ko vstavljamo urejene podatke. Zaporedoma vstavimo 1, 3, 4, 6, 8, 9, 10, 11 in 13.

Vstavimo 1, 3 in 4.

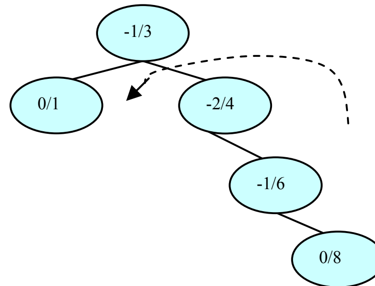
Diplomska naloga
Fakulteta za matematiko in fiziko



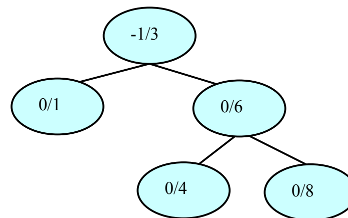
Pri vstavljanju vozlišča 4 se ravnotežje podre in potrebna je rotacija. Ker smo od vozlišča s pokvarjenim faktorjem ravnotežja šli pri vstavljanju dvakrat v desno, izvedemo rotacijo v levo.



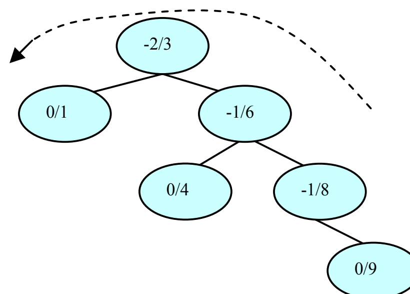
Vstavimo še 6 in 8.



Zopet smo pri vstavljanju podatka 8 od »neuravnoteženega« vozlišča 4 šli dvakrat v desno, zato okoli tega vozlišča izvedemo rotacijo v levo.



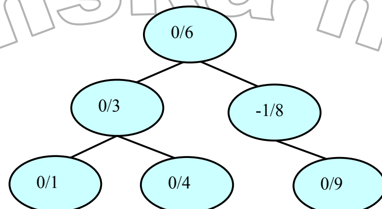
Nadaljujemo z vstavljanjem števila 9:



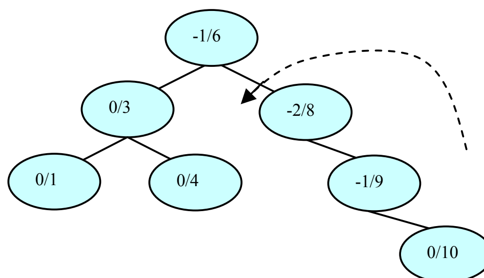
Kot prej, imamo tudi tu primer, ko gremo na poti k vstavljenemu vozlišču od najbližjega »pokvarjenega vozlišča« dvakrat desno in je potrebna rotacija v levo.

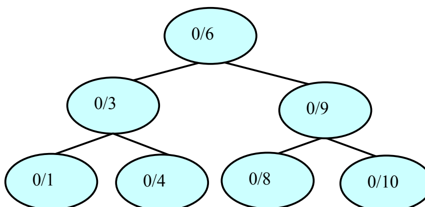
Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

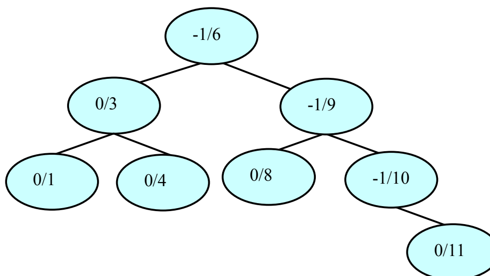


Sledi vstavljanje 10 z novo rotacijo v levo

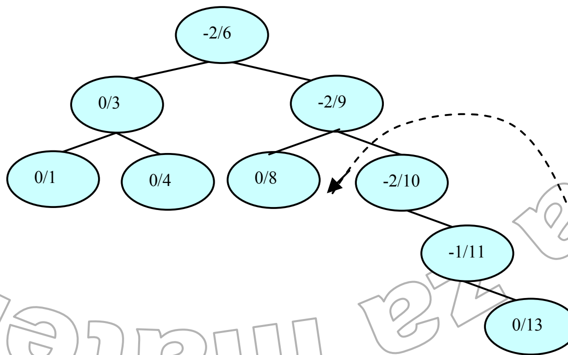




in vstavljanje 11, ki ne zahteva uravnoteževanja:

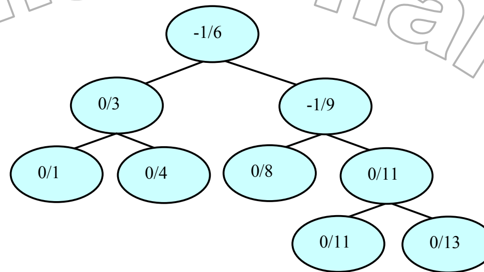


Na koncu vstavimo še vozlišče 13, ki pri vozlišču 10 povzroči rotacijo v levo:



Fakulteta za matematiko in fiziko

Na koncu je naše drevo videti takole:



Ugotovili smo, da se pri vstavljanju že urejenih podatkov v iskalno dvojiško drevo srečamo samo z enim tipom rotacije in sicer v našem primeru z levo rotacijo, saj smo vstavljali naraščajoče urejene podatke, torej smo vstavljali samo v desno. V nasprotnem primeru, če bi vstavljali padajoče zaporedje, bi uporabili pa samo desno rotacijo. Seveda pa bi, če bi vnaprej vedeli, da so podatki urejeni, drevo veliko lažje sestavili kar neposredno, tako da ne bi bila potrebna nobena rotacija.

Opisane postopke združimo v ustrezno metodo. Metodo *vstavi* sem gradila postopoma z začetkom pri razredu `Drevo`, zato ne deluje v optimalnem času. Metoda *vstavi* temelji na rekurzivnem postopku in zato faktor ravnotežja nastavljamo sproti na celotnem drevesu, potrebno pa bi ga bilo spreminjati le v vozliščih, kjer se dejansko spremeni višina poddreves. Vseeno sem se zaradi preglednosti nad celotnim načinom zgradbe programov odločila, da metodo vključim v diplomsko delo v taki obliki, saj metoda deluje pravilno.

Metoda VSTAVI v AVL drevo:

```
//metoda vstavi
public void vstavi(int x){
   koren = vstavi(koren, x);
}
public AVLVozel vstavi(ALVozel koren, int nPodatek){
   if(koren == null){
      koren = new AVLVozel(nPodatek);
   }
   else{
      if(nPodatek < koren.vrniPodatek()){
         // nov vozel v levo poddrevo
         Vozel tmp = vstavi(koren.vrniLevi(), nPodatek);
         //prestavimo kazalce
         koren.nastaviLevi(tmp);
         tmp.nastaviOce(koren);
      }
      else{
         // nov vozel v desno poddrevo
         AVLVozel tmp = vstavi(koren.vrnidesni(),nPodatek);
         //prestavimo kazalce
         koren.nastaviDesni(tmp);
         tmp.nastaviOce(koren);
      }
   }
   //nastavimo faktor ravnotežja
   koren.nastaviFaktor(ravnotezje(koren));


   //preverimo faktorje ravnotežja
   if(koren.vrniFaktor() < -1){
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
    //če je podatek večji od podatka v desnem sinu
        if(nPodatek > koren.vrnidesni().vrniPodatek()){
      // enojna rotacija v levo
            return levaRotacija(koren);
        }
        // dvojna rotacija
        else{
            //če je podatek manjši od podatka v desnem sinu
            if(nPodatek < koren.vrnidesni().vrniPodatek()){
                //rotacija v desno
                desnaRotacija(koren.vrnidesni());
            }
            //rotacija v levo
            return levaRotacija(koren);
        }
    }
    // če je faktor ravnotežja večji od 1
    else if(koren.vrniFaktor() > 1){
        //če je podatek manjši od podatka v levem sinu
        if(nPodatek < koren.vrnilevi().vrniPodatek()){
            // enojna roracija v desno
            return desnaRotacija (koren);
        }
        // dvojna rotacija
        else{
            //če je podatek večji od podatka v levem sinu
            if(nPodatek > koren.vrnilevi().vrniPodatek()){
                //rotacija v levo
                levaRotacija(koren.vrnilevi());
            }
            //rotacija v desno
            return desnaRotacija(koren);
        }
    }
    // na koncu vrnemo koren
    return koren;
}
```

BRISANJE:

Elemente brišemo tako, da brisano vozlišče nadomestimo z najbolj levim vozliščem v desnem poddrevesu, če ta obstaja, sicer pa z najbolj desnim vozliščem v levem poddrevesu. Če je element list (nima sinov), ga enostavno zbrišemo. Nato popravimo ravotežnostne faktorje ter po potrebi izvedemo ustrezne rotacije.
Podobno kot pri vstavljanju najprej rekurzivno poiščemo element, ki ga želimo odstraniti, nato pa ga nadomestimo. Faktorje ravnotežja popravimo takoj po brisanju v vsakem vozlišču na poti nazaj do korena. V vozlišču v katerem se nahajamo, preverimo razliko v višini levega in desnega poddrevesa, izračunamo nov faktor ravnotežja, nato se pomaknemo nivo višje.

Brisanje elementa v AVL drevesu v splošnem poteka takole:

1. Naj bo odstranjeno vozlišče list. Takrat popravimo faktor ravnotežja v vsakem vozlišču v tistem poddrevesu, v katerem se je nahajalo brisano vozlišče in preverimo faktor ravnotežja. Če je $-1 < fA(D) < 1$, je postopek končan, in spet nas rekurzija vrne nivo višje. V nasprotnem primeru gremo naprej na točko 2.a., 2.b. ali 2.c.

Diplomska naloga Fakulteta za matematiko in fiziko

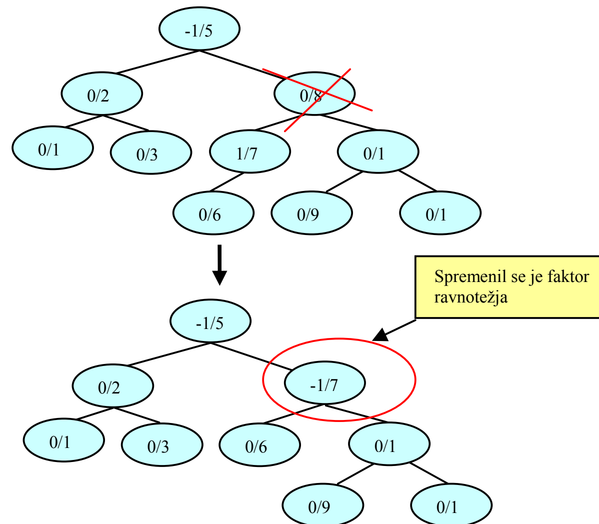
2. Odstranjeno vozlišče je notranje, zato ga nadomestimo z najbolj levim vozliščem v desnem poddrevesu. Če tega slučajno ni, ga nadomestimo z najbolj desnim vozliščem v levem poddrevesu. Popravimo faktorje ravnotežja na poti nazaj proti korenu.

    a. Faktor ravnotežja v nobenem vozlišču ni večji kot ena. Drevo je torej še vedno AVL drevo, zato je postopek končan.

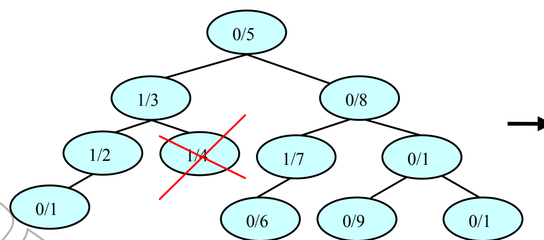
    b. Vozlišče s prevelikim faktorjem ravnotežja ima negativni predznak, kar pomeni, da je desno poddrevo višje od levega, zato izvedemo rotacijo v levo.

    c. Vozlišče s prevelikim faktorjem ravnotežja ima pozitivni predznak, kar pomeni da je levo poddrevo višje od desnega, zato izvedemo rotacijo v desno.

Če je bilo poddrevo pred odstranjevanjem popolnoma uravnoteženo ($f(D)=0$), je po odstranjevanju vozlišča ostalo uravnoteženo. Njegova višina se ni spremenila. Zato se pri vseh prednikih na poti proti korenu zagotovo ne spremeni faktor ravnotežja in s preverjanjem lahko končamo.

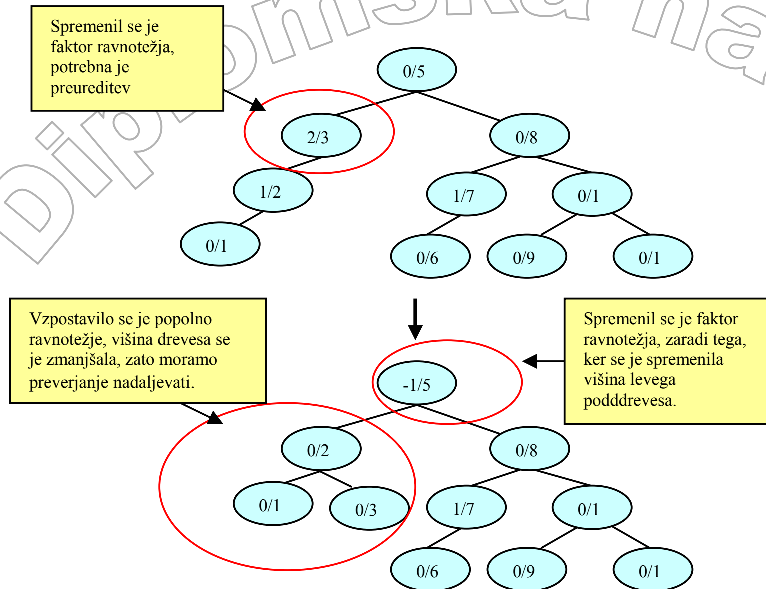


Če se je zaradi odstranjevanja vzpostavilo popolno ravnotežje, se je višina poddrevesa zmanjšala. Preverjanje nadaljujemo. Ko se ravnotežje podre, je potrebno uravnoteževanje.
Vsaka rotacija zmanjša višino drevesa. To pomeni, da po rotaciji ne moremo končati s preverjanjem ravnotežja. Ravnovesje se je lahko podrlo na višjem nivoju. To je bistvena razlika od uravnoteževanja pri vstavljanju, kjer po rotaciji lahko zaključimo postopek.



Diplomska naloga
Fakulteta za matematiko in fiziko

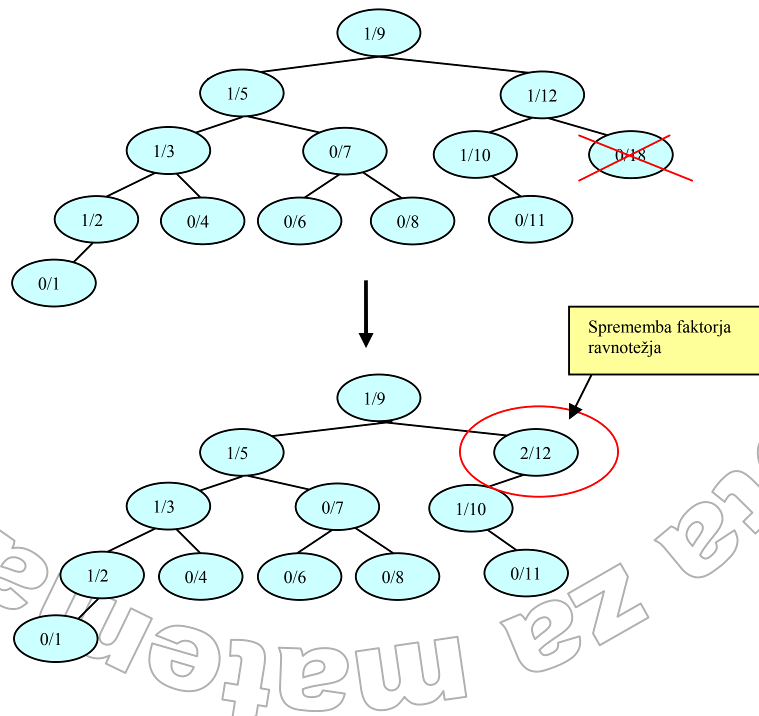
Diplomska naloga



Za razliko od vstavljanja tu po eni rotaciji drevo še ni uravnoteženo in ga je treba uravnoteževati dokler ne pridemo do korena, ali dokler ne naletimo na poddrevo, katerega višina se ni spremenila (faktor ravnotežja pred odstranjevanjem je bil $f(D) = 0$). V najslabšem primeru bo treba izvajati rotacije v vsakem vozlišču po iskalni poti.

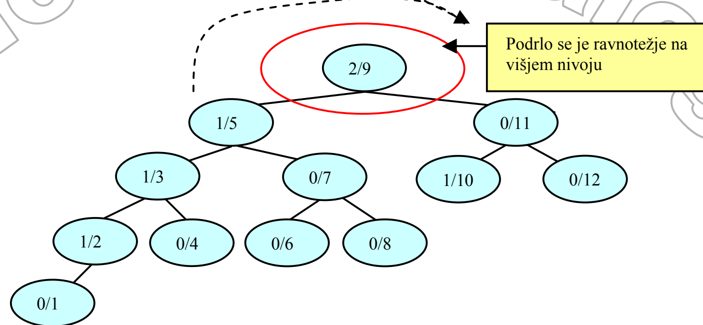
Oglejmo si nekaj primerov, kjer bomo opazovali, kako se spreminja faktor ravnotežja:

1. V AVL drevesu odstranimo vozlišče 18. S tem povzročimo spremembo ravnotežnostnega faktorja samo v vozlišču 12. Faktor ravnotežja je sedaj 2, kar pomeni, da je potrebna preureditev.

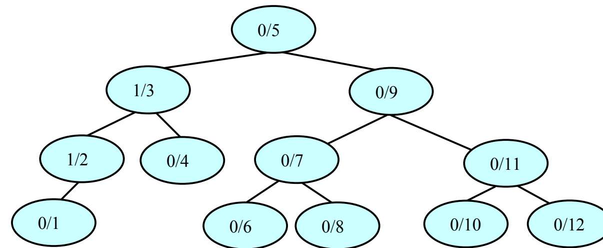


Fakulteta za matematiko in fiziko

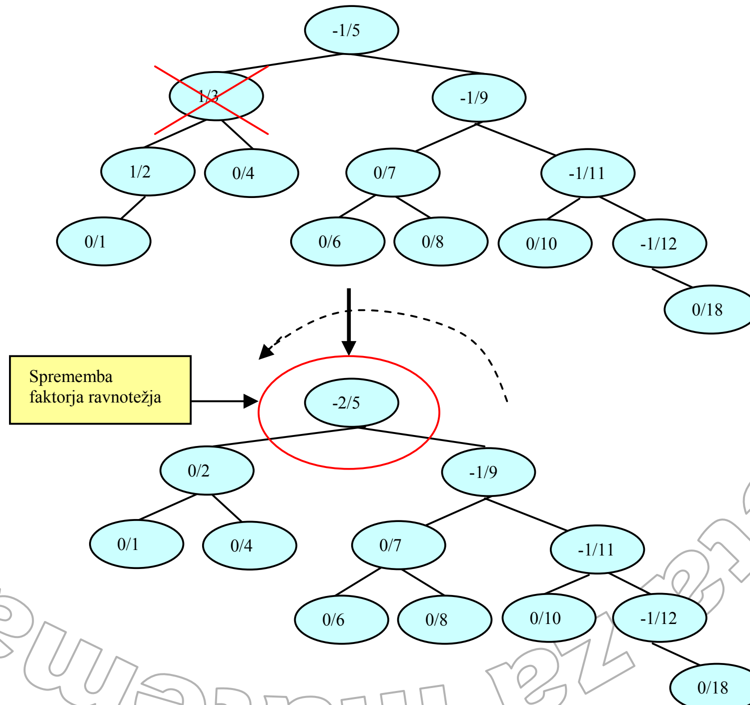
Vsaka rotacija zmanjša višino drevesa, kar pomeni, da po rotaciji ne moremo končati preverjanja ravnotežja, ker se je lahko podrlo na višjem nivoju.



Ravnotežje se je podrlo pri korenu drevesa in potrebno je uravnoteževanje. Faktor ravnotežja ima pozitivni predznak (levo poddrevo je višje od desnega) zato je potrebna rotacija v desno. Na mesto korena drevesa pride koren njegovega levega poddrevesa, vozlišče 5. Njegovo levo poddrevo ostane na njegovi levi in njegovo desno poddrevo bo sedaj na njegovi desni, vendar levo od vozlišča 9, saj je bilo levo že pred rotacijo.

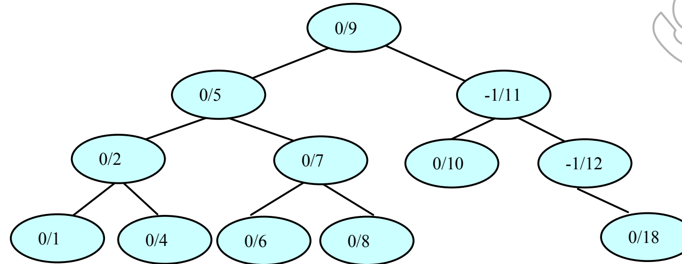


2. V drevesu odstranimo vozlišče 3. S tem povzročimo spremembo faktorja ravnotežja v korenu drevesa, v -2. Drevo je treba preurediti.

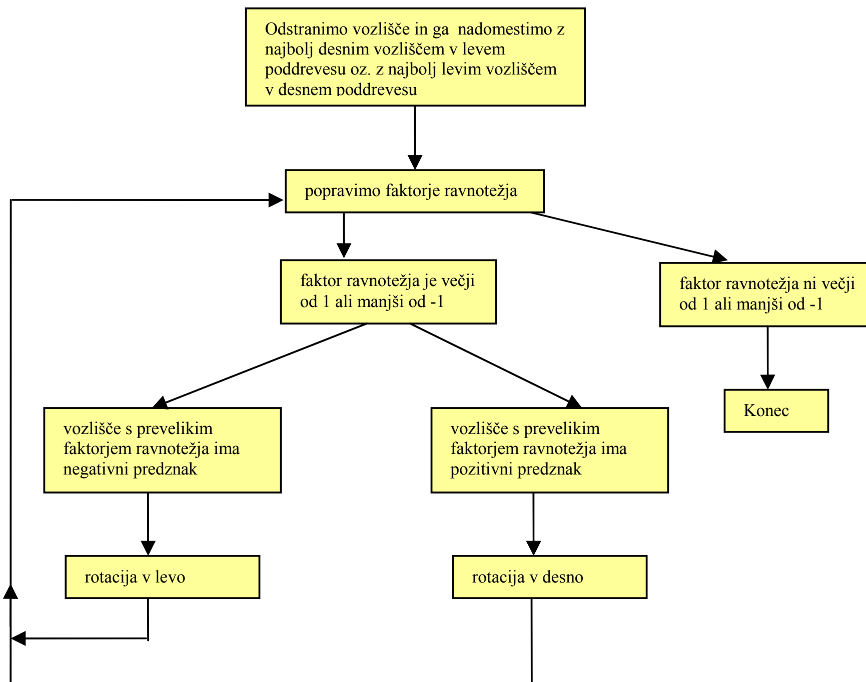


Diplomska naloga
Fakulteta za matematiko in fiziko

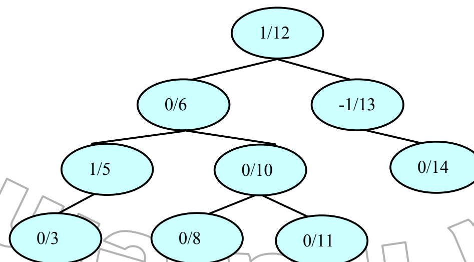
Ker je faktor ravnotežja negativnega predznaka (desno poddrevo je višje od levega), je potrebna rotacija v levo. Na mesto korena drevesa pride koren njegovega desnega poddrevesa, vozlišče 9. Njegovo desno poddrevo ostane na njegovi desni. Novo levo poddrevo bo imelo za koren stari koren (5). Levo poddrevo vozlišča 9 bo sedaj desno poddrevo vozlišča 5.



SHEMA ZA BRISANJE ELEMENTA V AVL DREVESU:

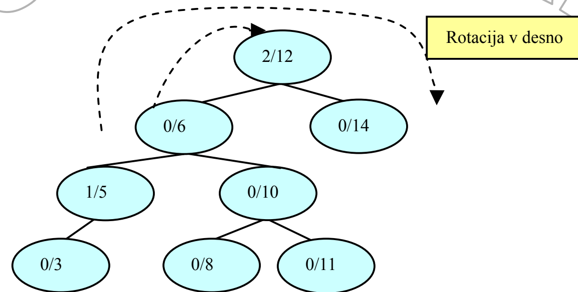


Poglejmo si na praktičnem primeru, kako brisanje poteka. Vzemimo kar AVL drevo, ki smo ga zgradili z vstavljanjem:

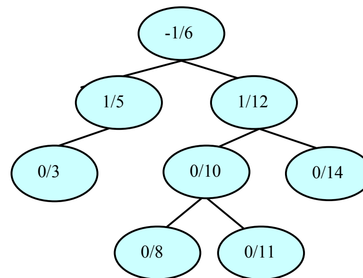


Diplomska naloga
Fakulteta za matematiko in fiziko

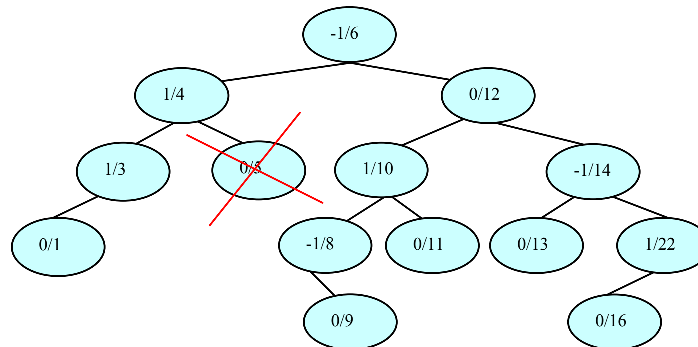
Izbrisali bomo vozlišče s podatkom 13. Vozlišče nadomestimo z najbolj levim vozliščem v desnem poddrevesu, torej z vozliščem 14. V našem primeru je to tudi edino vozlišče v njegovem poddrevesu in edini naslednik.



Ravnotežnostni faktor vozlišča 12 se je spremenil v 2. Ker je faktor pozitiven, uporabimo desno rotacijo. Levi sin vozlišča 12 postane novi koren, ki ohrani svoje levo poddrevo, koren desnega poddrevesa pa je vozlišče 12. Desno poddrevo vozlišča 6 postane levo poddrevo vozlišča 12 in dobimo

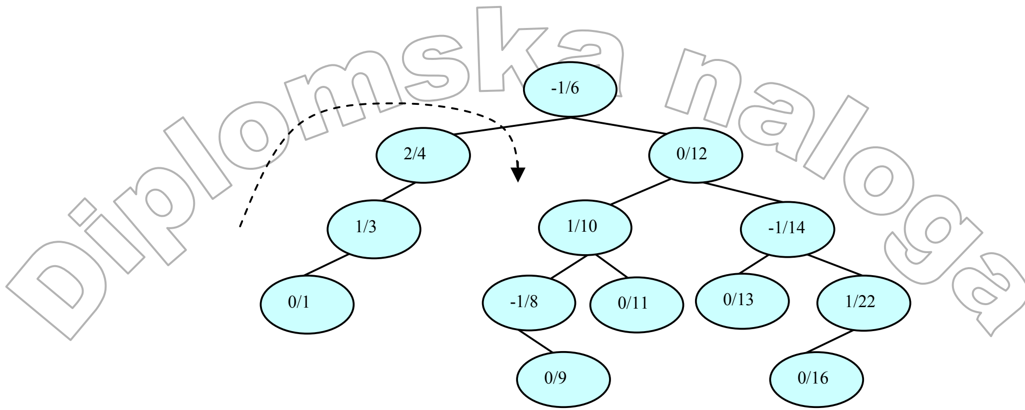


Oglejmo si še zapletenejši primer. Brišemo vozlišče 5.

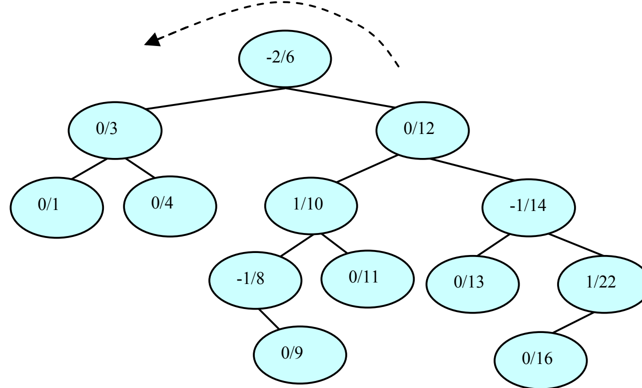


Po brisanju tega vozlišča in pred preureditvijo je položaj takle:

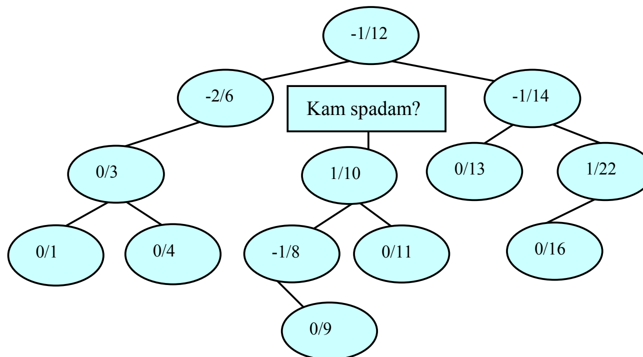
Diplomska naloga
Fakulteta za matematiko in fiziko



Faktor ravnotežja se je spremenil samo v očetu brisanega vozlišča. Faktor ravnotežja v vozlišču 4 je sedaj 2. Ker je faktor pozitiven, uporabimo desno rotacijo.

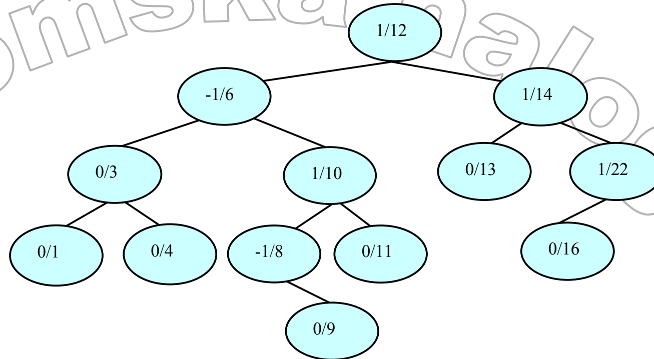


Z rotacijo povzročimo spremembo ravnotežnostnega faktorja v samem korenu drevesa. Spremenjeni faktor odstopa od meja za AVL drevo, zato je potrebna preureditev. Ker je novi faktor ravnotežja negativen, uporabimo levo rotacijo. Desni sin vozlišča 6, vozlišče 12, postane novi koren drevesa, ki ohrani svoje desno poddrevo. Kam pa bi dali njegovo levo poddrevo (s korenom 10)?



To poddrevo je bilo prej desno od korena 6, torej mora biti tudi zdaj desno od vozlišča 6 in levo od korena novega korena 12.

Diplomska naloga
Fakulteta za matematiko in fiziko



Metoda BRIŠI v AVL drevesu:

```
// metoda brisi, ki iz iskalnega dvojiškega drevesa odstrani  element
public void brisi(int nPodatek) throws Exception{
   koren = brisi(koren, nPodatek);
}

public AVLVozel brisi(AVLVozel koren, int nPodatek) throws Exception{
   if (koren == null){
      throw new Exception(" Drevo je prazno"); // če je drevo prazno ne naredimo nič
   }
   else{
      if(nPodatek == koren.vrniPodatek()){ // našli smo podatek
         koren = izbrisiVozel(koren);    // in zbrišemo ustrezni vozel
      }
      else if(nPodatek < koren.vrniPodatek()){ //podatek manjši od korena
         // gremo rekurzivno v levo in primerjamo podatek
         koren.nastaviLevi(brisi(koren.vrnilevi(), nPodatek));
      }
      else{
         // drugače gremo rekurzivno  v desno primerjamo podatek
         koren.nastaviDesni(brisi(koren.vrnidesni(), nPodatek));
      }
   }
   // nastavimo faktor ravnotežja
   koren.nastaviFaktor(ravnotezje(koren));
   if(koren.vrniFaktor() < -1){ // desno  je prevelika višina
      if(nPodatek < koren.vrnidesni().vrniPodatek()){   // enojna rotacija v levo
         return levaRotacija(koren);
      }
      else{   // dvojna rotacija
         if(nPodatek < koren.vrnidesni().vrniPodatek()){
            desnaRotacija(koren.vrnidesni());
         }
         return levaRotacija(koren);
      }
   }
   else if(koren.vrniFaktor() > 1){
      if(nPodatek > koren.vrniLevi().vrniPodatek()){   // enojna rotacija v desno
         return desnaRotacija  (koren);
      }
      else{   // dvojna rotacija
         if(nPodatek > koren.vrnilevi().vrniPodatek()){
            levaRotacija(koren.vrnilevi());
         }
      return desnaRotacija(koren);
      }
```


Diplomska naloga
Fakulteta za matematiko in fiziko

```
    }
    return koren;
}
// metoda za brisanje vozla  iz drevesa
private static AVLVozel izbrisiVozel(AVLVozel koren) {
    if(koren.vrnilevi() == null){   // levega poddrevesa ni
        koren = koren.vrnidesni();   // nastavimo koren desnega poddrevesa
    }
    else if(koren.vrnidesni() == null) { //desnega poddrevesa ni
        koren = koren.vrnilevi();   // nastavimo koren levega poddrevesa
    }
    else{ //imamo desno in levo poddrevo
        // poiščemo skrajno desni vozel v levem poddrevesu,
        AVLVozel zadnji = poisciSkrajniDesni(koren.vrnilevi());
        //zamenjamo podatke
        koren.nastaviPodatek(zadnji.vrniPodatek());
        //zbrišemo skrajno levega
        koren.nastaviLevi(zbrisiSkrajniDesni(koren.vrnilevi()));
    }
    return koren;
}

//metoda, ki vrne skrajno levi vozel
private static AVLVozel poisciSkrajniDesni(AVLVozel koren) {
    if(koren.vrnidesni() != null) { //levo poddrevo ni prazno
        //vrne skrajno levega
        return poisciSkrajniDesni(koren.vrnidesni());
    }
    else {
        //najmanjsi je kar koren
        return koren;
    }
}

// izbrišemo skrajni levi vozel
private static AVLVozel zbrisiSkrajniDesni(AVLVozel koren) {
    if(koren.vrnidesni() != null) { //levo poddrevo ni prazno
        //gremo v levo in ga zbrišemo
        koren.nastaviDesni(zbrisiSkrajniDesni(koren.vrnidesni()));
        return koren;
    }
    else{
        //vrne koren levega poddrevesa
        return koren.vrnilevi();
    }
}
```

Zahtevnost operacij nad AVL drevesom:

- **Iskanje podatkov:**
  Ker je AVL drevo uravnoteženo, je njegova višina enaka $\log_2(n+1)$. Zato iskanje v AVL drevesu izvedemo v najslabšem primeru v času $O(\log(n))$.

- **Vstavljanje podatkov:**
  Ko vstavljamo podatek v AVL drevo, moramo najprej zanj poiskati pravo mesto za vstavitev podatka. Da najdemo pravo mesto, moramo obiskati največ $\log_2(n+1)$ vozlišč. Ob vračanju obiščemo $\log_2(n+1)$ vozlišč in v nekaterih moramo izvesti

Diplomska naloga Fakulteta za matematiko in fiziko

rotacije. Časovna zahtevnost ene rotacije je $O(1)$. Če vse skupaj seštejemo in zanemarimo konstantne člene, je časovna zahtevnost postopka za vstavljanje v AVL drevo reda:

$$O(\log(n)).$$

- **Brisanje podatkov:**
  Podobno kot pri vstavljanju moramo tudi pri brisanju najprej poiskati podatek, ki ga bomo zbrisali. Pri tem moramo obiskati največ $\log_2(n+1)$ vozlišč. Samo brisanje in popravljanje zahtevata $(2\log_2(n+1))$ operacij, časovna zahtevnost je torej reda:

$$O(\log(n)).$$

AVL drevo je bilo prvo med uravnoteženimi drevesi. Vendar ga je v praksi kmalu izpodrinilo rdeče – črno drevo, ker je slednje bolj učinkovito.

Diplomska naloga
Fakulteta za matematiko in fiziko

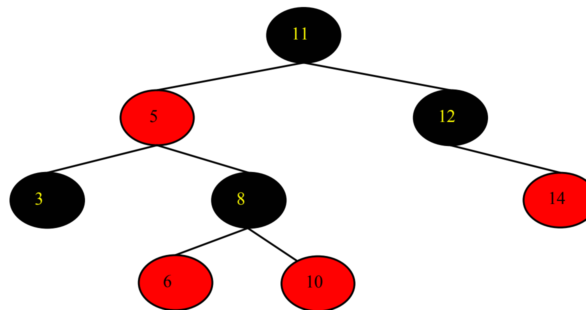
5.3. RDEČE-ČRNO DREVO

Rdeče-črno drevo je delno poravnano iskalno dvojiško drevo. O delni poravnanosti govorimo, ker ne dosežemo take poravnanosti kot pri AVL drevesu. Delno poravnanost dosežemo z dodatno informacijo v vozliščih: barvo. Vozlišča poleg kazalcev na očeta in sinova vsebujejo še informacijo o barvi.

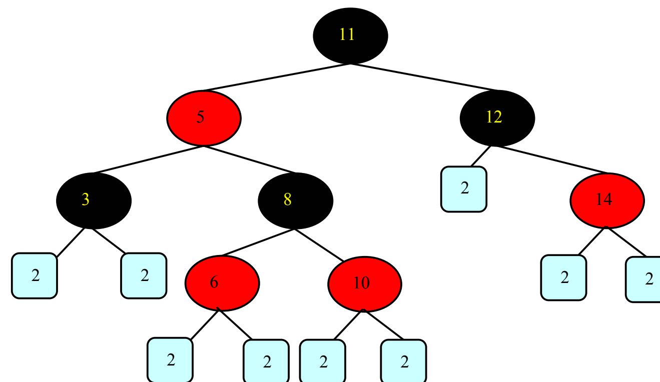
Da je neko drevo rdeče-črno drevo, mora ustrezati naslednjim pogojem:

1. vsako vozlišče je bodisi rdeče bodisi črno
2. rdeče vozlišče ima le črne sinove
3. koren je črn
4. vsaka pot od korena do praznega poddrevesa ima enako število črnih vozlišč.

Primer rdeče-črnega drevesa:



Vozlišča so črna ali rdeča, vsako rdeče vozlišče ima res le črne sinove. Tudi koren je črn. Poglejmo si veljavnost lastnosti 4. V svetlomodrih kvadratkih navedimo število črnih vozlišč na poti od korena do praznega poddrevesa.



Vidimo, da vsaka pot od korena do praznega poddrevesa res vsebuje enako število črnih vozlišč.

Če pri vstavljanju ali brisanju pride do kršitve teh lastnosti, vozlišča po potrebi prebarvamo in preuredimo, spet s pomočjo rotacij. V nasprotju z AVL drevesi, kjer je imel bistveni pomen faktor ravnotežja, rotacije temeljijo na barvi vozlišč.

Diplomska naloga
Fakulteta za matematiko in fiziko

V ta namen pripravimo razred `RCDrevo`, še prej pa definiramo razred `RCDVozel`. Tudi tu veljajo enake pripombe, kot pri AVL drevesu. Razred `RCDVozel` torej ni izpeljan iz razreda `Vozel`, razred `RCDrevo` pa ne iz razreda `Drevo`. Prav tako programi niso optimirani glede časovne zahtevnosti, ampak nam gre le za neposredni zapis ideje. Koda je navedena le za ključne metode. Za ostale metode so navedena le njihova imena (kot da bi definirali abstraktni razred). Njihovo polno kodo si lahko ogledate v prilogi diplomskega dela.

Razred RCDrevo:

```
public class RCDrevo{

  private RCDVozel koren;

   public RCDrevo() // konstruktor, ki ustvari prazno RCD drevo

   public RCDrevo(RCDVozel v)  // konstruktor, ki ustavri drevo z vozlom v

   public boolean prazno() // metoda, ki ugotovi ali je drevo prazno

   public RCDVozel koren() // metoda, ki vrne koren drevesa

   public int vrni()throws Exception // metoda vrne drevo

   public RCDrevo leviSin() //metoda, ki vrne levega sina (levo poddrevo)

   public RCDrevo desniSin() //  metoda, ki vrne desnega sina (desno poddrevo)

   public RCDrevo oce() //metoda, ki vrne očeta

   public static Vozel najmanjsi(Vozel x) // metoda, ki v iskalnem dvojiškem drevesu
   poišče najmanjši element

   // leva rotacija
   public RCDVozel levaRotacija(RCDVozel x)
      // y je x-ov desni sin
      RCDVozel y = x.vrnidesni();
      // x-u nastavimo desnega sina
      x.nastaviDesni(y.vrnilevi());
      // če y ima levega sina
      if(y.vrnilevi() != null){
         // y-ovemu levemu sinu nastavimo za očeta vozlišče x
         y.vrnilevi().nastaviOce(x);
      }
      // y-u nastavimo x-ovega očeta
      y.nastaviOce(x.vrnioce());
      //če ima x očeta
      if(x.vrnioce()!= null){
         // in če je x levi sin
         if(x == x.vrnioce().vrnilevi()){
            //potem x-ovemu očetu nastavimo y za levega sina
            x.vrnioce().nastaviLevi(y);
         }
         else{
            // drugače mu nastavimo y za desnega sina
            x.vrnioce().nastaviDesni(y);
         }
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
        }
        else {
            // drugače je y koren
            koren = y;
        }
        //y-u nastavimo x za levega sina
        y.nastaviLevi(x);
        // če x obstaja
        if(x != null){
            // mu za očeta nastavimo y
            x.nastaviOce(y);
        }
        // rezultat rotacije je kazalec na novi koren
        return y;
    }

    // desna rotacija
    public RCDVozel desnaRotacija(RCDVozel x){
        // y je x-ov levi sin
        RCDVozel y = x.vrnilevi();
        // x dobi kot levo poddrevo, desno poddrevo svojega levega sina
        x.nastaviLevi(y.vrnidesni());
        // če je y sploh imel desno poddrevo
        if(y.vrnidesni() != null){
            // bivši y-ov desni sin ima sedaj za očeta vozlišče x
            y.vrnidesni().nastaviOce(x);
        }
        // y dobi za očeta x-ovega očeta
        y.nastaviOce(x.vrnioce());
        // če x ima očeta
        if(x.vrnioce()!= null){
            // in če je x desni sin
            if(x == x.vrnioce().vrnidesni()){
                //potem x-ov oče dobi y kot novo desno poddrevo
                x.vrnioce().nastaviDesni(y);
            }
            else{
                // drugače dobi y kot novo levo poddrevo
                x.vrnioce().nastaviLevi(y);
            }
        }
        else{
            // drugače je y koren
            koren = y;
        }
        // y dobi x za desnega sina
        y.nastaviDesni(x);
        // mu za očeta nastavimo y
        x.nastaviOce(y);
        // vrnemo y
        return y;
    }
}
```

5.3.1. VSTAVLJANJE IN BRISANJE V RDEČE-ČRNEM DREVESU

VSTAVLJANJE:

Vstavljanje v rdeče črno drevo (od tu naprej bomo za rdeče črno drevo uporabljali kratico RČD) poteka v dveh korakih:

Diplomska naloga Fakulteta za matematiko in fiziko

Diplomska naloga

- Najprej novo vozlišče vstavimo na enak način, kot ga vstavimo v navadno iskalno dvojiško drevo. Podatek torej vstavimo v ustrezni list drevesa. Novo vozlišče pobarvamo rdeče.

- Na drugem koraku poskrbimo, da je novo drevo spet rdeče črno. Da ohranimo lastnosti RČD, določena vozlišča prebarvamo in po potrebi izvedemo nekaj rotacij.

Ugotovimo najprej, katere lastnosti RČD smo morda prekršili, ko smo vstavili novo vozlišče.

Pravilo 1 še vedno velja. Morda pa smo kršili drugo pravilo. Novo vozlišče sicer nima sinov, lahko pa smo vozlišče vstavili kot sina rdečega vozlišča. Prav tako smo lahko kršili pravilo 3. To se zgodi le, če smo vstavljali podatek v prazno drevo. Takrat pač vozlišče pobarvamo črno.
Kaj pa pravilo 4? Ker je novo vozlišče rdeče, ima tudi njegovo prazno poddrevo enako število črnih vozlišč na poti od korena do njega kot ga imajo ostala prazna poddrevesa. Pri slednjih se to število ni spremenilo, torej lastnost 4 še vedno velja.

Ko v RČD vstavimo novo vozlišče, torej lahko kršimo le pravilo, ki pravi, da rdeče vozlišče ne more imeti rdečega sina. Natančneje, ta lastnost je kršena le, če je oče vstavljenega vozlišča prav tako pobarvan rdeče, saj vstavljeno vozlišče vedno pobarvamo rdeče.

Kršitev lahko nastopi v šest različnih primerih. Tri med njimi so simetrične ostalim trem glede na to, ali je oče vstavljenega vozlišča levi ali desni sin svojega očeta.

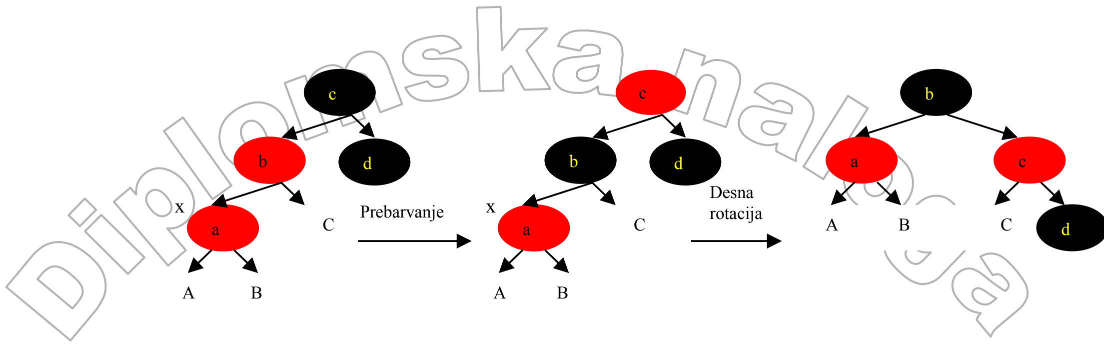
Poglejmo, kako ustrezno prebarvamo in rotiramo vozlišča. Naj bo x na začetku vstavljeno vozlišče, kasneje pa tekoče rdeče vozlišče.

1. Če vozlišče x nima starega očeta, postopek končamo. Oče vstavljenega vozlišča je lahko le koren drevesa in je zato črn.
2. Oče vozlišča x je rdeč in stari oče obstaja. Stari oče je nujno črn, saj bi bilo drugače prej kršeno drugo pravilo, ki pravi, da ima rdeče vozlišče le črne sinove.
   2.1. Stric vozlišča x je rdeč. To pomeni, da je stari oče imel oba sinova rdeča.
      - starega očeta obarvamo rdeče, očeta in strica pa obarvamo črno. S tem prebarvanjem se glede četrte lastnosti ni spremenilo prav nič.
      - Po prebarvanju stari oče postane trenutno vozlišče x (je rdeč!) in ponovimo postopek

   2.2. Stric vozlišča x je črn ali ga ni
      - Vozlišč ne moremo prebarvati kot v primeru 2.1. saj bi s tem ogrozili drugo lastnost. Oče postane trenutno vozlišče. Starega očeta obarvamo rdeče in izvedemo dvojno rotacijo, odvisno od orientacije poddreves (če smo na desni strani izvedemo levo rotacijo in nato desno, drugače ravno obratno) in ustrezno barvanje.

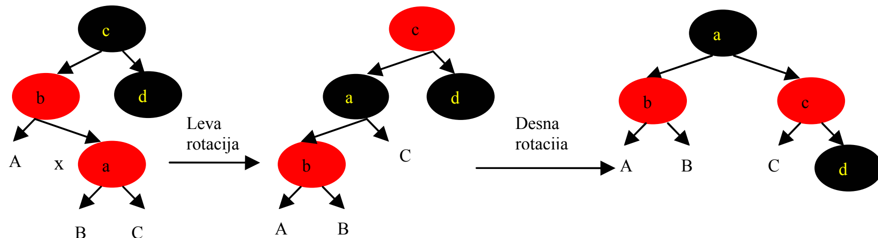
      2.2.1. Oče vozlišča x (vozlišče b) je levi sin svojega očeta:
         - Vozlišče x je levi sin svojega očeta in stric je črn (ali pa ga ni). V tem primeru najprej obarvamo očeta črno, starega očeta rdeče in izvedemo desno rotacijo nad x-ovim starim očetom.

Fakulteta za matematiko in fiziko



Druga lastnost ni več kršena. Prav tako se v tem poddrevesu ni spremenilo število črnih vozlišč na poti od korena poddrevesa do ustreznih praznih poddreves, torej še vedno drži četrta lastnost.

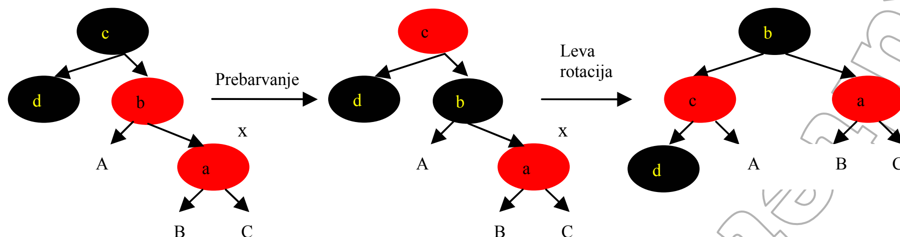
- Vozlišče x je desni sin svojega očeta in stric je črn (ali pa ga ni). Oče postane trenutno vozlišče, nad katerim izvedemo levo rotacijo. Nato očeta obarvamo črno, starega očeta rdeče in izvedemo desno rotacijo nad starim očetom.



Iz slike je razvidno, da 2. in 4. lastnost ohranita veljavo. Morebitni koreni vozlišč A, B in C so namreč že od prej črni, kar ohranja 2. lastnost. Prav tako pa se ni spremenilo število črnih vozlišč za nobeno prazno poddrevo v tem poddrevesu.

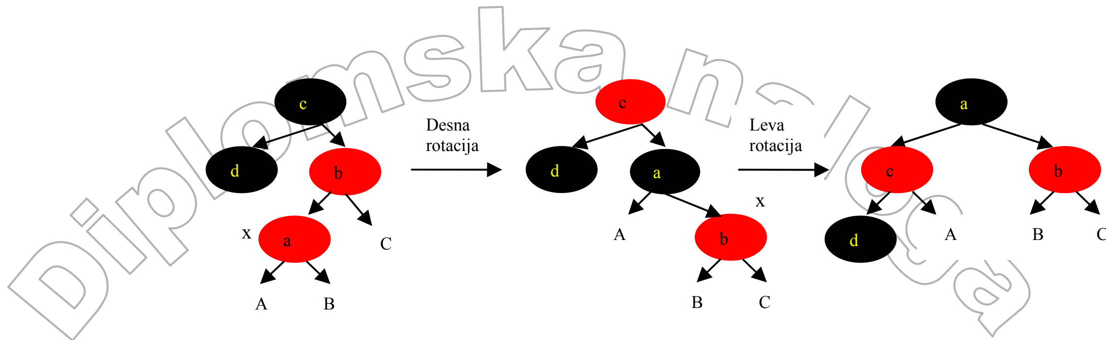
2.2.2. Oče vozlišča x (vozlišče b) je desni sin svojega očeta:

- Vozlišče x je desni sin svojega očeta in stric (vozlišče d) je črn (ali pa ga ni). V tem primeru najprej obarvamo očeta črno, starega očeta rdeče in izvedemo levo rotacijo nad x-ovim starim očetom. Slika nam pove, da smo s tem res dobili nazaj RČD.

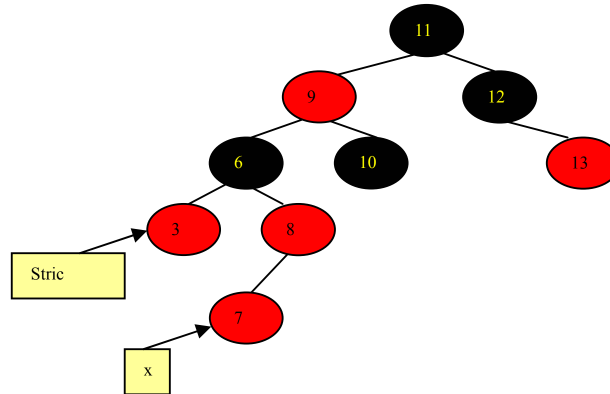


- Vozlišče x je levi sin svojega očeta in stric je črn (ali pa ga ni). Oče postane trenutno vozlišče, nad katerim izvedemo desno rotacijo. Nato očeta obarvamo črno, starega očeta rdeče in izvedemo levo rotacijo nad starim očetom.

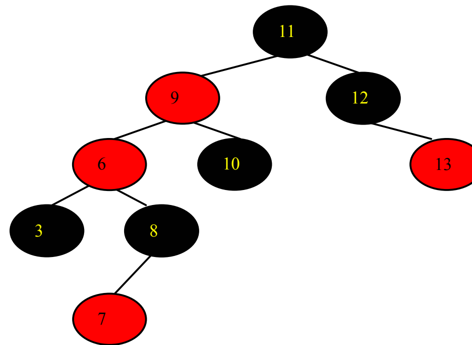
Diplomska naloga
Fakulteta za matematiko in fiziko



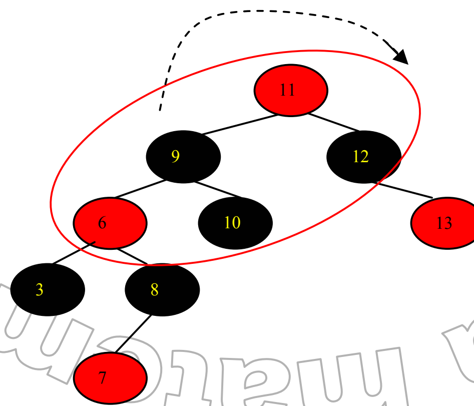
Poglejmo si zgled, ko v RČD vstavimo podatek 7.



Imamo primer 2.1. (stric je rdeč). Vozlišče 6 torej obarvamo rdeče, vozlišči 3 in 8 pa črno.



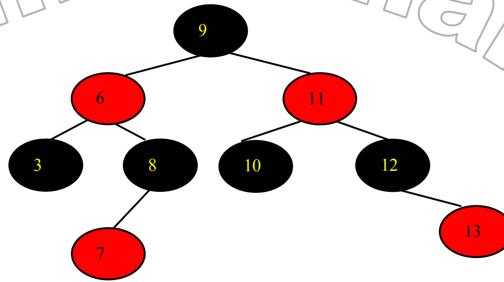
Vozlišče 6 postane trenutno vozlišče in preverjanje se rekurzivno nadaljuje. Tu nastopi primer 2.2.1., ko je stric črne barve. Zato očeta trenutnega vozlišča obarvamo črno, starega očeta rdeče in izvedemo desno rotacijo:



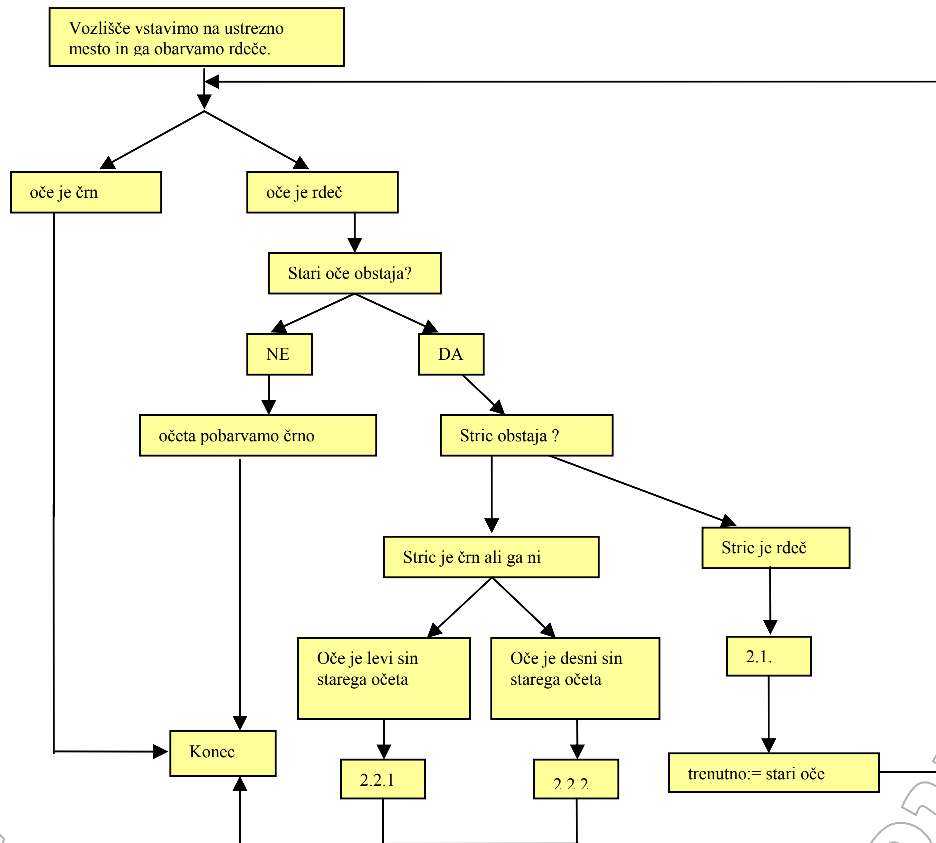
Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

Po končanem postopku res dobimo rdeče črno drevo.

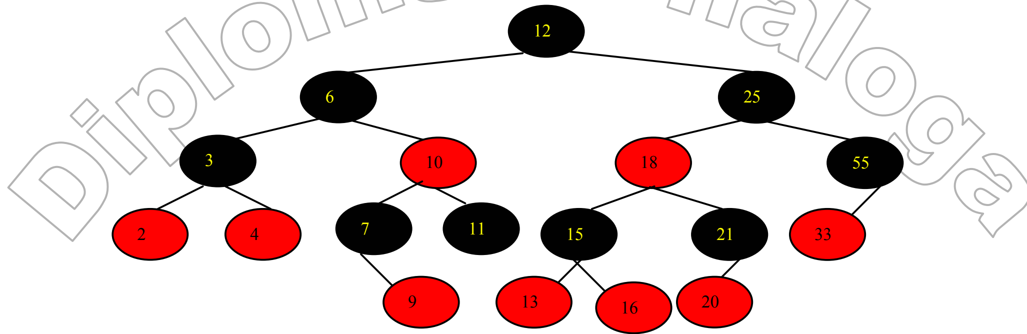


Za boljšo predstavo si predstavimo vstavljanja v RČD z naslednjo shemo:

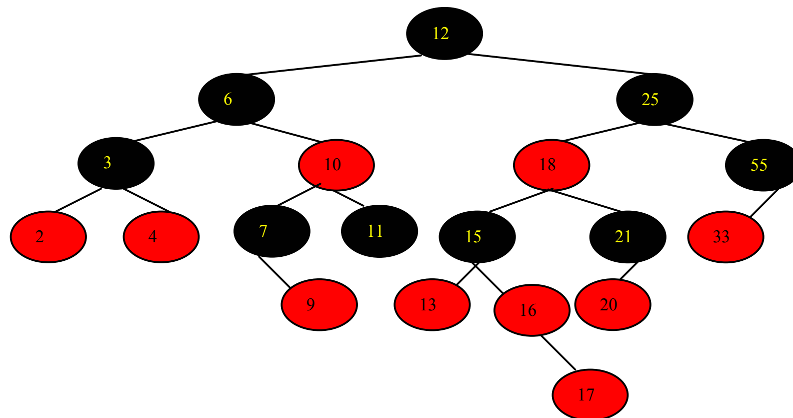


Fakulteta za matematiko in fiziko

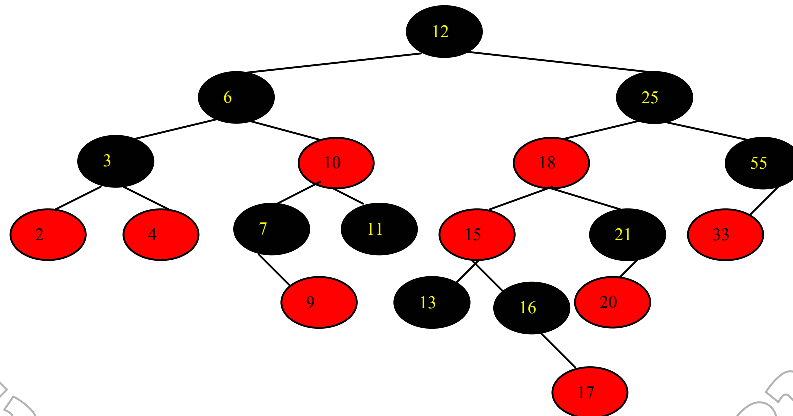
Oglejmo si delovanje sheme še na malo večjem primeru. Dano je naslednje RČD.



Poglejmo, kaj se zgodi po vstavljanju vozlišča 17. Mesto za vozlišče poiščemo po iskalni poti in ga obarvamo rdeče.

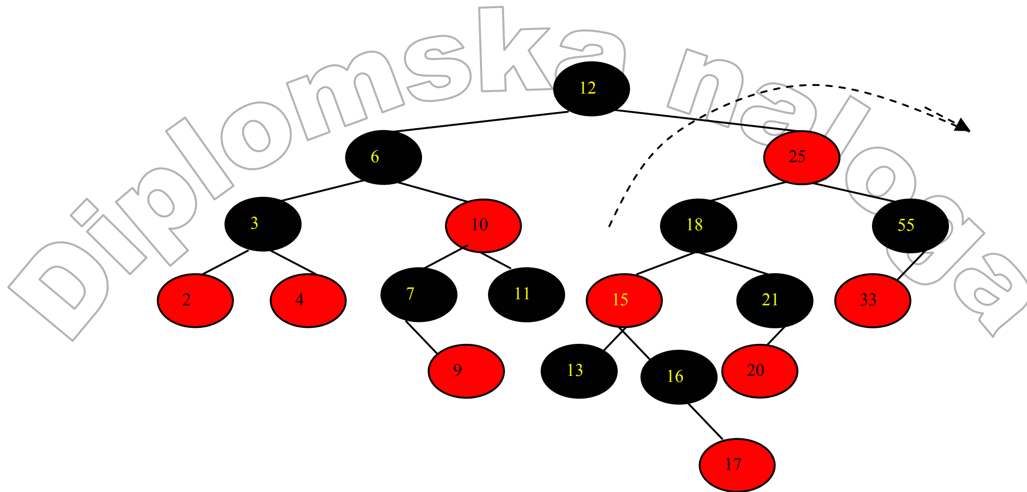


Kršimo pravilo 2, ki pravi, da ima rdeče vozlišče lahko le črne sinove. Stari oče vstavljenega vozlišča obstaja in je črne barve. Zato preverimo, kakšne barve je stric vstavljenega vozlišča. Stric novega vozlišča je rdeče barve in sledi prebarvanje. Starega očeta pobarvamo rdeče, očeta in strica črno.

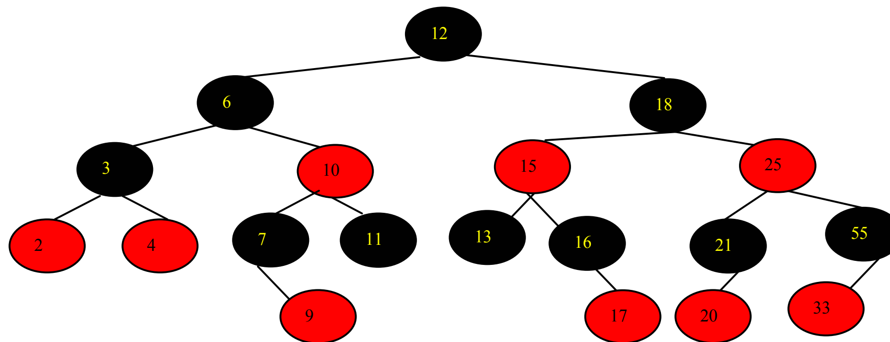


Ker je stari oče postal rdeč, lahko nastopi konflikt, če je njegov oče (vozlišče 18) rdeč. Zato preverjanje nadaljujemo. Stari oče postane trenutno vozlišče in postopek ponovimo. Preverimo barvo njegovega strica. Stric vozlišča 15 (vozlišče 55) je črne barve. Naleteli smo na primer 2.2.1., zato očeta obarvamo črno, starega očeta rdeče in izvedemo desno rotacijo.

Diplomska naloga
Fakulteta za matematiko in fiziko



Dobimo:

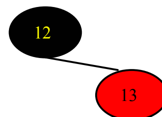


Preverimo, če vsa pravila RČD res držijo in imamo res RČ drevo. Pravilo 1 drži, pravilo 2 tudi drži, saj ima vsako rdeče vozlišče le črne sinove. Tudi pravili 3 in 4 držita. Koren drevesa je črn in tudi vsaka enostavna pot od korena do praznega poddrevesa vsebuje enako število črnih vozlišč.

Poglejmo si še primer, kako poteka sestavljanje RČD. V prazno drevo bomo vstavljali zaporedje števil 12, 13, 14, 5, 3, 11, 6, 10, 8.

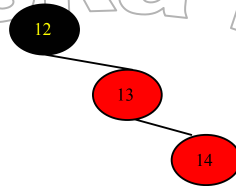
Vstavimo 12:

Ker je 12 koren drevesa, ki ga vstavljamo, postane koren in ga obarvamo črno. Vstavljamo naprej: 13

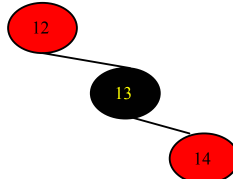


Vsako vstavljeno vozlišče se obarva rdeče (črno ga ne smemo obarvati zaradi pravila 2). Njegov oče je črn. Vsa pravila RČD držijo, zato nadaljujemo z vstavljanjem. Vstavimo 14:

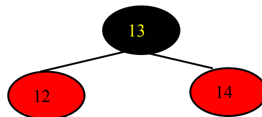
Diplomska naloga
Fakulteta za matematiko in fiziko



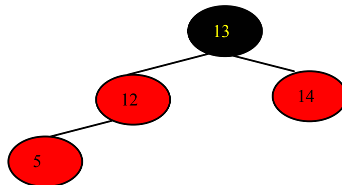
Z vstavljanjem vozlišča s podatkom 14 kršimo pravilo 2 (rdeče vozlišče ima le črne sinove). Drevo popravimo tako, da pogledamo kako je obarvan stric vozlišča 14 (primer 2.2.1.). Ker ta ne obstaja, obarvamo očeta črno, starega očeta rdeče in uporabimo levo rotacijo.



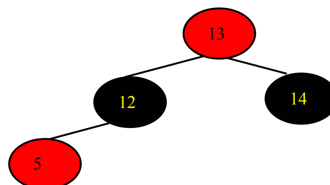
Po rotaciji je RČD:



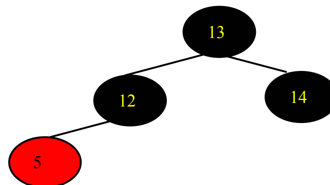
Nadaljujmo s 5:



Z vstavljanjem vozlišča s podatkom 5 smo ponovno kršili drugo pravilo. Pogledamo, kako je obarvano vozlišče 14 – stric vozlišča 5 - in kako je pobarvan oče vozlišča 5. Ker sta oba obarvana rdeče, ju obarvamo na črno, starega očeta (njunega očeta) pa na rdeče.



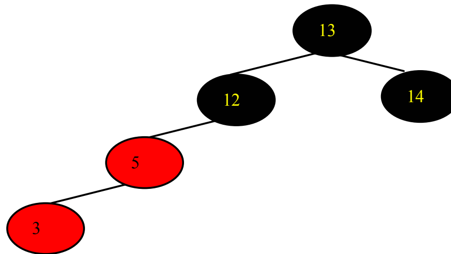
S tem smo prekršili pravilo 3, ki pravi, da mora biti koren črn. Zato prebarvamo koren na črno. S tem se sicer število črnih vozlišč na poti do praznih poddreves poveča, a ker je to koren, se poveča za vsako pot do praznega poddrevesa.



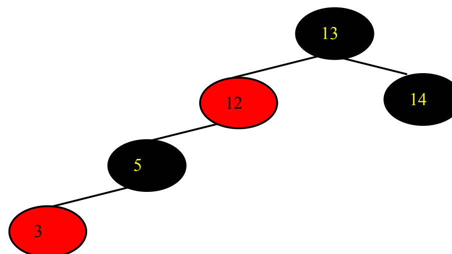
Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

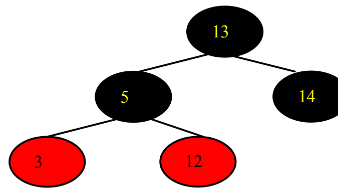
Vsako vozlišče je torej črno ali rdeče, rdeča vozlišča imajo le črne sinove, koren drevesa črn in vsaka pot od korena do praznih poddreves vsebuje enako število črnih vozlišč, torej pravila RDČ držijo. Nadaljujmo z vstavljanjem 3:



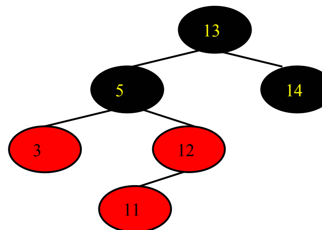
Ker je oče vozlišča 3 rdeč in ker vozlišče 3 nima strica, obarvamo očeta črno, starega očeta rdeče in izvedemo desno rotacijo.



Po rotaciji je ni kršeno nobeno pravilo več,



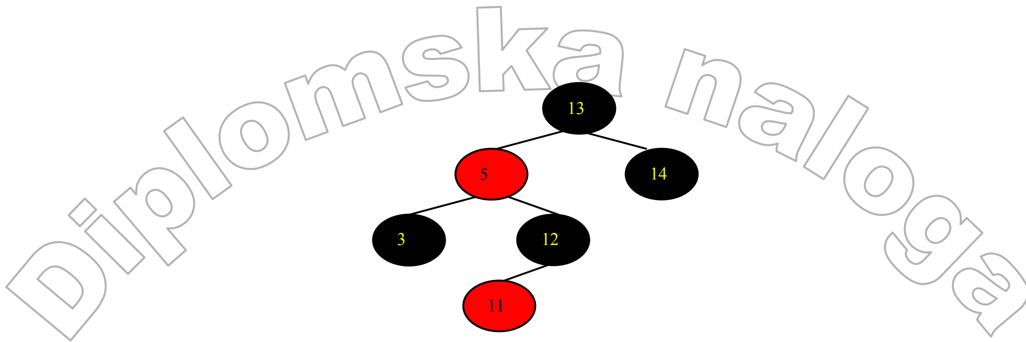
in nadaljujemo z vstavljanjem. Vstavimo 11:



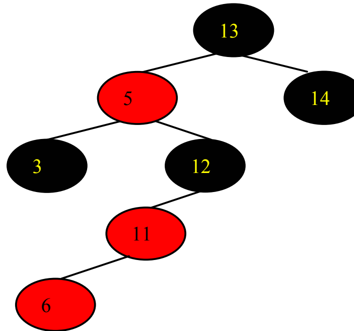
Oče vstavljenega vozlišča je rdeč, zato je ponovno kršeno pravilo, da ima rdeče vozlišče le črne sinove (pravilo 2). Ravnamo podobno kot v prejšnjem koraku. Pogledamo strica vozlišča 11, vozlišče 3 in ugotovimo, da sta stric kakor tudi oče vozlišča 11 rdeča, zato ju prebarvamo na črno in njunega očeta na rdeče. S tem dosežemo veljavnost pravil 2 in 4.

Fakulteta za matematiko in fiziko

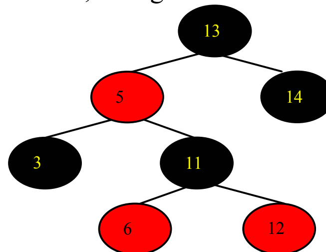
Diplomska naloga



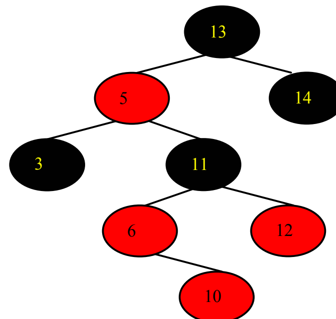
Preverimo še en korak višje in ugotovimo da je oče na rdeče prebarvanega vozlišča (vozlišča 5) črn. Preverimo še lastnost 4. Tudi ta lastnost je ohranjena. Torej so lastnosti RČD ohranjene. Nadaljujemo z vstavljanjem 6



Po vstavljanju vozlišča 6 ponovno kršimo pravila RČD. Ravnamo podobno kot v prejšnjem koraku. Kršimo pravilo 2, zato pogledamo strica vozlišča 11 in ugotovimo da ga ni (primer 2.2.1.). Zato prebarvamo očeta na črno, starega očeta na rdeče in uporabimo desno rotacijo.

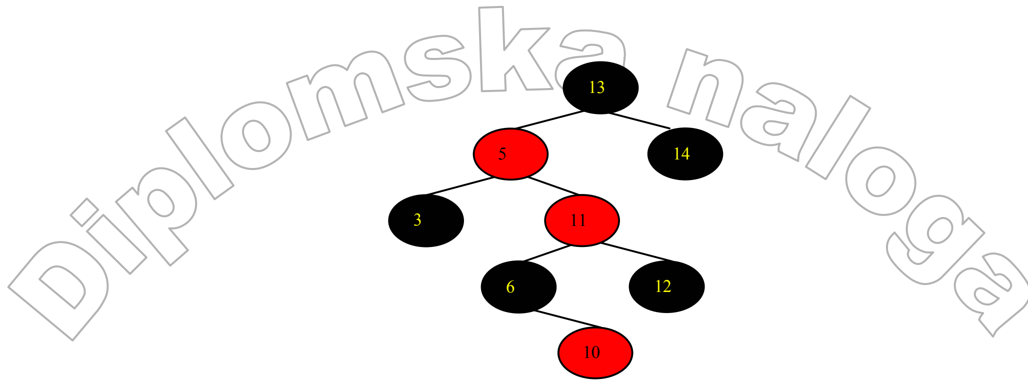


Preverimo stanje. Vsa pravila držijo, torej je dobljeno res RČD. Vstavimo 10:

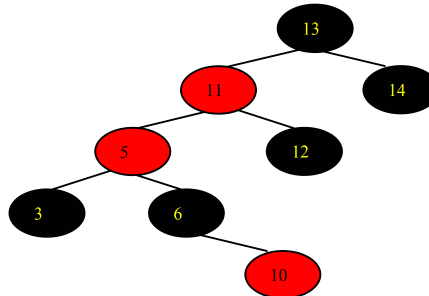


Stric vozlišča 10 je rdeče barve, prav tako je rdeče barve tudi oče vozlišča 10 (vozlišče 6) zato ju prebarvamo na črno, njunega očeta pa na rdeče. Lastnosti RČD moramo sedaj preveriti za stanje, kjer je "izhodiščno" vozlišče stari oče vozlišča 10, torej vozlišče 11:

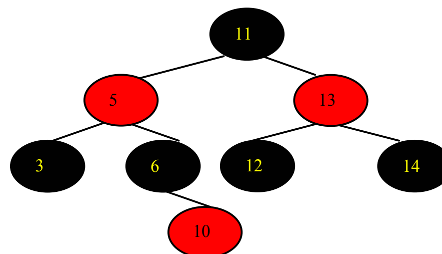
Fakulteta za matematiko in fiziko



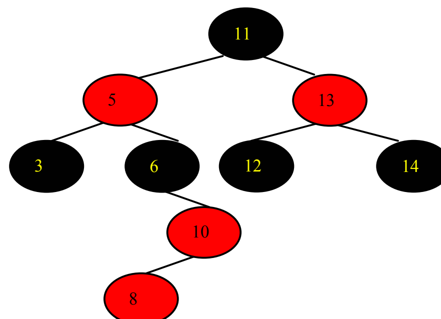
Vozlišče 11 in njegov oče vozlišče 5 sta rdeča, stari oče pa je črn. Toda tokrat je stric vozlišča 11, vozlišče 14, črne barve. Zato izvedemo rotacijo. Ker je vozlišče 11 desni sin svojega očeta, izvedemo levo rotacijo in dobimo:



Po rotaciji je zopet kršeno pravilo 2. Sedaj preverjamo vozlišče 5, ki je levi sin svojega očeta, zato prebarvamo očeta vozlišča 5 na črno, starega očeta pa na rdeče. Nastopi desna rotacija, ki privede do drevesa:



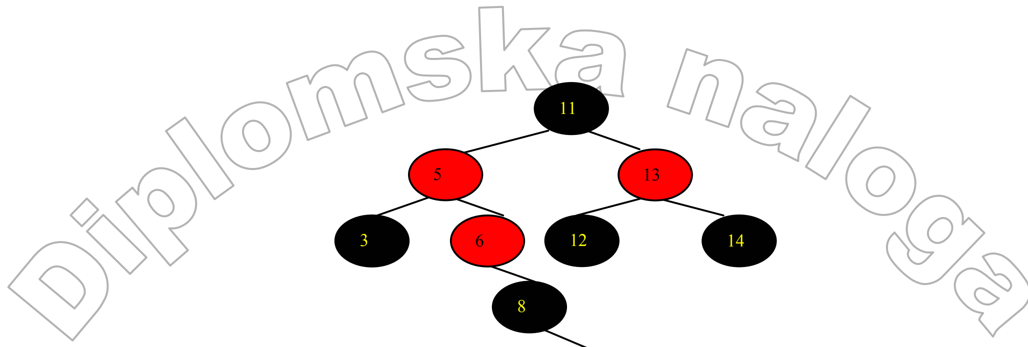
11 postane novi koren drevesa, njegovo desno poddrevo gre na njegovo desno, vendar na levo od vozlišča 13, levo poddrevo pa ostane levo od vozlišča 11.Vstavimo še: 8



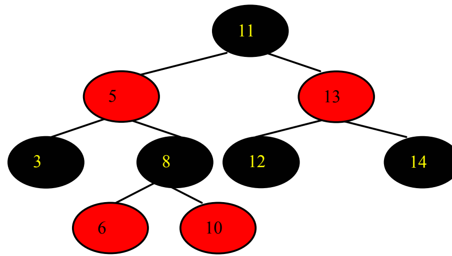
Z vstavljanjem vozlišča 8 ponovno kršimo pravila RČD. Ravnamo podobno kot v prejšnjem koraku. Kršimo pravilo 2, zato pogledamo strica vozlišča 10, in ugotovimo, da ga ni. Zato uporabimo najprej desno rotacijo, nato prebarvanje očeta in starega očeta.

Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

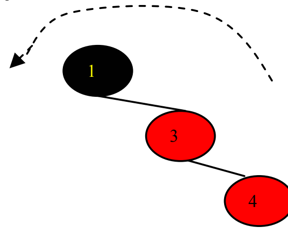


Sledi še leva rotacija okoli starega očeta (vozlišča 6).

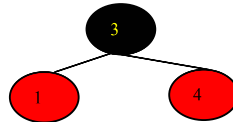


Po tem koraku ni kršeno nobeno pravilo več.

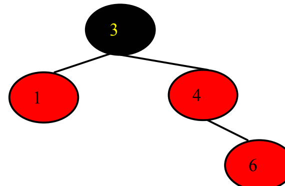
Poglejmo si še primer ko sestavljamo RČD iz urejenih podatkov. Vstavimo zaporedje števil 1, 3, 4, 6, 8, 9, 10, 11 in 13. Najprej so na vrsti 1,3 in 4.



Po vstavitvi vozlišča 4 kršimo pravilo 2. Stari oče je črn, strica pa ni, zato prebarvamo očeta in starega očeta in izvedemo levo rotacijo.



Nadaljujemo s 6:

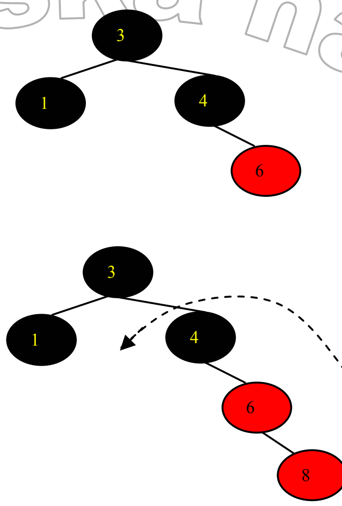


Zopet kršimo pravilo 2. Stari oče je črn, stric in oče sta rdeča, zato sledi prebarvanje.Stric in oče postaneta črna, stari oče pa rdeč. Da bo držalo še pravilo 3, koren pobarvamo črno.

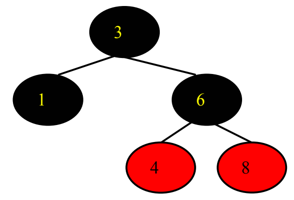
Fakulteta za matematiko in fiziko

Diplomska naloga

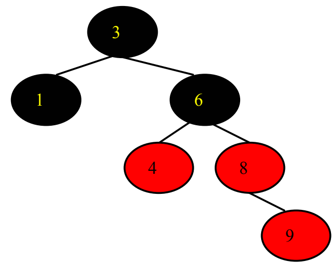
Vstavimo 8:



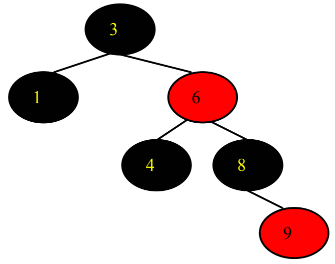
Z vstavljanjem kršimo pravilo 2, zato preverimo, ali stric obstaja. Ker ga ni je potrebno prebarvanje in leva rotacija.



Vstavimo 9:

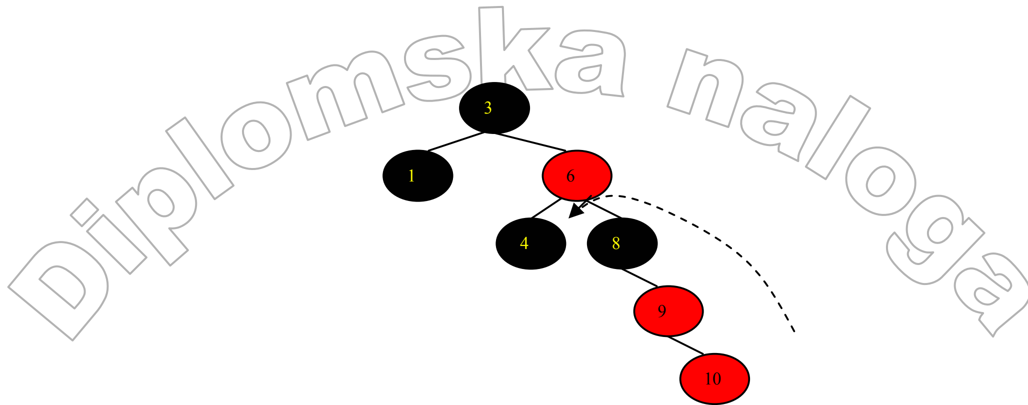


Zopet kršimo pravilo 2. Preverimo, kakšne barve je stric vstavljenega vozlišča. Stari oče je črn, stric in oče pa sta rdeča, zatorej sledi obarvanje.

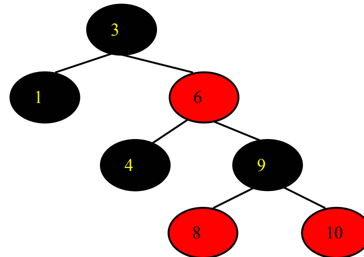


Vse je v redu, zato nadaljujemo in vstavimo 10:

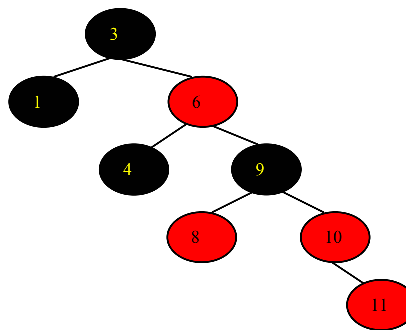
Fakulteta za matematiko in fiziko



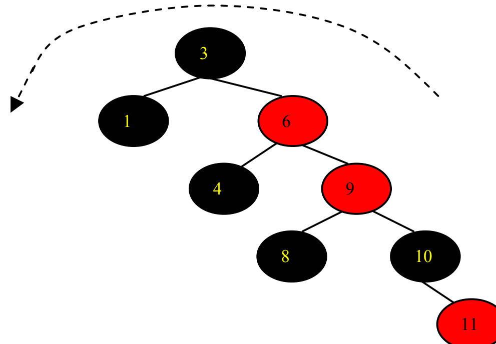
Preverimo pravila in vidimo, da spet kršimo pravilo 2. Sedaj strica ni, zato prebarvamo očeta in starega očeta in izvedemo levo rotacijo.



Vstavimo 11:

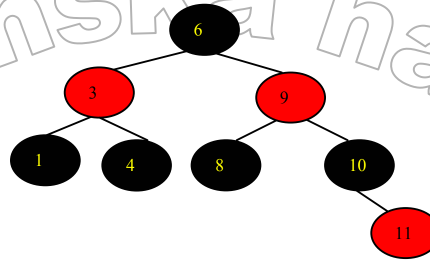


Zopet smo kršili pravilo 2. Preverimo barvo strica vstavljenega vozlišča. Oče in stric sta rdeče barve, stari oče pa črne. Obarvamo očeta in strica na črno, starega očeta pa na rdeče in preverimo dalje.

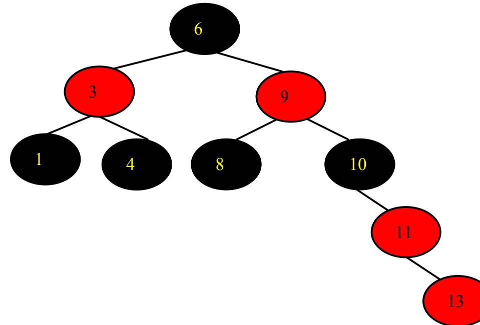


Trenutno vozlišče na katerem ponovimo postopek, postane stari oče (vozlišče 9). Njegov stric (vozlišče 1) je črne barve. Zato prebarvamo očeta in starega očeta in opravimo levo rotacijo. Dobimo drevo:

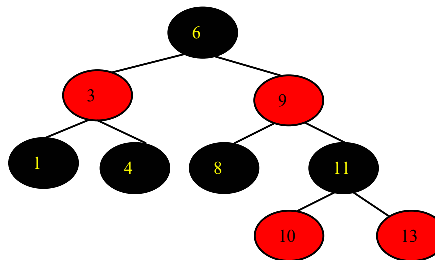
Diplomska naloga
Fakulteta za matematiko in fiziko



Vstavimo še podatek 13



Opravimo ustrezne spremembe. Na koncu je drevo takšno:



Poglejmo si sedaj ustrezne metode v jeziku Java. Metode so napisane v skladu z opisanimi postopki in pri njihovem zapisu je poudarek na enostavnosti razumevanja in ne na učinkovitosti.

Metoda VSTAVI:

```
// metoda vstavi, ki vstavi podatek v RCD
public void vstavi(int z)throws Exception{
   RCDVozel y = null;
   RCDVozel x = koren
   // ali je vstavljeno vozlišče levi sin
   boolean levi = false;
   // dokler ne bomo prišli do korena – trenutno vozlišče x ne obstaja več
   while(x != null){
      y = x;
      // če podatek z spada v levo poddrevo
      if(z < x.vrniPodatek()){
         // gremo v levo
         x = x.vrniLevi();
         levi = true;
      }
      else{
         // drugače gremo v desno
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
            x = x.vrniDesni();
            levi = false;
         }
      }
      // ko najdemo pravo mesto, naredimo nov RCDVozel s podatkom z
      RCDVozel pom = new RCDVozel(z);
      // in kazalcem na očeta y
      pom.nastaviOce(y);
      // v drevo smo vstavili prvi podatek
      if(y == null){
         koren = new RCDVozel(z);
      }
      else if(levi) y.nastaviLevi(pom);
         else y.nastaviDesni(pom);
         // in ga obarvamo rdeče
         pom.nastaviRdec(true);
         // ter preverimo barve po vstavljanju
         preveriBarve_poVstavljanju(pom);
}

// metoda, ki preveri pravilnost RCD po vstavljanju
public void preveriBarve_poVstavljanju(RCDVozel r)throws Exception{
      //oče obstaja in je rdeče barve
      while(r.vrnioce() != null && r.vrnioce().rdec()){
         // r-jev oče je levi sin
         if(r.vrnioce() == r.vrnioce().vrnioce().vrnilevi()){
            //smo na levi strani, oče r-ja je levi sin
            RCDVozel y = r.vrnioce().vrnioce().vrnidesni();
            //če je y tudi rdeč
            if(y != null && y.rdec()){
               //sledi prebarvanje očeta in strica
               //očeta obarvamo črno
               r.vrnioce().nastaviRdec(false);
               //strica obarvamo črno
               y.nastaviRdec(false);
               //starega očeta obarvamo na rdeče
               r.vrnioce().vrnioce().nastaviRdec(true);
               //stari oče postane trenutno vozlišče
               r = r.vrnioce().vrnioce();
            }
            // če je y črn oz. y-a ni
            else{
               // če je r desni sin
               if(r == r.vrnioce().vrnidesni()){
                  // zaradi rotacije trenutno vozlišče postane oče
                  r = r.vrnioce();
                  // izvedemo levo rotacijo
                  levaRotacija(r);
               }
               // oče postane črn
               r.vrnioce().nastaviRdec(false);
               // stari oče rdeč
               r.vrnioce().vrnioce().nastaviRdec(true);
               // rotiramo desno okoli starega očeta
               desnaRotacija(r.vrnioce().vrnioce());
            }
         }
         else {
            // smo na desni strani in y je r-jev stric
            RCDVozel y = r.vrnioce().vrnioce().vrnilevi();
            // če y sploh obstaja in je rdeč
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
            if(y != null && y.rdec()){
                // očeta obarvamo črno
                r.vrnioce().nastaviRdec(false);
                // strica obarvamo črno
                y.nastaviRdec(false);
                // starega očeta obarvamo rdeče
                r.vrnioce().vrnioce().nastaviRdec(true);
                // stari oče postane trenutno vozlišče
                r = r.vrnioce().vrnioce();
            }
            // če je y črn oziroma y-a ni
            else{
                // če je r levi sin
                if(r == r.vrnioce().vrnilevi()){
                    // oče postane trenutno vozlišče
                    r = r.vrnioce();
                    // izvedemo desno rotacijo
                    desnaRotacija(r);
                }
                // očeta obarvamo črno
                r.vrnioce().nastaviRdec(false);
                // starega očeta obarvamo rdeče
                r.vrnioce().vrnioce().nastaviRdec(true);
                // in izvedemo desno rotacijo na starem očetu
                levaRotacija(r.vrnioce().vrnioce());
            }
        }
    }
    // koren pobarvamo črno
    this.koren.nastaviRdec(false);
}
```

BRISANJE:

Brisanje v RČD zasnujemo podobno kot brisanje v običajnem iskalnem dvojiškem drevesu, le da moramo potem poskrbeti za ustrezne popravke s katerimi ohranimo lastnost RČD.
Spomnimo se sheme brisanja elementa v iskalnem drevesu:

- če element nima sina, ga lahko neposredno odstranimo
- če element ima ali samo levega ali samo desnega sina, ga odstranimo in prevežemo ustrezen kazalec njegovega očeta
- če ima oba sina, ga nadomestimo s prvim največjim, torej največjim iz levega poddrevesa (ali prvim najmanjšim iz desnega poddrevesa)

Kadar brišemo element, ki ima oba sinova, se na njegovem mestu pojavi nov podatek (nadomestno vozlišče y). Barva vozlišča se ohrani, torej se glede lastnosti RČD ne spremeni prav nič.

Ob odstranjevanju vozlišč z enim sinom, ga enostavno odstranimo in prevežemo kazalec njegovega očeta na njegovega sina. Pri tem lahko pride le do kršitve drugega pravila (da ima rdeče vozlišče le črne sinove), pa tudi četrto pravilo je sedaj v tem poddrevesu lahko kršeno. Za oboje bo poskrbela spodaj omenjena metoda `preveriBarve_poBrisanju`.

Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

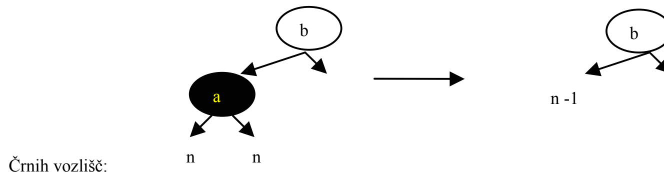
Preveriti moramo še to, kaj se zgodi, ko pobrišemo list (bodisi je to ravno vozlišče, ki ga odstranjujemo iz drevesa, bodisi je to vozlišče, ki se je »preselilo« na mesto vozlišča, ki ga odstranjujemo).

Ko odstranimo list, je treba preveriti lastnosti RČ drevesa. Prva, druga in tretja lastnost se z brisanjem lista tako ali tako nikoli ne spreminjajo. Kaj pa četrto pravilo – da vsaka pot v drevesu od korena do praznega poddrevesa še vedno vsebuje enako število črnih vozlišč?

Metodo `preveriBarve_poBrisanju` kličemo takrat, kadar je bilo brisano vozlišče y črne barve. Če je bilo vozlišče y rdeče barve, RČD lastnosti še vedno veljajo iz naslednjih razlogov:

- vsaka pot od korena do praznega poddrevesa še vedno vsebuje enako število črnih vozlišč,
- nobenega rdečega vozlišča nismo pridobili naknadno, ki bi kršil lastnost 2.
- y ne more biti koren drevesa, ker je bil rdeče barve, koren ostaja črn in lastnost 3. se ohrani.

Če je odstranjeni list črno vozlišče, na njegovo mesto pride prazno drevo. Na poti od korena do tega praznega drevesa sedaj manjka eno črno vozlišče. Potrebna je korekcija.



Vozlišče x, ki je posredovano metodi `preveriBarve_poBrisanju`, je lahko eno od dveh vozlišč:

- vozlišče je bilo y-ov edini sin, preden je bilo vozlišče y odstranjeno. Sedaj x postane y-ov in mu nastavimo njegovega očeta, zamenjamo podatke in izvedemo prevezave med vozlišči.
- y ni imel sinov in je x »stražar« ali prazno vozlišče. V tem primeru x-ov oče postane y-ov oče. Izvedemo še preostale prevezave med x-om in y-ovim očetom. Vozlišču x nastavimo podatek y-ona, nato pa na vozlišču x preverimo lastnosti RČD.

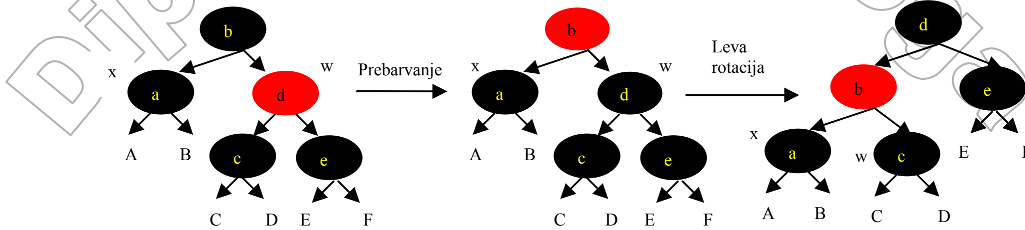
x-ov oče postane sedaj bivši y-ov oče (y je vozlišče ki je prišlo na mesto brisanega vozlišča). Če je bilo odstranjeno vozlišče črne barve, je potrebna preureditev. Če je bil y koren drevesa in rdeči sin y-a postane novi koren, smo kršili lastnost 3. Če sta oba x in oče y-a rdeča, potem kršimo lastnost 2. Odstranitev črnega vozlišča y, povzroči eno črno vozlišče manj v vseh poteh od korena do praznega poddrevesa. Kadar brišemo črno vozlišče y, se njegova črnina prenese na njegovega očeta. Problem nastane pri vozlišču x, ki ni niti črn niti rdeč, kar krši lastnost 1.
Z metodo `preveriBarve_poBrisanju` poskrbimo, da se ohranijo vse lastnosti 1, 2 in 3. Poglejmo kako se ohrani tudi lastnost 4 v vseh štirih primerih. Poglavitna ideja je ta, da se v vsakem primeru število črnih vozlišč od korena do praznega poddrevesa ohrani.

Poglejmo kako ustrezno prebarvamo in rotiramo vozlišča po brisanju črnega lista v drevesu. Vozlišče x je nadomestilo brisano vozlišče.

1. Vozlišče w, x-ov brat je rdeče barve. Ker mora imeti vozlišče w oba sina črna, ga obarvamo črno. Očeta obarvamo rdeče in izvedemo levo rotacijo okoli očeta. S tem ne kršimo nobene lastnosti RČD. Na poti do ustreznih poddreves v A, B, C, D, E IN F smo

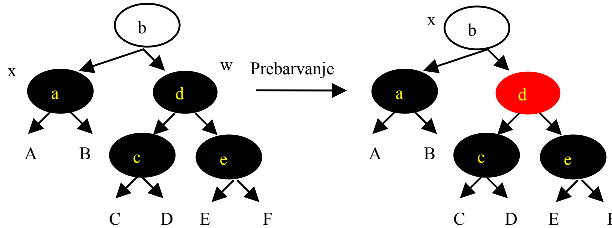
Fakulteta za računalništvo in informatiko

v tem delu prej imeli dve črni vozlišči. Enako je tudi po izvedeni spremembi, torej četrta lastnost drži, ostale tri pa glede na sliko tudi držijo. Po prvem koraku so ohranjene vse lastnosti RČD. Na sliki vidimo, da je število črnih vozlišč na vseh poteh od korena do praznega poddrevesa enako pred in po preurejanju.

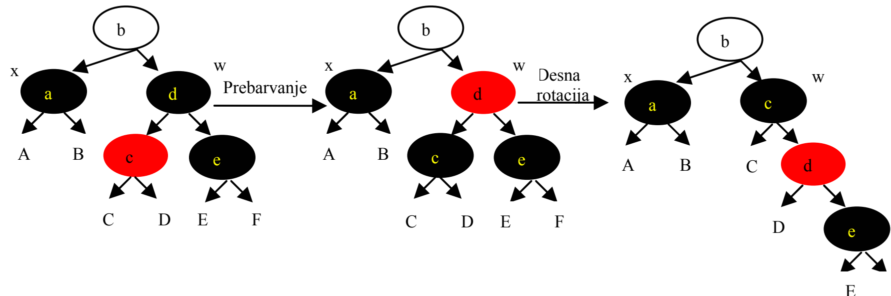


Primeri 2, 3 in 4 nastopijo takrat, kadar je brat črne barve. Med seboj pa se razlikujejo v barvah bratovih sinov. Če je vseeno, kakšne barve je vozlišče, je na sliki brez barve.

2. Vozlišče w (x-ov brat), je črn in oba njegova sinova sta črna. V tem primeru w obarvamo rdeče. Očeta obarvamo črno, če ni že črn. Vse poti skozi vozlišče d imajo po eno črno vozlišče manj od ostalih. S postopkom torej še nismo končali. Njegov oče postane trenutno vozlišče in ponovimo celotni postopek. Kot vidimo, bomo nadaljevali v stanju, ko so pravila RČD lahko kršena (konkretno 4. pravilo!).

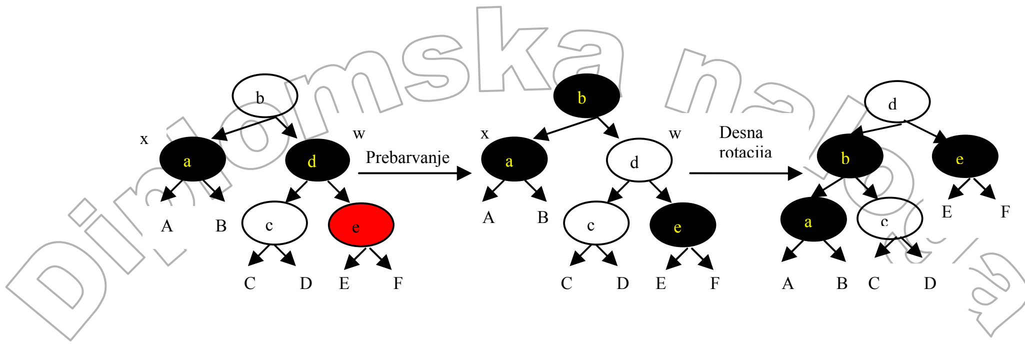


3. Vozlišče w je črno, njegov levi sin je rdeč in desni sin je črn. Prebarvamo vozlišče w na rdeče, njegovega levega sina na črno in izvedemo desno rotacijo okoli vozlišča w. Novi brat w vozlišča x je sedaj črno vozlišče z rdečim sinom desnim sinom. Drevo še vedno ni nujno RČD, zato preverjanje nadaljujemo v primeru 4.

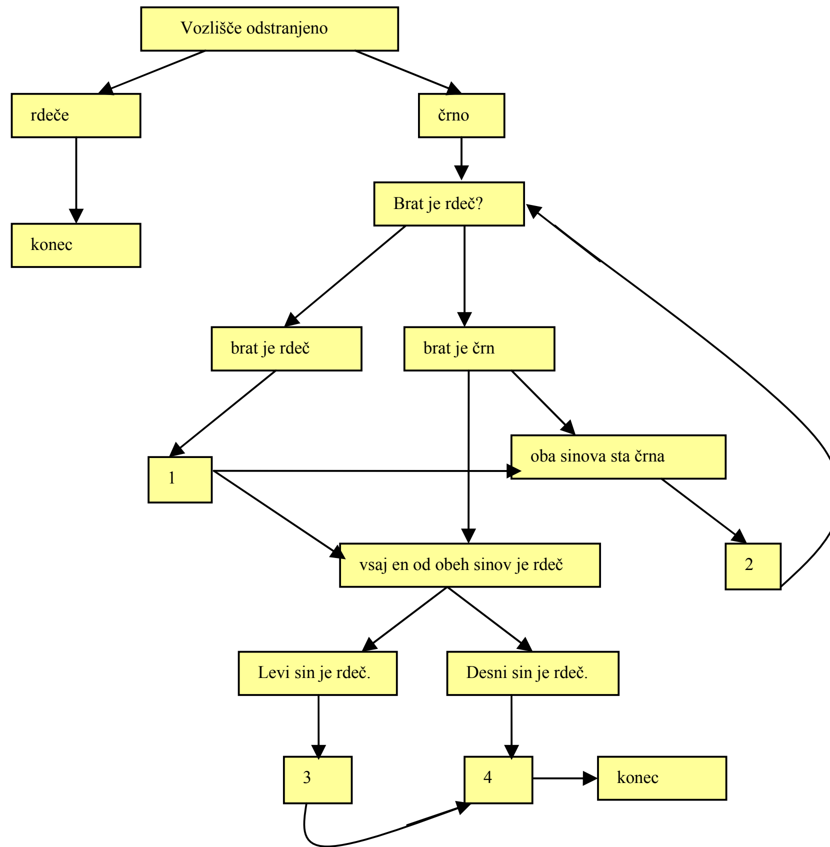


4. Vozlišče w je črno in njegov desni sin je rdeč. Prebarvamo vozlišče w tako kot je njegov oče, očeta obarvamo črno, desnega sina obarvamo črno in izvedemo desno rotacijo okoli očeta.

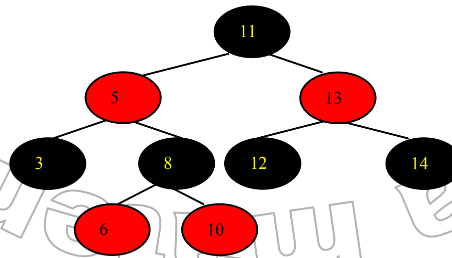
Diplomska naloga
Fakulteta za matematiko in fiziko



Preverimo ali na koncu veljajo vse lastnosti RČD
Shema brisanja vozlišč v črno-rdečem drevesu:



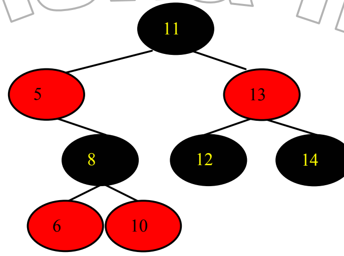
Poglejmo si primer, kako dejansko poteka brisanje v rdeče-črnem drevesu.
Dano imamo RČ drevo:



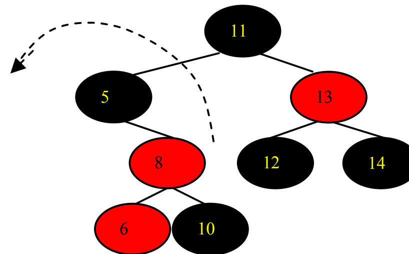
Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

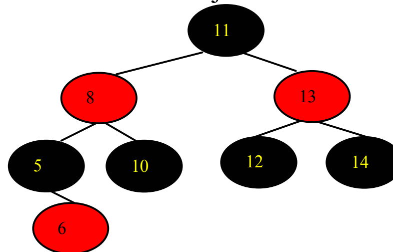
Zbrišemo črno vozlišče s podatkom 3:



Zbrisali smo črno vozlišče s podatkom 3. Ker je to list drevesa, ga enostavno odstranimo. S tem smo kršili lastnost 4, saj sedaj ni več v vseh poteh od korena do praznega poddrevesa enako število črnih vozlišč. Zato je potrebno preurejanje. Brat brisanega vozlišča 3 je črn in desni nečak je rdeč (Primer 4). Sledi prebarvanje. Oče in nečak postaneta črna, brat pa rdeč.

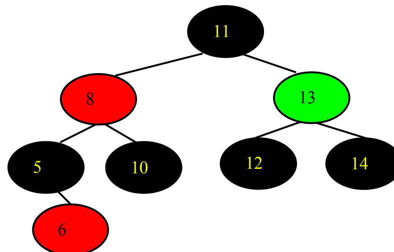


S tem smo kršili pravilo 2, zato sledi leva rotacija.



Po rotaciji so lastnosti RČD ohranjene.

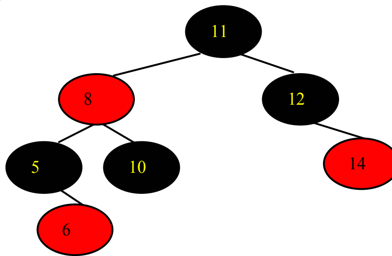
Poglejmo, kaj se zgodi, če zbrišemo rdeče vozlišče s podatkom 13?



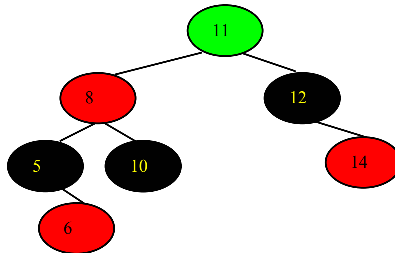
Kot vidimo, ima vozlišče s podatkom 13 (drugače je rdeče, a na sliki je zelene obarvano) levega in desnega sina. Zato podatek v tem vozlišču nadomestimo z največjim podatkom v njegovem levem poddrevesu, to je podatkom 12. Vozlišče ki je prej vsebovalo ta podatek, zbrišemo. Ker je brisani list črne barve, kršimo pravilo 4, zato sledi preurejanje.

Fakulteta za matematiko in fiziko

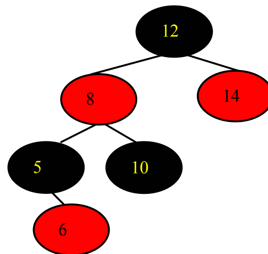
Vozlišče 14 le prebarvamo na rdeče. Sedaj je vse v redu in postopek je končan.



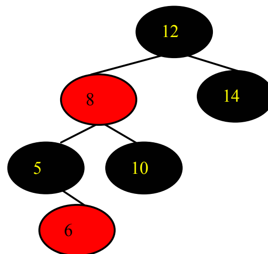
Poglejmo še, kaj se zgodi, kadar brišemo koren drevesa:



Torej radi bi zbrisali koren drevesa s podatkom 11 (zeleno vozlišče). Vozlišče ima tako kot prej oba sinova. Tokrat za ilustracijo naredimo drugačno izbiro. Na njegovo mesto dajmo najmanjše vozlišče v njegovem desnem poddrevesu, to je vozlišče s podatkom 12. V korenu torej dobimo 12, prejšnje vozlišče s tem podatkom pa moramo izbrisati. Ker ima le desnega sina, enostavno naredimo prevezavo. Oče (novi koren) dobi kot svojega desnega sina to desno poddrevo.



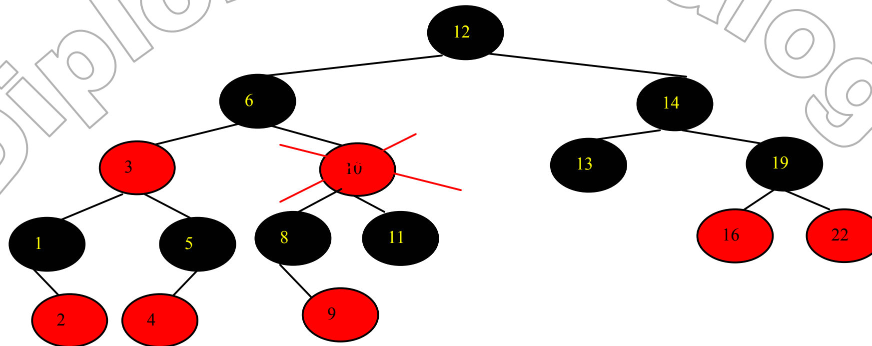
Kršeno je pravilo 4, ker je eno črno vozlišče manj v vseh poteh v desnem poddrevesu, zato moramo prebarvati vozlišče 14 črno. Vse lastnosti RDČ so tako ohranjene, zato končamo. Končno RČD je videti takole:



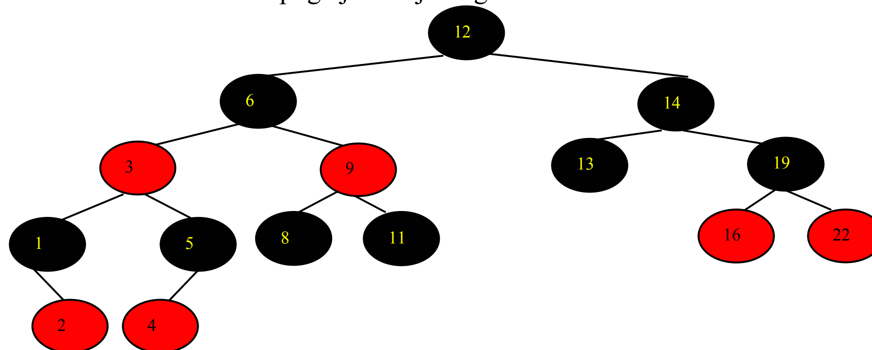
Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

Poglejmo si še primer v večjem drevesu:



Odstranimo rdeče vozlišče 10 in poglejmo kaj se zgodi.



Ko odstranimo vozlišče 10, pride na njegovo mesto največje vozlišče v njegovem levem poddrevesu, torej vozlišče 9. Ker smo brisali rdeči list (ohranila se je lastnost 4), je postopek končan.

Metoda ODSTRANI:

```
// metoda odstrani, ki odstrani podatek iz RCD
public RCDVozel odstrani(int x){
    // poiščemo vozlišče, ki vsebuje podatek x
    RCDVozel pom = poisci(x);
    // če vozlišče obstaja (podatek je v drevesu) ga izbrišemo iz drevesa
    if(pom != null) return odstrani(pom);
        return koren;
}

//metoda poišči, ki poišče vozlišče, ki vsebuje podatek, ki ga brišemo
public RCDVozel poisci(int x){
    // poiščemo vozlišče, ki vsebuje podatek x
    RCDVozel pom = koren; // pomožni kazalec na trenutno vozlišče
    while(pom != null){ // dokler ne pridemo do »dna« drevesa
        if(pom.vrniPodatek() == x) return pom; // našli smo vozlišče s podatkom x
            if(pom.vrniPodatek() < x) pom = pom.vrniDesni(); // gremo v desno
                else pom = pom.vrniLevi(); // gremo v levo
    }
    return null; // nismo našli vozlišča, zato vrnemo null
}

public RCDVozel odstrani(Vozel z) throws NullPointerException{
```

Fakulteta za matematiko in fiziko

```
// odstranimo vozlišče z
RCDVozel x = new Vozel();   // x je
RCDVozel y = new Vozel();   // y je
RCDVozel strazar = new RCDVozel();
strazar.nastaviRdec(false);    // stražar je črn

// z nima dveh sinov
if (z.vrnilevi() == null || z.vrnidesni() == null){
    y = z;
}
else{
    // y je naslednik z-ja
    y = naslednik(z);
}
//x je y-ov edini sin
if (y.vrnilevi() != null){
   // y ima levega sina
    x = y.vrnilevi();
}
else{
     // x je y-ov edini sin
    if (y.vrnidesni() != null){
        // y ima desnega sina
        x = y.vrnidesni();
    }
    else{
        // y nima nobenega od sinov, zato x-u nastavimo stražarja
        x = strazar;
    }
}
// x-u nastavimo y-ovega očeta
x.nastaviOce(y.vrnioce());
// če y nima očeta
if (y.vrnioce() == null){
    // potem je x koren
    this.koren = x;
}
else{
    // priključi očeta k svojima novima sinovoma
    // kot levi ali desni sin
    if (y == y.vrnioce().vrnilevi()){ // če je y levi sin
        y.vrnioce().nastaviLevi(x); // potem tudi x levi sin y-ovega očeta
        // x-u nastavimo še y-ov podatek
        x.nastaviPodatek(y.vrniPodatek());
    }
    else{
        // drugače je x desni sin
        y.vrnioce().nastaviDesni(x);
    }
}
if(y != z){
    // z-ju nastavimo y-ov podatek
    z.nastaviPodatek(y.vrniPodatek());
}
// če je y rdeče barve
if (!y.rdec()){
    // preverimo pravilnost RČD po brisanju
    preveriBarve_poBrisanju(x);
}
// odstranimo vse reference do stražarja
// drugače je stražar lahko viden v drevesu
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
    if (this.koren != strazar ){
       if (strazar != null && (strazar.vrnioce()).vrnilevi() == strazar){
          strazar.vrnioce().nastaviLevi(null);
       }
       if (strazar != null && (strazar.vrnioce()).vrnidesni() == strazar){
          strazar.vrnioce().nastaviDesni(null);
       }
       strazar = null;
    }
    else{
       this.koren = null;
       strazar = null;
    }
    return y;
}

// metoda, ki preveri pravilnost RČD po brisanju
public void preveriBarve_poBrisanju(RCDVozel b){
    RCDVozel w = new RCDVozel();
    // dokler b črn
    while(b != this.koren && !b.rdec()){
        // b je levi sin
       if (b == (b.vrnioce()).vrnilevi()){
          // w je desni sin, ali b-jev brat
          w = (b.vrnioce()).vrnidesni();
          // w je rdeč
          if (w.rdec()==true){
             // obarvamo w črno
             w.nastaviRdec(false);
              // očeta obarvamo rdeče
              (b.vrnioce()).nastaviRdec(true);
              rotiramo v levo okoli očeta
             levaRotacija(b.vrnioce());
             // nastavimo novi w
             w = (b.vrnioce()).vrnidesni();
          }
          // če sta oba sinova črna
          if ( w.vrnidesni() != null && w.vrnilevi() != null &&
               !w.vrnidesni().rdec() && !w.vrnilevi().rdec()){
             // w obarvamo rdeče
             w.nastaviRdec(true);
             // oče postane trenutno vozlišče
             b = b.vrnioce();
          }
          else{
              // če je desni sin črne barve
             if ( w.vrnidesni() != null && !w.vrnidesni().rdec())
                // levega obarvamo črno
                 (w.vrnilevi()).nastaviRdec(false);
                // w obarvamo rdeče
                w.nastaviRdec(true);
                // rotiramo v desno okoli w
                desnaRotacija(w);
                // w postane trenutno vozlišče
                w = (b.vrnioce()).vrnidesni();
          }
          // če je oče črn
          if (!b.vrnioce().rdec()){
             // w obarvamo črno
             w.nastaviRdec(false);
          }
          else{
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
            // drugače w obarvamo rdeče
            w.nastaviRdec(true);
         }
         // očeta obarvamo črno
         (b.vrnioce()).nastaviRdec(false);
         // desni sin obstaja
         if (w.vrnidesni() != null){
            // obarvamo ga črno
            (w.vrnidesni()).nastaviRdec(false);
         }
         // rotiramo v levo okoli očeta
         levaRotacija(b.vrnioce());
         // koren postane trenutno vozlišče
         b = this.koren;
      }
   }
   else{
      // w je levi sin
      w = (b.vrniOce()).vrniLevi();
      // če je w rdeč
      if (w.rdec()){
         // ga obarvamo črno
         w.nastaviRdec(false);
         // očeta obarvamo rdeče
         (b.vrnioce()).nastaviRdec(true);
         // rotiramo v desno okoli očeta
         desnaRotacija(b.vrnioce());
         // levi sin b-ja postane trenutno vozlišče
         w = (b.vrnioce()).vrnilevi();
      }
      // oba sinova sta črna
      if (w.vrnidesni() != null && w.vrnilevi() != null
         && (!w.vrnidesni()).rdec() && (!w.vrnilevi()).rdec())
        {
         // w obarvamo rdeče
         w.nastaviRdec(true);
         // oče postane trenutno vozlišče
         b = b.vrnioce();
      }
      else{
         // če je levi različen od nič in je črne barve
         if ( w.vrnilevi() != null && !w.vrnilevi().rdec()){
            // desnega obarvamo črno
            (w.vrnidesni()).nastaviRdec(false);
            // w obarvamo rdece
            w.nastaviRdec(true);
            // rotiramo v levo okoli w
            levaRotacija(w);
            // levi sin b-ja postane trenutno vozlišče
            w = (b.vrnioce()).vrnilevi();
         }
         // če je oče črn
         if (!b.vrnioce().rdec()){
            // w obarvamo črno
            w.nastaviRdec(false);
         }
         else{
            // drugače ga obarvamo rdeče
            w.nastaviRdec(true);
         }
         // očeta obarvamo črno
```

Diplomska naloga
Fakulteta za matematiko in fiziko

```
            (b.vrnioce()).nastaviRdec(false);
            if (w.vrnilevi() != null){
                // drugače je črn
                (w.vrnilevi()).nastaviRdec(false);
            }
            // rotiramo v desno okoli očeta
            desnaRotacija(b.vrnioce());
            //koren postane trenutno vozlišče
            b = this.koren;
        }
    }
}
// obarvamo ga črno
b.nastaviRdec(false);
}
```

Diplomska naloga
Fakulteta za matematiko in fiziko

Zahtevnost operacij nad rdeče-črnim drevesom:

- **Iskanje podatkov:**
  Čas iskanja podatka v rdeče-črnem drevesu je enak času iskanja podatka v iskalnem dvojiškem drevesu, saj karakteristike rdeče-črnih dreves med samim iskanjem niso uporabljene. Torej je časovna zahtevnost iskanja podatka enaka

$$O(\log(n)).$$

- **Vstavljanje podatkov:**
  Glede na klasično vstavljanje v iskalno drevo, se čas vstavljanja podatka v RČD poveča za konstantni faktor zaradi prebarvanj in rotacij na poti navzdol. V povprečju potrebujemo pri vstavljanju eno samo rotacijo.
  Tako kot pri vstavljanju v navadno iskalno drevo tudi iskanje ustreznega mesta za novi podatek zahteva največ obisk $(2\log_2(n+1))$ vozlišč. Dodajanje in eno barvanje vozlišč izvedemo v konstantnem času ($O(1)$). Za primerjanje, prebarvanje in rotacije pa je potrebno opraviti $O(\log(n))$ operacij. Ko seštejemo vse potrebne operacije, ugotovimo, da je časovna zahtevnost reda

$$O(\log(n)).$$

- **Brisanje podatkov:**
  Podobno kot vstavljanje podatkov tudi brisanje zahteva $(2\log_2(n+1))$ operacij, časovna zahtevnost je torej reda

$$O(\log(n)).$$

Diplomska naloga

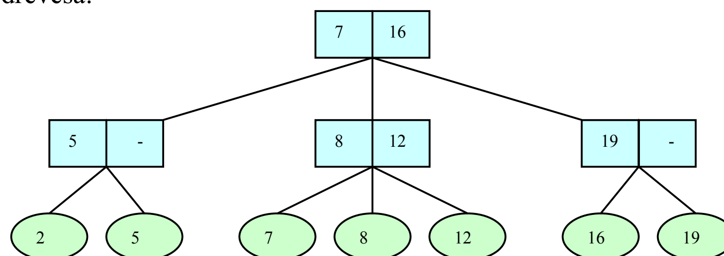
5.4. 2-3 DREVESA

2-3 drevesa so svoje ime dobila, ker ima lahko vsako vozlišče po dva ali tri sinove. So popolnoma poravnana drevesa z lastnostmi:

1. Podatki so shranjeni samo v listih drevesa.
2. Vsako notranje vozlišče ima dva ali tri sinove in en ali dva ključa. Prvi ključ je ključ najmanjšega elementa v poddrevesu, katerega koren je drugi sin, drugi ključ pa je ključ najmanjšega elementa v poddrevesu, katerega koren je tretji sin.
3. Vsi listi so na istem nivoju.

V listih drevesa imamo podatke, v ostalih vozliščih pa informacijo, kako do podatkov pridemo. Tudi tu zahtevamo da so vsi podatki različni. Podatki so urejeni naraščajoče z leve v desno, tako, da je v najbolj levem listu najmanjši, v najbolj desnem pa največji podatek

Primer 2-3 drevesa:



Zaradi lastnosti 3 je očitno, da je drevo avtomatično uravnoteženo. Glede na to, da to ni dvojiško drevo, ne moremo uporabiti formule glede minimalne in maksimalne možne višine drevesa v katerem hranimo n podatkov.

V najslabšem primeru je 2-3 drevo pravo dvojiško drevo. Vsi podatki so na zadnjem, k – tem nivoju in k je tudi višina drevesa. Dvojiško drevo ima na nivoju k :

$$2^{k-1}$$

vozlišč.Iz tega torej sledi, da pri hranjenju n podatkov v 2 – 3 drevesu dobimo drevo z višino:

$$k = \log_2(n+1) - 1 \cong \log_2(n)\,;(n >>)$$

V najboljšem primeru je 2-3 drevo pravo trojiško drevo. Trojiško drevo višine k ima na zadnjem nivoju 3^{k-1} vozlišč. Torej je najmanjša možna višina 2 – 3 drevesa z n podatki

$$k = \log_3(n) + 1 \cong \log_3 n;(n >>)$$

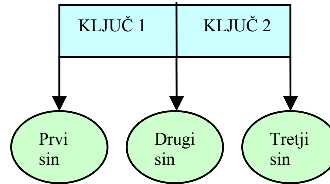
Višina 2-3 drevesa je torej med $\log_2 n$ in $\log_3 n$.

Tudi tu seveda želimo, da je višina minimalna možna. V ta namen si prizadevamo, da ima kar se da veliko vozlišč stopnjo 3 (ima po tri sinove).

5.4.1. PREDSTAVITEV 2-3 DREVES

2-3 drevo se tako imenuje zato, ker ima lahko vsako vozlišče po dva ali tri sinove. To naredi drevo za predstavitev malce bolj zapleteno kot samo dvojiško drevo. Vozlišče 2-3 drevesa je

tako sestavljeno iz enega ali dveh ključev in dveh ali treh sinov. Grafično prikazano izgleda takole:



Temu primerno torej sestavimo pravila:

1. Vozlišče mora vedno imeti Ključ 1, vendar ni potrebno da vsebuje tudi Ključ 2. Če vozlišče vsebuje oba ključa v vozlišču, potem je Ključ 1 (levi podatek) vedno manjši od Ključa 2 (desni podatek).
2. Vsako vozlišče ima lahko dva ali tri sinove. Če vozlišče vsebuje samo Ključ1, vozlišče ne sme imeti več kot dva sinova.
3. Sinovi so nastavljeni tako, da so podatki v prvem poddrevesu manjši od Ključa 1, podatki v drugem poddrevesu so večji od Ključa 1 in manjši od Ključa 2 in podatki v zadnjem poddrevesu so večji od Ključa 2. Če vozlišče vsebuje samo Ključ 1, potem ima samo prvega in drugega sina.
4. Vsi listi so na istem nivoju.

5.4.2. ISKANJE V 2-3 DREVESIH

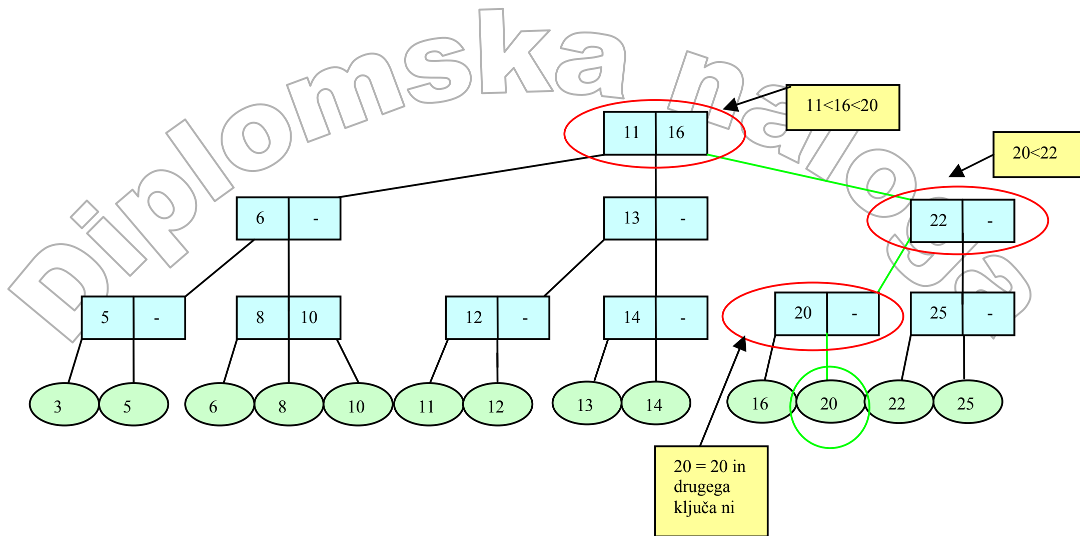
Preden si pogledamo vstavljanje in brisanje elementa v 2-3 drevesu, je smiselno pogledati kako poteka iskanje v 2-3 drevesu. Ker 2-3 drevo ni dvojiško drevo, je seveda iskanje nekoliko drugačno. Kot vemo, ima pri 2-3 drevesu vsako notranje vozlišče dva ali tri sinove. Algoritem za iskanje elementa v 2-3 drevesu je takle (iščemo element x):

1. če je x $< k_1$, se premaknemo k prvemu sinu
2. če je x $\geq k_1$ in ima vozlišče samo dva sinova, se premaknemo k drugemu sinu
3. če je x $\geq k_1$ in ima vozlišče tri sinove, se premaknemo k drugemu sinu, če je x $< k_2$ in k tretjemu sinu če je x $\geq k_2$

Primer:

Poglejmo kako poiščemo podatek 20 v 2-3 drevesu na sliki. Podatek 20 primerjamo s podatkom v korenu drevesa. Podatek 20 je večji od podatka 11. Vozlišče ima tri sinove, zato primerjamo podatek 20 še z drugim podatkom v korenu, to je 16. Ker je podatek 20 večji od podatka 11 in večji tudi od podatka 16, se premaknemo k tretjemu sinu in postopek ponovimo. Sedaj podatek primerjamo s podatkom 22. Ker je 20 manjši od 22, se premaknemo k prvemu sinu in postopek ponovimo. Primerjamo podatek 20 s podatkom 20 v vozlišču. Podatek 20 je enak ključu 20. Vozlišče ima samo dva sinova. Premaknemo se k drugemu sinu. Prišli smo do lista in našli smo podatek 20. Pot iskanja je označena zeleno.

Diplomska naloga
Fakulteta za matematiko in fiziko

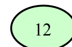


5.4.3. VSTAVLJANJE IN BRISANJE V 2-3 DREVESU

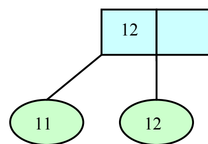
Vstavljanje in brisanje elementa iz 2-3 drevesa je nekoliko bolj zapleteno kot pri prejšnjih dveh tipih uravnoteženih dreves. Izkaže se da sta tudi ti dve operaciji na 2-3 drevesih učinkoviti, reda $O(\log n)$.

VSTAVLJANJE

Vstavljanje v prazno drevo ali v drevo z enim elementom je preprosto. V prvem primeru dobimo en list,



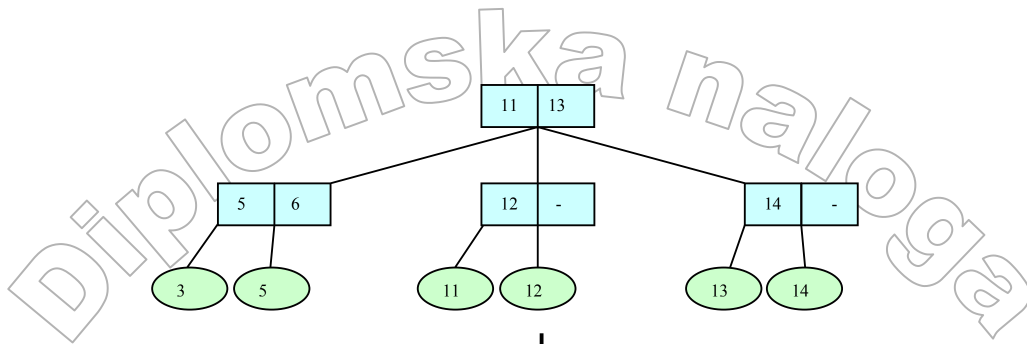
v drugem pa drevo z enim notranjim vozliščem in dvema sinovoma.

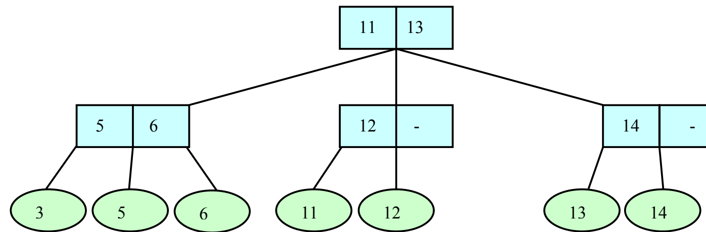


V splošnem pa najprej poiščemo očeta lista, kamor naj bi element vstavili. Možna sta dva primera:

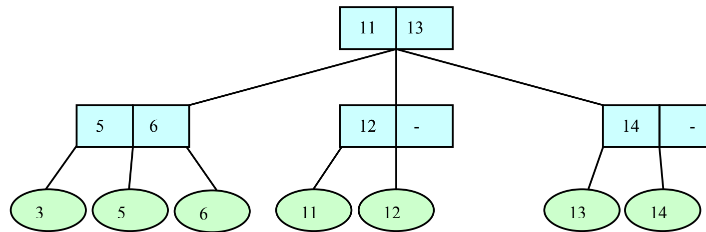
1. Če ima oče samo dva sina, vrinemo nov element na ustrezno mesto, tako da ima oče tri sinove. Ko ustrezno popravimo še ključa v očetu, je postopek končan. Ključev po drevesu navzgor ni treba spreminjati, saj algoritem iskanja elementa zagotavlja, da se najmanjši element v drugem in tretjem poddrevesu ne more spremeniti med vstavljanjem novega elementa.

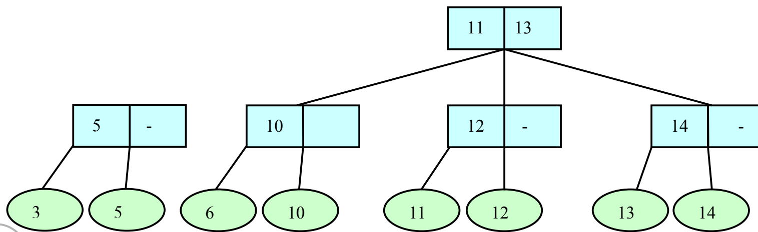
Diplomska naloga
Fakulteta za matematiko in fiziko





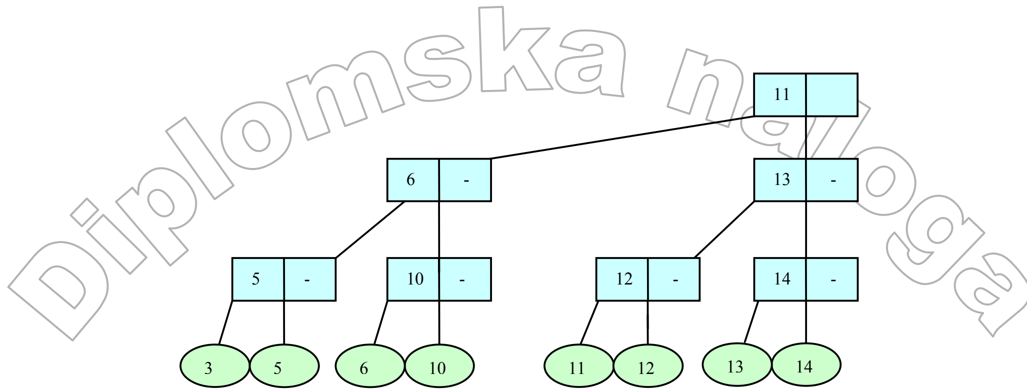
2. Če ima oče tri sinove, potem tvorimo novo notranje vozlišče(strica), tako da si oče in stric delita vsak po dva sinova ter ustrezno popravimo ključe.



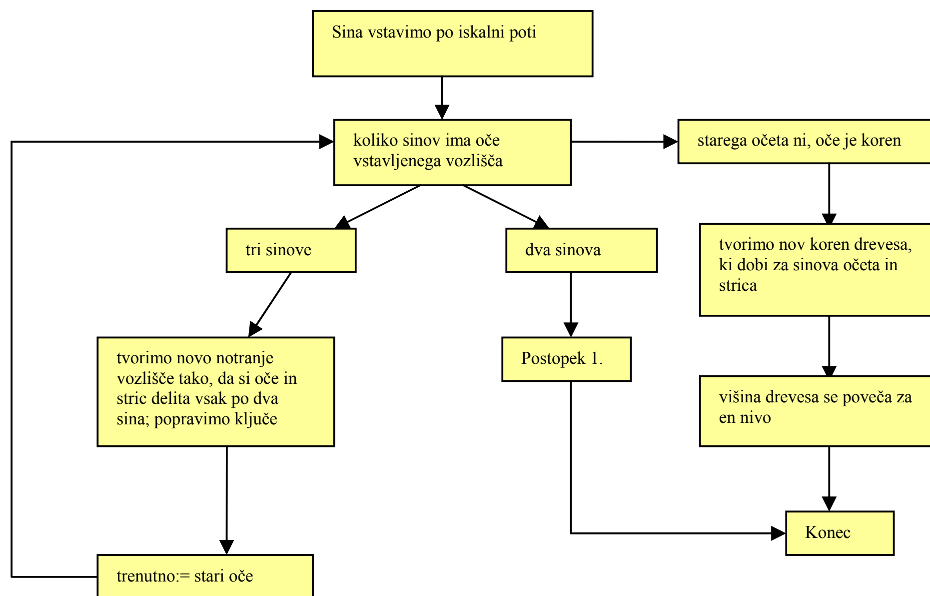


S tem smo dodali novega sina nivo višje in ponovimo postopek. Rekurzija se izteče bodisi, ko je imel oče le dva sinova (pri točki 1), ali pa v primeru, ko starega očeta ni; ko je oče koren drevesa. V tem primeru tvorimo nov koren drevesa (novega starega očeta), ki dobi za sinova očeta in strica. S tem se višina drevesa poveča za ena.

Diplomska naloga
Fakulteta za matematiko in fiziko

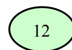


SHEMA VSTAVLJANJA ELEMENTA V 2-3 DREVO:

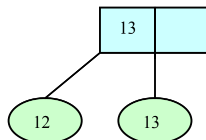


PRIMER VSTAVLJANJA:

Iz elementov 12, 13, 14, 5, 3, 11, 6, 10 in 8 bomo zgradili 2-3 drevo. Vstavimo 12:

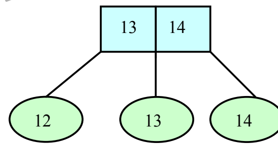


Nadaljujemo s 13:

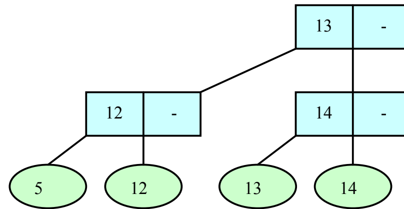


Vstavljamo naprej, sedaj 14. Vstavljamo list vozlišča, ki ima dva sinova, zato novi element vrinemo na ustrezno mesto. Oče ima sedaj tri sinove. Ustrezno nastavimo še ključ.

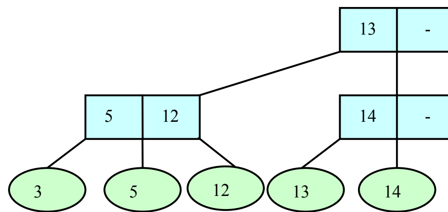
Diplomska naloga
Fakulteta za matematiko in fiziko



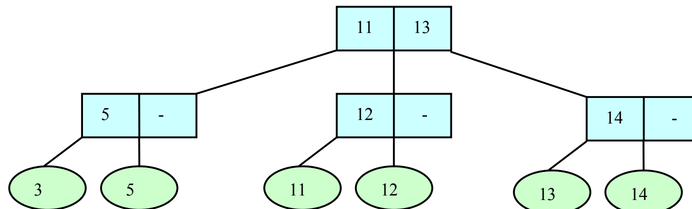
Vstavimo 5. Ker ima oče tri sinove, tvorimo novo notranje vozlišče, tako da si oče in stric delita vsak po dva sinova ter ustrezno popravimo ključe. Ker smo na vrhu drevesa, moramo narediti novo korensko vozlišče. Dobimo naslednje drevo:



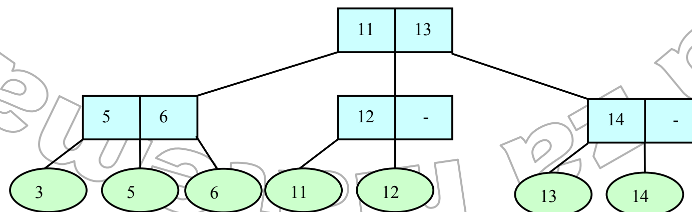
Nato vstavimo 3. Poiščemo očeta lista, kjer naj bi podatek vstavili. To je vozlišče 12 - . Ker ima samo dva sinova, vrinemo novi element na ustrezno mesto, tako da ima sedaj tri sinove. Ko ustrezno popravimo še ključe, je postopek končan.



Vstavimo 11. V tem primeru ima oče tri sinove, zato tvorimo novo notranje vozlišče, tako da si oče in stric delita vsak po dva sinova, ter ustrezno popravimo ključe. Zatem rekurzivno ponovimo dodajanje vozlišča en nivo višje; dodamo strica kot sina k staremu očetu. Postopek se izteče, ker je imel stari oče samo dva sinova.

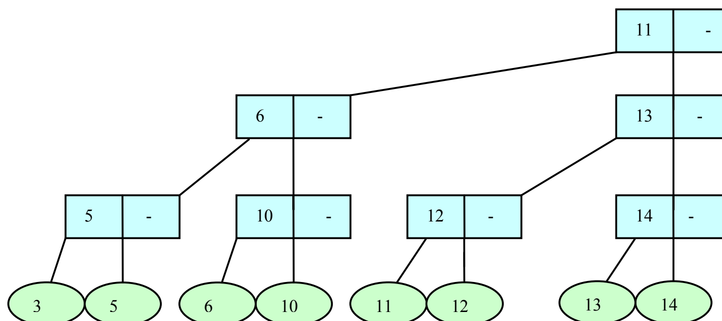


Nadaljujemo z vstavljanjem podatka 6. Ravnamo podobno kot dva primera višje in dobimo:

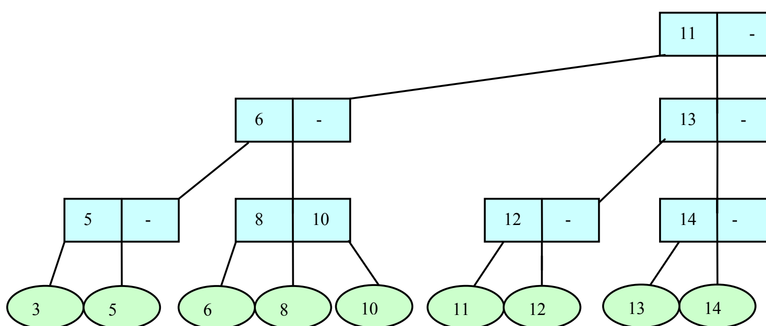


Diplomska naloga
Fakulteta za matematiko in fiziko

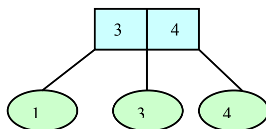
Vstavljajmo naprej in vstavimo 10. Primeren oče za vstavljanje lista s podatkom 10 že ima tri sinove, zato tvorimo novo notranje vozlišče tako, da si oče in stric delita delita vsak po dva sina ter ustrezno popravimo ključe. Zatem rekurzivno ponovimo dodajanje vozlišča en nivo višje; dodamo strica kot sina k staremu očetu. Postopek se izteče, ko starega očeta ni. Takrat je oče koren drevesa. Tvorimo nov koren drevesa, ki dobi za sinova očeta in strica. S tem se višina drevesa poveča za ena.



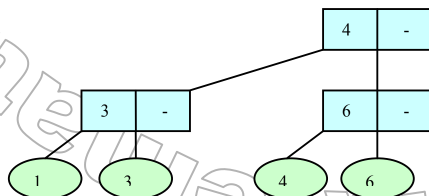
Vstavimo še podatek 8. Ker ima oče vozlišča, v katerega bomo vstavili nov podatek, samo dva sinova, vrinemo novi podatek na ustrezno mesto, tako da ima sedaj tri sinove. Ko ustrezno popravimo še ključe, je postopek končan.



Poglejmo si še primer za urejene podatke. Vstavljamo: 1, 3, 4, 6, 8, 9, 10, 11, 13, 15 in 16.
Vstavimo podatke 1, 3 in 4.

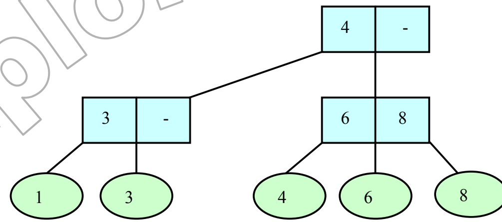


Ko vstavimo podatek 4, pridemo do položaja kjer ima oče tri sinove. Zato tvorimo novo notranje vozlišče (strica), tako da si oče in stric delita vsak po dva sinova, ter popravimo ključe. S tem smo dodali novega sina nivo višje in višina drevesa se poveča za ena.

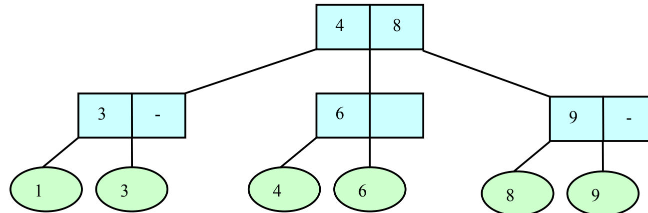


Diplomska naloga
Fakulteta za matematiko in fiziko

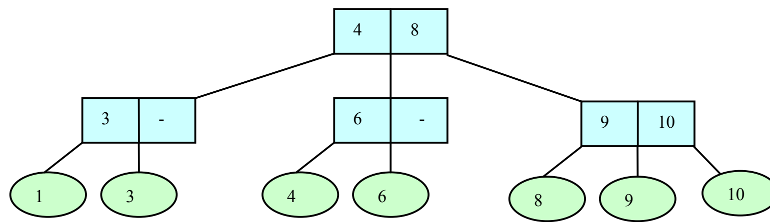
Nadaljujemo in vstavimo podatek 8. Oče ima dva sinova zato podatek samo vrinemo na ustrezno mesto.



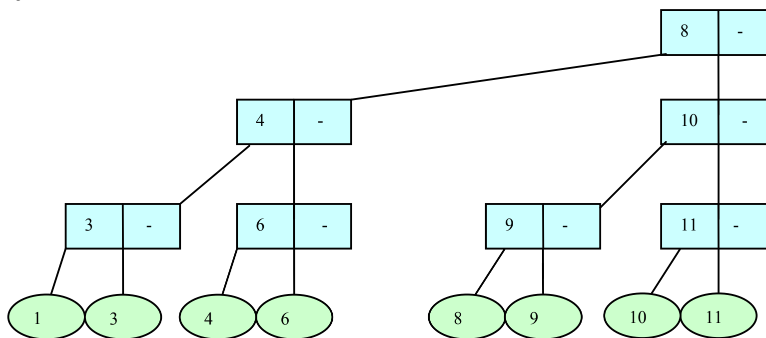
Nato vstavimo 9 in podobno kot že prej dobimo.



Vstavimo še podatek 10.



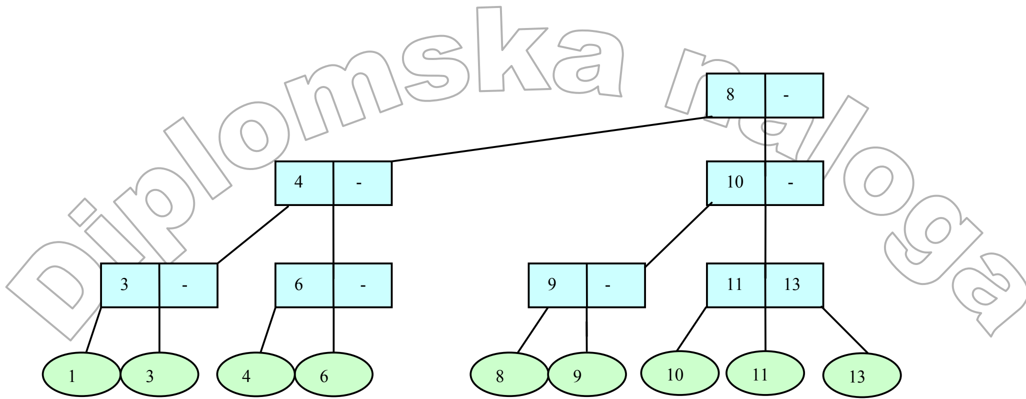
Pri vstavljanju podatka 11 ponovno naletimo na primer, ko ima oče tri sinove. Težavo odpravimo z delitvijo vozlišča. Najprej tvorimo notranje vozlišče, da si oče in sin delita vsak po dva sinova, ter ustrezno popravimo ključe. Rekurzivno ponovimo dodajanje vozlišča en nivo višje.



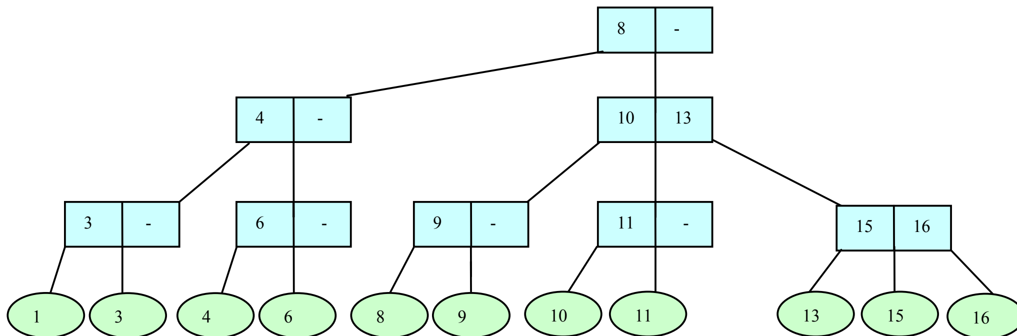
Vstavimo 13. Podatek v tem primeru samo vrinemo na pravo mesto.

Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga



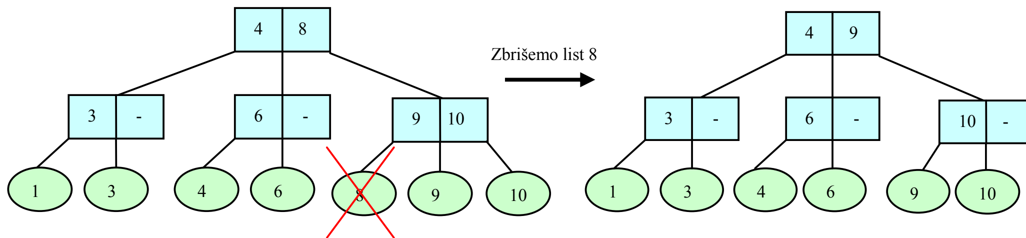
Vstavimo še 15 in 16 in dobimo končno 2-3 drevo.



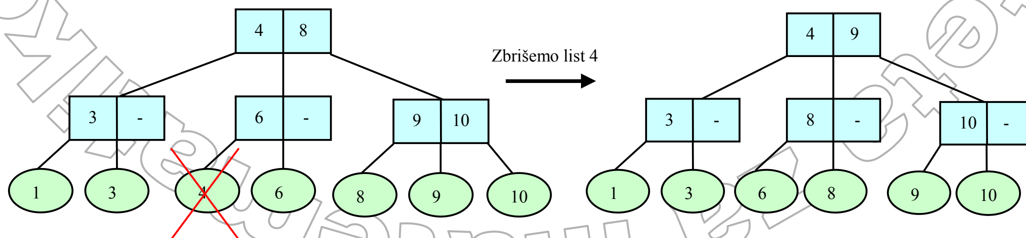
BRISANJE:

Ko želimo podatek zbrisati iz 2-3 drevesa, ga najprej poiščemo. Ko ga najdemo, zbrišemo ustrezni list. Nastopijo tri možnosti:

1. Če je oče zbrisanega lista imel prej tri sinove, premaknemo druga dva sina tako, da postaneta prvi in drugi z leve ter ustrezno popravimo ključe v očetu.

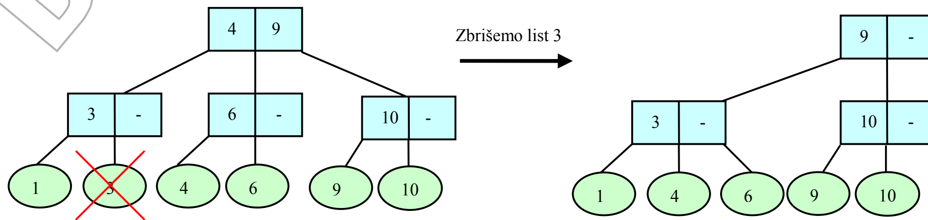


2. Če je imel oče zbrisanega lista prej samo dva sina in ima kateri od sosednjih očetovih bratov tri sinove, potem temu bratu vzamemo bližnjega sina in ga priključimo na ustrezno mesto k očetu. Pri tem je potrebno ustrezno spremeniti ključe. S tem je postopek končan.

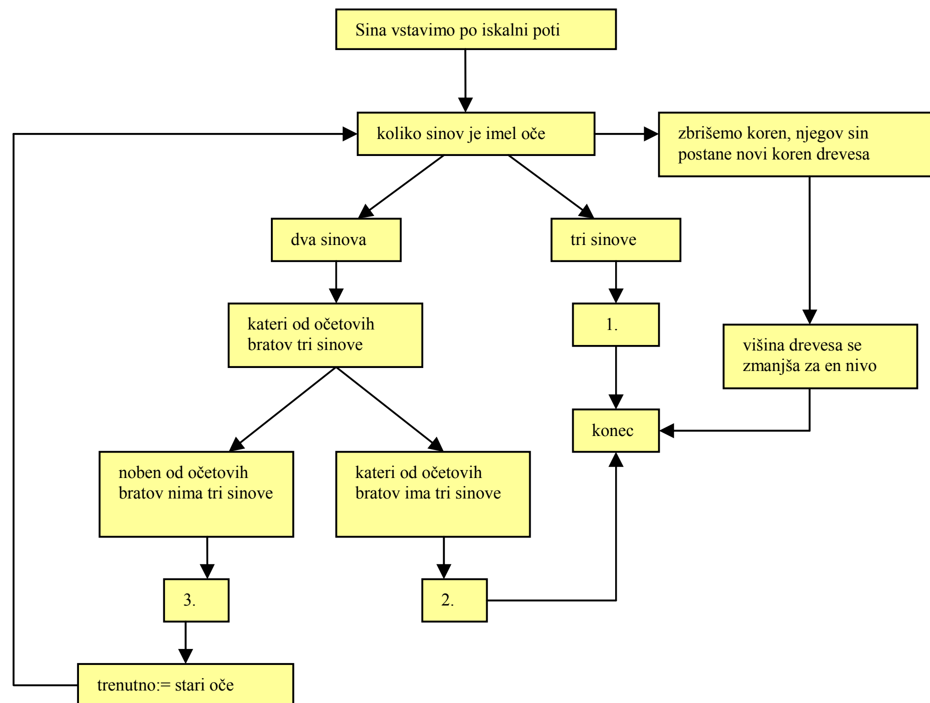


Fakulteta za matematiko in fiziko

3. Če noben od očetovih bratov nima treh sinov, priključimo preostalega očetovega sina enemu od bratov kot tretjega sina ter očeta zbrišemo. Sedaj celi postopek rekurzivno ponovimo pri starem očetu (očetovemu očetu, ki smo mu zbrisali enega od sinov). Postopek se konča bodisi pri 1. ali 2. točki ali pa nam ostane koren drevesa z enim samim sinom. V slednjem primeru zbrišemo koren in njegov sin postane novi koren drevesa. S tem se višina drevesa zmanjša za ena.



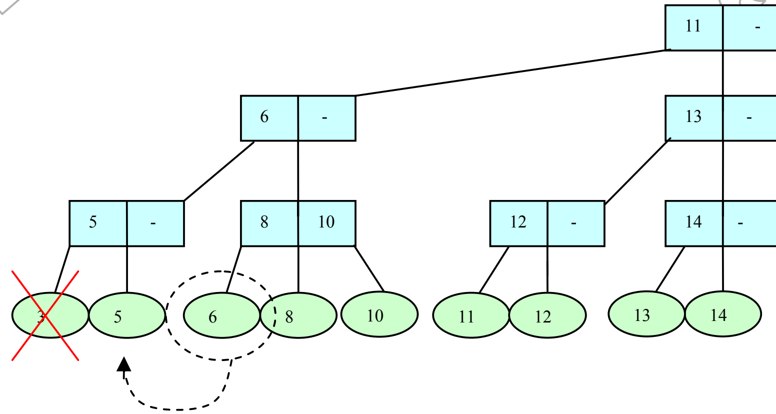
SHEMA BRISANJA ELEMENTA V 2-3 DREVESU:



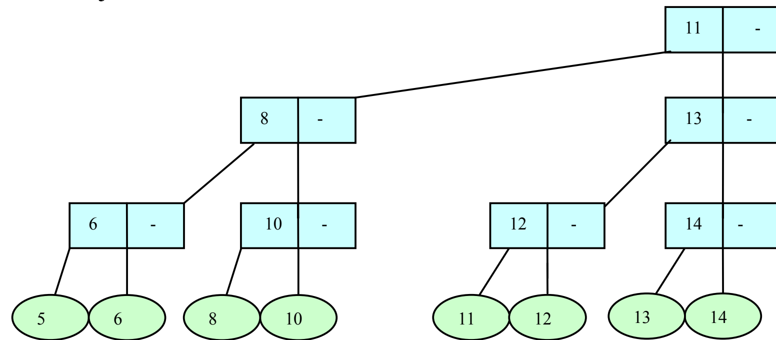
Diplomska naloga
Fakulteta za matematiko in fiziko

PRIMER BRISANJA ELEMENTA:

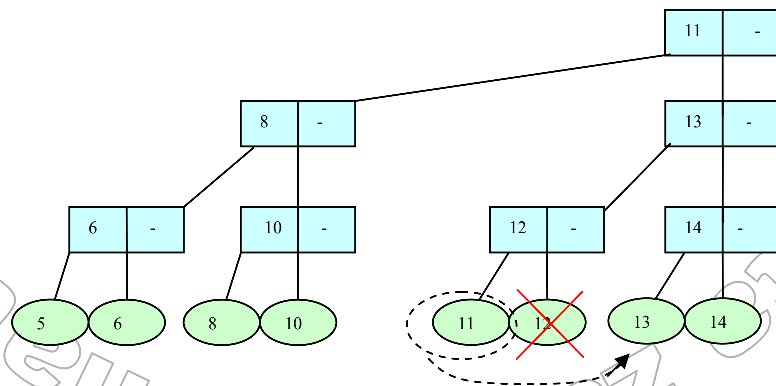
Vzemimo kar prej zgrajeno drevo in si poglejmo primer brisanja elementov.
Dano imamo torej naslednje 2-3 drevo:



Zbrisali bomo vozlišče s podatkom 3. Ker ima oče zbrisanega vozlišča dva sinova in ima sosednji (desni) brat tri sinove, vzamemo bližnjega sina in ga priključimo k očetu. Potem še popravimo ustrezne ključe.



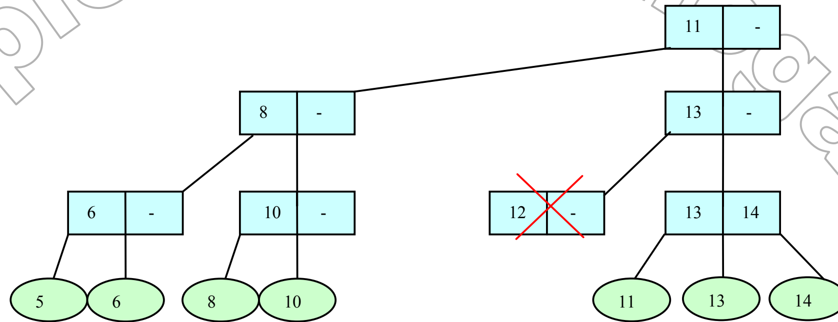
Zbrišimo vozlišče s podatkom 12:



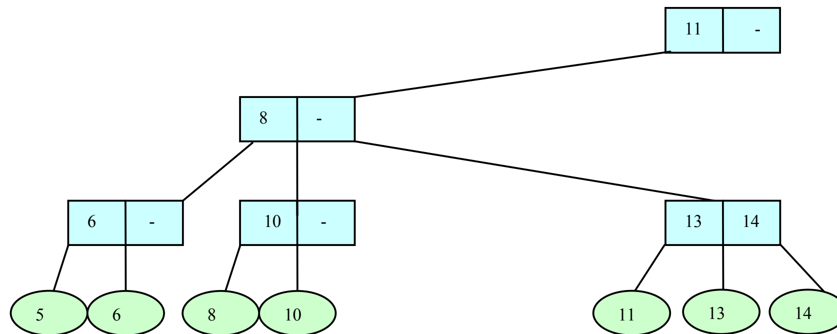
Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga
Fakulteta za matematiko in fiziko

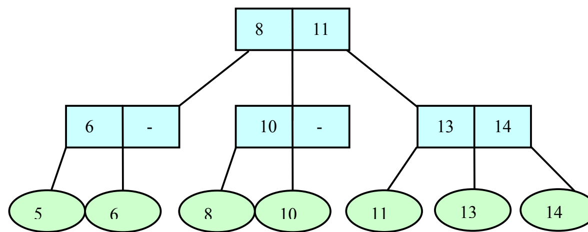
Oče zbrisanega vozlišča je imel dva sina. Noben od njegovih bratov (je samo eden) nima treh sinov, zato preostalega očetovega sina priključimo bratu ter očeta zbrišemo.



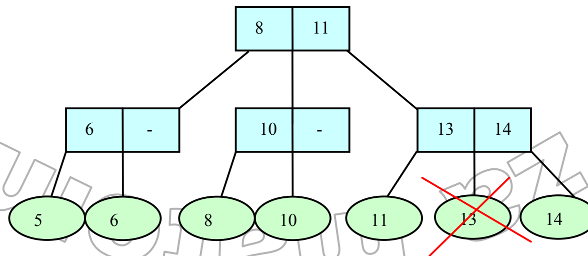
Sedaj celi postopek rekurzivno ponovimo pri starem očetu, to je pri vozlišču 13 - .
Noben od njegovi bratov nima treh sinov, zato priključimo preostalega sina k bratu kot tretjega sina, ter očeta zbrišemo.



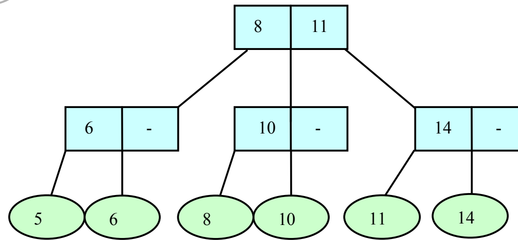
Rekurzivno smo prišli do korena drevesa z enim samim sinom, zato zbrišemo koren in njegov sin postane novi koren drevesa. Višina drevesa se je zmanjšala za ena. S tem je postopek končan in drevo je videti takole:



Zbrišimo še vozlišče s podatkom 13:



Pobrisali smo vozlišče, katerega oče je imel tri sinove. Preostala dva sina premaknemo tako, da postaneta prvi in drugi z leve ter ustrezno popravimo ključe. Dobimo drevo:



Zahtevnost operacij nad 2-3 drevesom:

- **Iskanje podatkov**
  2-3 drevo z n podatki ima višino med $1+\log_3 n$ in $1+\log_2 n$. Torej je dolžina poti od korena do lista reda $O(\log n)$. Zato je iskanje elementa v 2-3 drevesu vedno reda $O(\log(n))$.
- **Vstavljanje podatkov**
  Iskanje pravega mesta za vstavljanje v 2-3 drevo je reda $O(\log(n))$. Dodajanje podatke nas stane 1 operacijo, saj samo vstavimo podatek na pravo mesto, ki smo ga že prej našli. Preverjanje in preureditev pa nas stane $O(\log(n))$ operacij, ker moramo v najslabšem primeru preveriti vsa vozlišča na poti nazaj proti korenu. Teh vozlišč je toliko kot je nivojev, torej kot je višina drevesa ($O(\log(n))$) in ena preureditev nas stane eno operacijo.
- **Brisanje podatkov:**
  Prav tako kot vstavljanje je tudi brisanje elementa v 2-3 drevesu zapletena operacija in prav tako učinkovita - reda $O(\log(n))$.

Diplomska naloga
Fakulteta za matematiko in fiziko

Diplomska naloga

6. ZAKLJUČEK

V diplomskem delu sem obravnavala uravnotežena drevesa. Najprej sem obravnavala drevesa na splošno, nato dvojiška drevesa in posebno vrsto dvojiških dreves – iskalna dvojiška drevesa. Pri analizi lastnosti iskalnih dvojiških dreves se izkaže, da dvojiška iskalna drevesa lahko dokaj hitro pridobijo na razliki v višini poddreves in se v skrajnem primeru celo izrodijo – postanejo podobna linearnim seznamom. To poglavitno vpliva na časovno zahtevnost algoritmov nad iskalnimi dvojiškimi drevesi, saj je le ta odvisna od višine drevesa. Zato želimo pri danem številu vozlišč kar se da omejiti višino drevesa. Preprečiti želimo, da bi se drevo izrodilo. Zato je pomembno vzdrževati dvojiško drevo kot uravnoteženo drevo. V ta namen vpeljemo uravnotežena drevesa, kjer pri vsakem vstavljanju in brisanju podatkov sproti preverjamo zgradbo drevesa. To pomeni, da sproti kontroliramo razliko med višinama levega in desnega poddrevesa. V diplomski nalogi je predstavljeno nekaj tipov uravnoteženih dreves. Pri tem je poudarek na operacijah *vstavi* in *briši*, ker le ti dve prispevata svoj delež k preoblikovanju zgradbe drevesa. Pri preoblikovanju ne gre brez rotacij, vsaj pri AVL in rdeče črnih drevesih ne, medtem ko pri 2-3 drevesih izvajamo združevanje in razdruževanje vozlišč. Pri vseh treh oblikah dreves je časovna zahtevnost operacij enaka $O(\log n)$, kjer je n število podatkov, ki jih hranimo v drevesu.

Vse tri opisane podatkovne strukture imajo nekaj skupnih točk. Vsi trije tipi dreves imajo skupno hitro iskanje, vstavljanje in brisanje, ter to, da drevo po izvedbi teh rotacij ostane uravnoteženo. Vendar pa so algoritmi zato malce bolj zahtevni in kompleksni.
Kot strukturi sta si predvsem podobni AVL drevo in rdeče-črno drevo. Obe vrsti dreves sta posebna primera dvojiških iskalnih dreves. Pri obeh so podatki shranjeni v vsakem vozlišču. Obe podatkovni strukturi združuje tudi dejstvo, da uravnoteženost doseženta s pomočjo rotacij. Te so za obe vrsti dreves enake, le pravila, ki do njih privedejo, se razlikujejo.
Podatkovna struktura 2-3 se kar precej razlikuje od drugih dveh. Poleg tega, da to ni več dvojiško drevo, so pri 2-3 drevesu podatki shranjeni le v listih, v ostalih vozliščih pa je le informacija, kako do podatkov pridemo.
Na prvi pogled je torej 2-3 drevo precej drugačno drevo kot ostali dve vrsti. Vendar se izkaže, da sta si 2-3 drevo in rdeče-črno drevo precej podobni. Prvo se lahko transformira v drugo z uporabo nekaj preprostih pravil. Celo operacije, potrebne za uravnoteževanje, so enakovredne. Zgodovinsko je bilo razvito najprej 2-3-4 drevo, sledilo je 2-3 drevo, šele kasneje pa rdeče-črno drevo, izhajajoče iz 2-3-4 drevesa. Vsa je vpeljal Bayer leta 1972. Takrat so bila AVL drevesa »stara« že 10 let, saj so bila prvič predstavljena leta 1962. Tudi AVL drevo lahko z uporabo nekaj preprostih pravil preoblikujemo v rdeče-črno drevo.

Fakulteta za matematiko in fiziko

LITERATURA:

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to algorithms*, second edition, MIT Press, 2001

I. Kononenko, *Načrtovanje podatkovnih struktur in algoritmov*, Založba FE in FRI, 1996, Ljubljana.

J. Kozak, *Podatkovne strukture in algoritmi*, Društvo matematikov, fizikov in astronomov, 1986, Ljubljana.

R. Lafore, *Data structures and algoritms in Java*, The Waite group, 1998.

R. Sedgewick, *Algorithms in Java (Parts 1 - 4)*, Addison-Wesley, 2003

N. Wirth, *Računalniško programiranje I. in II.*, Društvo matematikov, fizikov in astronomov, 1983.

Diplomska naloga
Fakulteta za matematiko in fiziko