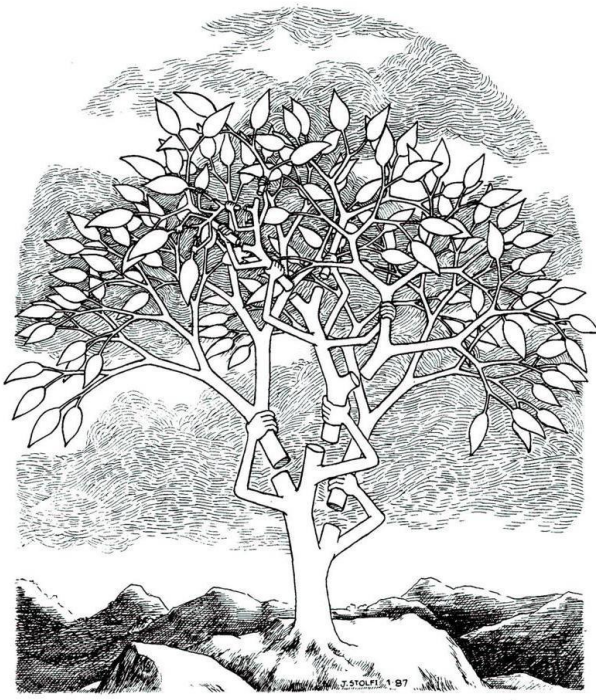
LOMLJENA DRVESA



Hana Kranjec Kelbel
Računalništvo1
Praktična matematika, Fakultete za matematiko in fiziko

KAZALO VSEBINE

DEFINICIJA 3

Zgodovinski del 4

Zakaj ravno lomljenje dreves? 5

Glavna metoda: lomljenje dreves 6

Tehnika preoblikovanja : ROTACIJE 7

VRSTE ROTACIJ 7

Tipi rotacij 7

ČASOVNA ZAHTEVNOST 7

ROBNA PRIMERA 8

SPLOŠNA PRIMERA 11

IMPLEMENTACIJA 17

OPERACIJE 18

1. ISKANJE ELEMENTA V DREVESU 18
2. VSTAVLJANJE ELEMENTA V DREVO 26
3. BRISANJE ELEMENTA V DREVESU 30

VIRI 37

DEFINICIJA

Lomljena drevesa so dvojiška iskalna drevesa, ki se lomijo oziroma preoblikujejo ob vsakem posegu v drevo.

Uporabljamo tri osnovne operacije na drevesu, kot so vstavljanje, brisanje in iskanje elementa. Nato pa izvedemo zaporedje rotacij, kjer obravnavan element postavimo v koren drevesa. Zaporedju rotacij pravimo lomljenje drevesa.

- Vstavljanje elementa v drevo
  Element vstavimo v list drevesa. Nato uporabimo zaporedje rotacij, da vstavljen element oziroma list postane koren drevesa.

- Brisanje elementa v drevesu
  Element, katerega želimo izbrisati z zaporedjem rotacij postane koren drevesa. Nato ga izbrišemo iz drevesa.

- Iskanje elementa v drevesu
  Uporabimo zaporedje rotacij, da iskan element postane koren drevesa.

Dvojiško drevo je drevesna podatkovna struktura, kjer ima vsako vozlišče največ dva otroka. Levi in desni sin.

Iskalno drevo je dvojiško drevo, za katera velja, da so vsi elementi v levem poddrevesu manjši od korena in v desnem poddrevesu večji od korena. Levo in desno poddrevo je iskalno poddrevo.

Zgodovinski del

Podatkovno strukturo lomljenja drevesa sta odkrila profesorja Daniel Dominic Sleator in Robert Endre Tarjan leta 1985.
Daniel Dominic Sleator je profesor na Univerzi Pittshburg, US.
Robert Endre Tarjan predava na univerzi Princeton . Odkril je nekaj grafičnih algoritmov. Med drugim tudi Fibonaccijeve kopice.
Leta 1999 sta osvojila nagrado The ACM Paris Kanellakis Award za odkritje podatkovne strukture Lomljena drevesa.
Skupaj sta odkrila več algoritmov podatkovnih struktur kot so povezovanje in brisanje v drevesih ter kopice.

ZAKAJ RAVNO LOMLJENJE DREVES?

Ker pri posegih v drevo pogosto pride do izrojevanja dreves, saj se lahko višini levega in desnega poddrevesa razlikujeta. Z lomljenjem dreves pa ravno to preprečimo. Nikoli ne bo prišlo do izrojenega drevesa.

Prav tako pa bo vedno obravnavano vozlišče v korenu drevesa.

Glavna metoda: lomljenje dreves

Lomljenje drevesa je metoda, s katero pomaknemo trenutni element proti korenu drevesa. S tem želimo doseči, da je drevo bolj uravnoteženo ter da je obravnavano vozlišče X bližje korenu drevesa. To storimo z zaporedjem rotacij tako, da premikamo opazovan element proti korenu.

Tehnika preoblikovanja : ROTACIJE

Za izvedbo rotacij na lomljenih drevesih potrebujemo poleg podatka o obeh sinovih tudi podatek o očetu in starem očetu. Rotacije so postopki, ki zamenjajo vlogo sina in očeta. Rotacije uporabljamo pri vsakem posegu v drevo.
S pomočjo rotacij željen element iz drevesa premaknemo v koren drevesa.

VRSTE ROTACIJ

- Enojna rotacija
  Enojne rotacije so enostavnejše. Uporabimo jih ko je oče vozlišča, ki ga rotiramo kar koren drevesa.
  V večini primerov enojne rotacije ne zadoščajo pogoju, da drevo ne sme biti izrojeno.

- Dvojna rotacija
  Dvojne rotacije so kombinacije dveh enojnih rotacij.

TIPI ROTACIJ

- Enojna rotacija
  Poznamo leve in desne rotacije. Leva rotacija je zrcalna desni rotaciji.

- Dvojna rotacija
  Dvojna rotacija je lahko sestavljena iz dveh levih ali dveh desnih rotacij. Uporabimo pa lahko tudi kombinirano dvojno rotacijo.
  Torej poznamo levo-desno, desno-levo in desno-desno ali levo-levo dvojno rotacijo.

ČASOVNA ZAHTEVNOST

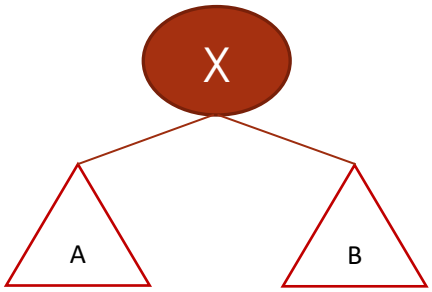
- Enojno rotacijo končamo v konstantnem času 0(1) kar v korenu.
- Ker lomimo drevo od korena do obravnavanega elementa je časovna zahtevnost sorazmerna z dolžino poti. Menim, da je pričakovana časovna zahtevnost 0(log(n)).

Zakaj logaritemska časovna zahtevnost?

Saj se v iskalnem drevesu premikamo po levem in desnem poddrevesu in ne potrebujemo preveriti vseh elementov v drevesu.

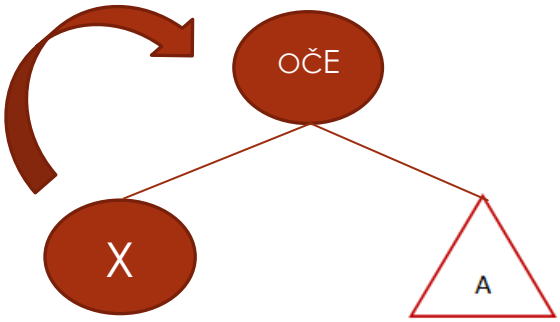
ROBNA PRIMERA

1. *Robni primer* : Ustrezen element je v koren, rotacij ne potrebujemo.

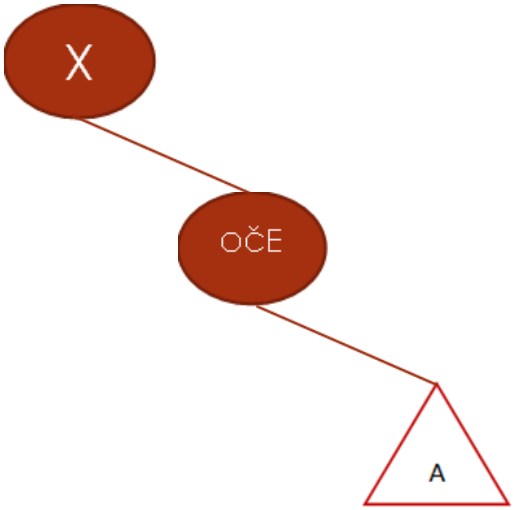


2. *Robni primer* : Oče obravnavanega vozlišča je koren drevesa. Zato potrebujemo le enojno levo ali desno rotacijo, da obravnavani element premaknemo v koren drevesa.

2.1. Robni primer, kjer je obravnavan element levi sin očeta. Potrebujemo enojno desno rotacijo.



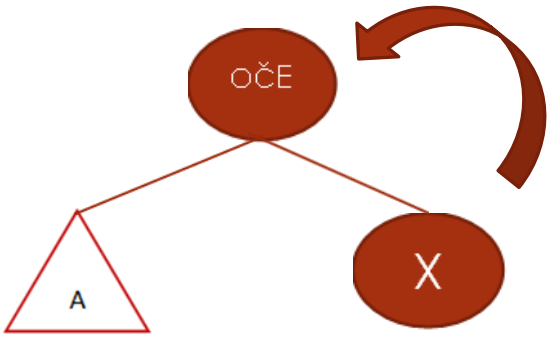
Desno rotacijo naredimo tako, da premaknemo vozlišče X na mesto korena oziroma očeta. Oče postane desni sin korena. Desno vozlišče pa očetovo desno poddrevo.



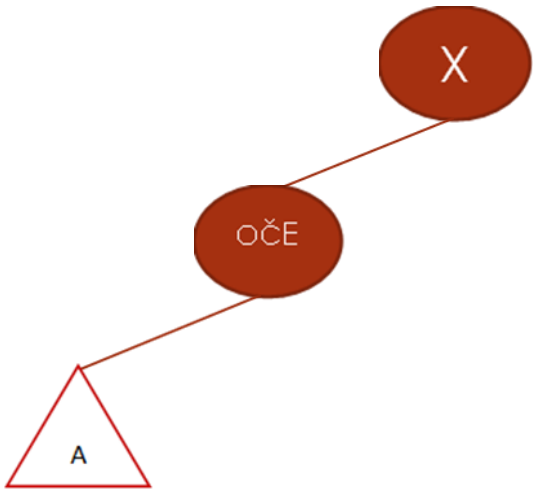
Preverimo ali je po izvedeni rotaciji drevo še vedno iskalno.

Vozlišče X je levi sin korena, torej je manjše od očeta. Pri desni rotaciji oče postane desni sin korena, kar je pravilno saj so elementi v desnem poddrevesu večji od korena. Desno poddrevo očeta ostane desno poddrevo, kar je tudi pravilno, saj so tudi tu elementi večji od obravnavanega elementa X in od očeta.

2.2. Robni primer, kjer je obravnavan element levi desni očeta. Potrebujemo enojno levo rotacijo.



Levo rotacijo naredimo tako, da premaknemo vozlišče X na mesto korena oziroma očeta. Oče postane levi sin korena. Desno vozlišče pa očetovo desno poddrevo.



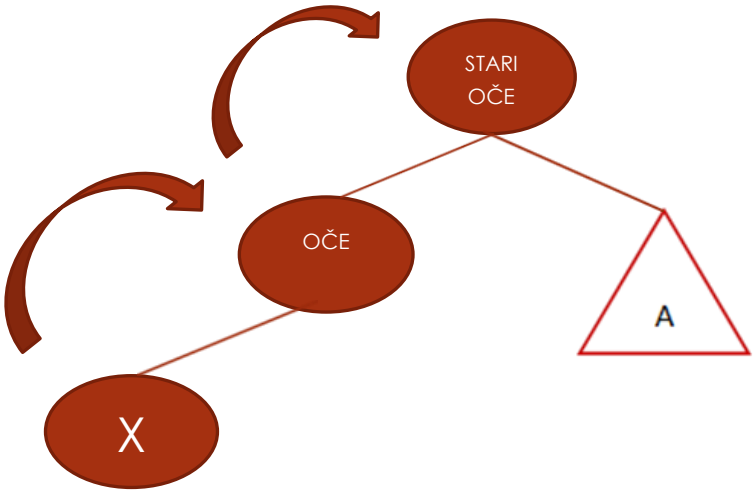
Preverimo ali je po izvedeni rotaciji drevo še vedno iskalno.

Vozlišče X je desni sin korena, torej je X največji element v tem drevesu. Pri levi rotaciji X postane koren, njegov levi sin postane oče, saj je manjši element. Levo poddrevo ostane levo poddrevo očeta, saj levo poddrevo vsebuje manjše elemente.

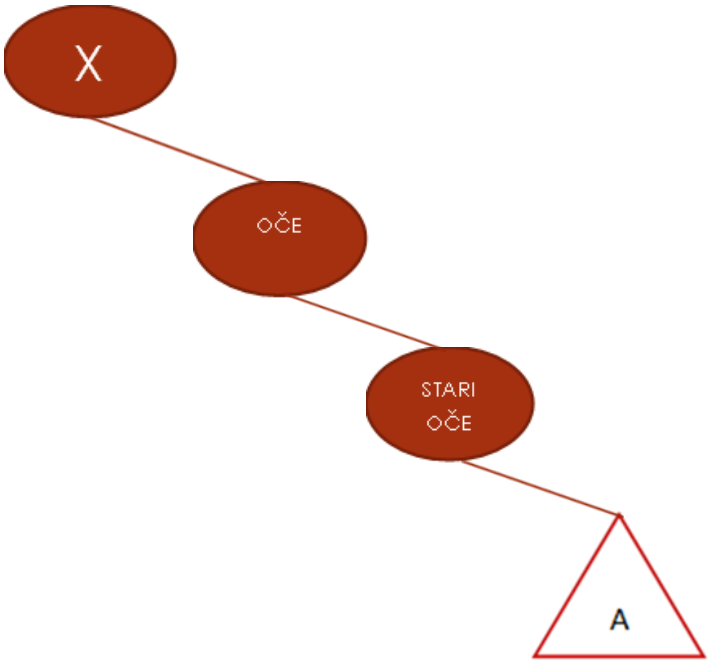
SPLOŠNA PRIMERA

1. *Splošni primer* : Dvojno rotacije uporabimo, če sta obravnavano vozlišče in njegov oče leva (desna) sinova.

1.1. Splošni primer, kjer je obravnavano vozlišče levi sin očeta. Oče pa je prav tako levi sin starega očeta oziroma korena. Torej uporabimo dvojno desno-desno rotacijo, da premaknemo obravnavano vozlišče v koren drevesa.



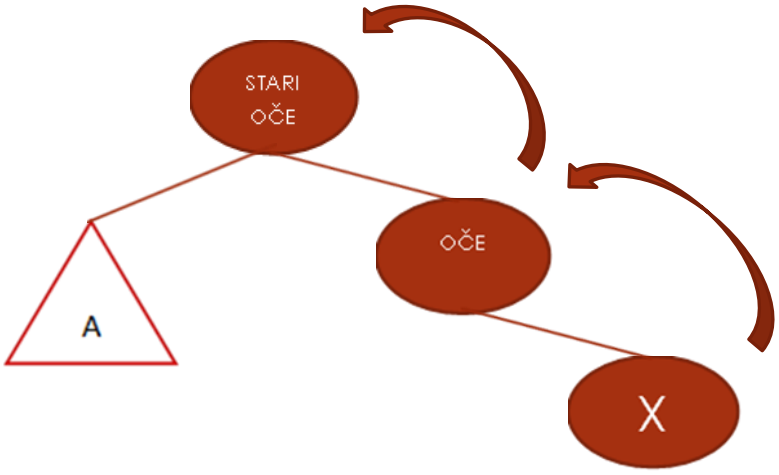
Dvojno desno-desno rotacijo storimo tako, da najprej premaknemo obravnavan element X v koren levega poddrevesa, oče postane koren drevesa. Stari oče postane koren desnega poddrevesa. Desno poddrevo A pa ostane desno poddrevo starega očeta. Naredili smo prvo desno rotacijo, nadaljujemo še z eno desno rotacijo. Obravnavan element X postane koren drevesa, oče koren desnega poddrevesa. Njegovo desno poddrevo postane stari oče. Poddrevo A pa ostane desno poddrevo starega očeta.



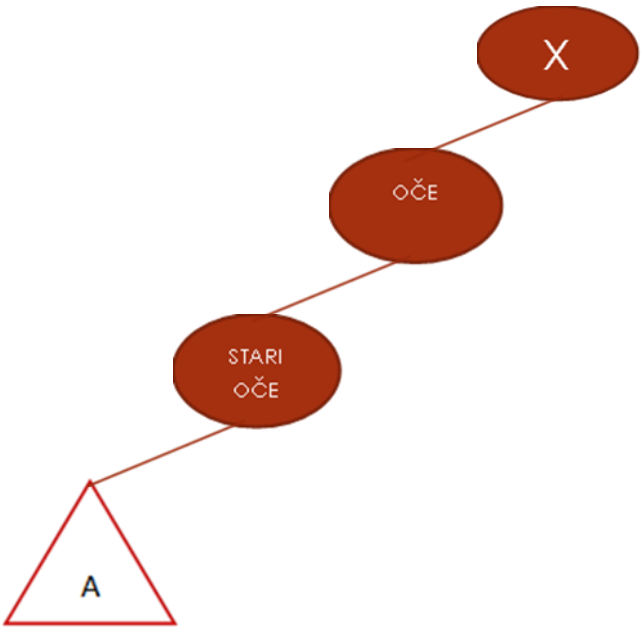
Preverimo ali je po izvedeni rotaciji drevo še vedno iskalno.

Obravnavan element X je najmanjši element v drevesu, sedaj je v korenu. Njegov oče in stari oče sta postala desna sinova, saj sta večja od korena. Desno poddrevo A ostane desno poddrevo starega očeta je prav tako pravilo, ker so notri največji element drevesa. Drevo je ostalo iskalno drevo.

1.2. Splošni primer, kjer je obravnavano vozlišče levi desni očeta. Oče pa je prav tako desni sin starega očeta oziroma korena. Torej uporabimo dvojno levo-levo rotacijo, da premaknemo obravnavano vozlišče v koren drevesa



Obravnavano vozlišče premaknemo v koren tako, da naredimo dvojno levo-levo rotacijo tako, da najprej z levo rotacijo premaknemo obravnavano vozlišče X v koren desnega poddrevesa. Oče postane koren drevesa, stari oče koren levega poddrevesa. Poddrevo A pa ostane levo poddrevo starega očeta. Nato naredimo še eno levo rotacijo tako, da obravnavano vozlišče postane koren drevesa. Oče njegovo levo poddrevo, stari oče postane levo poddrevo očeta. Poddrevo A pa ostane levo poddrevo starega očeta.

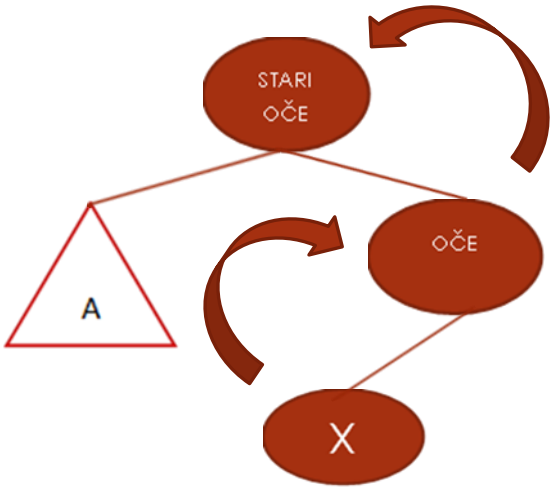


Preverimo ali je po izvedeni rotaciji drevo še vedno iskalno.

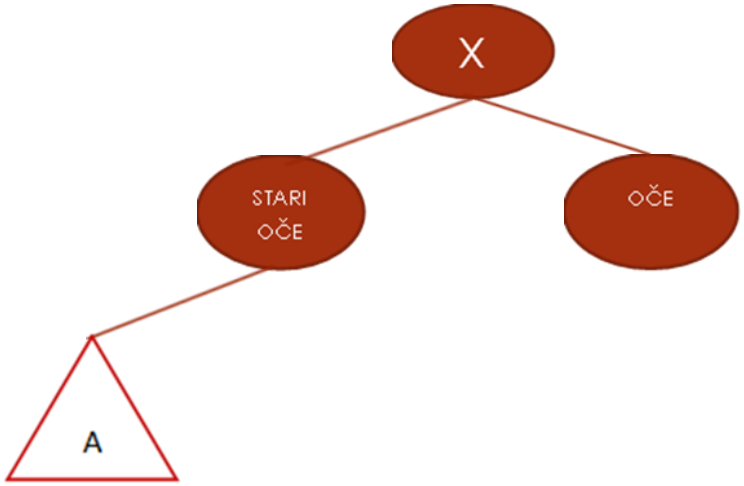
Obravnavan element X je največji element v drevesu, sedaj je v korenu. Torej bodo preostala vozlišča drevesa v njegovem levem poddrevesu. Oče je bil koren desnega poddrevesa, sedaj je koren levega poddrevesa. Ta del je prav tako pravilen, saj se je iskalnost ohranila. Poddrevo A je ostalo levo poddrevo starega očeta, saj so notri manjši elementi.

2. *Splošni primer* : Dvojno rotacijo uporabimo, če je obravnavano vozlišče levi sin očeta. Oče je pa desni sin svojega očeta. (in obratno)

2.1. Splošni primer, kjer je obravnavano vozlišče levi sin očeta. Oče pa je desni sin starega očeta oziroma korena. Torej uporabimo dvojno desno-leva rotacijo, da premaknemo obravnavano vozlišče v koren drevesa.



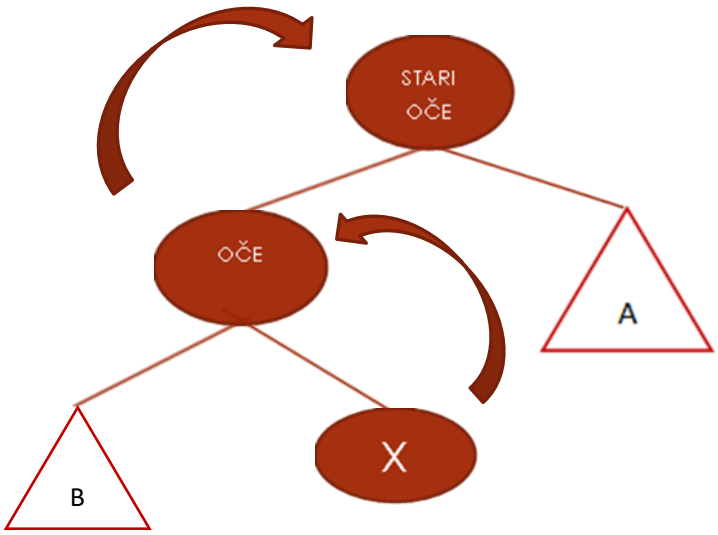
Obravnavano vozlišče X premaknemo v koren z dvojno desno-levo rotacijo tako, da najprej naredimo desno rotacijo in X postane koren desnega poddrevesa. Oče pa postane desno poddrevo vozlišča X. Stari oče in poddrevo A sta nespremenjena. Sedaj naredimo še levo rotacijo in X postane koren drevesa, oče njegovo desno poddrevo. Stari oče je sedaj koren levega poddrevesa in poddrevo A ostane levo poddrevo starega očeta.



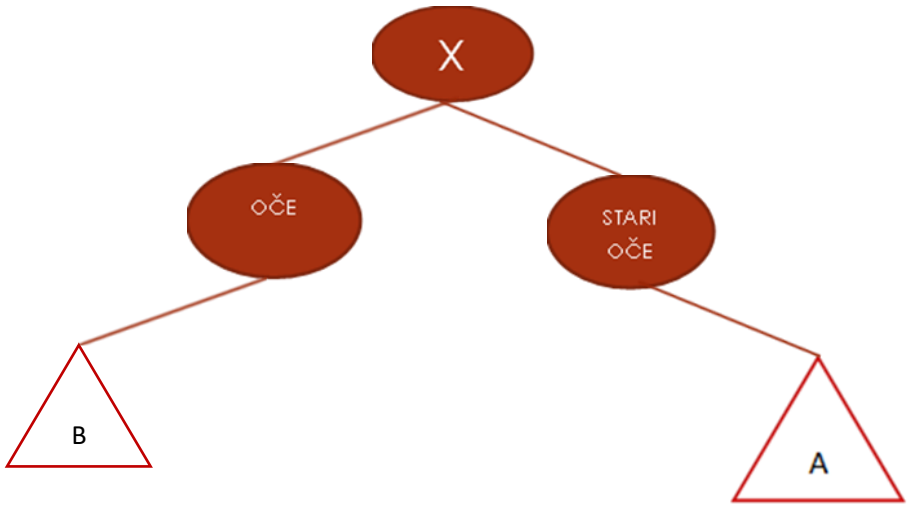
Preverimo ali je po izvedeni rotaciji drevo še vedno iskalno.

Vozlišče X se pred dvojno rotacijo nahajava v desnem poddrevesu. Sedaj ga najdemo v korenu drevesu, zato postane stari oče njegov levi sin, saj je manjši. Levo poddrevo A vsebuje manjše elemente drevesa, zato ostane levo poddrevo starega očeta. Vozlišče X je bil levi sin očeta pred rotacijo, sedaj ko je vozlišče X koren je oče njegov desni sin. Saj se v vozlišču oče nahaja večja vrednost, kakor je lahko v vozlišču X.

2.2. Splošni primer, kjer je obravnavano vozlišče desni sin očeta. Oče pa je levi sin starega očeta oziroma korena. Torej uporabimo dvojno levo-desno rotacijo, da premaknemo obravnavano vozlišče v koren drevesa.



Obravnavano vozlišče premaknemo v koren drevesa tako, da naredimo dvojno levo-desno rotacijo. Najprej z levo rotacijo premaknemo X v koren levega poddrevesa. S tem pa očeta premaknemo v levo poddrevo vozlišča X. Poddrevo B pa ostane levi sin očeta. Stari oče in poddrevo A ostaneta nespremenjena. Nato naredimo še desno rotacijo in vozlišče X postane koren drevesa, stari oče pa njegov desni sin. Poddrevo A ostane desno poddrevo starega očeta. Oče in poddrevo B ostaneta leva sinova.



Preverimo ali je po izvedeni rotaciji drevo še vedno iskalno.

Obravnavano vozlišče X je sedaj v korenu drevesa. Oče postane levi sin, saj je X pred rotacijo bil desni sin. Torej je vrednost očeta manjša. Stari oče postane desni sin, saj je stari oče bil koren drevesa. Torej je stari oče večji od vrednosti v vozlišču X. Desno poddrevo po dvojni rotaciji ostane desno poddrevo starega očeta, prav tako levo poddrevo B ostane levo poddrevo očeta. Torej drevo je po dvojni levo-desni rotaciji ostalo iskalno.

```
Lomljenje_drevesa(x)  # obravnavan element  želimo premakniti v koren drevesa
    while x.oče NOT NULL #izvajamo zanko dokler x ne postane koren drevesa oz.
dokler ima očeta
        if x.oče.levi == NULL  #če je stari oče X element NULL oz. element X je
koren levega ali desnega poddrevesa
            if x == x.oče.oče.levi  # če je X element levo poddrevo očeta
                // desna rotacija
                desna-rotacija(x.oče)
            else
                // leva rotacija  # če je pa X element desno podrevo naredimo
levo rotacijo
                leva-rotacija(x.oče)
        elseif x==x.oče.levi and x.oče == x.oče.oče.levi #če je X očetovo levo
podrevo in prav tako levo poddrevo starega očeta
                // desno-desna rotacija
                desna-rotacija(x.oče)
                desna-rotacija(x.oče)
        elseif x==x.oče.desni and x.oče == x.oče.oče.desni #če je X očetovo in
staro-očetovo desno poddrevo
                // levo-levo rotacijo
                leva-rotacija(x.oče)
                leva-rotacija(x.oče)
         elseif x==x.oče.desni and x.oče == x.oče.oče.levi  #če je X očetovo levo
desno poddrevo in levo poddrevo starega očeta
                // desno-leva rotacija
                leva-rotacija(x.oče)
                desna-rotacija(x.oče)
        else
                // levo-desna rotacija   # X je očetovo levo podrevo in desno
poddrevo starega očeta
                desna-rotacija(x.oče)
                leva-rotacija(x.oče)
...
```

OPERACIJE

1. ISKANJE ELEMENTA V DREVESU

Zanima nas ali je nek element v drevesu. Iskanje začnemo v korenu drevesa in nato se pomikamo proti listom drevesa.

Pri iskanju elementa v drevesu naletimo na tri možnosti:

- Prazno drevo
  Drevo je prazno, torej iskanega elementa ni v drevesu.

- Iskan element je v korenu drevesa
  Iskan element je v korenu drevesa, zato smo ob pregledu končali z iskanjem.

- Iskan element je nekje v drevesu
  Rekurzivno poiščemo element v drevesu.

- Iskanega elementa ni v drevesu
  Rekurzivno pogledamo drevo, ko pridemo do zadnjega lista končamo z iskanjem in si zapomnimo zadnji element.

Sedaj ko smo bodisi našli iskani element, bodisi našli zadnji element na poti iskanja elementa. Začnemo z lomljenjem drevesa.

Saj smo z iskanjem elementa posegali v drevo, zato moramo sedaj z zaporedjem rotacij iskani element ali zadnji element na poti premakniti v koren drevesa. S tem bomo preprečili izrojenost našega drevesa.

Kadar je iskan element v korenu drevesa, lomljenja ne potrebujemo. Saj je iskani element že v korenu drevesa.

ČASOVNA ZAHTEVNOST

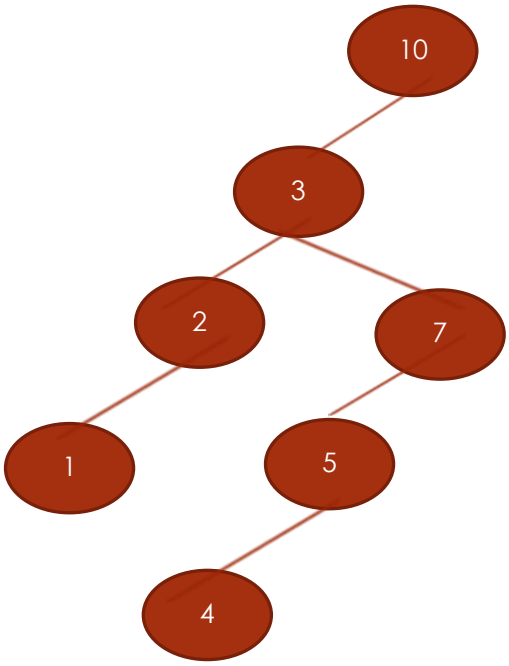
- Iskanje elementa v drevesu je sorazmerno z višino drevesa. Pričakovana časovna zahtevnost je 0(log(n)). Nato pa sledi lomljenje, kar že vemo, da je pričakovana časovna zahtevnost 0(log(n)).

IMPLEMENTACIJA

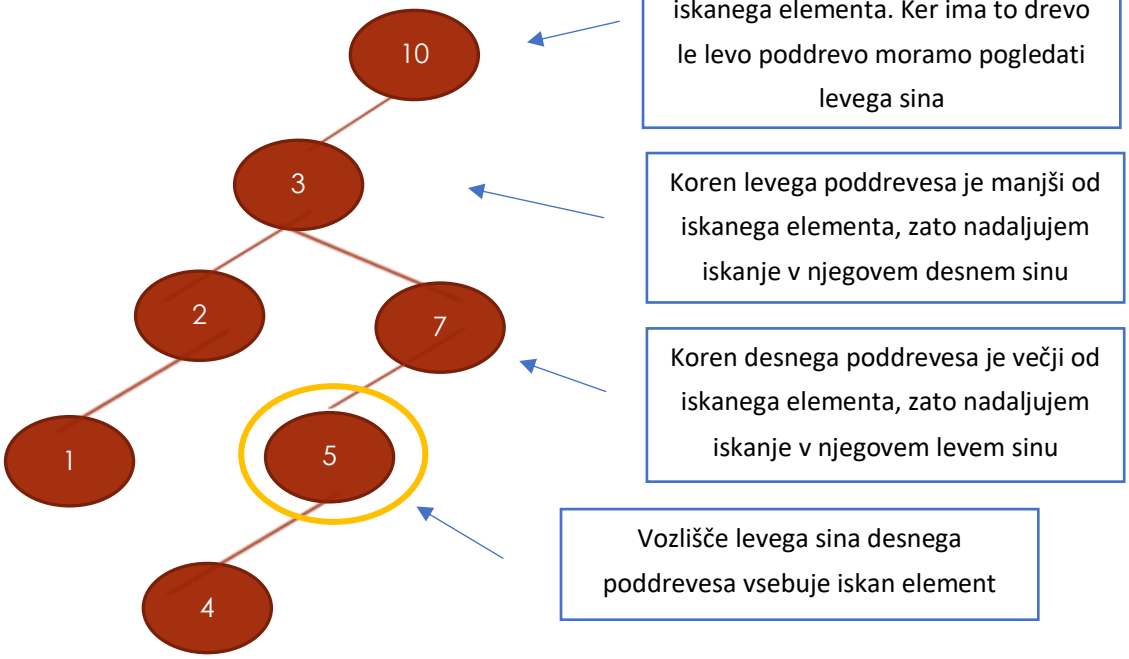
```
1.korak : poiščemo element v drevesu
2. korak : prestavimo x v koren drevesa
poišči_element(ključ)
     x = poišči_element_v_drevsu(ključ)
     if x NOT  None # iskan element je v drevesu
         Lomljenje_drevesa(x)  # prični z lomljenjem dreveasa
```

Primer : Iskanje elementa 5 v drevesu

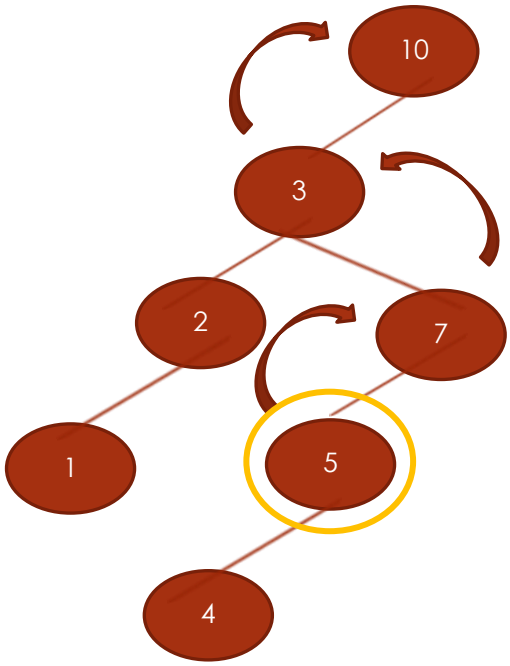
1.1. Iskan element je v drevesu



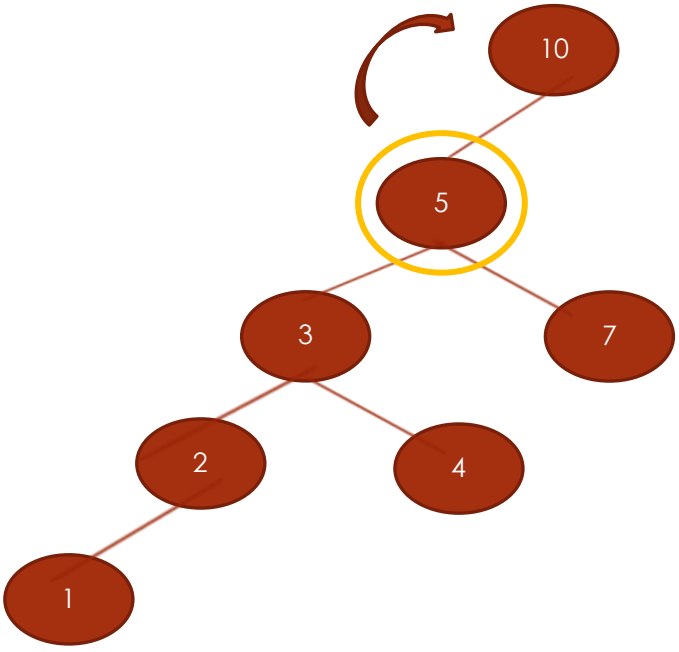
Najprej pogledamo ali je element v drevesu. Začnemo v korenu.



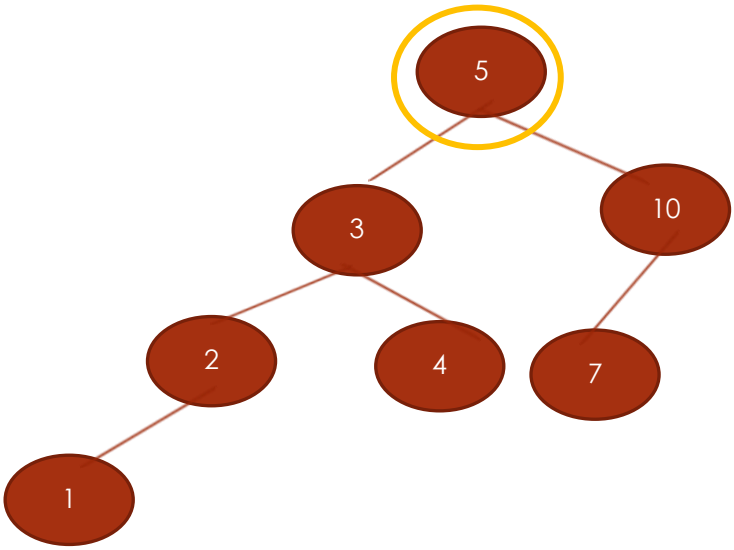
Našli smo iskani element v drevesu. Sedaj se začne lomljenje dreves.



Začnemo z dvojno desno-levo rotacijo. Nato nadaljujemo z enojno desno rotacijo.



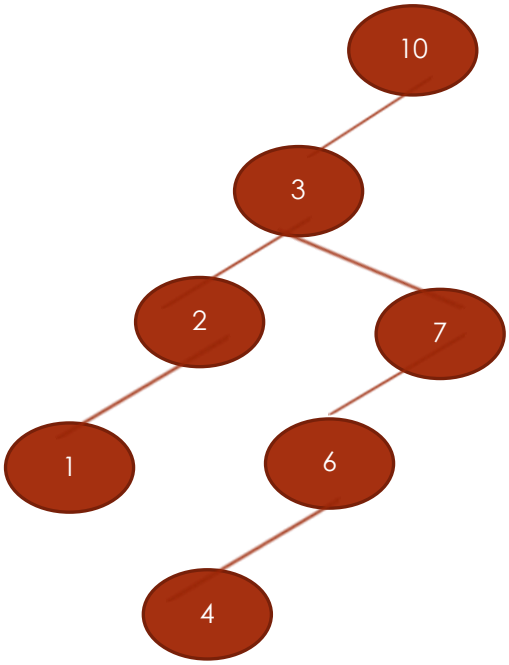
Sedaj naredimo še enojno desno rotacijo.



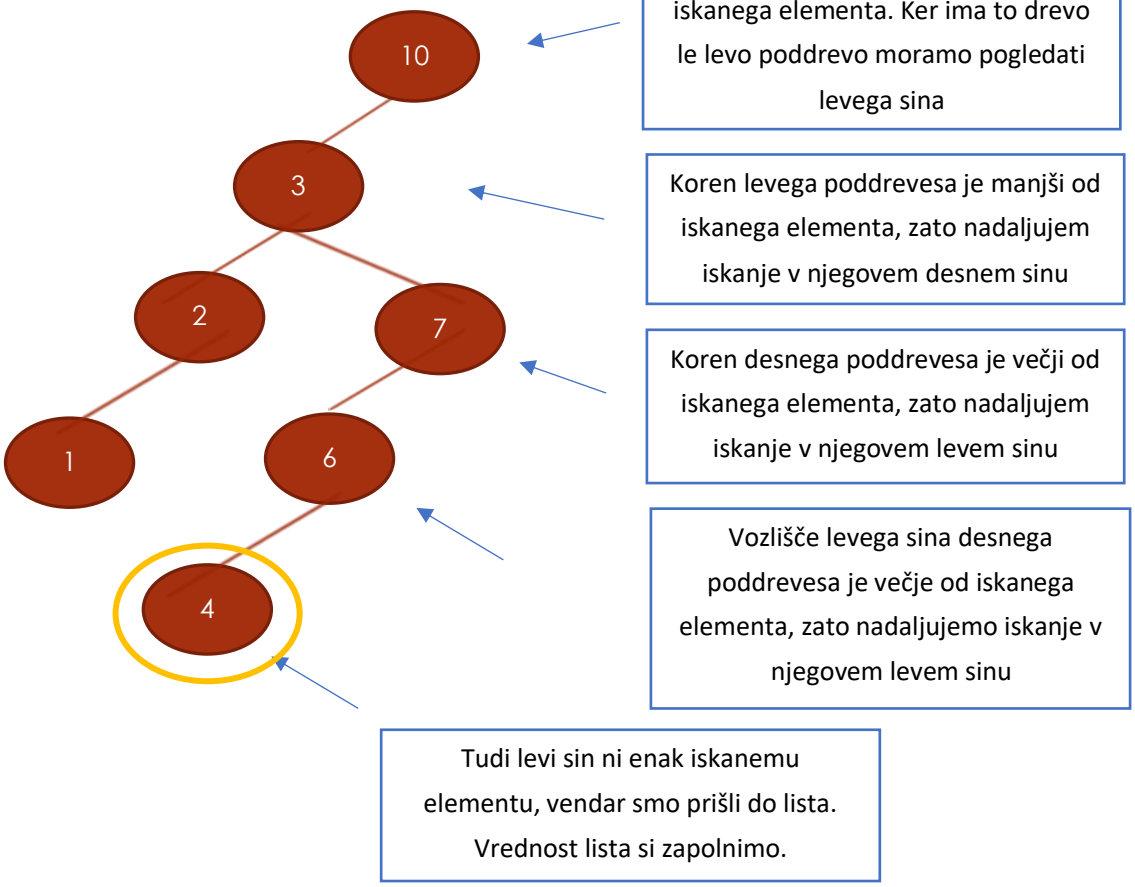
Uspeli smo premakniti iskani element v koren drevesa, sedaj preverimo ali je drevo iskalno.

Elementi v desnem poddrevesu so večji od iskanega elementa 5 in v levem poddrevesu so manjši od korena. Drevo po končanem lomljenju ostane iskalno.

1.2. Iskanega elementa ni v drevesu

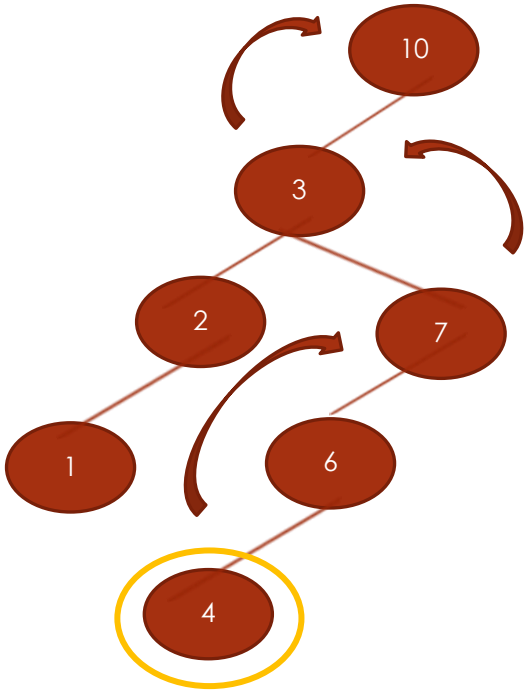


Najprej pogledamo ali je element v drevesu. Začnemo v korenu.

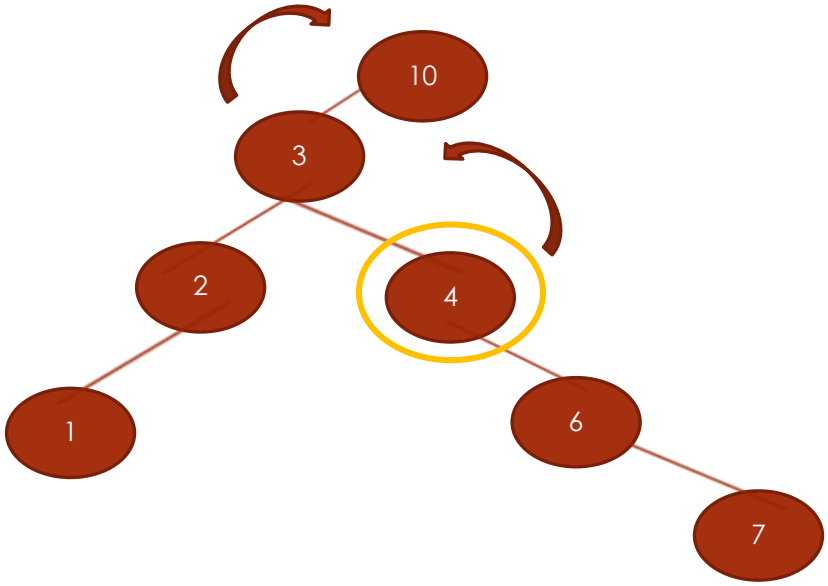


Iskanega elementa nismo našli v drevesu. Kljub temu moramo zadnji element na poti iskanja premakniti v koren, saj pri vsakem iskanju posegamo v drevo. S tem ko zadnji element v drevesu premaknemo v koren preprečimo izrojeno drevo.

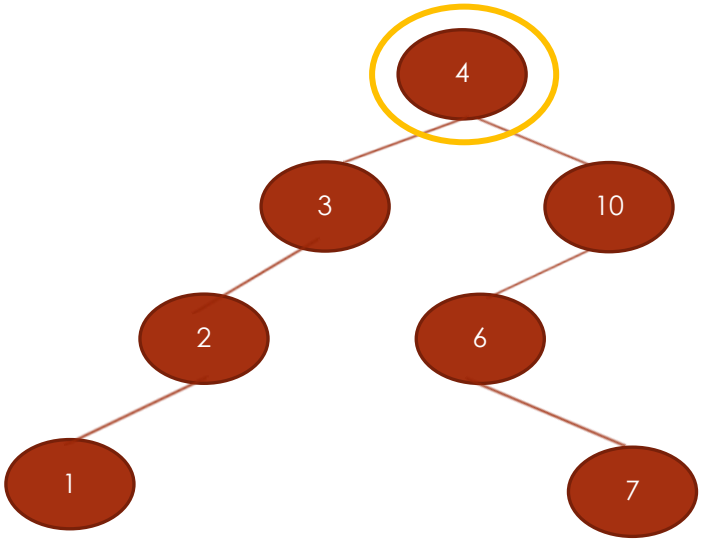
Začnimo z lomljenjem drevesa.



Najprej naredim dvojno desno-desno rotacijo.



Nadaljujem z dvojno desno-levo rotacijo.



Preverimo ali je po izvedeni rotaciji drevo še vedno iskalno.

V levem poddrevesu so manjši elementi od korena, v desnem pa večji. Tako, da lastnost iskalnost drevesa se je ohranila.

2. VSTAVLJANJE ELEMENTA V DREVO

V drevo želimo dodati element, vendar moramo najprej poiskati ustrezno mesto kamor ga bomo dodali. Element vedno vstavimo kot nov list v drevo.

Ločimo tri primere pri vstavljanju elementa v drevo:

- Drevo je prazno
  Element postane koren drevesa

- Drevo ni prazno, element katerega želimo dodati ni v drevesu
  Preverimo ali je element katerega želimo dodati v drevesu. V tem primeru ni elementa v drevesu in ga dodamo na ustrezno mesto in nadaljujemo z lomljenjem drevesa.

- Drevo ni prazno, element katerega želimo dodati je že v drevesu
  Preverimo ali je element katerega želimo dodati v drevesu. V tem primeru element je v drevesu. Seveda ga ne dodamo, saj v iskalnih drevesih ni podvojenih elementov. Ampak element, kateri je že v drevesu premaknemo v koren oziroma naredimo lomljenje drevesa.

Sprehodimo se rekurzivno skozi drevo in poiščemo ustrezno mesto kamor vstavimo element kot nov list v drevo. Nato pa naredimo lomljenje drevesa.

ČASOVNA ZAHTEVNOST

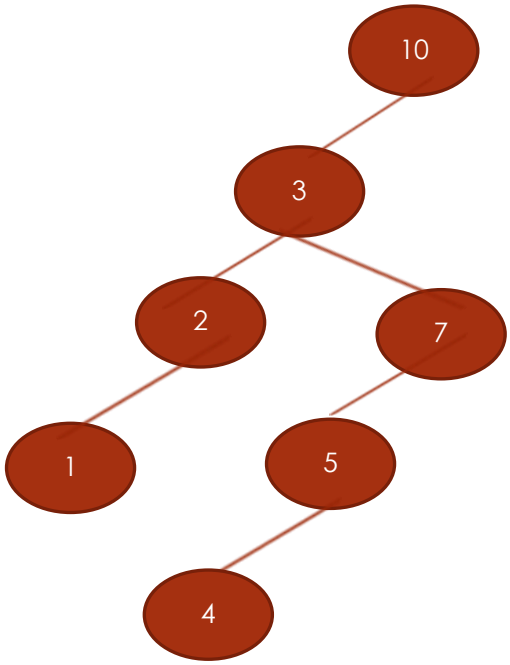
- Sprehodimo se po drevesu do ustreznega lista kamor bomo vstavili element. Iskanje primernega mesta za vstavitev elementa je prav tako sorazmerno z višino drevesa. Pričakovana časovna zahtevnost je 0(log(n)). Nato začnemo z lomljenjem. Časovna zahtevnost lomljenja pa vemo, da je 0(log(n)).

IMPLEMENTACIJA

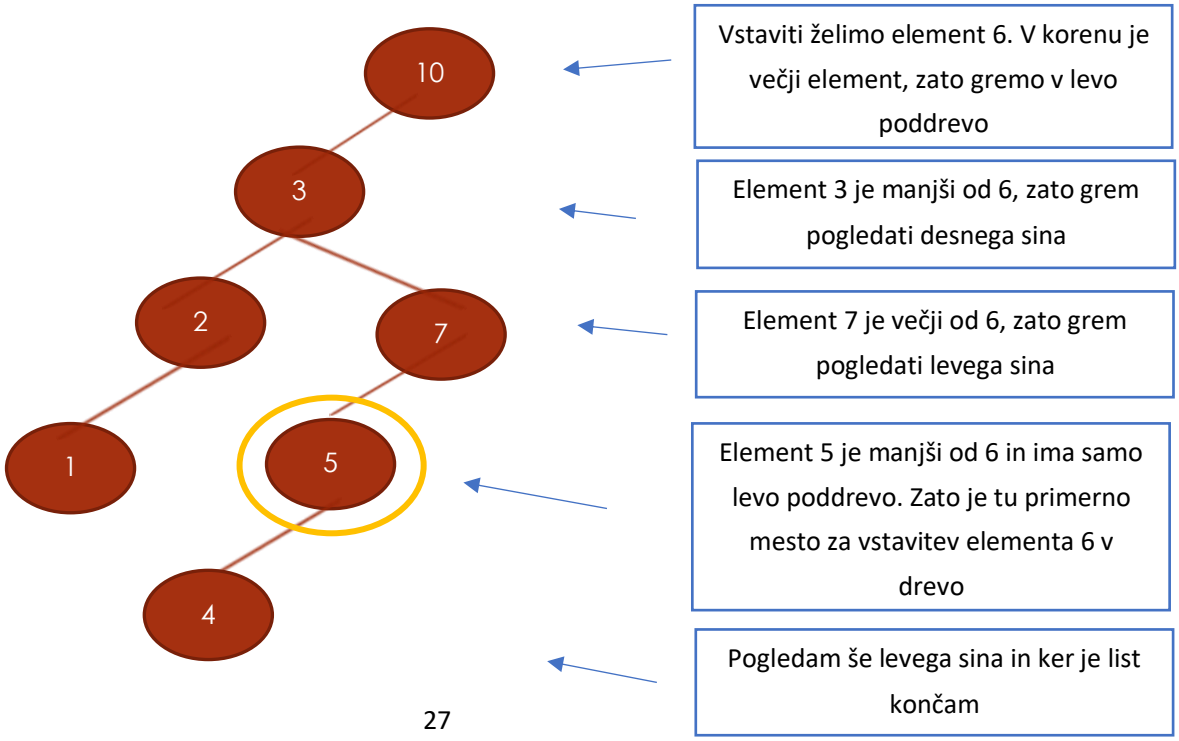
```
1.korak : Vstavimo element v drevo
2.korak : prestavimo x v koren drevesa

vstavi_element(x)
    vstavi_element_v_drevo(x)
    Lomljenje_drevesa(x)
```

Primer : Vstavljanje elementa 6 v drevesu

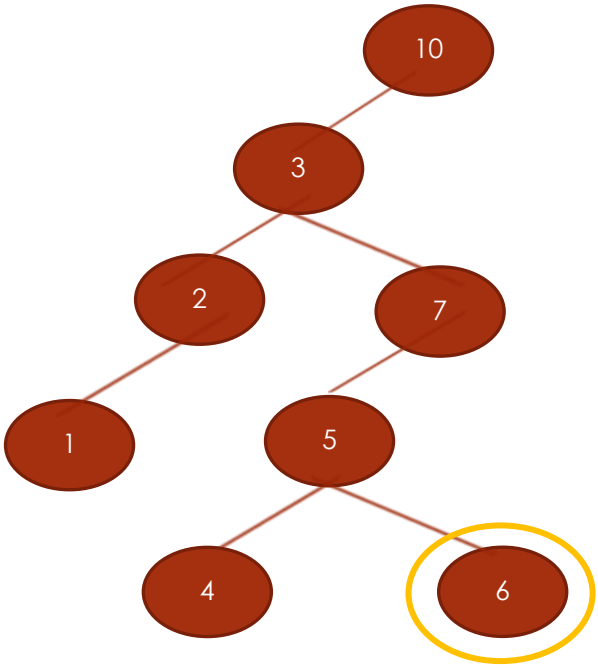
2.1. Elementa ni v drevesu



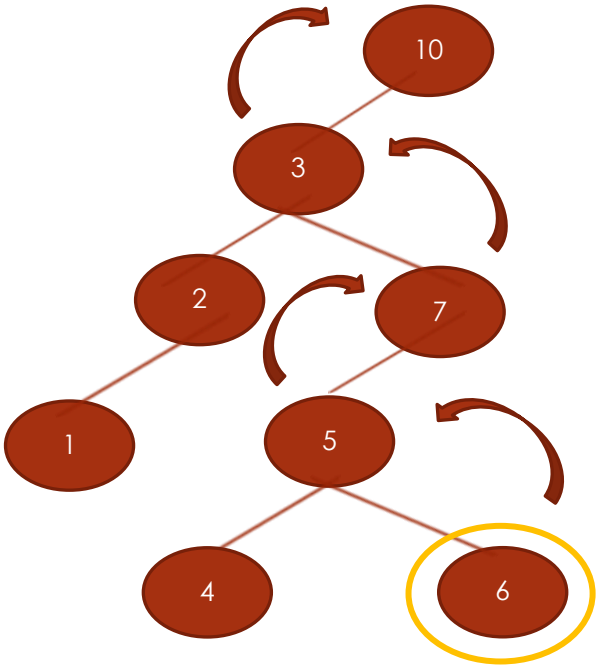
Najprej pogledamo ali je element katerega želimo dodat v drevesu. To storimo tako, da se rekurzivno sprehodimo po drevesu.



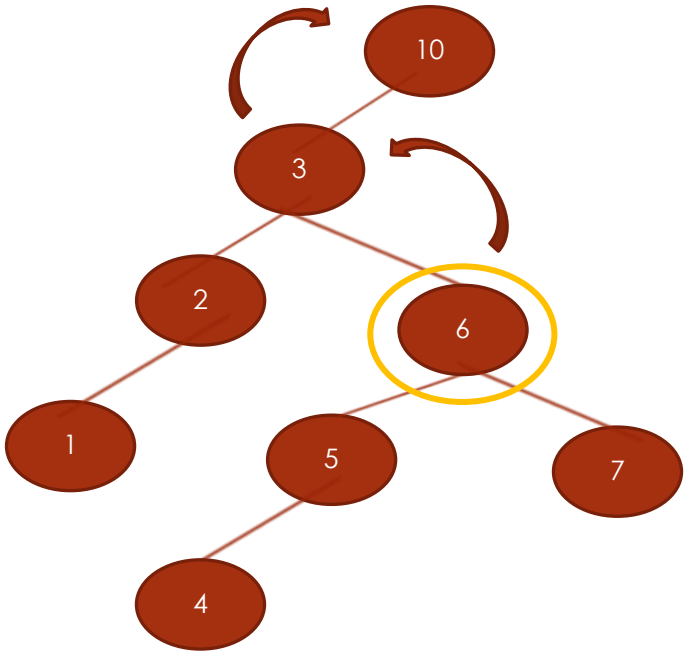
Element 6 dodam v drevo.



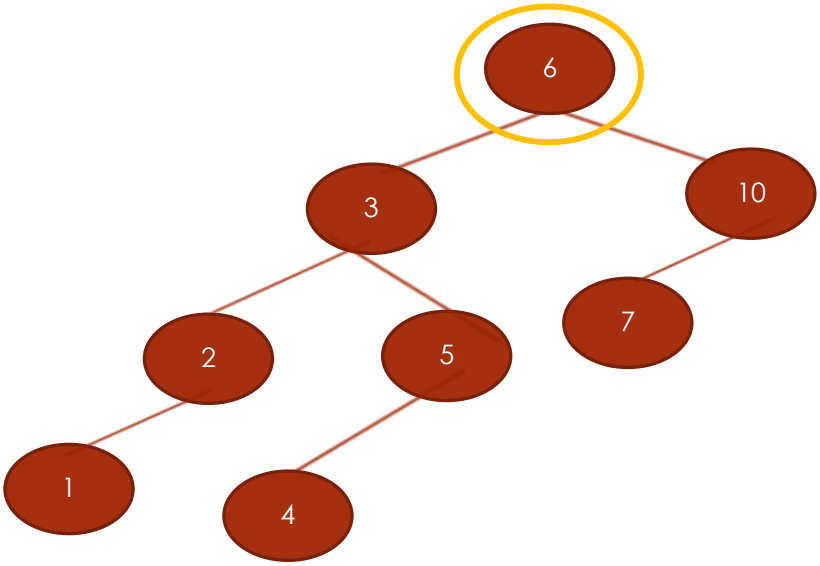
Sedaj je na vrsti lomljenje. Z zaporedjem rotacij bomo premaknili element 6 v koren drevesa. Potrebujemo dvojno levo-desno rotacijo in nato ponovimo dvojno levo-desno rotacijo.



Začnimo s prvo dvojno levo-desno rotacijo.



Nadaljujemo še z naslednjo dvojno levo-desno rotacijo in dobili bomo vstavljeni element v korenu drevesa.



3. BRISANJE ELEMENTA V DREVESU

Če želimo izbrisati element iz drevesa ga najprej z rekurzivnim sprehodom po drevesu poiščemo. S tem smo posegali v drevo, zato sledi lomljenje oziroma premik željenega elementa v koren drevesa. Sedaj ko imamo željeni element v korenu sledi brisanje elementa iz drevesa.

Ločimo tri primere pri brisanju elementa iz drevesa:

- Drevo je prazno
  V primeru kadar je drevo prazno, očitno elementa katerega želimo zbrisati ni v drevesu, zato končamo.

- Drevo vsebuje element katerega želimo izbrisati
  Z rekurzivnim sprehodom poiščemo element katerega želimo izbrisati in element premaknemo v koren drevesa.

  Ločimo tri primere:

  ➔ Levo poddrevo je prazno
  Kadar je levo poddrevo prazno, postane desni sin koren drevesa. Izbrišemo element. Vlogo korena dobi levi sin.

  ➔ Desno poddrevo je prazno
  V primeru, da je desno poddrevo prazno vlogo korena dobi levi sin.

  ➔ Desno in levo poddrevo ni prazno
  V levem poddrevesu poiščemo vozlišče z največjim elementom. Premaknemo ga v koren levega poddrevesa. Ko je levo drevo spremenjeno, velja, da je v korenu levega poddrevesa maksimalni element, torej nima desnega sina. Za desnega sina lahko uporabimo desno poddrevo celotnega drevesa. S tem levo poddrevo postane novo drevo in je element zbrisan.

- V drevesu ni elementa, katerega želimo izbrisati
  Rekurzivno poiščemo element katerega želimo izbrisati. Ampak ker elementa nismo našli, zadnjega na poti iskanja premaknemo v koren drevesa.

ČASOVNA ZAHTEVNOST

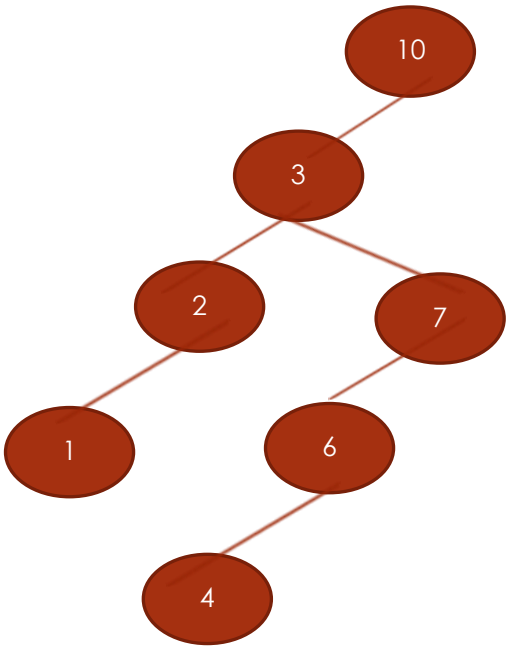
- Najprej uporabimo operacijo za iskanje, kar ima pričakovana časovno zahtevnost 0(log(n)), nadaljujemo z lomljenjem drevesa. Kar prav tako vemo, da ima pričakovano časovno zahtevnost 0(log(n)). Brisanje elementa ima konstantno količino časa 0(1).

IMPLEMENTACIJA

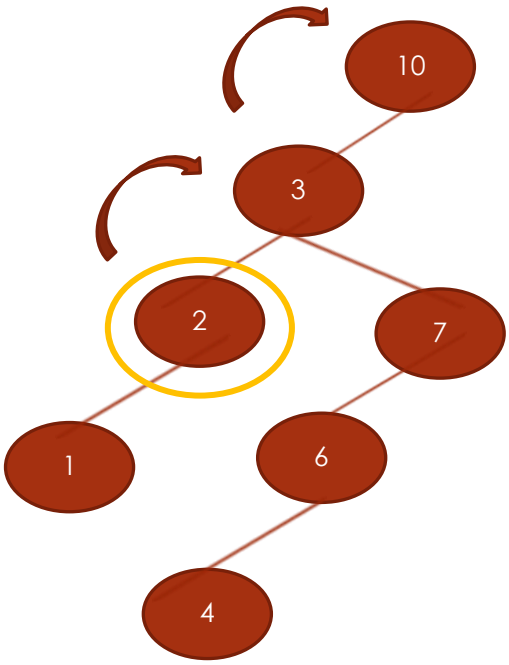
```
1. korak : Zlomimo poddrevo A in B
2. korak : Odstranimo x iz korena drevesa A
3. korak : Združimo A in b

lomljenje_brisanje(x)
     B, A = Lomljenje_drevesa(x)
     if A.levi IS NOT None
         A.levi.oče IS None
     JOIN(A.levi, B)
```

Primer : Brisanje elementa 2 v drevesu

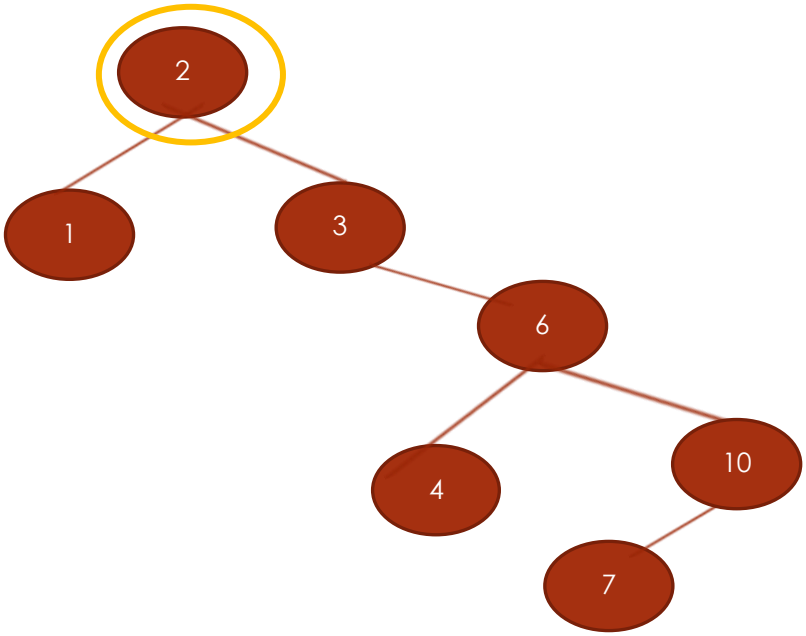
3.1. Elementa je v drevesu, levo/desno poddrevo ni prazno



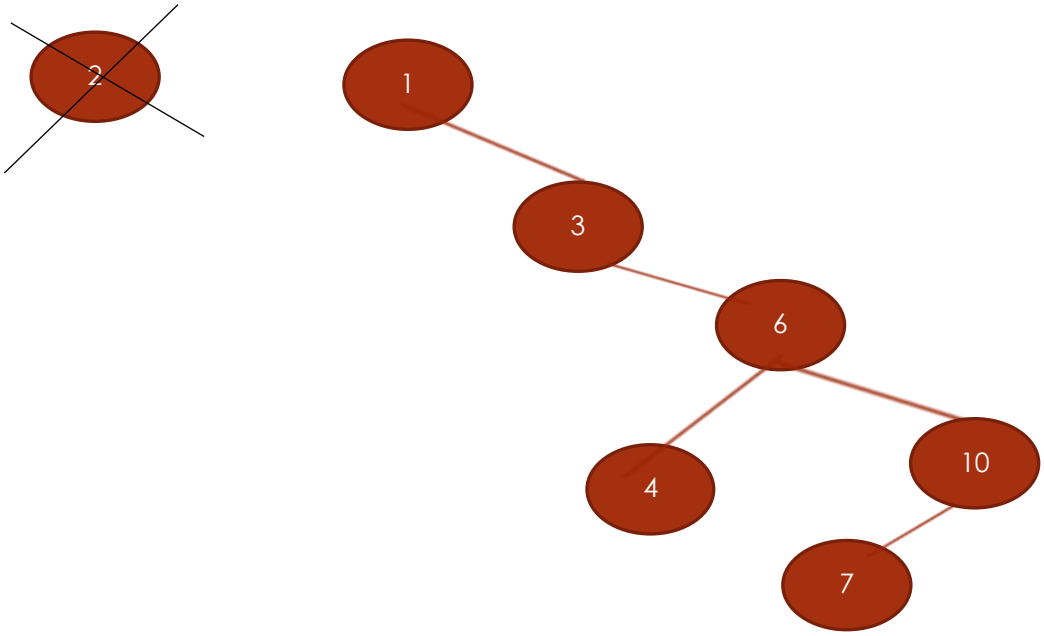
Najprej se rekurzivno sprehodimo po drevesu in poiščimo element 2 v drevesu. V korenu je element 10, zato vemo, da bomo iskali element 2 v levem poddrevesu. Koren levega poddrevesa je 3, zato pogledamo njegovega levega sin, saj je 3 večje od 2. V levem sinu smo našli element 2.



Sedaj sledi lomljenje. Potrebujemo dvojno desno-desno rotacijo.

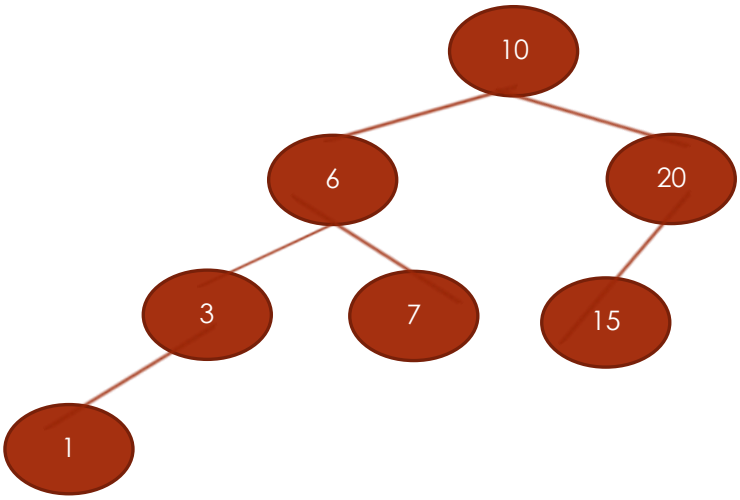


Element 2 izbrišemo in največji element v levem poddrevesu. V tem primeru je element 1, saj ima koren samo levega sina.

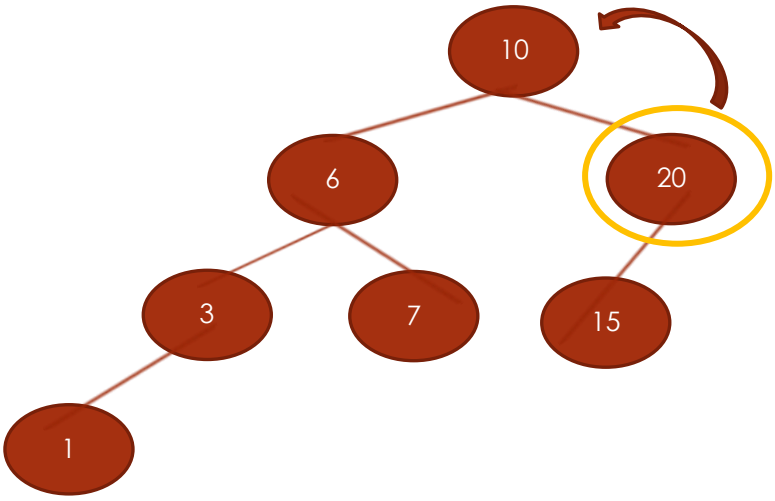


Primer: Brisanje elementa 20

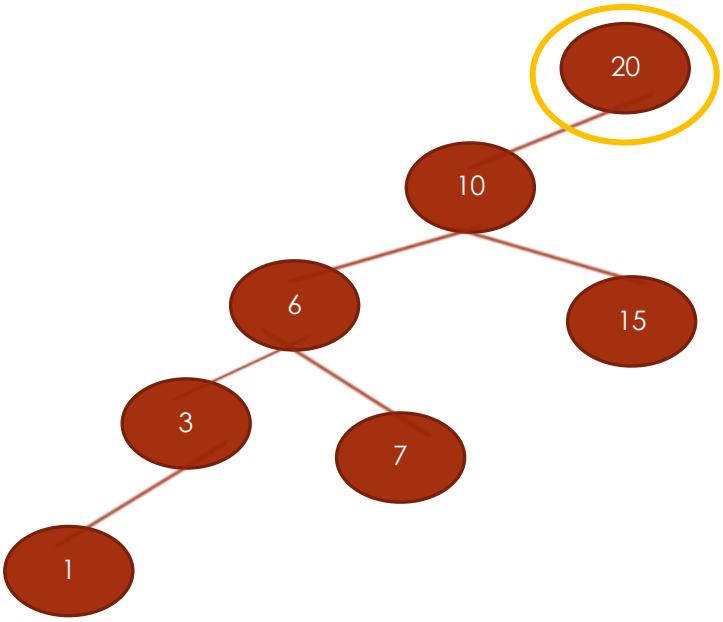
3.2. Elementa je v drevesu, levo/desno poddrevo je prazno



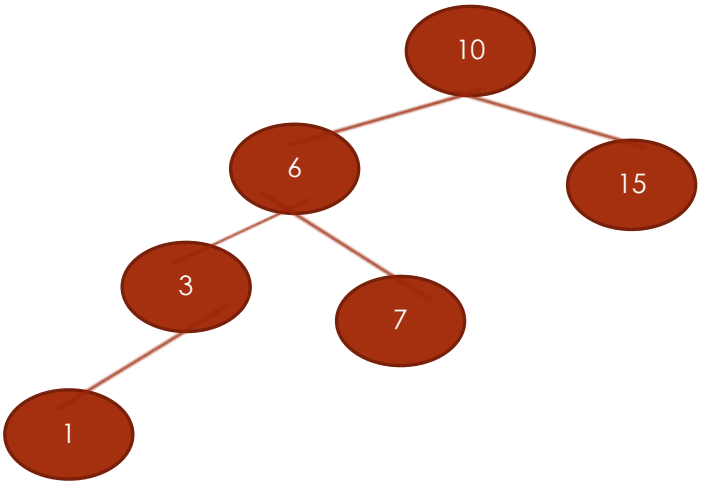
Rekurzivno poiščemo element v drevesu. Začnemo pri korenu. V korenu je element 10, zato gremo v desno poddrevo. V korenu desnega poddrevesa je element 20. Našli smo element.



Nadaljujemo z lomljenjem. Potrebujemo enojno levo rotacijo in tako premaknemo element katerega želimo izbrisati v koren drevesa.



Sedaj, ko imamo element katerega želimo izbrisati v korenu drevesa. Izbrišemo



VIRI

- https://www.codesdope.com/course/data-structures-splay-trees/ [dostop:10.02.2022]
- https://wiki.lokar.fmf.uni-lj.si/psawiki/images/archive/6/64/20090731110137!Lomljena_drevesa.pdf [dostop:10.02.2022]
- https://wiki.lokar.fmf.uni-lj.si/r2wiki/index.php/Lomljena_drevesa [dostop:10.02.2022]
- https://lokar.fmf.uni-lj.si/www/osebno/OpravljeneDiplome/Simon_Butalic_diploma.pdf [dostop:10.02.2022]
- https://wiki.lokar.fmf.uni-lj.si/r2wiki/images/e/e9/S%26D_LomljenoDrevo_prosojnice.pdf [dostop:10.02.2022]
- https://slidetodoc.com/splay-tree-1-seminarska-naloga-splay-tree-lomljena/ [dostop:10.02.2022]