UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika– Praktična matematika (VSŠ)

Renata Jere

B – DREVESA

Diplomska naloga

Ljubljana, junij 2013

KAZALO

ZAHVALA

Posebna zahvala mentorju, profesorju mag. Matiji Lokarju, za vse napotke, ves čas in trud ter potrpežljivost v času nastajanja diplomske naloge.

Zahvaljujem se tudi staršem, bratu in vsem prijateljem, ki so me tako ali drugače podpirali in mi stali ob strani v času študija in opravljanja diplomske naloge.

Diplomsko nalogo posvečam svojim staršem in bratu Aljažu.

PROGRAM DELA

V diplomski nalogi predstavite B-drevesa in njihovo izpeljanko B^+-drevesa.

Osnovna literatura:

- Comer, Douglas, *The Ubiquitous B-Tree*, Computing Surveys, 11(2):123, 1979.
- Knuth, Donald, *Sorting and Searching*, The Art of Computer Programming, Volume 3, Addison-Wesley.
- Cormen, Thomas; Leiserson, Charles; Rivest, Ronald; Stein, Clifford (2001), *Introduction to Algorithms*, MIT Press and McGraw-Hill.
- R. Bayer, E. McCreight, *Organization and Maintenance of Large Ordered Indexes*, Acta Informatica, Vol. 1, Fasc. 3, 1972.

Mentor: mag. Matija Lokar

POVZETEK

V diplomski nalogi na začetku predstavim podatkovno strukturo drevo in z njo povezane pojme. Definiram posebno obliko dreves, to so dvojiška drevesa, kjer ima vsako vozlišče največ dva sinova. Govorimo o urejenih drevesih stopnje 2. Za predstavitev dvojiških dreves lahko uporabimo različne načine – predstavitev s tabelo, oklepajni zapis,… V drugem poglavju so opisane nekatere oblike dvojiških dreves, s katerimi se srečamo skozi celotno diplomsko nalogo.

Posebna oblika dvojiških dreves, ki je namenjena predvsem učinkovitemu iskanju podatkov, so iskalna dvojiška drevesa. Zaradi postopkov vstavljanja in brisanja nad iskalnimi dvojiškimi drevesi, lahko ta drevesa hitro postanejo izrojena. Časovna zahtevnost algoritmov nad takimi drevesi pa je bistveno slabša od časovne zahtevnosti algoritmov nad optimalnimi drevesi.

Rešitev so večsmerna in uravnotežena drevesa, ki so opisana v tretjem poglavju. Večsmerna drevesa v vozliščih hranijo večje število podatkov. Uravnoteženo drevo pa je tako drevo, v katerem se višini levega in desnega poddrevesa razlikujeta za največ 1. To mora veljati za vsako vozlišče drevesa. Pri operacijah nad uravnoteženimi drevesi si pomagamo z ravnotežnim faktorjem. Kadar drevo zaradi vstavljanja ali brisanja postane neuravnoteženo, ga moramo preoblikovati. Za uravnoteževanje dreves uporabljamo rotacije, ki so natančneje opisane v tem delu diplomske naloge.

Z združitvijo idej o večsmernih in uravnoteženih drevesih pridemo do B-dreves. V četrtem poglavju je predstavljena struktura B-drevesa, njene značilnosti in operacije iskanja, vstavljanja in brisanja nad B-drevesi.

Posebna oblika B-dreves so B^+-drevesa. V B^+-drevesu so celotni podatki shranjeni le v listih drevesa. V notranjih vozliščih so le ključi. Posebna lastnost teh dreves je, da so ključi v njih lahko podvojeni.

V zaključku diplomske naloge povzamem osnovne značilnosti B-dreves in B^+-dreves kot posebnih oblik uravnoteženih dreves.

Math. Subj. Class. (2010): 68P05, 68P10.

Computing Review Class. System (1998): D.1.7, D.2.3, D.3.3, G.2.2.

Ključne besede: podatkovna struktura, drevo, iskalno drevo, večsmerno drevo, uravnoteženo drevo, B-drevo, B^+-drevo

Keywords: data structure, tree, search tree, multi-way tree, balanced tree, B-tree, B^+-tree

SUMMARY

The diploma thesis starts with introduction to tree data structure and associated concepts. A special tree form is defined, namely binary tree, where each node has at most two child nodes. These are ordered trees of level 2. In order to introduce binary trees, various ways can be used – presentation using charts, parentheses, etc. The second chapter focuses on some of the binary tree forms that appear throughout the diploma thesis.

A special form of binary trees designed mostly for effective data search is binary search tree. Using the methods of insertion and deletion on binary search trees can cause trees to quickly become degenerated. Time complexity of algorithms on such trees is considerably worse than time complexity of algorithms on optimum trees.

The solutions are multi-way and balanced trees, described in the third chapter. Multi-way trees store higher number of data in the nodes, whereas a balanced tree is a tree, where the level of the left and right subtree is distinguished for 1 at most. This must be valid for each tree node. In operations on balanced trees a balance factor is in use. If due to insertion and deletion the tree becomes unbalanced, it has to be reformed. For tree balance the rotations are used, which are specified in this part of the thesis.

By combining the ideas on multi-way and balanced trees, B-trees develop. The fourth chapter represents the structure of a B-tree, its properties and search engines, insertion and deletion on B-trees.

A special form of a B-tree is a B+-tree. In a B+-tree entire data is stored only in tree leaves. Internal nodes store only keys. A special feature of those trees is that the keys inside them can be duplicated.

The final part of the diploma thesis sums up basic properties of B-trees and B+-trees as special forms of balanced trees.

1. UVOD

Podatkovna struktura je način shranjevanja podatkov z namenom, da jih lahko čim bolj učinkovito uporabimo. Ena od osnovnih podatkovnih struktur je drevo. Drevo je nelinearna podatkovna struktura, kjer podatke hranimo v vozliščih drevesa. Vsako vozlišče ima natanko enega prednika (očeta), lahko pa ima več naslednikov (sinov). Edino vozlišče brez prednika se imenuje koren drevesa.

Iskalno dvojiško drevo, v katerem ni podvojenih elementov, je poseben tip drevesa, namenjen predvsem takemu hranjenju podatkov, kjer je najbolj pogosta operacija iskanje podatkov. Časovna zahtevnost algoritmov (iskanje, vstavljanje in brisanje) nad drevesi je odvisna od višine drevesa. Pri gradnji in vzdrževanju velikih iskalnih dreves je pomembna uporaba večsmernih dreves, ki v vozlišču vsebujejo večje število podatkov. Pri tem moramo skrbeti, da je naše drevo čim bolj uravnoteženo. To pomeni, da je razlika med višino levega in desnega poddrevesa vsakega vozlišča omejena. Pri preoblikovanju neuravnoteženega drevesa si pomagamo z rotacijami, kjer v konstantnem času spet uravnotežimo drevo, ki je zaradi izvedbe določene operacije postalo neuravnoteženo.

Glavna tema te diplomske naloge so B-drevesa. Gre za posebno vrsto večsmernih iskalnih dreves. B-drevo reda *m* je namreč *m*-smerno iskalno drevo z določenimi dodatnimi lastnostmi. V poglavju o B-drevesih si bomo natančneje pogledali postopke iskanja, vstavljanja in brisanja nad B-drevesi.

Posebna oblika B-dreves so B^+-drevesa. B^+-drevo je podatkovna struktura, kjer so celotni podatki shranjeni le v listih drevesa, v notranjih vozliščih pa so le ključi. Listi B^+-drevesa so med seboj s kazalci povezani v verižni seznam. V nasprotju z do sedaj omenjenimi drevesnimi strukturami, so v B^+-drevesu ključi lahko podvojeni. Samo strukturo in lastnosti B^+-dreves si bomo natančneje pogledali v zadnjem razdelku diplomske naloge.

2. DREVO

Podatkovna struktura (datastructure) je način organizacije podatkov, ki nam omogoča, da lahko te podatke učinkovito uporabimo. Določa jo tip podatkov, ki jih hranimo v njej in operacije, ki jih izvajamo nad podatkovno strukturo. Pri študiju podatkovnih struktur nas zanimajo lastnosti operacij, kako podatkovno strukturo predstaviti v določenem programskem jeziku in kako operacije učinkovito izvesti.

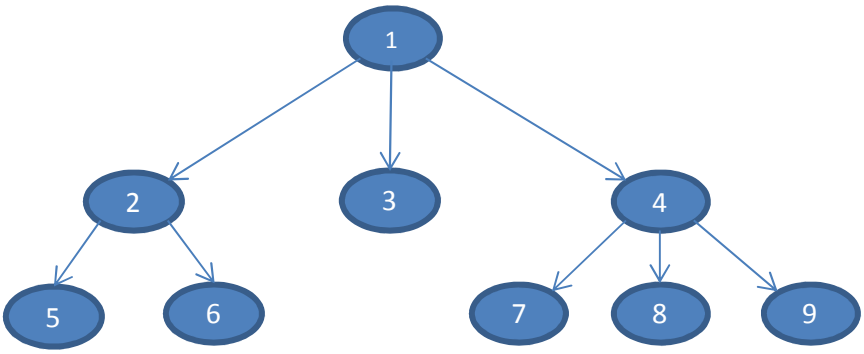
Ena od osnovnih podatkovnih struktur je drevo. Drevo je končna neprazna množica vozlišč, za katero velja: eno od vozlišč je posebej odlikovano in ga imenujemo koren, druga vozlišča pa razpadejo v disjunktno unijo n dreves T_1, T_2, …, T_n, kjer je $_n$ nenegativno celo število. Množice T_1, T_2, …, T_n, kjer je n nenegativno celo število, imenujemo poddrevesa.

1.1. Definicija drevesa

Drevo je nelinearna podatkovna struktura, katere osnovni elementi so vozlišča. V vozliščih hranimo podatke. Pogosto je najpomembnejši del teh podatkov ključ podatka. To je tisti del podatka, ki je značilen za ta podatek in ga po njem lahko prepoznamo. Pri enostavnih podatkih (števila, nizi,…) je ključ kar podatek sam. Če ne bo možnosti zamenjave, bomo podatkom v vozliščih večkrat rekli kar ključi oziroma ključem podatki. Elementi drevesa (vozlišča) so razporejeni hierarhično v razmerju »oče« - »sin«, ter so med seboj povezani. Vsako vozlišče ima lahko več naslednikov (sinov), vendar natanko enega prednika (očeta vozlišča). Le vrhnje vozlišče, imenovano koren, prednika nima. Je torej edino vozlišče v drevesu brez prednika.

Poznamo še posebno drevo, to je prazno drevo. Nima nobenega vozlišča in v njem ne hranimo nobenega podatka.

Primer podatkovne strukture drevo je prikazan na sliki Slika 1.



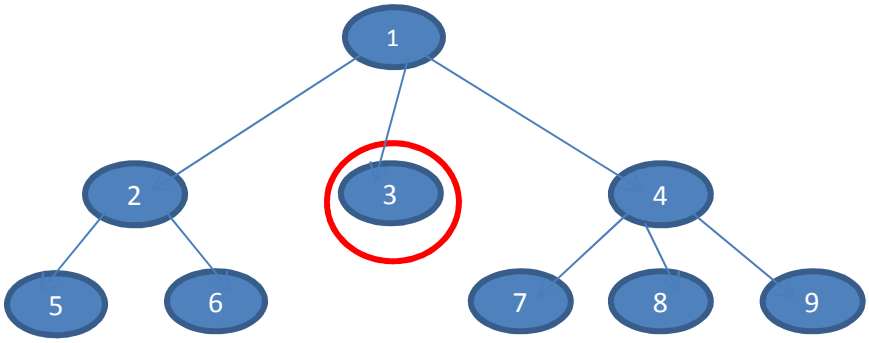
Slika 1: Primer podatkovne strukture drevo

Drevo torej opiše povezavo vozlišč, izbira podatkov, ki jih vstavljamo v vozlišča, pa je odvisna od primera uporabe.

Oglejmo si nekaj pojmov, ki jih pogosto srečamo ob delu z drevesi.

1.2. Terminologija

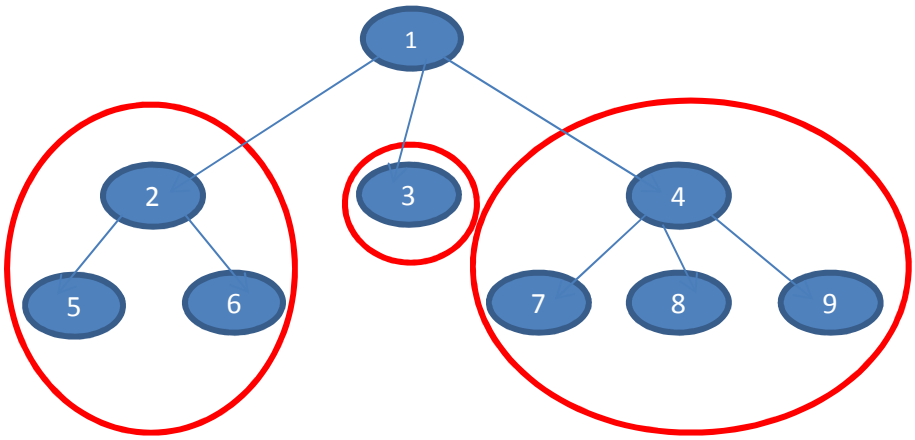
Vozlišče: element drevesa, ki hrani podatek (ali podatke) izbranega tipa in informacijo o iz njega izhajajočih poddrevesih.



Slika 2: Vozlišče drevesa

Na sliki Slika 2 je posebej označeno tisto vozlišče drevesa, ki kot podatek hrani število 3. Ostala vozlišča hranijo podatke 1, 2, 4, 5, 6, 7, 8 in 9.

Poddrevo: celotno drevo, ki izhaja iz vozlišča.



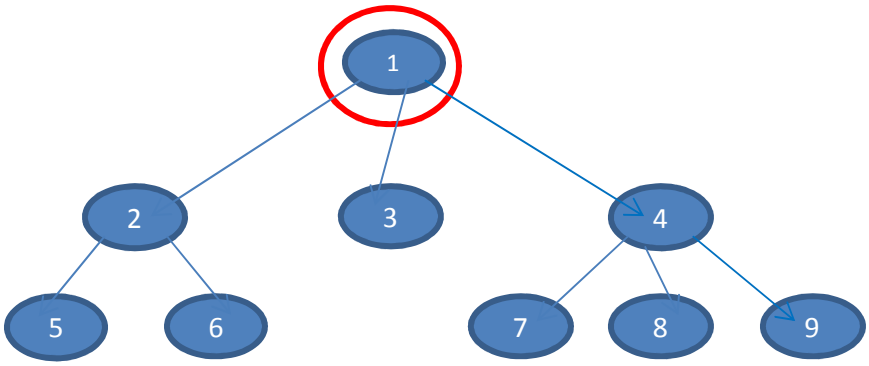
Slika 3: Poddrevo

V konkretnem primeru (Slika 3) iz vozlišča, ki hrani podatek 1, izhajajo 3 poddrevesa. Označena so z rdečo barvo.

Potomci: vsa vozlišča v vseh poddrevesih danega vozlišča.

Potomca vozlišča s podatkom 2 v drevesu na sliki Slika 1 sta vozlišče s podatkom 5 in vozlišče s podatkom 6. Opazimo, da vozlišče, ki hrani podatek 3, nima nobenega potomca. Vozlišče s podatkom 4 pa ima za svoje potomce vozlišča s podatki 7,8 in 9. Potomci vozlišča s podatkom 1 pa so vsa preostala vozlišča.

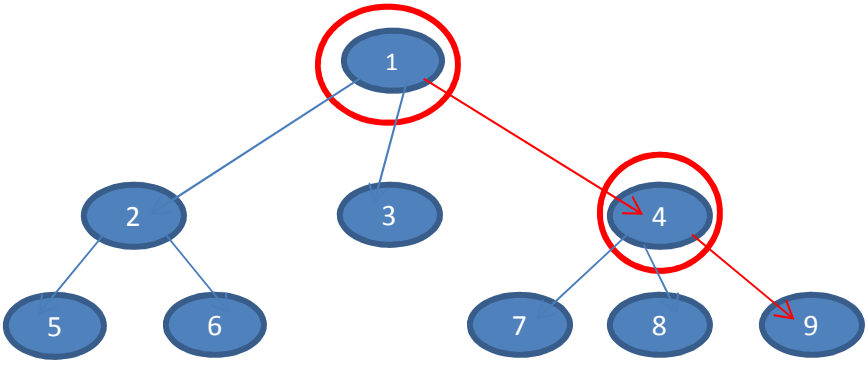
Koren: vozlišče brez predhodnika (očeta).



Slika 4: Koren drevesa

Koren drevesa na sliki Slika 4 je vozlišče s podatkom 1. Vidimo da nima nobenega predhodnika. Pravimo tudi, da je koren »začetno vozlišče«.

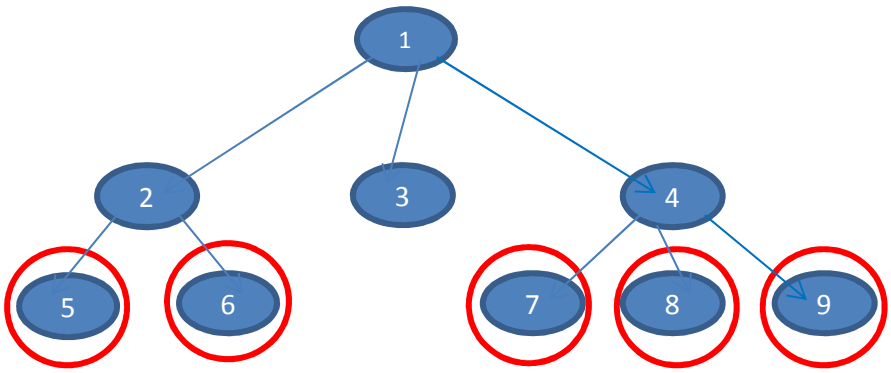
Predniki: vozlišča na poti od korena do tega vozlišča.



Slika 5: Predniki vozlišča

Prednika vozlišča s podatkom 9 sta vozlišči s podatkoma 4 in 1. Rdeče puščice na sliki Slika 5 kažejo pot od korena do vozlišča, ki hrani podatek 9.

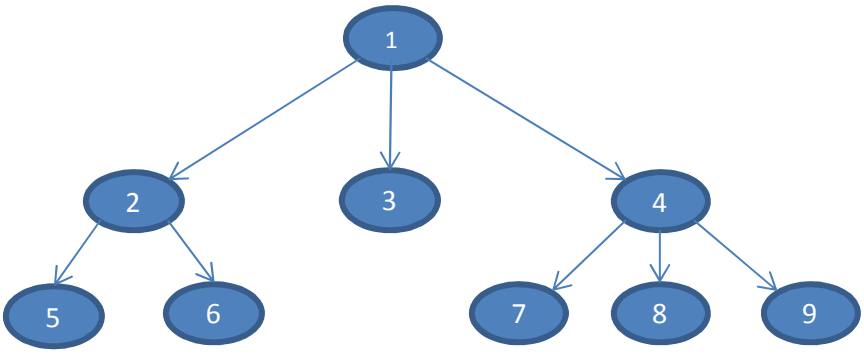
List: vozlišče brez potomcev, ki mu pravimo tudi končno vozlišče.



Slika 6: Končna vozlišča (listi) drevesa

Listi ali končna vozlišča drevesa so na sliki Slika 6 označeni z rdečimi krogi. To so vozlišča s podatki 5, 6, 7, 8 in 9.

Oče: predhodnik vozlišča.



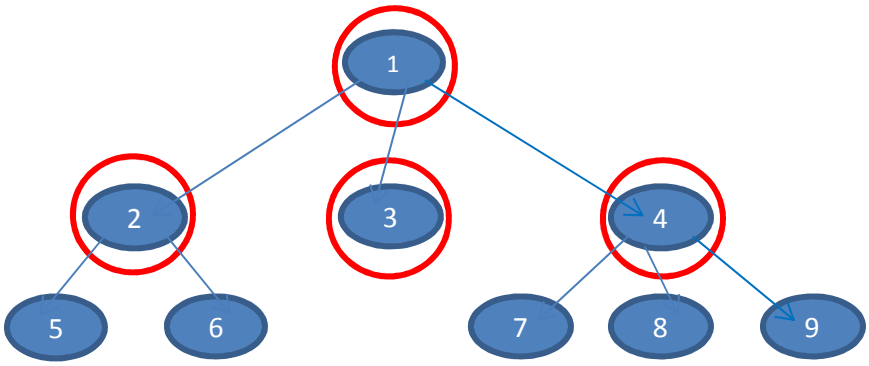
Slika 7: Drevo

V drevesu na sliki Slika 7 imata lista s podatkoma 5 in 6 istega predhodnika, to je vozlišče s podatkom 2. Tudi vozlišča s podatki 7, 8 in 9 imajo skupnega predhodnika (očeta) – vozlišče, ki hrani podatek 4. Oče vozlišč s podatki 2, 3 in 4 je vozlišče s podatkom 1. Ker to vozlišče nima predhodnika, ga imenujemo koren drevesa. Koren je edino vozlišče v drevesu brez očeta.

Sin (naslednik): koren poddrevesa nekega vozlišča.

Koren drevesa na sliki Slika 7 ima sinove s podatki 2, 3 in 4. Vozlišče s podatkom 2 ima dva sinova in sicer vozlišči s podatkoma 5 in 6. To sta lista drevesa, kar pomeni da nimata sinov. Tudi vozlišče s podatkom 3 je list in nima naslednika. Vozlišče s podatkom 4 ima za svoje sinove vozlišča s podatki 7, 8 in 9. To so spet listi drevesa.

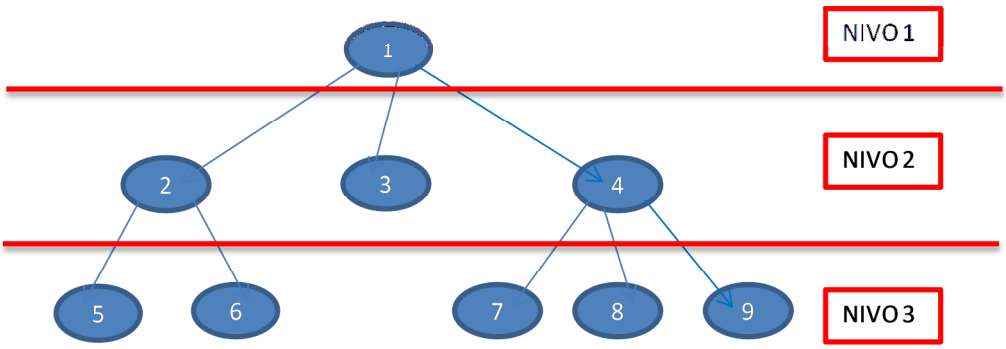
Notranja vozlišča: vsa vozlišča drevesa, razen listov.



Slika 8: Notranja vozlišča drevesa

Na sliki Slika 8 so z rdečo barvo označena notranja vozlišča drevesa. To so vozlišča s podatki 1, 2, 3 in 4. Med notranja vozlišča spada tudi koren drevesa. Vozlišča s podatki 5, 6, 7, 8 in 9 so listi drevesa in zato niso notranja vozlišča.

Nivo/raven vozlišča: koren je na nivoju 1, sinovi korena na nivoju 2, sinovi sinov so na nivoju 3…



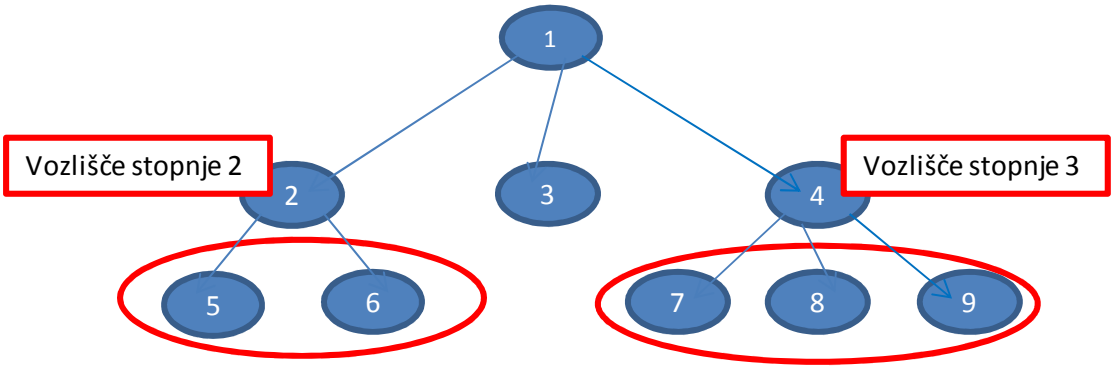
Slika 9: Nivo vozlišča

Na sliki Slika 9 so z rdečo barvo prikazani posamezni nivoji drevesa. V konkretnem primeru so listi s podatki 5, 6, 7, 8 in 9 na nivoju 3, list s podatkom 3 pa na nivoju 2. Z največjim nivojem je določena višina drevesa.

Višina drevesa: je enaka najvišjemu nivoju vozlišča v drevesu.

Višina drevesa na sliki Slika 9 je 3.

Stopnja vozlišča drevesa je podana s številom iz njega izhajajočih poddreves.



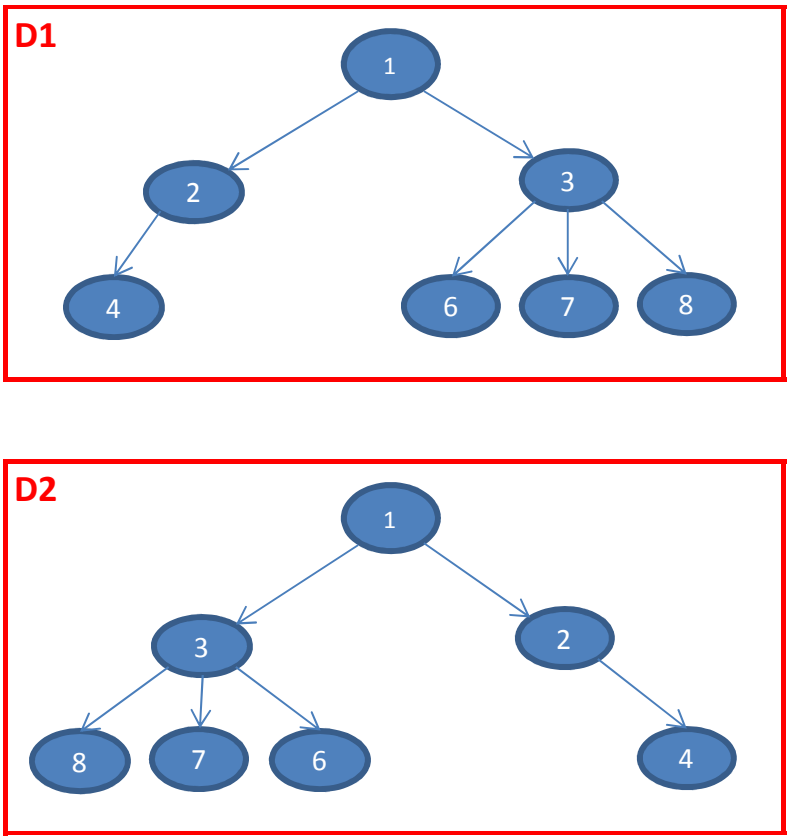
Slika 10: Stopnja vozlišča

Vozlišče s podatkom 2 je stopnje 2, saj ima 2 sinova (vozlišči s podatkoma 5 in 6). Stopnja vozlišča s podatkom 4 pa je 3. Ta ima za svoje sinove vozlišča s podatki 7, 8 in 9.

Stopnja drevesa je najvišja stopnja njegovih vozlišč.

Stopnja drevesa na sliki Slika 10 je 3.

Urejeno drevo je tisto drevo, pri katerem je vrstni red sinov (poddreves) pomemben. Če na drevesi D1 in D2 s slike Slika 11 gledamo kot na urejeni drevesi, sta različni, sicer pa sta enaki. Urejenost drevesa se nanaša le na obliko drevesa in ne na urejenost podatkov v drevesu.



Slika 11: Urejeni drevesi

2. DVOJIŠKO DREVO

Posebna vrsta drevesa so urejena drevesa stopnje 2. Vsako vozlišče ima torej največ dva sinova. Tem drevesom pravimo dvojiška drevesa.

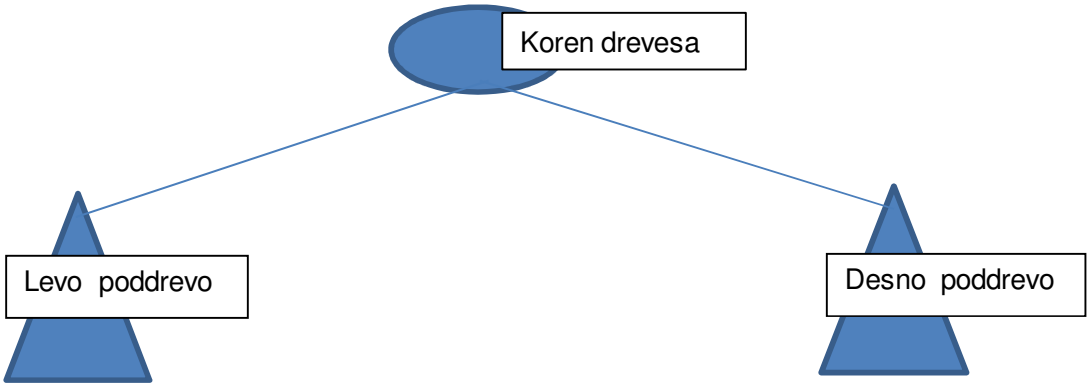
2.1. Definicija dvojiškega drevesa

Dvojiško drevo običajno definiramo na naslednji način: »*Dvojiško drevo je bodisi prazno ali pa ga sestavlja posebej odlikovano vozišče koren, ki ima levo in desno poddrevo. Levo in desno poddrevo sta spet dvojiški drevesi.*«

Dvojiško drevo je urejeno drevo, saj je vrstni red sinov pomemben; govorimo o levem in desnem poddrevesu.

Neprazno dvojiško drevo D lahko zapišemo v obliki trojice

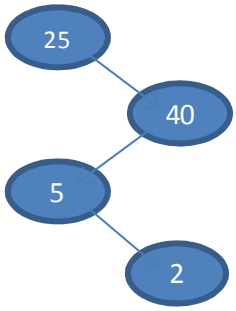
D = (koren,LevoPoddrevo,DesnoPoddrevo).



Slika 12: Shema dvojiškega drevesa

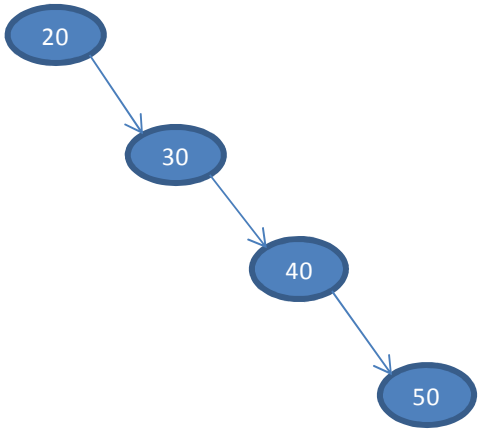
Omenimo sedaj nekatere posebne oblike dvojiških dreves, ki jih bomo srečali v nadaljevanju diplomske naloge:

Izrojeno dvojiško drevo je drevo, v katerem ima vsako vozlišče (razen lista (ki je v izrojenem drevesu en sam)) le enega sina.



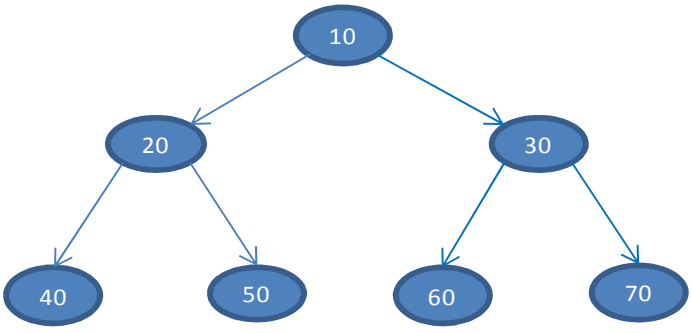
Slika 13: Izrojeno dvojiško drevo

Desno izrojeno dvojiško drevo je drevo, v katerem ima vsako vozlišče (razen lista (ki je v izrojenem drevesu en sam)) le desnega sina.



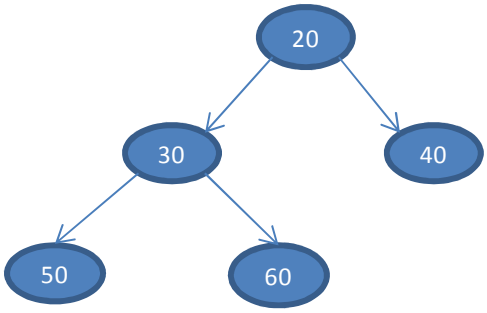
Slika 14: Desno izrojeno dvojiško drevo

Polno dvojiško drevo je drevo, kjer so vsi listi drevesa na istem nivoju, vsa notranja vozlišča pa imajo natanko dva sinova.



Slika 15: Polno dvojiško drevo

Levo poravnano dvojiško drevo je polno dvojiško drevo ali »skoraj« polno drevo, kjer se nivoji listov razlikujejo kvečjemu za ena. Ta razlika se v drevesu, ko liste pregledujemo od leve proti desni, zgodi samo enkrat.



Slika 16: Levo poravnano dvojiško drevo

2.2. Predstavitve dvojiških dreves

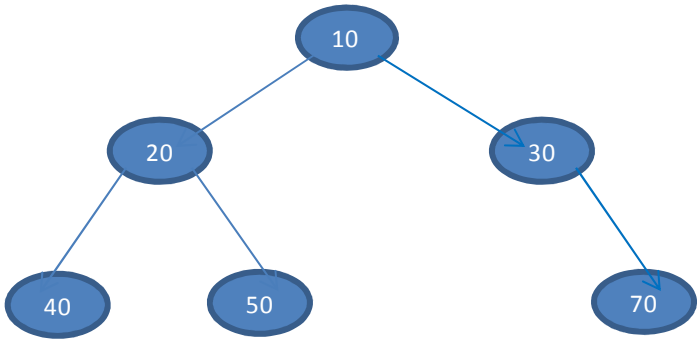
Če želimo dvojiško drevo predstaviti v določenem programskem jeziku, lahko uporabimo različne načine. Oglejmo si tri – oklepajni zapis, predstavitev s tabelo in predstavitev s kazalci.

- Oklepajni zapis

Definicija zapisa dvojiškega drevesa z oklepaji je rekurzivna:

- Prazno drevo predstavimo kot niz »[]«.
- Če drevo ni prazno, je sestavljeno iz korena, levega in desnega poddrevesa.
  Takrat oklepajni zapis zapišemo kot niz: »[podatek v korenu, oklepajni zapis levega poddrevesa, oklepajni zapis desnega poddrevesa]«.

Drevo na sliki Slika 17 želimo zapisati z oklepajnim zapisom:



Slika 17: Dvojiško drevo

Oklepajni zapis drevesa je: [10,[20,[40,[],[],],[50,[],[]],[30,[],[70,[],[]]].

Če zapis razdelimo na dele »10«, »[20[40[][]][50[][]]« in »[30[][70[][]]]«, se lepo vidi struktura opisa. Prvi niz opiše koren dvojiškega drevesa, drugi niz njegovo levo in tretji niz desno poddrevo.

- Predstavitev s tabelo

Tu podatke hranimo v tabeli. Elemente drevesa razvrstimo po nivojih, na posameznem nivoju pa jih vodimo od leve proti desni in pri tem upoštevamo »manjkajoče« elemente. Tabela nam torej služi kot skladišče podatkov v drevesu, strukturo drevesa pa nam določajo indeksi elementov. Prvo mesto v tabeli (mesto z indeksom 0) pustimo neuporabljeno in začnemo elemente vpisovati od indeksa 1 dalje. Če je vozlišče drevesa na i-tem mestu v tabeli, veljajo za indekse njegovega očeta in sinov naslednje formule:

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

Za primer vzemimo drevo na sliki Slika 17, ki ga zapišemo s tabelo:

Tabela 1: Predstavitev drevesa s tabelo

	10	20	30	40	50		70

Predstavitev s tabelo je dobra predvsem takrat, ko potrebujemo hiter dostop do očeta oziroma sinov določenega vozlišča. Manj primerna pa je takrat, ko potrebujemo celo levo (ali desno) poddrevo nekega vozlišča.

Predstavitev s tabelo je smiselna, kadar je drevo polno ali levo poravnano. Kadar pa drevo ni tako, moramo za vsako nezasedeno vozlišče izpustiti ustrezno prazno mesto v tabeli, da ohranimo indeksiranje. Tako lahko izgubimo veliko prostora. To bi se zgodilo, če bi s tabelo želeli predstaviti desno izrojeno dvojiško drevo (Slika 14). Za desno izrojeno dvojiško drevo z n vozlišči namreč potrebujemo tabelo velikosti $2^n - 1$, v kateri pa bi bilo le n zasedenih mest.

- Predstavitev s kazalci

Za predstavitev dreves pogosto uporabimo kazalčno predstavitev. Tu vozlišča drevesa predstavimo s strukturo, kjer poleg podatka hranimo še kazalce na vozlišča, ki vsebujejo očeta in sinova.

Možnosti, kaj hranimo v vozliščih so:

- Podatek, kazalec na očeta.
- Podatek, kazalec na levega in desnega sina.
- Podatek, kazalec na očeta, kazalec na levega in desnega sina.

Kako dostopamo do samega drevesa, je odvisno od izbrane možnosti. Če izberemo drugo ali tretjo, potrebujemo le en kazalec (kazalec na korensko vozlišče), če pa prvo, potrebujemo kazalce na vse liste.

2.3. Pregledi dvojiških dreves

Pregled dvojiškega drevesa imenujemo kakršenkoli sistematičen obisk vseh vozlišč drevesa v določenem vrstnem redu. Lahko ga opravimo na različne načine: v globino, po nivojih, po listih,...

V dvojiškem drevesu imamo tri pomembnejše vrstne rede obiska. Odločimo se dati prednost levemu sinu danega vozlišča. Tako dobimo tri najpomembnejše preglede:

- Premi pregled
  Če je drevo prazno, pregled ni potreben. Sicer pa najprej obiščemo podatek v korenu, sledi premi pregled levega poddrevesa in na koncu še premi pregled desnega poddrevesa.

- Vmesni pregled
  Že samo ime pove, da tu podatek v korenu obiščemo »vmes«. Najprej vmesno pregledamo levo poddrevo, obiščemo koren in na koncu vmesno pregledamo še desno poddrevo.

- Obratni pregled
  Najprej obratno pregledamo levo poddrevo, nato obratno pregledamo desno poddrevo, na koncu obiščemo koren.

Zapis postopkov, s katerimi pregledamo dvojiško drevo, je že po svoji naravi rekurziven. Zanima nas, kakšna predstavitev dvojiškega drevesa bo primerna za pregled le-tega. V vsakem vozlišču potrebujemo napotek, kje iskati levo in kje desno poddrevo. Zato bo najbolj primerna kazalčna predstavitev, kjer vsako vozlišče hrani podatek in kazalca na oba sinova.

Zapišimo algoritem `premiPregled`, ki vozlišča dvojiškega drevesa obišče v premem vrstnem redu:

```
algoritem premiPregled(drevo);
začni
      če drevo ni prazno
            začni
                  obišči(drevo.koren);
                  premiPregled(drevo.levoPoddrevo);
                  premiPregled(drevo.desnoPoddrevo);
            končaj
končaj
```

Pri pregledu praznega drevesa, ki je tudi dvojiško drevo, algoritem ne stori ničesar. Kadar drevo ni prazno, najprej obišče koren, premo pregleda levo poddrevo in nato še desno.

Poglejmo si primer. Denimo, da je dano dvojiško drevo s slike Slika 17. Zapišimo sedaj, v kakšnem redu bi pri posameznem pregledu dobivali podatke tega drevesa.

Premi pregled: 10, 20, 40, 50, 30, 70.

Vmesni pregled: 40, 20, 50, 10, 70, 40.

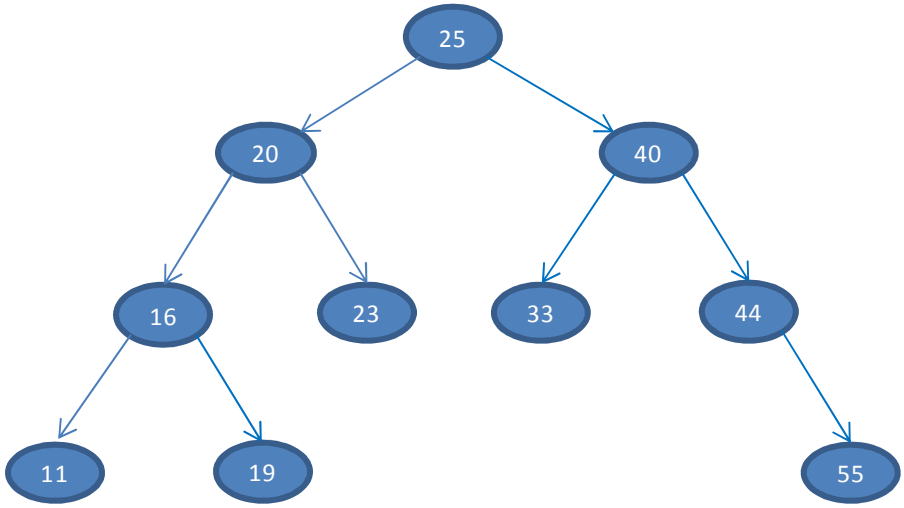
Obratni pregled: 40, 50, 20, 70, 30, 10.

2.4. Iskalno dvojiško drevo

Drevesa kot podatkovne strukture so zelo uporabna za organizacijo podatkov. Poseben tip drevesa, ki ga imenujemo iskalno dvojiško drevo, je namenjen predstavitvi podatkov, med katerimi želimo učinkovito iskati. Iskalno dvojiško drevo ima v vozliščih shranjene podatke, ki jih lahko primerjamo po velikosti. Za iskalno dvojiško drevo velja, da so vsi elementi v levem poddrevesu (ki mora biti tudi iskalno dvojiško drevo) manjši od podatka v korenu, vsi elementi v desnem poddrevesu (ki mora tudi

biti iskalno dvojiško drevo) pa so večji od podatka v korenu. Iz definicije je torej razvidno, da v njem ni podvojenih elementov.

Primer iskalnega dvojiškega drevesa je prikazan na sliki Slika 18.



Slika 18: Iskalno dvojiško drevo

Drevo na zgornji sliki ustreza vsem zahtevam iskalnega dvojiškega drevesa. Vsako vozlišče ima največ dva sinova, kar pomeni da je dvojiško drevo. Hkrati so vsi elementi v levem poddrevesu (11, 16, 19, 20 in 23) manjši, in vsi elementi v desnem poddrevesu (33, 40, 44 in 55) večji od elementa v korenu (25). Da so levi potomci manjši, desni pa večji velja za vsako vozlišče drevesa. Zato je to drevo iskalno dvojiško drevo velikosti 10 in globine 4.

2.5. Pogoste operacije nad iskalnimi dvojiškimi drevesi

Pogoste operacije, definirane na iskalnih dvojiških drevesih, so:

- iskanje(koren, podatek) ... vrne true, če v dvojiškem iskalnem drevesu s podanim korenom obstaja vozlišče z danim podatkom, false sicer
- vstavi(koren, vozlišče) ... v dvojiško iskalno drevo s podanim korenom vstavi novo vozlišče tako, da se ohrani lastnost dvojiškega iskalnega drevesa
- odstrani(koren, ključ) ... iz dvojiškega iskalnega drevesa s podanim korenom odstrani element z danim ključem, če obstaja
- minimum(koren) ... vrne kazalec na vozlišče, ki vsebuje najmanjšo vrednost v drevesu s podanim korenom
- maximum(koren) ... vrne kazalec na vozlišče, ki vsebuje največjo vrednost v drevesu s podanim korenom
- naslednik(vozlišče) ... vrne kazalec na vozlišče, ki vsebuje vrednost, ki je naslednja po vrsti glede na vrednost v vozlišču vozlišče

Podrobneje si bomo ogledali operacije iskanja, vstavljanja in brisanja.

2.5.1. Iskanje

Dvojiška drevesa se tipično uporabljajo za shranjevanje podatkov, kjer potrebujemo hitro iskanje. Mi se bomo omejili na drevesa, ki za podatke hranijo števila.

Pri iskanju elementa v iskalnem dvojiškem drevesu najprej iskani element primerjamo s korenom. Če se podatka ujemata je naloga končana. Če je podatek manjši od korena, nadaljujemo z iskanjem v levem poddrevesu, sicer v desnem.

Algoritem za iskanje elementa x v iskalnem dvojiškem drevesu s korenom koren:

```
algoritem iskanje(d, x);
začni
      dokler d ni prazno
            če(d.koren == x) vrni true;
            če(d.koren < x)
                  d = d.levoPoddrevo;
            če(d.koren > x)
                  d = d.desnoPoddrevo;
      vrni false;
končaj
```

Za primer vzemimo kar iskalno dvojiško drevo s slike Slika 18. V danem drevesu želimo poiskati element 33. Postopek iskanja elementa:

- 33 primerjamo s podatkom v korenu. Ker je 33>25, se pomaknemo v desno poddrevo.
- Naslednje je vozlišče s podatkom 40. Ker je 33<40, gremo v levo poddrevo.
- Pridemo do vozliča s podatkom 33. Ker je ta podatek enak iskanemu, smo našli iskani element in z iskanjem zaključimo.

2.5.2. Minimum in maksimum

Če je dano splošno drevo, moramo pri iskanju najmanjšega in največjega podatka v drevesu pregledati vsak element. Pri iskalnem dvojiškem drevesu je iskanje teh dveh elementov načeloma lažje in hitrejše.

Ker so vsi podatki v levem poddrevesu manjši od korena bomo najmanjši podatek (minimum) iskali v levem poddrevesu. V primeru, ko koren nima levega poddrevesa, je najmanjši podatek kar podatek v korenu. Vemo, da je levo poddrevo spet iskalno dvojiško drevo, torej postopek ponovimo. V danem iskalnem dvojiškem drevesu se torej pomikamo v levo, dokler ne pridemo do najbolj levega vozlišča drevesa. Podatek v njem je iskani minimum. Pomikamo se torej v levo poddrevo toliko časa, da iskalno dvojiško drevo nima levega poddrevesa. Takrat je najmanjši podatek kar podatek v korenu drevesa.

Podobno velja pri iskanju največjega podatka v iskalnem dvojiškem drevesu (maksimum). Ker so vsi podatki v desnem poddrevesu, ki je tudi iskalno dvojiško drevo, večji od korena, bomo največji podatek zagotovo našli v desnem poddrevesu. To bo najbolj desno vozlišče v drevesu. Tudi tu velja, da je v primeru, ko drevo nima desnega poddrevesa, največji podatek kar podatek v korenu drevesa.

Algoritem za iskanje minimuma:

```
algoritem minimum(d);
začni
      če d == prazno
            vrži izjemo »Drevo je prazno.«;
      sicer
            dokler d ni prazno
                  min = d.koren;
                  d = d.levoPoddrevo;
      vrni min;
končaj
```

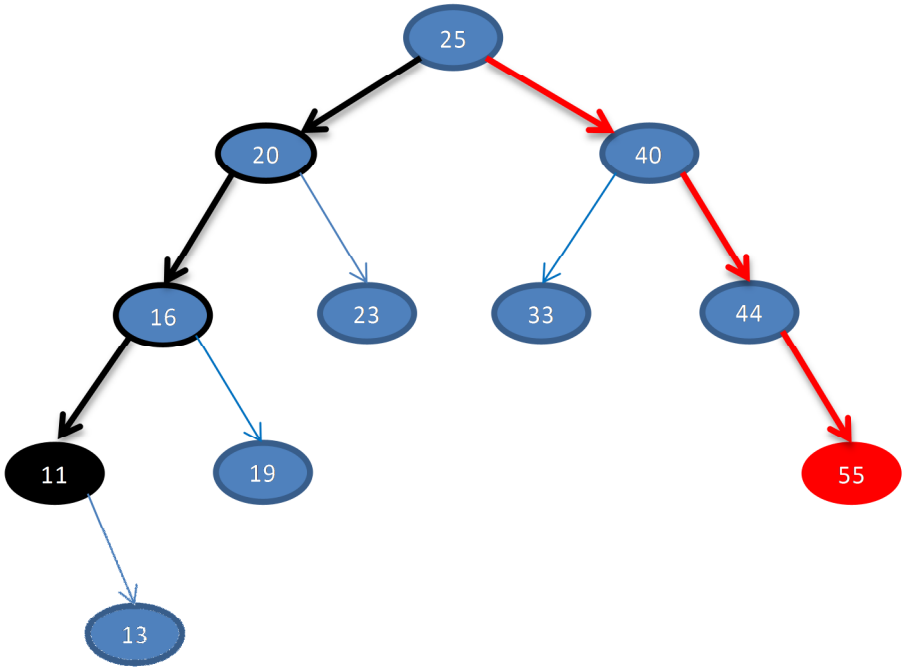
Algoritem `minimum` v iskalnem dvojiškem drevesu *d* poišče minimum. Če je drevo prazno, vrže izjemo *»Drevo je prazno.«*. Sicer se pomikamo po drevesu do skrajno levega vozlišča. Ko pridemo do vozlišča, ki nima levega sina, algoritem vrne podatek v tem vozlišču (*min*). To je minimum.

Algoritem za iskanje maksimuma:

```
algoritem maksimum(d);
začni
      če d == prazno
            vrži izjemo »Drevo je prazno.«;
      sicer
            dokler d ni prazno;
                  max = d.koren;
                  d = d.desnoPoddrevo;
      vrni max;
končaj
```

Algoritem `maksimum` v iskalnem dvojiškem drevesu *d* poišče največjo vrednost v tem drevesu. Če je drevo prazno, vrže izjemo *»Drevo je prazno.«*. Sicer se pomikamo po drevesu do skrajno desnega vozlišča. Algoritem vrne podatek v tem vozlišču (*max*). To je maksimum.

Primer:



Slika 19: Iskanje minimuma in maksimuma v iskalnem dvojiškem drevesu

V iskalnem dvojiškem drevesu na sliki Slika 19 je s črno barvo označena pot, po kateri smo prišli do minimuma (11) in z rdečo barvo pot, po kateri smo prišli do maksimuma (55).

2.5.3. Vstavljanje

Pri običajnih dvojiških drevesih operacije vstavi nimamo. Razlog je v tem, da v dvojiškem drevesu ni pravila o tem, kako so podatki v vozliščih razporejeni. Zato ne vemo, na katero mesto bi vstavili novo vozlišče. Pri iskalnem dvojiškem drevesu pa je točno določeno, kam spada podatek, ki ga vstavljamo. Če na primer v iskalno dvojiško drevo na sliki Slika 19 vstavljajmo podatek 42, vemo, da ga moramo dodati v vozlišče, ki je levi sin vozlišča s podatkom 44.

V primeru, da je drevo prazno, vstavimo podatek v koren in to je naše novo drevo. Kadar drevo ni prazno, moramo poiskati ustrezno mesto, kamor bomo podatek vstavili. Najprej preverimo, ali je podatek manjši od korena – v tem primeru, ga vstavimo na primerno mesto v levo poddrevo. Če pa je podatek večji, ga vstavimo v desno poddrevo.

Algoritem:

```
algoritem vstavi(d, x);
//v iskalno dvojiško drevo d vstavimo podatek x
začni
      če d == prazno;
            vrni d(x);
            //drevo ima eno samo vozlišče-koren s podatkom x
      sicer
            če x < d.podatek
                  d.levoPoddrevo = vstavi(d.levoPoddrevo,x);
            //če je podatek, ki ga vstavljamo, manjši od podatka v
            trenutnem vozlišču, ga vstavimo v levo poddrevo
            če x > d.podatek
```

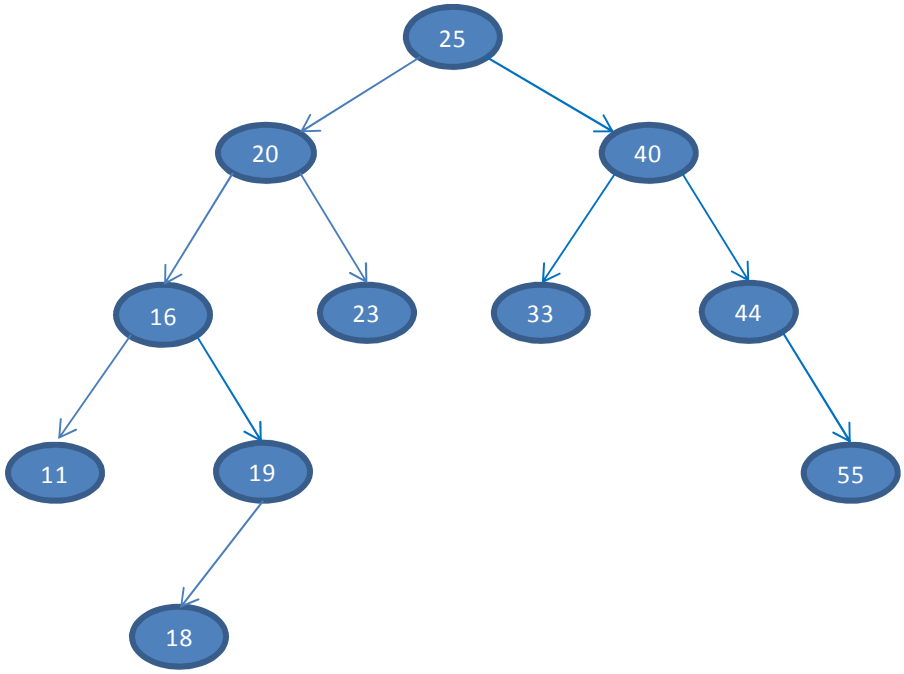
```
                d.desnoPoddrevo = vstavi(d.desnoPoddrevo,x);
            //če je podatek, ki ga vstavljamo, večji od podatka v
            trenutnem vozlišču, ga vstavimo v desno poddrevo
      vrni d;
      //algoritem vrne novo iskalno dvojiško drevo
končaj
```

Primer:

V iskalno dvojiško drevo na sliki Slika 18 želimo vstaviti podatek 18. Postopek je naslednji:

- Podatek 18 primerjamo s podatkom v korenu. Ker je 18<25, se pomaknemo v levo poddrevo.
- Naslednji podatek je 20. Ker je 18<20, postopek ponovimo.
- Pridemo do vozliča s podatkom 16. Ker je 18>16, se pomaknemo v desno.
- V tem koraku pridemo do lista. Ker je 18<19, vstavimo podatek v levo poddrevo lista.

Novo iskalno dvojiško drevo je prikazano na sliki Slika 20.



Slika 20: Iskalno dvojiško drevo po vstavljanju

2.5.4. Brisanje

Pri brisanju podatka iz iskalnega dvojiškega drevesa je pomembno, da takrat, ko podatek izbrišemo, ohranimo lastnost iskalnega dvojiškega drevesa.

Pri brisanju podatka iz drevesa imamo 3 možnosti:

- Če je vozlišče, ki ga želimo izbrisati, list, ni težav z brisanjem. Enostavno ga izbrišemo in v očetu označimo, da nima več ustreznega poddrevesa (oziroma natančneje, da je to poddrevo prazno).

- Če ima vozlišče le enega sina, ga nadomestimo z njim. Kazalec očeta brisanega vozlišča popravimo tako, da kaže na sina tega vozlišča. Vozlišče lahko sedaj izbrišemo.
- Če ima vozlišče tako levega kot desnega sina, je brisanje vozlišča nekoliko bolj zapleteno. Vozlišče nadomestimo z najmanjšim elementom v desnem poddrevesu ali pa z največjim elementom v levem poddrevesu.

Algoritem:

```
algoritem brisanje(d, x);
//iz iskalnega dvojiškega drevesa d brišemo podatek x
začni
      če d je prazno
            vrži izjemo »Drevo je prazno.«;
      če x < d.koren
      // če je podatek, ki ga želimo izbrisati manjši od podatka v
      // trenutnem vozlišču, brišemo v levem poddrevesu
            d.levoPoddrevo = brisanje(d.levoPoddrevo,x);
      sicer
      // brišemo v desnem poddrevesu
            če x > d.koren
                  d.desnoPoddrevo = brisanje(d.desnoPoddrevo,x);
            sicer
            //našli smo vozlišče, ki ga brišemo – to je koren
            če d.desnoPoddrevo in d.levoPoddrevo nista prazni
                  d.koren = minimum(d.desnoPoddrevo);
                  d.desnoPoddrevo = brisanjeMinimum(d.desnoPoddrevo);
                  // vozlišče nadomestimo z najmanjšim elementom v
                  // desnem poddrevesu
                  // ali z največjim elementom v levem poddrevesu
                  // d.koren = maksimum(d.levoPoddrevo);
                  // d.levoPoddrevo =brisanjeMaksimum(d.levoPoddrevo);
            sicer
                  če d.levoPoddrevo ni prazno
                        d = d.levoPoddrevo;
                  sicer
                        d = d.desnoPoddrevo;
vrni d; //algoritem vrne novo iskalno dvojiško drevo
končaj
```

Algoritem *brisanje* iz iskalnega dvojiškega drevesa *d* izbriše podatek *x*. Če je drevo prazno, algoritem vrže izjemo »Drevo je prazno.«. Sicer preverimo podatek v trenutnem vozlišču. Začnemo s korenom. Če je podatek, ki ga želimo izbrisati, manjši od podatka v korenu, s postopkom brisanja nadaljujemo v levem poddrevesu, sicer v desnem. Ko pridemo do vozlišča s podatkom *x*, podatek izbrišemo. Če je vozlišče s tem podatkom list, ga izbrišemo brez nadaljnjega preurejanja drevesa. Sicer *x* nadomestimo z najmanjšim podatkom v desnem poddrevesu (algoritem *minimum* na strani 15) ali največjim podatkom v levem poddrevesu (algoritem *maksimum*). Vozlišče, iz katerega smo podatek prenesli, izbrišemo iz drevesa (pomožna algoritma *brisanjeMinimum* ali *brisanjeMaksimum*) . Algoritem vrne novo iskalno dvojiško drevo.

Pomožna algoritma za brisanje najmanjšega in največjega elementa iz iskalnega dvojiškega drevesa *d*:

```
//algoritem za brisanje najmanjšega elementa iz dvojiškega drevesa
```

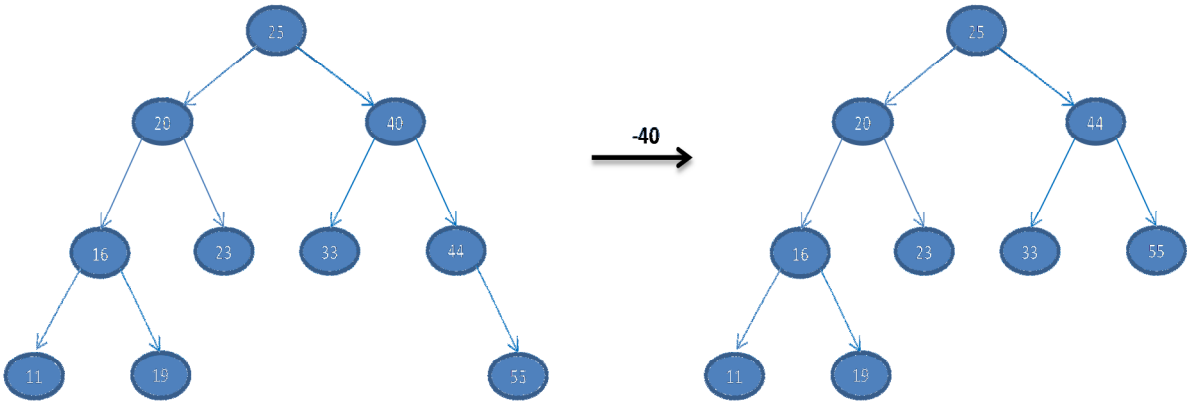
```
algoritem brisanjeMinimum(d);
začni
      če d je prazno
            vrži izjemo »Drevo je prazno.«;
      sicer
      če d.levoPoddrevo ni prazno
            d.levoPoddrevo = brisanjeMinimum(d.levoPoddrevo);
            vrni d;
      sicer // brišemo podatek v korenu drevesa, ki nima levega
             // poddrevesa, torej dobimo desno poddrevo
      vrni d.desnoPoddrevo;
končaj


//algoritem za brisanje največjega elementa iz dvojiškega drevesa
algoritem brisanjeMaksimum(d);
začni
      če d je prazno
            vrži izjemo »Drevo je prazno.«;
      če d.desnoPoddrevo ni prazno
      // novo desno poddrevo bo torej prvotno desno poddrevo brez
      // maksimalnega elementa
            d.desnoPoddrevo = brisanjeMaksimum(d.desnoPoddrevo);
            vrni d;
      sicer // desno poddrevo je prazno, brišemo torej koren
             // ostane le levo poddrevo
vrni d.levoPoddrevo;
končaj
```

Primer:

Na primeru poglejmo, kako deluje naš algoritem *brisanje* po posameznih korakih. Denimo, da želimo iz iskalnega dvojiškega drevesa na sliki Slika 18 odstraniti element 40.

Najprej preverimo, da drevo ni prazno. Vidimo, da ni, zato je naša naloga poiskati vozlišče s podatkom, ki ga želimo izbrisati (40). Primerjamo ga s podatkom v korenu. Ker je podatek, ki ga želimo izbrisati, večji od podatka v korenskem vozlišču, bomo brisali v desnem poddrevesu. Pridemo do vozlišča s podatkom 40. To vozlišče ima tako levega kot desnega sina, zato podatek nadomestimo z najmanjšim elementom v desnem poddrevesu. Pri tem vozlišče s tem najmanjšim podatkom zbrišemo.

Tako dobimo novo iskalno dvojiško drevo na sliki Slika 21.



Slika 21: Iskalno dvojiško drevo pred in po brisanju

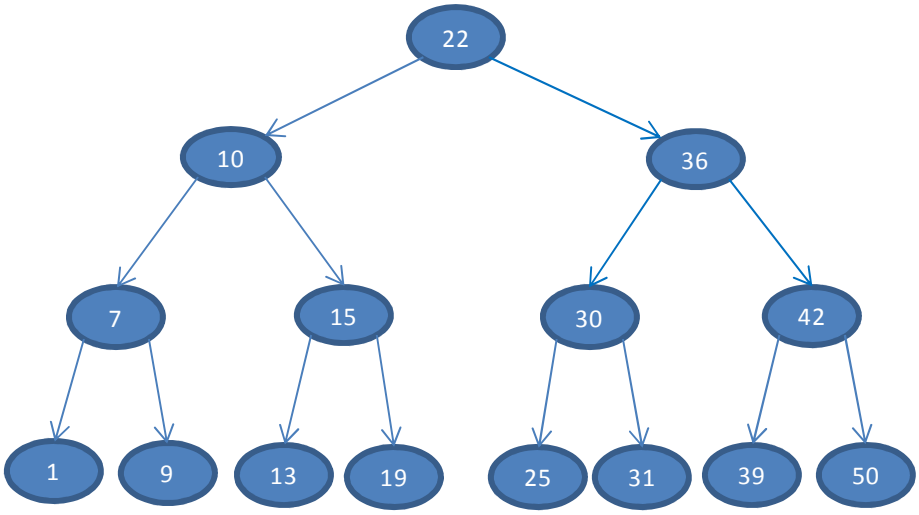
2.6. Časovna zahtevnost algoritmov

Časovna zahtevnost vseh treh algoritmov (`iskanje`, `vstavljanje` in `brisanje`) je odvisna od višine drevesa. Višina iskalnega dvojiškega drevesa z n vozlišči je najmanj $\log_2(n+1)$ in največ n. Torej:

- Časovno najugodnejša oblika iskalnega dvojiškega drevesa za iskanje, vstavljanje in brisanje elementov je v primeru polnega ali »vsaj« levo poravnanega iskalnega dvojiškega drevesa. V tem primeru je časovna zahtevnost zgoraj navedenih algoritmov logaritemska (glede na število vozlišč v drevesu).
- Časovna zahtevnost algoritmov nad izrojenimi drevesi je linearna glede na število podatkov v drevesu.

Tukaj omenimo še, da je polno dvojiško drevo mogoče dobiti le, če gradimo drevo z določenim številom podatkov, torej, če imamo 1, 3, 7, 15, 31, 63, 127, 255, 511,… podatkov. Ustrezno levo poravnano dvojiško drevo pa je mogoče sestaviti za vsako število podatkov.

Primer polnega iskalnega dvojiškega drevesa višine 4, ki je seveda tudi levo poravnano, je prikazan na sliki Slika 22.



Slika 22: Polno iskalno dvojiško drevo

Časovna zahtevnost algoritmov nad izrojenimi drevesi je linearna glede na število podatkov v drevesu.

Seveda si želimo, da bi bilo naše iskalno dvojiško drevo polno in s tem časovna zahtevnost algoritmov nad tem drevesom čim bolj ugodna. Četudi imamo na začetku polno dvojiško drevo, pa se lahko z vstavljanjem in brisanjem podatkov, njegova struktura hitro spremeni. V najslabšem primeru dobimo izrojeno ali izrojenemu podobno drevo in posledično linearno (in ne več logaritemsko) časovno zahtevnost pri osnovnih algoritmih.

Eden od načinov, da bi ves čas zagotovili ugodno časovno zahtevnost algoritmov je, da v vozliščih hranimo spremenljivo število podatkov. Takrat govorimo o večsmernih drevesih.

3. VEČSMERNA IN URAVNOTEŽENA DREVESA

Do sedaj smo se v diplomski nalogi ukvarjali z dvojiškimi iskalnimi drevesi. Osnovna značilnost iskalnih dvojiških dreves je, da vsako od vozlišč nosi informacijo, na osnovi katere se pregled drevesa nadaljuje v levem ali desnem poddrevesu. Kadar so takšna drevesa zelo velika, morajo biti shranjena na disku. Takrat je poseg v vozlišče (torej dostop do podatka) razmeroma dolgotrajen v primerjavi z obdelavo informacije v njem. Zato bi želeli z enim samim posegom na disk prenesti več informacij. Vsakič, ko dostopamo do podatkov na disku, želimo doseči večjo skupino elementov brez dodatnega dela in časa. Zato v posameznih vozliščih hranimo več podatkov. Ker pa želimo še vedno ohraniti možnost učinkovitega iskanja, to pomeni, da se pot iz vozlišča nadaljuje v več kot dveh smereh. Drevesom, kjer so vozlišča zgrajena tako, da je iskanje možno v več kot dveh smereh, rečemo večsmerna drevesa.

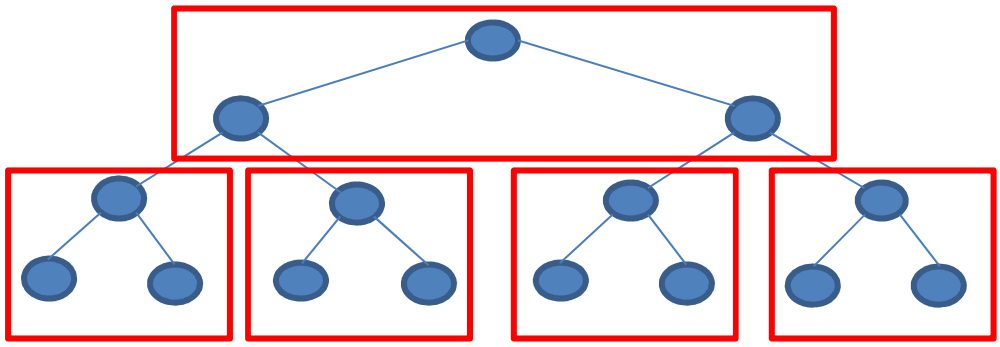
3.1. Večsmerna drevesa

Večsmerna drevesa v nasprotju z dvojiškimi drevesi, ki vsebujejo le en podatek, v vozlišču vsebujejo več podatkov.

Uporaba večsmernih dreves je pomembna pri gradnji in vzdrževanju velikih iskalnih dreves, kjer sta potrebni operaciji vstavljanja in izločanja (brisanja).

Če uporabimo dvojiška drevesa, vemo, da za predstavitev n podatkov v najslabšem primeru potrebujemo drevo višine n. Za milijon podatkov bi v najslabšem primeru torej potrebovali drevo višine 10^6 in prav toliko dosegov na disk. Če pa za množico podatkov z milijon elementi uporabimo idealno iskalno dvojiško drevo (npr. levo poravnano iskalno dvojiško drevo), potem je potrebnih $\log_2 10^6 = 20$ iskalnih korakov. Pri tem vsak korak terja en doseg do diska. Želeli bi si tako organizacijo podatkov, ki bi terjala še manj dosegov.

To nas pripelje do delitve drevesa na poddrevesa, pri čemer poddrevesa delujejo kot celote, ki jih dosegamo na enkrat. Takim poddrevesom bomo v nadaljevanju večkrat rekli strani.

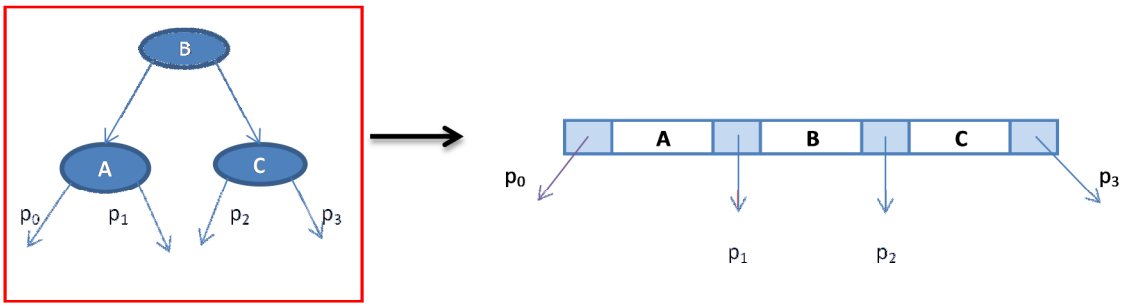


Slika 23: Polno dvojiško drevo, razdeljeno na strani

Na sliki Slika 23 je prikazano polno dvojiško drevo s 15 elementi, ki je razdeljeno na 5 strani. Na vsaki strani so 3 vozlišča. Sedaj terja le vsaka stran en doseg. Za iskanje podatka bi potrebovali torej le en, oziroma največ dva posega na disk.

Sedaj stran iskalnega dvojiškega drevesa stisnemo v eno samo vozlišče. Podatke (ključe) naslednikov vstavimo med podatke predhodnikov. Tako dobimo vozlišče, v katerem hranimo večje število

podatkov in kazalcev na naslednike. Podatki si sledijo v naraščajočem vrstnem redu od leve proti desni. Med podatke vstavimo še kazalce na sinove (Slika 24).



Slika 24: Stran drevesa, predstavljena kot vozlišče

Dvojiško iskalno drevo s tremi vozlišči smo torej združili v eno samo vozlišče. Vemo, da veljata relaciji A<B in B<C, zato je v vozlišču podatek A pred podatkom B in podatek B pred podatkom C.

Kazalec p_0 je sedaj kazalec na levega sina in kazalec p_1 kazalec na desnega sina vozlišča s podatkom A. Podobno je p_2 kazalec na levega sina in kazalec p_3 kazalec na desnega sina vozlišča s podatkom C. Če torej iščemo podatek, ki je večji kot A in manjši kot B, bomo šli v smeri p_1. Če pa iščemo podatek, večji kot C, bomo nadaljevali po p_2.

Poglejmo si primer, ko je dano idealno po višini uravnoteženo iskalno drevo z milijon vozlišči. Na vsaki strani drevesa naj bo 100 vozlišč. Takrat potrebujemo le

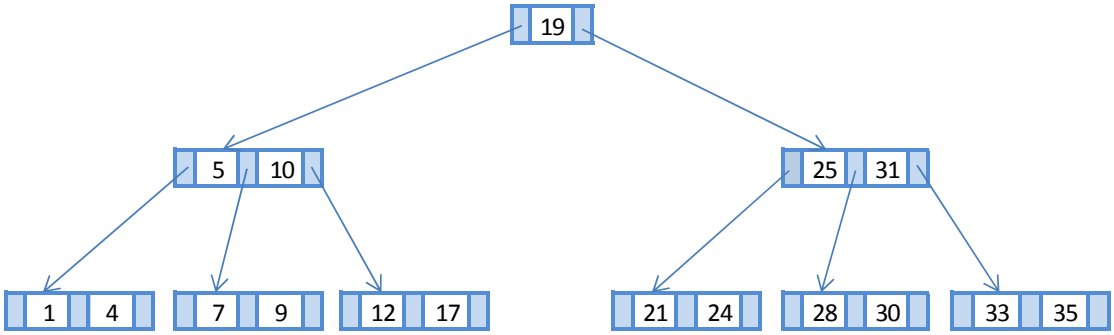
$$\log_{100} 10^6 = 3$$

dosege do diska in ne več 20.

V primeru, ko dopustimo, da drevo raste naključno, v najslabšem primeru (ko je drevo izrojeno) še vedno potrebujemo 10^4 dosegov. Tako vidimo, da tudi pri večsmernih drevesih nujno potrebujemo shemo nadzirane rasti.

Za *m*-smerno iskalno drevo velja, da je bodisi prazno ali pa je višine vsaj 2. Vsako notranje vozlišče ima najmanj *m*/2 in največ *m* sinov ter en podatek manj kot ima sinov.

Primer večsmernega iskalnega drevesa:



Slika 25: Večsmerno iskalno drevo reda 3

Na sliki Slika 25 je prikazano večsmerno iskalno drevo reda 3. Vsako vozlišče drevesa (razen korena) ima največ 2 elementa in največ 3 sinove.

Sedaj pa si poglejmo osnovne operacije iskanja, vstavljanja in brisanja nad večsmernimi drevesi.

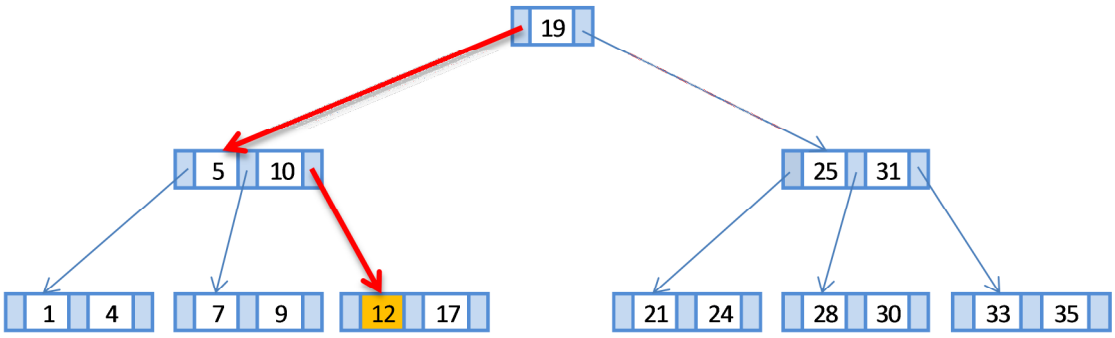
3.1.1. Osnovne operacije nad večsmernimi drevesi

3.1.1.1. Iskanje

Tako kot v iskalnem dvojiškem drevesu, so tudi v večsmernih iskalnih drevesih podatki urejeni. Vsako vozlišče večsmernega drevesa lahko vsebuje večje število podatkov in ima več naslednikov. Število podatkov v vozlišču in število naslednikov vozlišč je odvisno od reda drevesa.

Postopek iskanja v večsmernem iskalnem drevesu si poglejmo na primeru. Denimo, da v drevesu na sliki Slika 25 iščemo podatek 12. Iskani podatek najprej primerjamo s podatkom v korenu. Ker je 12 < 19, pot iskanja nadaljujemo v levem poddrevesu. Ker sta oba podatka (5 in 10) manjša od iskanega, nadaljujemo v vozlišču, na katerega kaže desni kazalec podatka 10. Prvi podatek v tem vozlišču (12) je enak iskanemu. Iskani element smo torej našli, zato lahko s postopkom iskanja zaključimo.

Pot iskanja podatka 12 v večsmernem drevesu s slike Slika 25 je prikazana na sliki Slika 26.



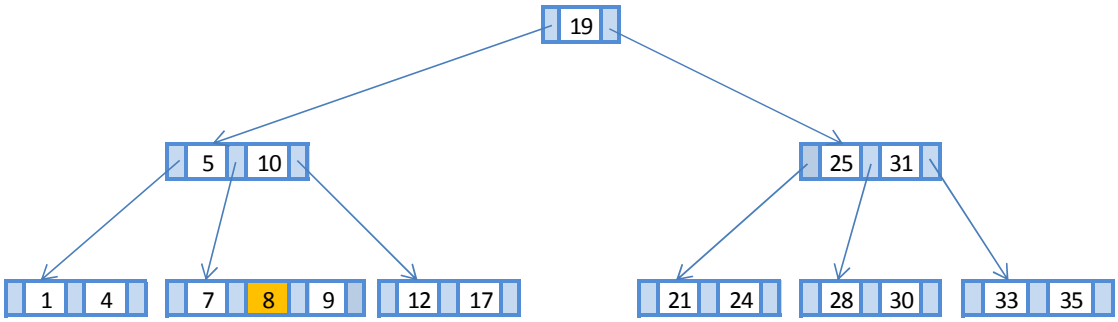
Slika 26: Pot iskanja v večsmernem drevesu

3.1.1.2. Vstavljanje

Podatke v večsmerno drevo vedno vstavljamo v liste drevesa. Najprej moramo poiskati ustrezno mesto, kamor bomo podatek vstavili. Pri tem upoštevamo, da si podatki v listu sledijo od najmanjšega do največjega.

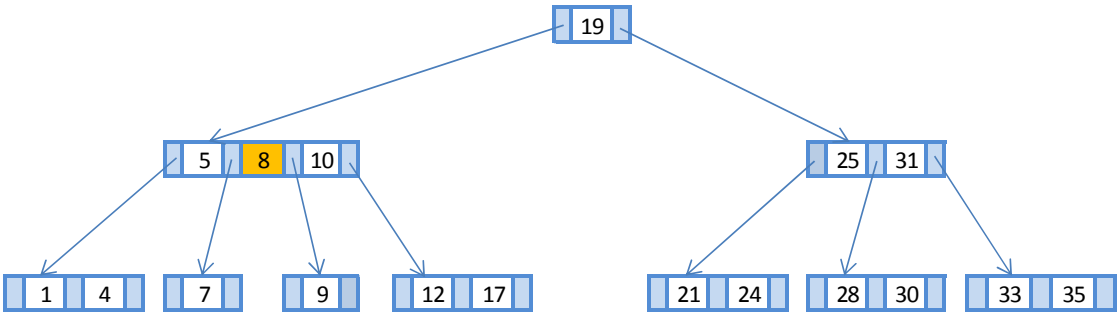
Ko podatek vstavimo na ustrezno mesto v listu, preverimo, kakšno je število elementov v tem listu. To ne sme presegati zgornje meje glede števila elementov v posameznem vozlišču večsmernega drevesa. V primeru, ko je v vozlišču preveč elementov, moramo vozlišče razdeliti.

Podrobneje si postopek vstavljanja podatka v večsmerno iskalno drevo poglejmo na primeru. V drevo na sliki Slika 25 želimo vstaviti podatek 8. Najprej poiščemo ustrezni list, kamor bomo podatek vstavili. To je list, na katerega kaže levi kazalec podatka 10 (Slika 27).



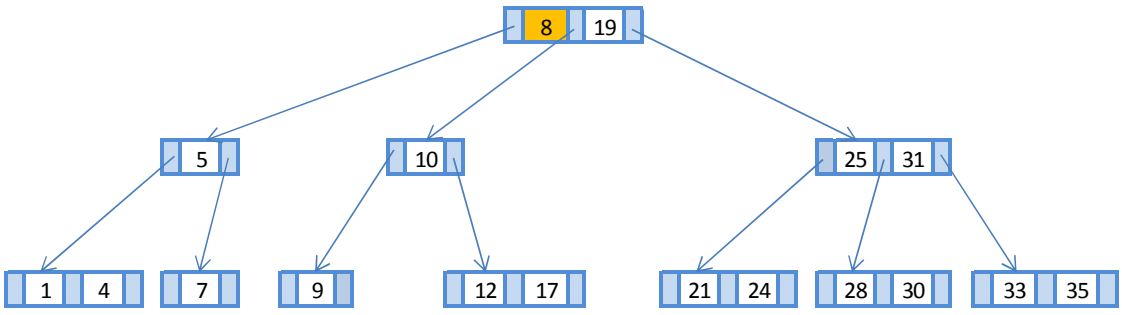
Slika 27: Vstavljanje podatka v večsmerno iskalno drevo

Po vstavljanju je število podatkov v vozlišču (listu) preseglo zgornjo mejo. Drevo moramo zato preurediti. To storimo tako, da list razdelimo na dva dela, srednji podatek pa prestavimo v očeta (Slika 28).



Slika 28: Delitev vozlišča pri postopku vstavljanja v večsmerno iskalno drevo

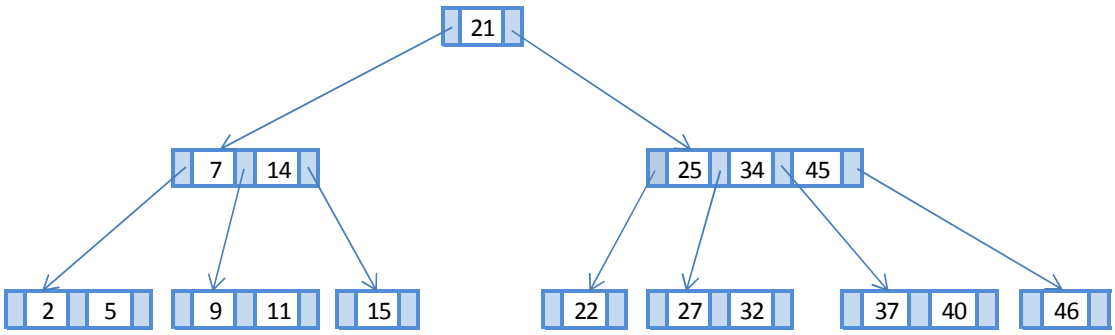
Ko podatek prestavimo v očeta lista, ki smo ga razpolovili, število podatkov v očetu preseže zgornjo mejo. Postopek delitve vozlišča ponovimo. Podatek 8 prestavimo nivo višje, v koren drevesa. tako dobimo novo večsmerno iskalno drevo na sliki Slika 29.



Slika 29: Večsmerno iskalno drevo reda 3 po vstavljanju podatka 8

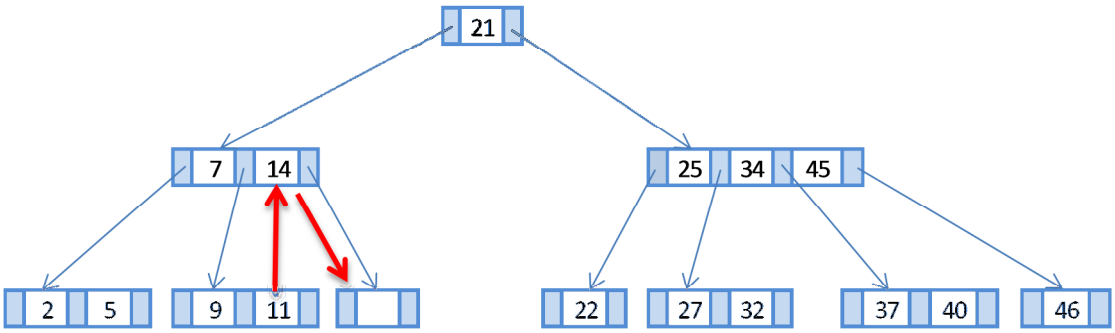
3.1.1.3. Brisanje

Podobno kot pri postopku vstavljanja, se tudi postopek brisanja podatka iz večsmernega drevesa izvaja v listih drevesa. Kadar je podatek, ki ga brišemo v listu, ga enostavno izbrišemo. Če je v listu še vedno ustrezno število podatkov, postopek brisanja zaključimo. Kadar pa število podatkov pade pod spodnjo mejo, moramo drevo preoblikovati. Tak primer brisanja podatka si poglejmo na primeru.



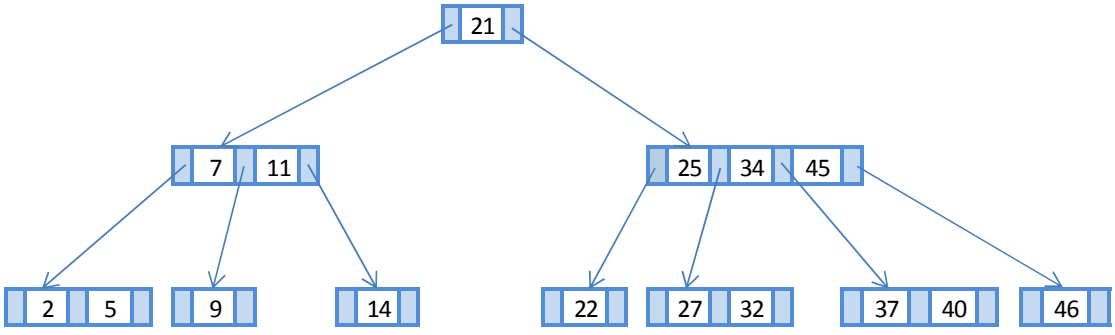
Slika 30: Večsmerno iskalno drevo reda 4

Iz večsmernega iskalnega drevesa na sliki Slika 30 želimo izbrisati podatek 15. Ko podatek izbrišemo, postane vozlišče (list) prazno. Ker njegov levi brat vsebuje več kot minimalno število podatkov, prestavimo enega izmed podatkov v očeta, podatek iz očeta pa v prazen list (Slika 31).



Slika 31: Preurejanje večsmernega iskalnega drevesa po brisanju podatka

Tako dobimo večsmerno iskalno drevo na sliki Slika 32, ki ustreza vsem zahtevam večsmernega iskalnega drevesa reda 4.



Slika 32: Večsmerno iskalno drevo po brisanju

Posebna oblika večsmernih iskalnih dreves so B-drevesa. Podrobneje so operacije iskanja, vstavljanja in brisanja nad večsmernimi iskalnimi drevesi opisane v poglavju o B-drevesih (razdelek 4.3).

3.2. Uravnotežena drevesa

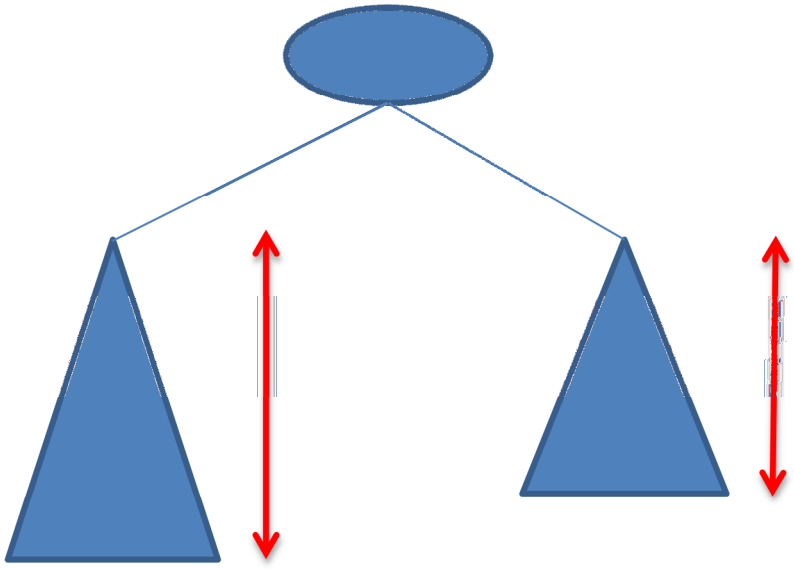
Da bo drevo raslo nadzorovano, bomo morali skrbeti, da bo v vsakem levem poddrevesu približno enako število elementov kot v desnem. Ta kriterij je dosežen pri polnih drevesih (glej razdelek 2.1 - Slika 15). Takrat govorimo o popolnem ravnotežju. Ker pa nimamo vedno idealno število podatkov, lahko »dovolimo«, da je drevo skoraj polno. To pomeni, da je drevo višine *v* na nivoju *v-1* polno, na zadnji nivo pa razporedimo ustrezno število vozlišč. Če še v zadnji nivo »uvedemo red«, kar pomeni, da si tu vozlišča zaporedno sledijo od leve proti desni ali od desne proti levi, dobimo levo ali desno poravnana drevesa.

Žal v dinamičnem sestavljanju drevesa ne moremo takšne strukture ohranjati brez zapletenega dodatnega dela, zato samo ravnotežje definiramo nekoliko manj strogo.

Kot smo videli v poglavju o iskalnih dvojiških drevesih, je časovna zahtevnost osnovnih operacij (iskanje, vstavljanje in brisanje) nad iskalnim dvojiškim drevesom odvisna od višine drevesa (glej razdelek 2.4). Časovna zahtevnost je v najboljšem primeru (kadar je drevo polno ali levo poravnano) logaritemska in v najslabšem primeru (ko je drevo izrojeno) linearna glede na število elementov v drevesu. Seveda bi se primerom, ko je časovna zahtevnost algoritmov linearna, radi izognili. Zato moramo ves čas paziti, da drevo ne postane izrojeno. To najlažje storimo tako, da pri vsakem vstavljanju ali brisanju elementa v drevesu, preverimo njegovo strukturo in jo po potrebi spremenimo.

3.2.1. Definicija uravnoteženega drevesa

Dvojiško drevo je k-uravnoteženo, kadar se v vsakem vozlišču višini levega in desnega poddrevesa razlikuje za največ k in ko sta obe poddrevesi k-uravnoteženi. Če je k=1, izrazoslovje poenostavimo in namesto 1-uravnoteženo rečemo, da je drevo uravnoteženo.



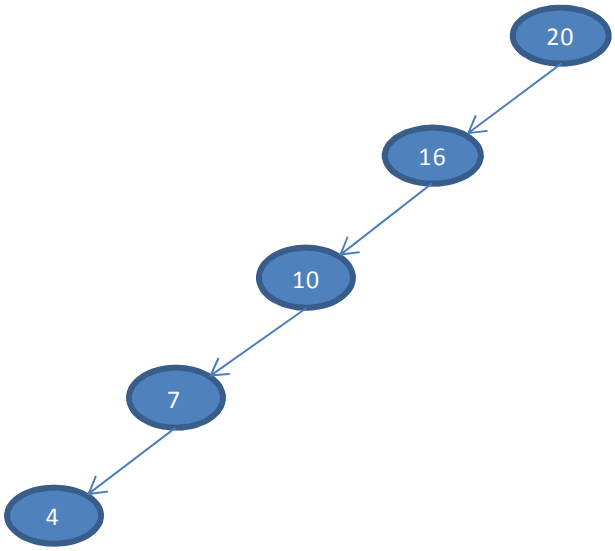
Slika 33: Shema drevesa, kjer je levo poddrevo višje od desnega

Uravnoteženo dvojiško drevo je torej ali prazno ali pa je to dvojiško drevo, v katerem sta levo in desno poddrevo obe uravnoteženi in se njuni višini razlikujeta kvečjemu za 1,

$$|\text{visina}(\text{d.levoPoddrevo}()) - \text{visina}(\text{d.desnoPoddrevo}())| \leq 1.$$

Pojem uravnoteženega drevesa sta leta 1962 vpeljala Adelson-Velskii in Landis . Iskalnim dvojiškim drevesom, ki so 1-uravnotežena, rečemo AVL drevesa.

Uravnotežena drevesa se zaradi razmeroma preproste izvedbe in sprejemljive časovne zahtevnosti algoritmov nad njimi pogosto uporabljajo tam, kjer je treba zagotoviti predvidljiv odzivni čas ali tam, kjer lahko pričakujemo neugodna zaporedja operacij vstavljanja in brisanja podatkov. Če bi uporabili klasičen algoritem za vstavljanje v iskalno dvojiško drevo (glej razdelek 2.5.3) in iskalno drevo polnili s podatki, ki so urejeni po velikosti, bi dobili izrojeno drevo.



Slika 34: Izrojeno drevo

Če smo v prazno iskalno dvojiško drevo zaporedoma vstavljali podatke 20, 16, 10, 7 in 4, smo dobili v levo izrojeno drevo, ki je podobno enojno povezanemu verižnemu seznamu (Slika 34). Časovna zahtevnost večine algoritmov nad tem drevesom je linearna. Temu se želimo izogniti, zato bo takšno drevo potrebno preoblikovati.

Na tem mestu moramo omeniti tudi pojem ravnotežnega faktorja, s katerim se bomo srečali v nadaljevanju diplomske naloge.

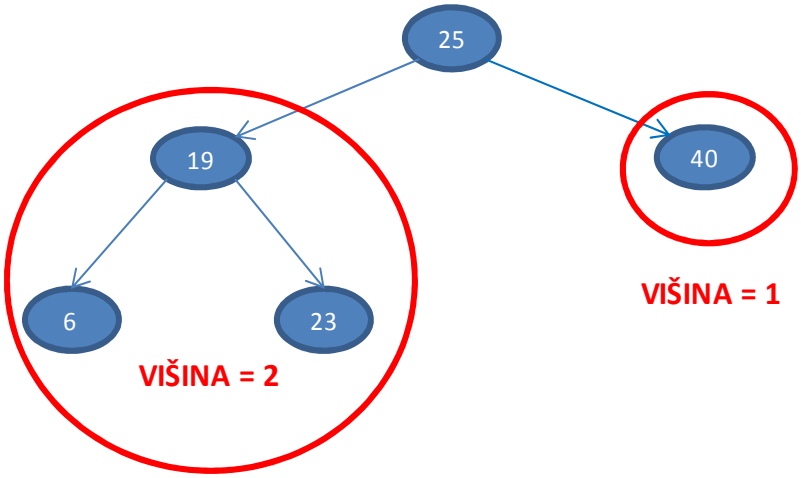
3.2.2. Ravnotežni faktor

Pri operacijah nad uravnoteženimi drevesi, ki si jih bomo podrobneje pogledali v nadaljevanju, si pomagamo z ravnotežnim faktorjem. Tega določimo vsakemu vozlišču drevesa. Izračunamo ga tako, da odštejemo višino desnega poddrevesa od višine levega poddrevesa.

$$f = višina(d.levoPoddrevo()) - višina(d.desnoPoddrevo())$$

Kadar je ravnotežni faktor -1, 0 ali 1, je drevo uravnoteženo, sicer pa ne.

Primer uravnoteženega drevesa je prikazan na sliki Slika 35.



Slika 35: Uravnoteženo drevo

Ravnotežni faktor korena drevesa na sliki Slika 35 je 1, saj je višina levega poddrevesa enaka 2, višina desnega poddrevesa pa enaka 1. To po definiciji pomeni, da je drevo v korenu uravnoteženo. Ker sta uravnoteženi tudi obe poddrevesi, je drevo uravnoteženo.

Pri vstavljanju novega ali brisanju obstoječega vozlišča v uravnoteženem drevesu, se lahko ravnotežje poruši. Takrat moramo drevo preoblikovati. Neravnotežje popravimo tako, da del drevesa zavrtimo v levo ali desno. Govorimo o levi oziroma desni rotaciji.

Bistvena razlika med operacijama vstavljanja in brisanja vozlišča v uravnoteženem drevesu je v številu rotacij za preoblikovanje neuravnoteženega drevesa. Med tem, ko vstavljanje podatka lahko povzroči kvečjemu eno vrtenje (dveh ali treh vozlišč), pa lahko zahteva brisanje vozlišča vrtenje pri vsakem vozlišču na poti od izbrisanega vozlišča do korena drevesa.

3.2.3. Rotacije uravnoteženega drevesa

Preden si pogledamo osnovne operacije nad uravnoteženimi drevesi, moramo podrobneje spoznati rotacije, ki jih potrebujemo za preoblikovanje drevesa. Poznamo dve skupini rotacij; kadar gre za enojno vrtenje govorimo o levi ali desni rotaciji. Kadar pa imamo dvojno vrtenje so rotacije naslednje:

- levo-leva rotacija (ali krajše LL rotacija),
- desno-desna (DD rotacija),
- levo-desna (LD rotacija) in
- desno-leva rotacija (DL rotacija).

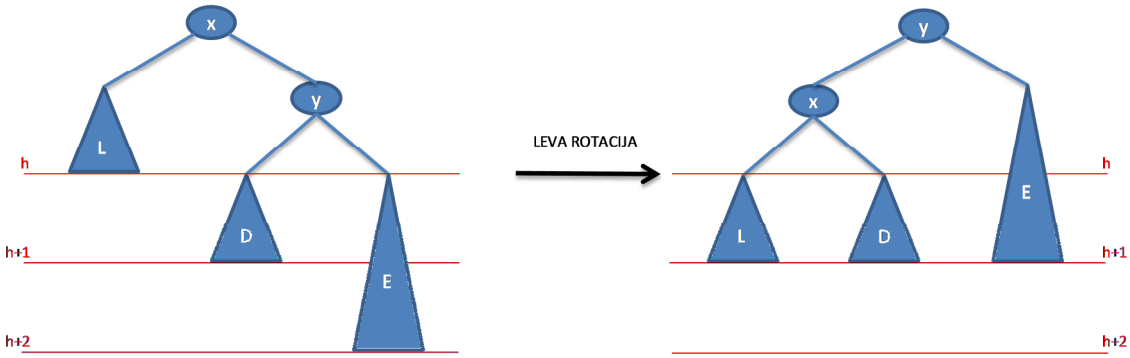
Ker je desna rotacija simetrična levi, imamo v primeru dvojnega »zasuka« dejansko le dva različna tipa rotacij.

Poglejmo si posamezne primere:

- Enojni zasuk
  1. Leva rotacija

Levo rotacijo uporabimo, kadar je ravnotežni faktor nekega vozlišča drevesa enak -2. To vozlišče mora biti edino vozlišče z »napačnim« ravnotežnim faktorjem v poddrevesu, ki mu je »pokvarjeno«

vozlišče koren. Na tem vozlišču opravimo levo rotacijo tako, da njegov desni sin zasede njegovo mesto, sam pa postane njegov levi sin.

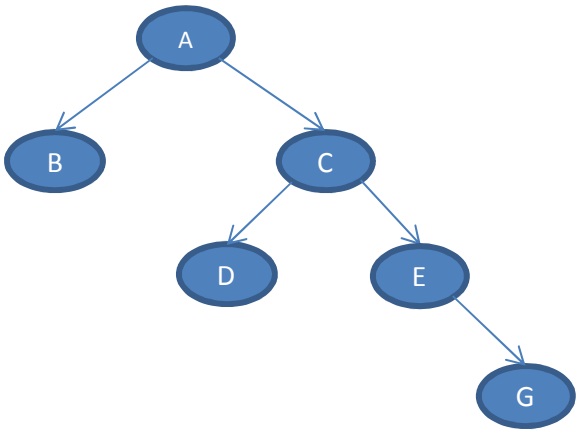


Slika 36: Leva rotacija

Na sliki Slika 36 je prikazano preoblikovanje drevesa s pomočjo leve rotacije. Višini levega in desnega poddrevesa sta se razlikovali za 2, zato drevo ni bilo uravnoteženo. Z zasukom v levo smo drevo preoblikovali tako, da imata levo in desno poddrevo sedaj enaki višini. Koren novonastalega drevesa je sedaj vozlišče s podatkom y, ki ima za svojega levega sina vozlišče s podatkom x. To vozlišče (vozlišče x) ima sedaj za svojega desnega sina poddrevo D, ki je prejšnji levi sin vozlišča y. Pri tem je seveda potrebno spremeniti kazalca v obeh vozliščih.

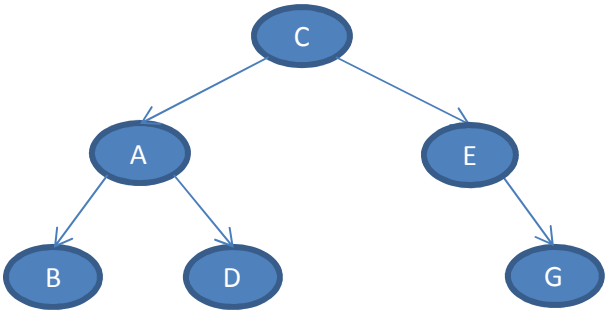
Ker rotacije uporabljamo na iskalnih drevesih, pokažimo še, da se pri tem preoblikovanju lastnost iskalnega drevesa ohrani. Pokazali bomo torej, da so po preoblikovanju elementi v levem poddrevesu manjši in elementi v desnem poddrevesu večji od elementa v korenu.

Predpostavimo, da je drevo na sliki Slika 37 iskalno dvojiško drevo. Zanj torej velja: B<A<D<C<E<G.



Slika 37: Neuravnoteženo iskalno dvojiško drevo

Ker je drevo neuravnoteženo, ga uravnotežimo s pomočjo leve rotacije pri korenu drevesa. Tako dobimo drevo na sliki Slika 38.

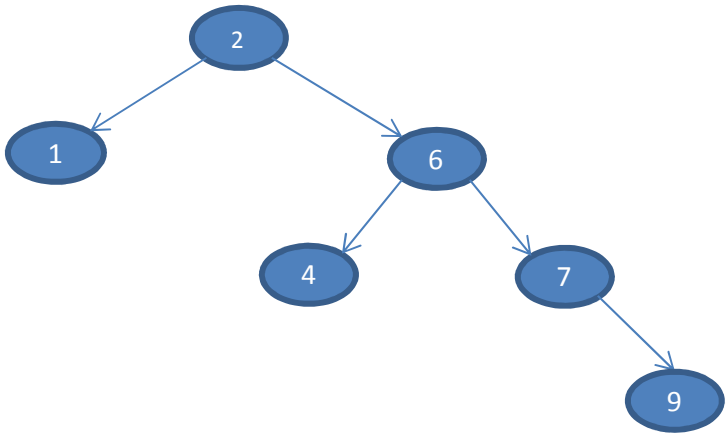


Slika 38: Iskalno dvojiško drevo po levi rotaciji

Za drevo na sliki Slika 38 mora veljati:

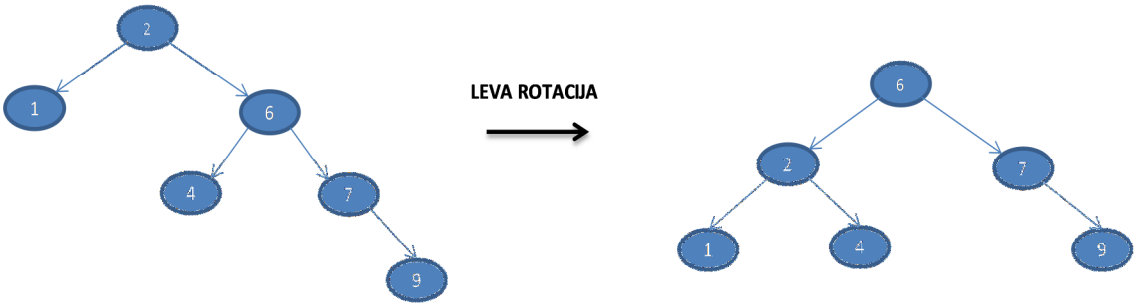
- B<A<D<C. To velja, saj je bilo vozlišče s podatkom B v levem poddrevesu (B<A) in vozlišče s podatkom C v desnem poddrevesu korena s podatkom A (A<C). Vozlišče s podatkom C je imelo za svojega levega sina vozlišče s podatkom D (D<C).
- C<E<G. Vozlišče s podatkom C je imelo za svojega desnega sina vozlišče s podatkom E (C<E), vozlišče s podatkom E pa za svojega desnega sina vozlišče s podatkom G (E<G).

Poglejmo sedaj še konkreten primer. Drevo na sliki Slika 39 je neuravnoteženo, saj je višina levega poddrevesa enaka 1, višina desnega poddrevesa pa 3. Da bomo drevo uravnotežili, uporabimo levo rotacijo pri vozlišču s podatkom 2 (koren drevesa).



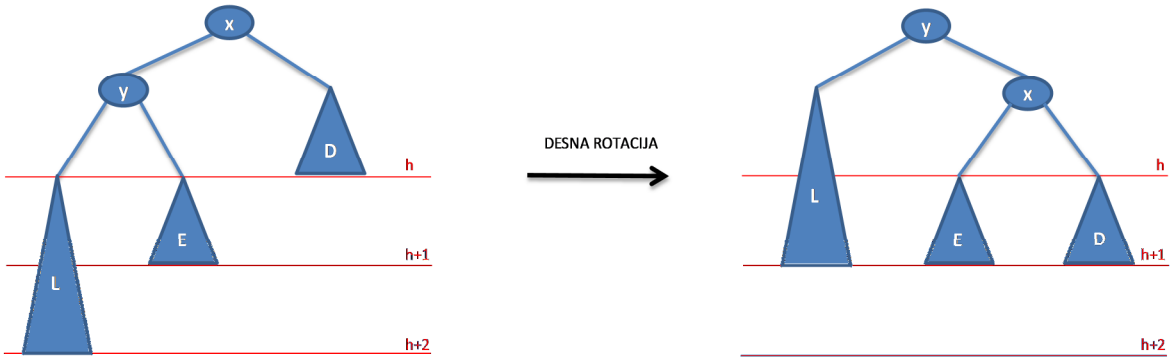
Slika 39: Neuravnoteženo iskalno dvojiško drevo_konkreten primer

Z levo rotacijo drevo preoblikujemo tako, da je koren novega drevesa vozlišče s podatkom 6, ki ima za svojega levega sina vozlišče s podatkom 2 in za svojega desnega sina vozlišče s podatkom 7. Vozlišče s podatkom 2 ima za svojega levega sina list s podatkom 1 in za svojega desnega sina list s podatkom 4. Vozlišče s podatkom 7 pa ima le desnega sina. To je list drevesa s podatkom 9. Za vsako vozlišče drevesa velja, da so vsi podatki v levem poddrevesu manjši in vsi podatki v desnem poddrevesu večji od podatka v korenu. Torej je novo drevo na sliki Slika 40 še vedno iskalno dvojiško drevo, ki je sedaj uravnoteženo.



Slika 40: Uravnoteženo iskalno dvojiško drevo po levi rotaciji

– Desna rotacija

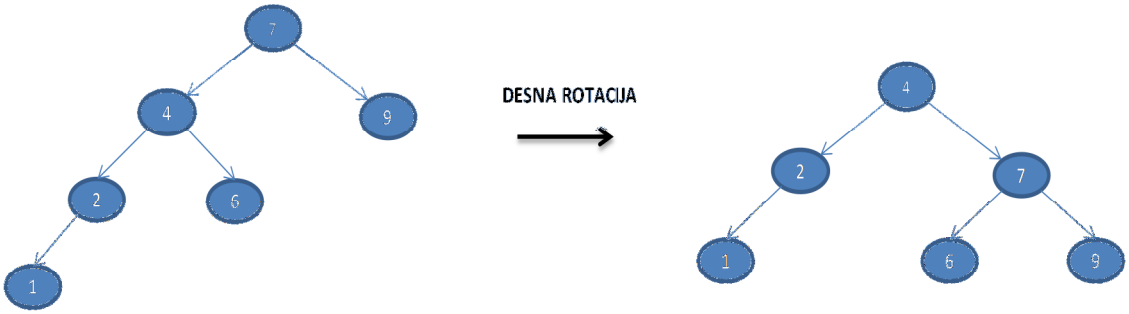


Slika 41: Desna rotacija

Desna rotacija je simetrična levi. Uporabimo jo, kadar je ravnotežni faktor nekega vozlišča enak 2. Desno rotacijo na tem vozlišču opravimo tako, da njegov levi sin zasede njegovo mesto, sam pa postane njegov desni sin.

Vidimo, da je desna rotacija zelo podobna levi, le da tu drevo »zasukamo« v desno. V tem primeru je višina levega poddrevesa za 2 večja od višine desnega poddrevesa. Drevo uravnotežimo tako, da koren drevesa postane vozlišče s podatkom y, ki ima za svojega desnega sina sedaj prejšnji koren drevesa-vozlišče s podatkom x. Levo poddrevo vozlišča x sedaj postane prejšnje desno poddrevo vozlišča, ki hrani podatek y. Zopet moramo spremeniti oba kazalca.

Da je dobljeno drevo še vedno iskalno se prepričamo na podoben način kot pri levi rotaciji (Slika 42).

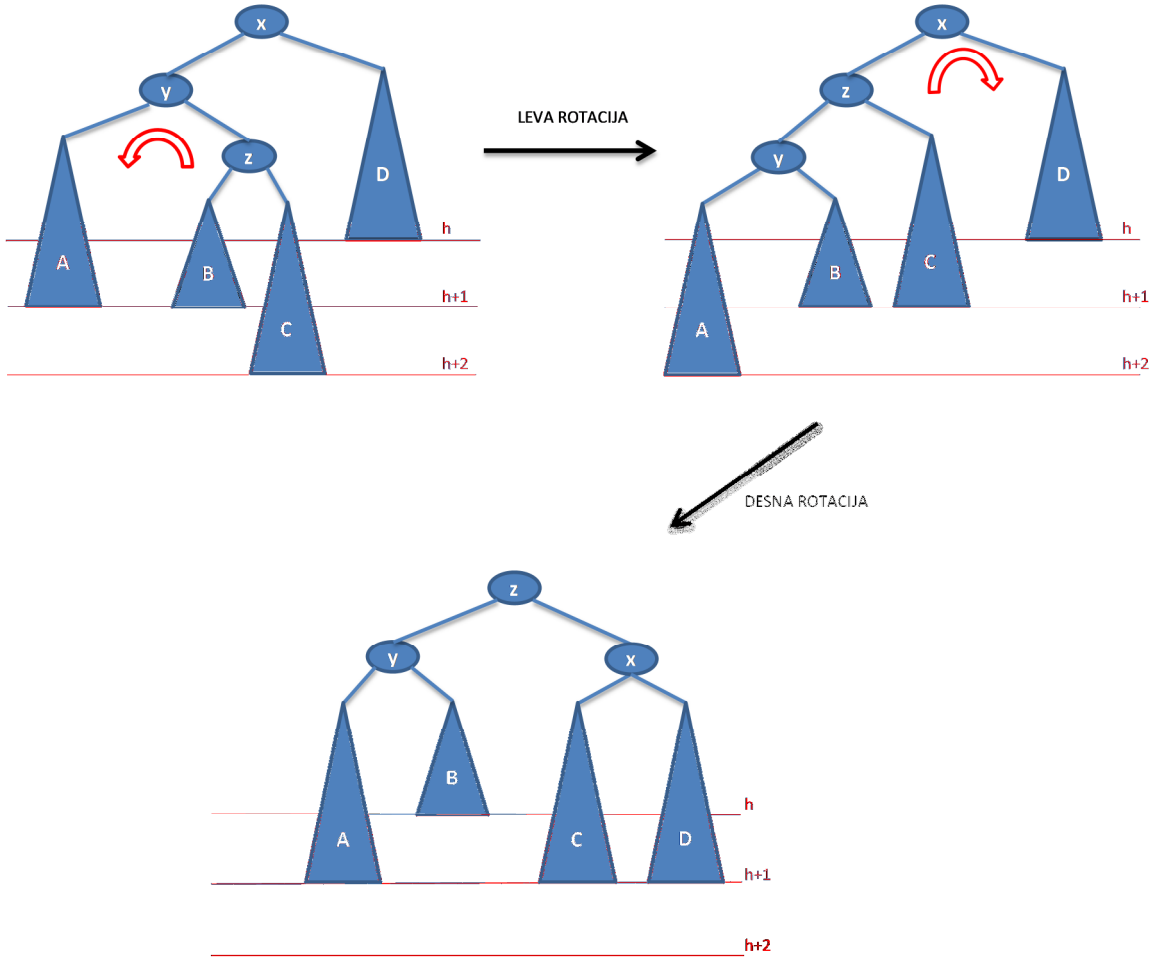


Slika 42: Uravnoteženo drevo po desni rotaciji

- Dvojni zasuk

Podrobneje si bomo pogledali levo-desno (LD) in desno-levo (DL) rotacijo. Postopek pri levo-levi oziroma desno-desni rotaciji je zelo podoben, zato ga ne bomo natančneje opisovali.

Levo-desno rotacijo uporabimo, kadar želimo uravnotežiti drevo, katerega levo poddrevo je za 2 višje od desnega. Kadar pa je desno poddrevo danega drevesa za 2 višje od levega, izvedemo desno-levo rotacijo.



Slika 43: Levo-desna rotacija

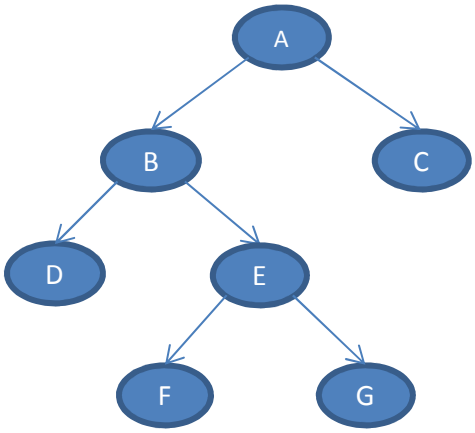
Na sliki Slika 43 je prikazan postopek uravnoteževanja drevesa, v katerem je višina desnega poddrevesa za 2 manjša od višine levega poddrevesa. Preurejanje tega drevesa zahteva eno vrtenje LD.

Višina levega poddrevesa vozlišča y je za 1 manjša od višine desnega poddrevesa (poddrevesa A). Zato najprej izvedemo levo rotacijo pri vozlišču y. Ko rotacijo izvedemo, vidimo, da je levo poddrevo drevesa še vedno za 2 višje od desnega. Koren drevesa ostane nespremenjen, spremeni pa se struktura levega poddrevesa. Koren levega poddrevesa je sedaj vozlišče z, ki za svoje levega sina dobi vozlišče y, desni sin ostane nespremenjen- koren poddrevesa C. Desni sin vozlišča y je v preurejenem drevesu koren poddrevesa B (prejšnje levo poddrevo vozlišča z).

Sledi desna rotacija okrog korena x. Nov koren drevesa postane vozlišče z, ki ima za svojega levega sina vozlišče y in za desnega sina vozlišče x. Struktura levega poddrevesa s korenom y ostane nespremenjena, Vozlišče x, kot koren desnega poddrevesa pa ima sedaj za levega sina bivšega desnega sina vozlišča z; to je koren poddrevesa C, njegov desni sin pa ostane koren poddrevesa D. Višini

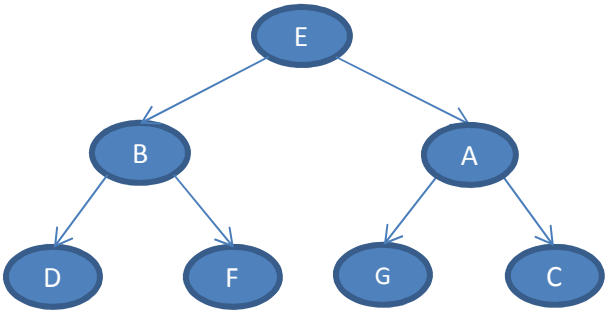
poddreves se sedaj razlikujeta samo še za 1, kar pomeni da je novo drevo uravnoteženo. V tem primer se višina drevesa za 1 zmanjša.

Zopet preverimo, da se lastnost iskalnega drevesa po rotaciji ohrani. Uravnotežiti želimo iskalno dvojiško drevo na sliki Slika 44. Zanj velja: D<B<F<E<G<A<C.



Slika 44: Neuravnoteženo drevo, nad katerim izvedemo LD rotacijo

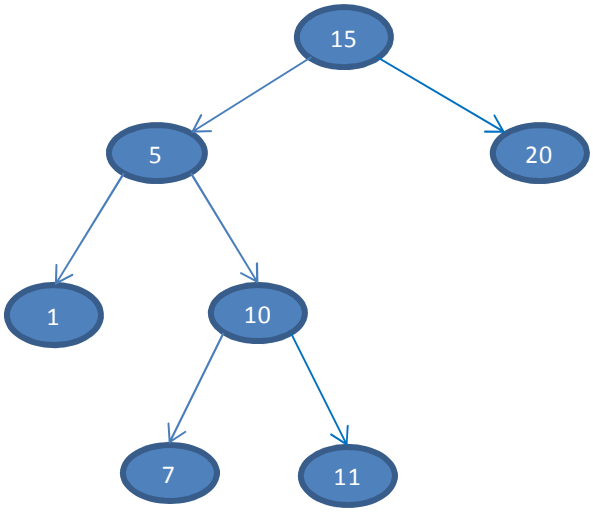
Z uporabo levo-desne rotacije dobimo uravnoteženo drevo na sliki Slika 45.



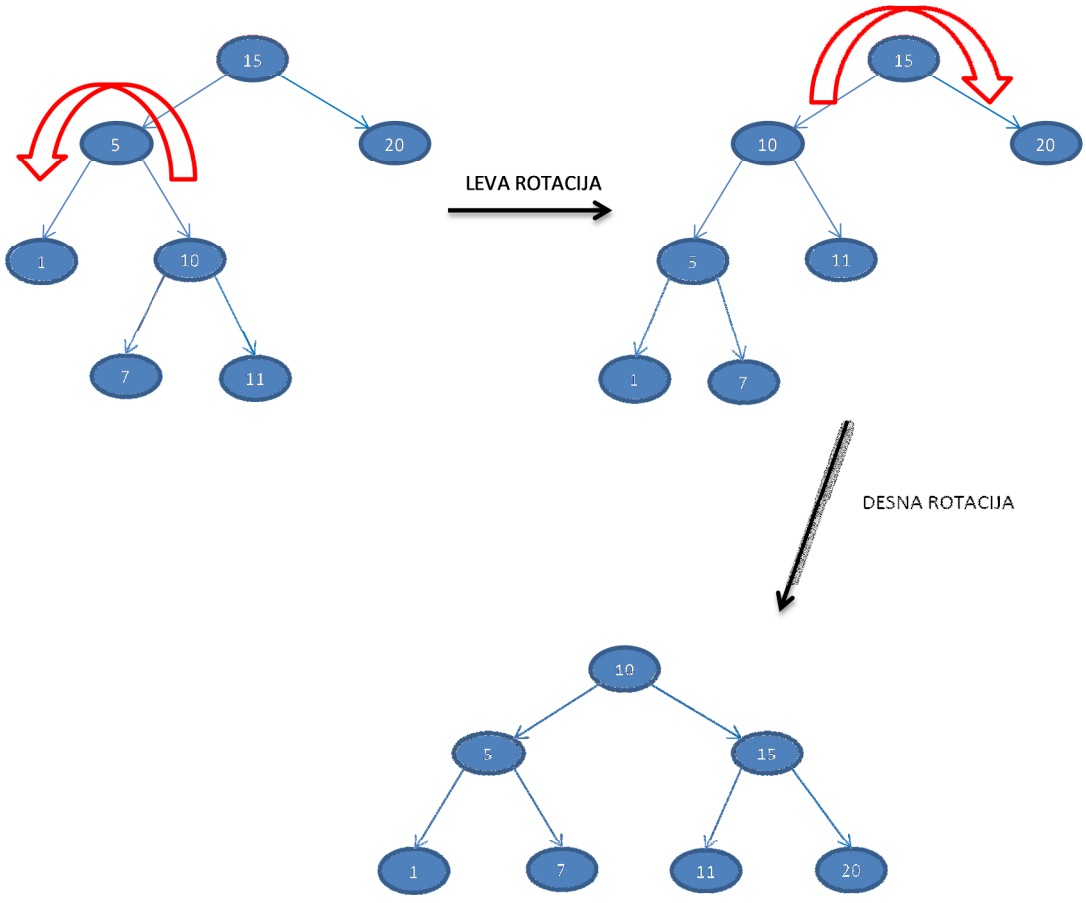
Slika 45: Uravnoteženo drevo po uporabi LD rotacije

Preverimo, ali velja, da je drevo na sliki Slika 45 iskalno. Vozlišče s podatkom D je še vedno levi sin vozlišča s podatkom B, kar pomeni, da velja D<B. Podatek F je bil že prej v desnem poddrevesu vozlišča s podatkom B (B<F). Ker je bil podatek E v desnem sinu vozlišča s podatkom B, velja B<E. Torej so vsi podatki v levem poddrevesu korena res manjši od podatka v korenu drevesa (D<B<F<E). Podatek G je bil v desnem poddrevesu vozlišča s podatkom E (E<G) in v levem poddrevesu korena s podatkom A (G<A). Vozlišče s podatkom C pa je bil desni sin korena drevesa, torej A<C. Vsi podatki v desnem poddrevesu korena drevesa na sliki Slika 45 so torej večji od A.

Da se lastnost iskalnega drevesa po uporabi LD rotacije ohrani, si poglejmo še na konkretnem primeru. Začetno iskalno drevo, ki ni uravnoteženo, je na sliki Slika 46.



Slika 46: Neuravnoteženo drevo, nad katerim izvedemo LD rotacijo_konkreten primer

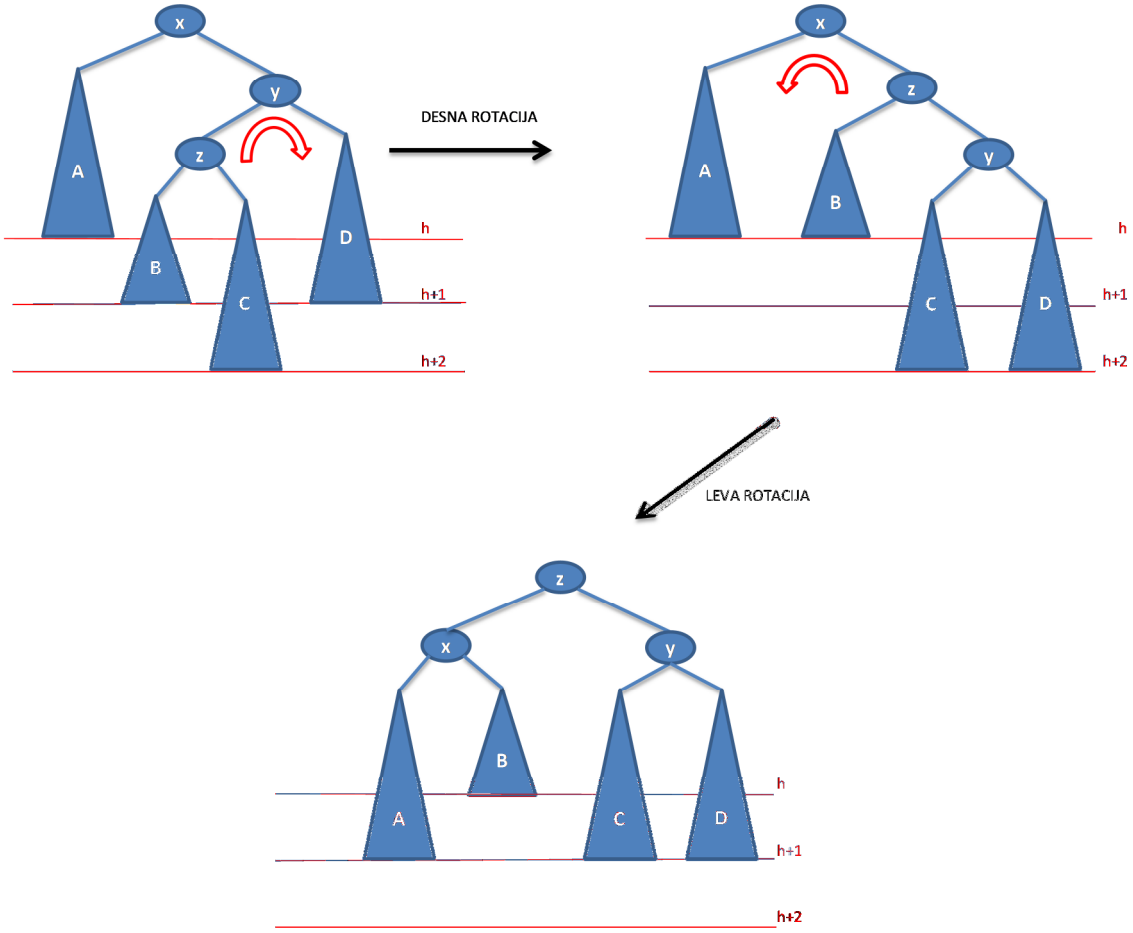


Slika 47: LD rotacija na konkretnem primeru

Da drevo uravnotežimo najprej izvedemo levo rotacijo pri vozlišču s podatkom 5. Na njegovo mesto pride vozlišče s podatkom 10, ki ima za svojega levega sina vozlišče s podatkom 5 in za svojega desnega sina vozlišče s podatkom 11. Vozlišče s podatkom 5 ima za svojega levega sina list s podatkom 1 in za svojega desnega sina list s podatkom 7. Novo drevo je torej še vedno iskalno dvojiško drevo. Višini levega in desnega poddrevesa korena s podatkom 15 se še vedno razlikujeta za 2. Sedaj pri korenu drevesa izvedemo desno rotacijo. Koren novega drevesa je sedaj vozlišče s

podatkom 10. Ta ima za svojega levega sina vozlišče s podatkom 5 in za svojega desnega sina vozlišče s podatkom 15. Levi sin vozlišča s podatkom 5 je sedaj list s podatkom 1, desni sin pa je list s podatkom 7. Levi sin vozlišča s podatkom 15 je list s podatkom 11, desni pa list s podatkom 20. Končno drevo je polno iskalno dvojiško drevo višine 3 (Slika 47).

Poglejmo sedaj še primer, ko izvedemo desno-levo (DL) rotacijo.



Slika 48: Desno-leva rotacija

Situacija na sliki Slika 48 je precej podobna prejšnji, le da je tu slika nekako »obrnjena«. Na začetku je podano drevo, v katerem je višina levega poddrevesa za 2 manjša od višine desnega poddrevesa. Za preurejanje drevesa uporabimo vrtenje DL.

Višina levega poddrevesa vozlišča y je za 1 večja od višine poddrevesa D. Zato najprej izvedemo desno rotacijo pri vozliščih y in z. Koren desnega poddrevesa (vozlišče y) zamenja vozlišče z, pri čemer se višina levega poddrevesa vozlišča z (poddrevo B) za 1 zmanjša. Vozlišče y, ki je sedaj koren desnega poddrevesa vozlišča z, za svojega levega sina dobi prejšnjega desnega sina vozlišča z, koren poddrevesa C.

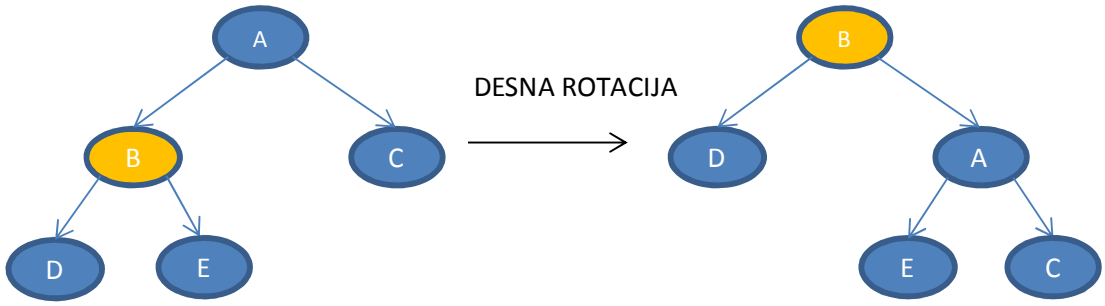
Sedaj sledi leva rotacija okrog korena x. Nov koren drevesa postane vozlišče z, ki ima za svojega levega sina vozlišče x in za desnega sina vozlišče y. Vozlišče x ima sedaj kot desnega sina prejšnjega levega sina vozlišča z; to je koren poddrevesa B. Razlika med višinama levega in desnega poddrevesa je sedaj 1. Novo drevo je torej uravnoteženo. Da pa je drevo še vedno iskalno, se prepričamo podobno kot pri LD rotaciji.

Potrebne operacije preurejanja oziroma rotacije si lahko predstavljamo kot zaporedja prestavljanj kazalcev. Kadar rotiramo vozlišče, katerega oče je koren, uporabimo enojne rotacije. Dvojne rotacije

pa so kombinacije enojnih. Kadar sta obravnavano vozlišče in njegov oče oba leva ali oba desna sinova uporabimo desno-desno (DD) ali levo-levo (LL) rotacijo. Če pa je vozlišče z obravnavanim podatkom na nasprotni strani očeta, tako kot oče glede na svojega očeta, uporabimo levo-desno (LD) oziroma desno-levo (DL) rotacijo.

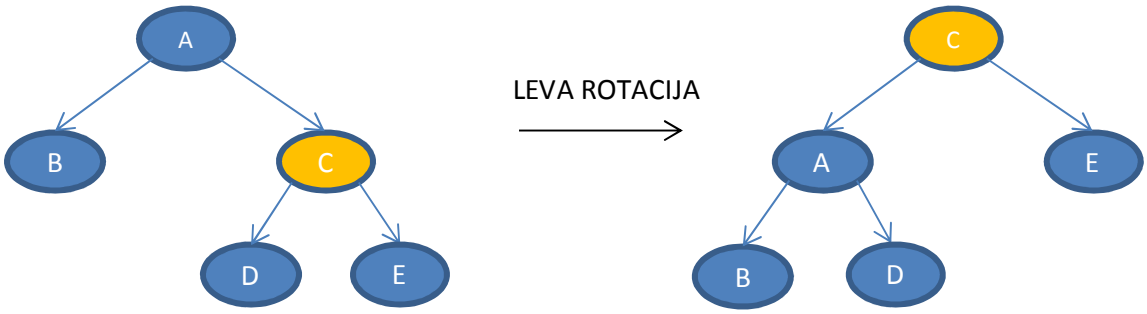
Poglejmo sedaj vseh 6 možnih primerov neuravnoteženega drevesa in pri vsakem od njih povejmo, kateri tip rotacije uporabimo.

Kadar je oče obravnavanega elementa koren, uporabimo enojno rotacijo. Desno rotacijo uporabimo, kadar je vozlišče obravnavanega elementa (ta je na sliki označen z oranžno barvo) levi sin korena drevesa (Slika 49).



Slika 49: 1. primer: desna rotacija

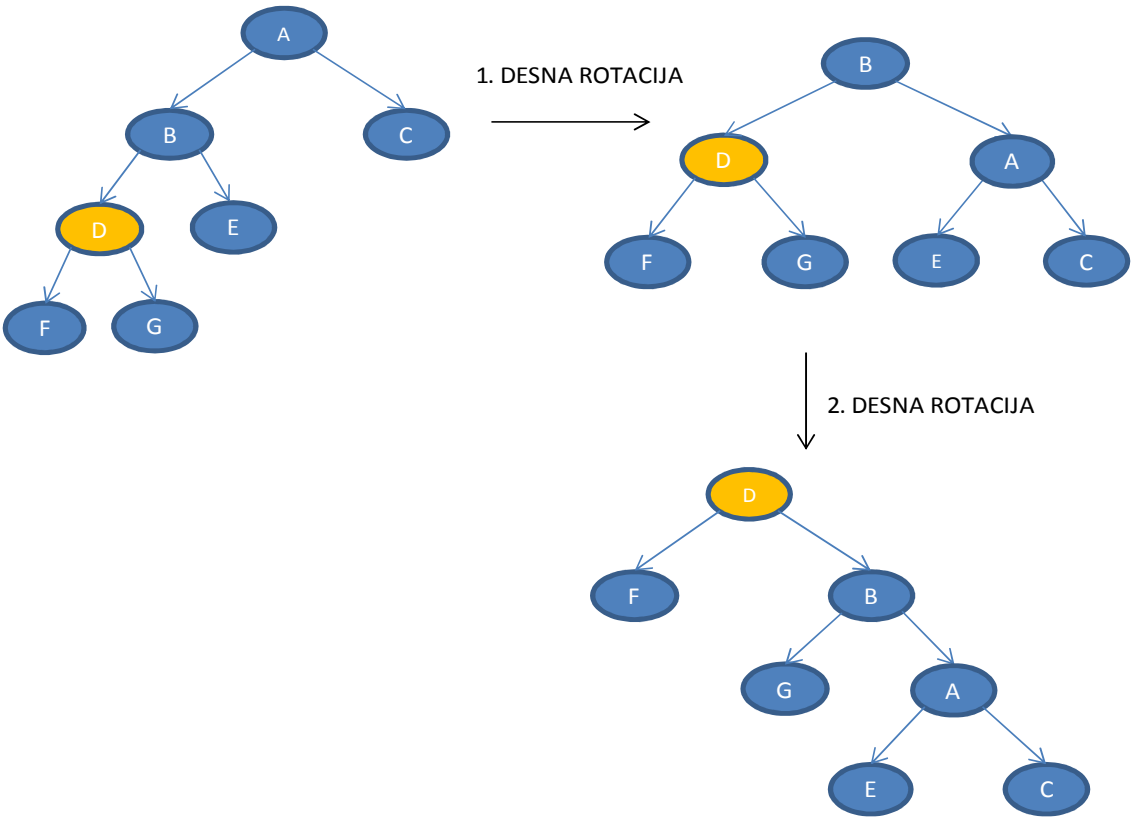
Če je vozlišče obravnavanega elementa desni sin korena, uporabimo levo rotacijo. Leva rotacija je zrcalna desni (Slika 50).



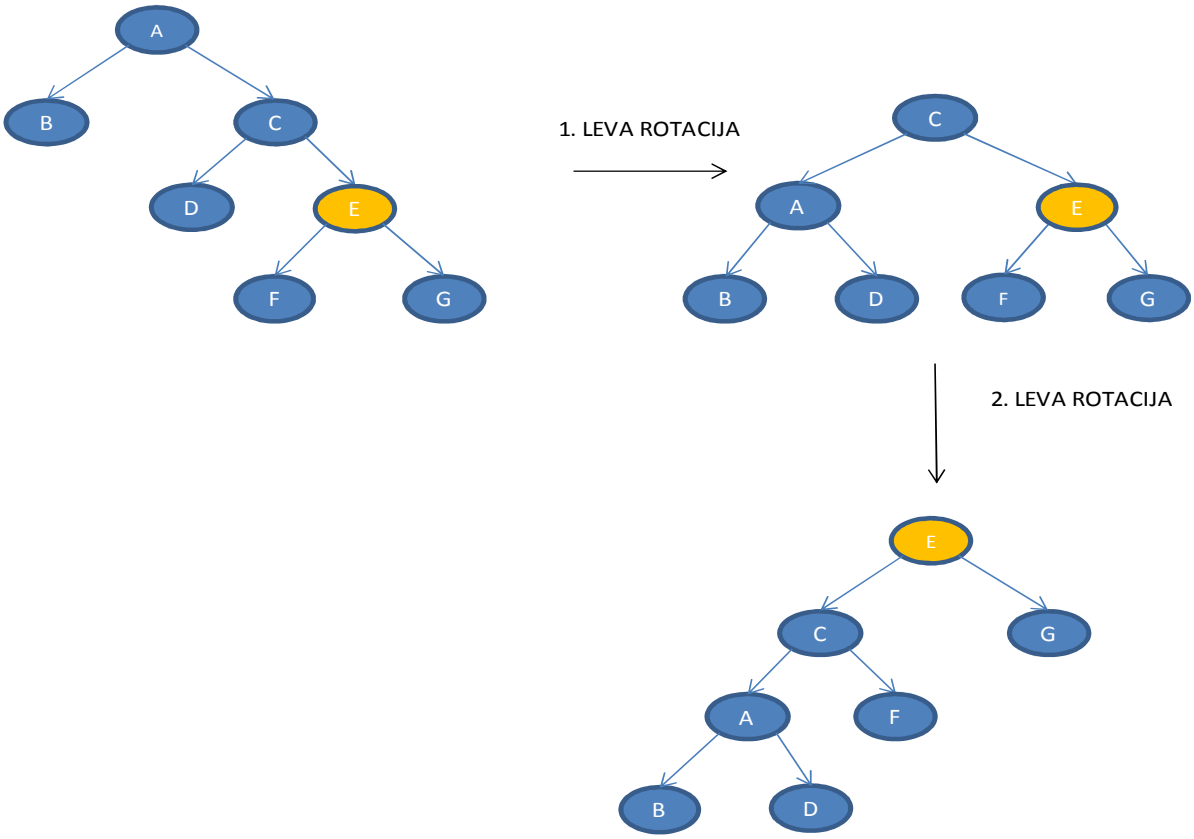
Slika 50: 2. primer: leva rotacija

Kadar sta obravnavano vozlišče in oče leva sinova svojih očetov uporabimo desno-desno rotacijo (Slika 51).

Če pa sta obravnavano vozlišče in njegov oče desna sinova svojih očetov, je uporabljena rotacija levo-leva (Slika 52). Ta rotacija je zrcalna desno-desne rotacije.

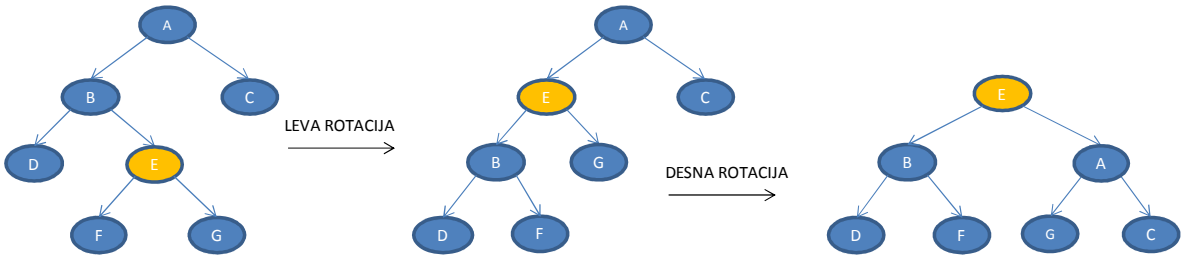


Slika 51: 3. primer: desno-desna rotacija



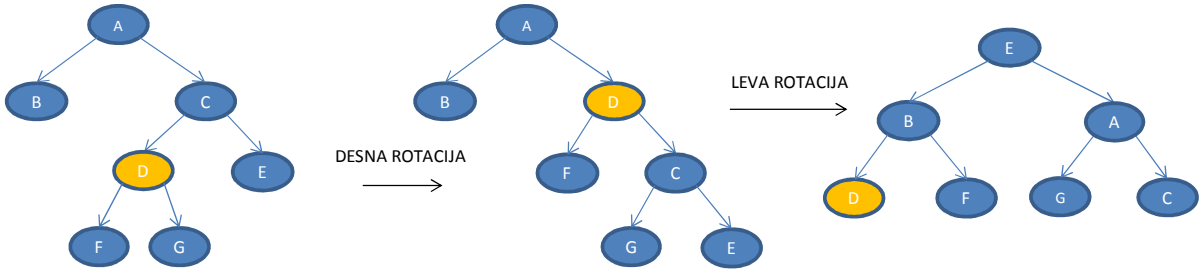
Slika 52: 4. primer: levo-leva rotacija

Kadar je vozlišče z obravnavanim podatkom desni sin, njegov oče pa levi sin svojega očeta, uporabimo levo-desno rotacijo (Slika 53).



Slika 53: 5. primer: levo-desna rotacija

Zrcalna slika levo-desne rotacije je desno-leva rotacija. To uporabimo, kadar je obravnavani element levi sin, njegov oče pa desni sin svojega očeta (Slika 54).



Slika 54: 6. primer: desno-leva rotacija

3.2.4. Vstavljanje in brisanje v uravnoteženem drevesu

Vemo, da je drevo uravnoteženo, kadar se pri vsakem vozlišču višini njegovih poddreves razlikujeta za največ 1. Pri vstavljanju ali brisanju vozlišč se lahko to ravnotežje drevesa poruši (višini levega in desnega poddrevesa se razlikujeta za več kot 1), zato moramo takšno drevo preoblikovati. Za preoblikovanje potrebujemo enojne in dvojne rotacije. Pri tem si pomagamo tudi z ravnotežnim faktorjem. Postopek vstavljanja novega in brisanja obstoječega vozlišča v uravnoteženem drevesu, si bomo pogledali v tem poglavju.

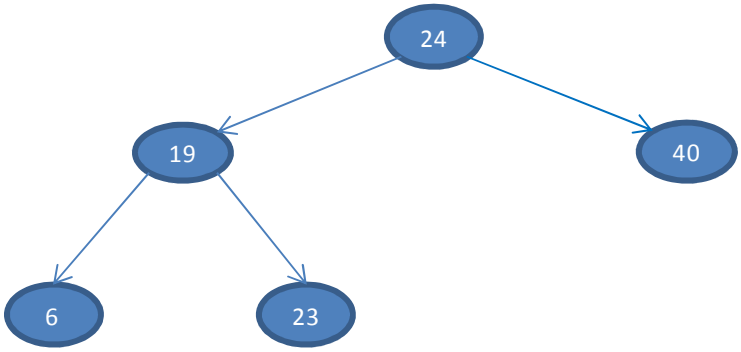
3.2.4.1. Vstavljanje

Poglejmo, kaj se lahko zgodi, ko v uravnoteženo drevo vstavimo nov podatek. Denimo, da imamo koren, katerega levo poddrevo označimo z L, njegovo desno poddrevo pa z D. Predpostavimo, da moramo podatek vstaviti v poddrevo L, pri čemer se višina slednjega poveča za 1. Nastopijo lahko trije primeri. Pri tem s h_L in s h_D označimo višini poddreves L in D pred vstavljanjem. Ker je bilo drevo pred vstavljanjem uravnoteženo, sta se h_L in h_D razlikovali kvečjemu za 1.

- $h_L < h_D$: Drevesi L in D postaneta enako visoki. V tem primeru se torej ravnotežje celo izboljša in je drevo v korenu popolnoma uravnoteženo.
- $h_L = h_D$: Drevesi L in D sta sedaj različnih višin, vendar je ravnotežni kriterij še vedno izpolnjen (levo poddrevo je za 1 višje od desnega).

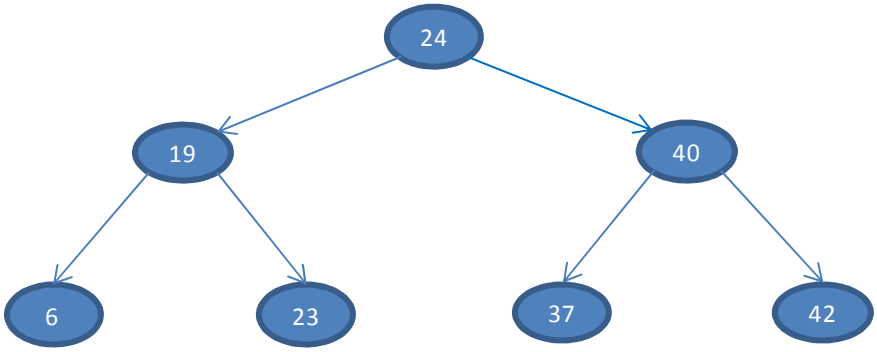
- $h_L > h_D$: ravnotežje se poruši (višina L je sedaj za 2 večja od višine D), zato moramo drevo preurediti.

Preureditev drevesa si poglejmo na primeru. Naj bo dano uravnoteženo drevo na sliki Slika 55. Vanj želimo vstaviti podatke: 37, 42 in 26.



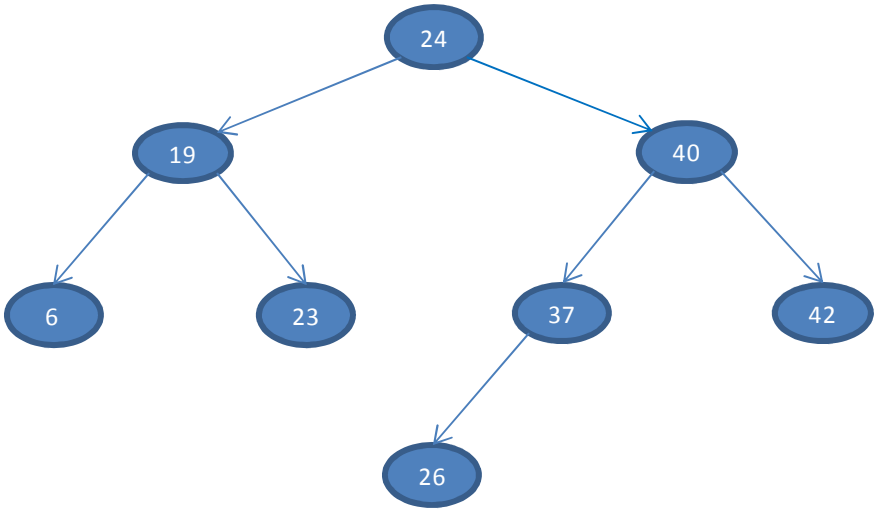
Slika 55: Uravnoteženo drevo pred vstavljanjem

Za vstavljanje podatkov ustvarimo nova vozlišča, kamor bomo te podatke vstavili. Vozlišči s podatkoma 37 in 42 lahko vstavimo brez preurejanja. Vidimo, da se ravnotežje drevesa celo izboljša. Drevo postane polno in popolnoma uravnoteženo (Slika 56).



Slika 56: Uravnoteženo drevo po vstavljanju elementov 37 in 42

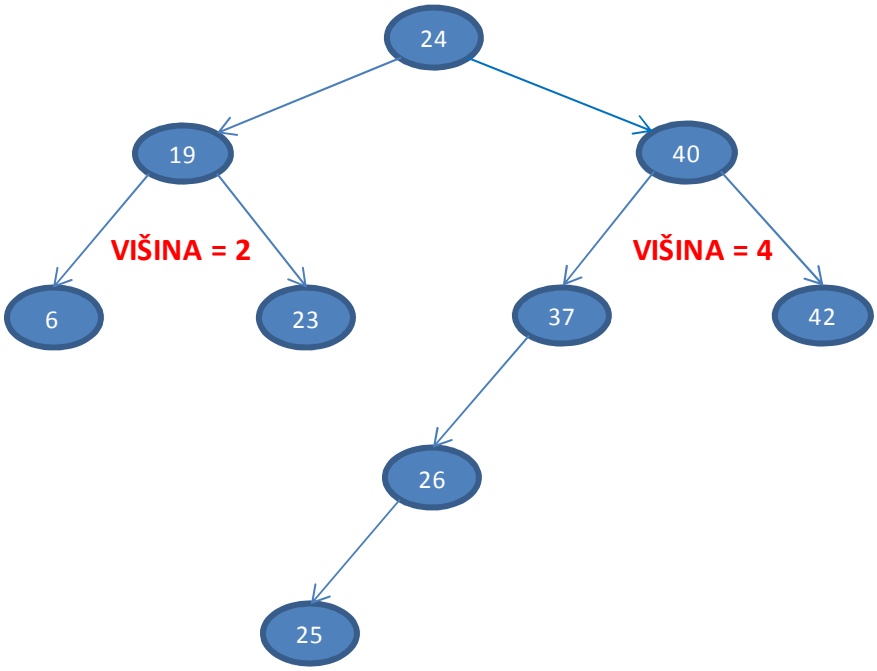
Ko v drevo vstavimo 26, se višina desnega poddrevesa za 1 poveča, vendar je ravnotežni kriterij še vedno izpolnjen. Dobljeno drevo je prikazano na sliki Slika 57.



Slika 57: Uravnoteženo drevo po vstavljanju elementa s podatkom 26

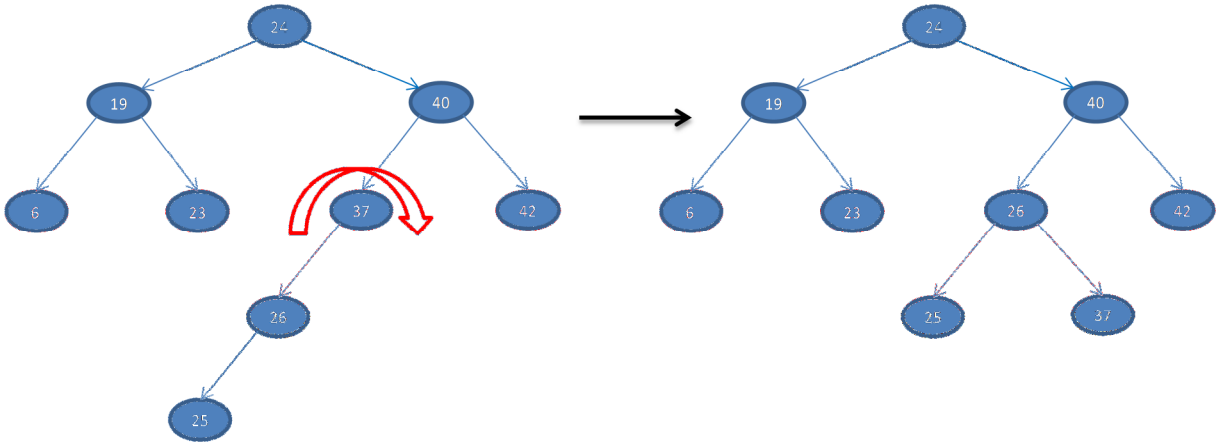
Sedaj želimo v drevo na sliki Slika 57 vstaviti podatek 25. To vstavljanje terja preurejanje drevesa.

Ko v prvotno drevo dodamo vozlišče s podatkom 25, dobimo drevo, v katerem se višini obeh poddreves korena razlikujeta za 2 (višina levega poddrevesa je enaka 2, višina desnega poddrevesa pa 4). S tem se lastnost uravnotežena drevesa poruši (Slika 58) in drevo moramo preoblikovati.



Slika 58: Porušena lastnost uravnoteženega drevesa

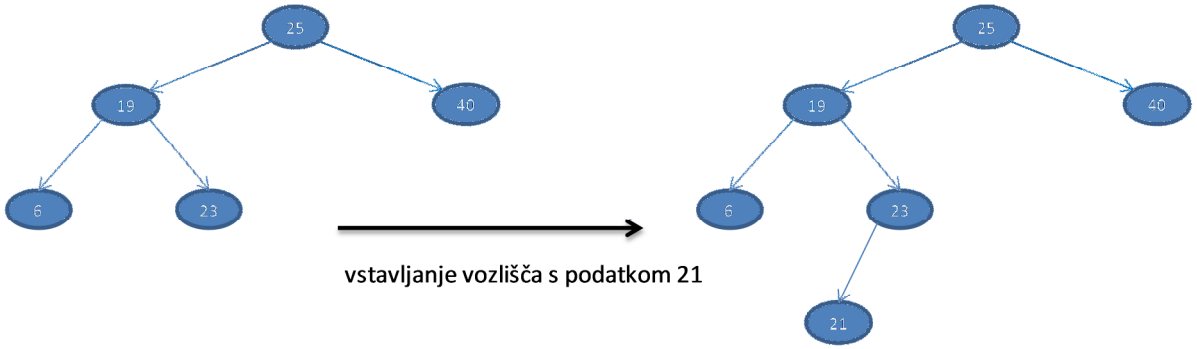
Najprej preverimo ravnotežne faktorje vozlišč na poti od mesta vstavljanja novega vozlišča do korena. Vozlišče s podatkom 26 ima ravnotežni faktor 1. Problem nastane pri vozlišču s podatkom 37, kjer je ravnotežni faktor 2. Od tu smo šli pri iskanju ustreznega mesta za vstavljanje novega vozlišča enkrat v levo, zato bomo sedaj drevo preoblikovali z desno rotacijo pri vozlišču s podatkom 37. Tako dobimo uravnoteženo iskalno dvojiško drevo višine 3.



Slika 59: Uravnoteženo drevo po vstavljanju vozlišča s podatkom 25

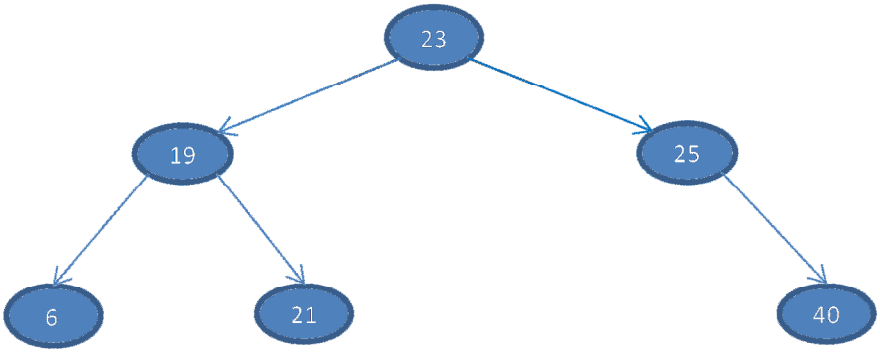
Sedaj pa v prvotno drevo s slike Slika 55 vstavimo vozlišče z elementom 21. Drevo postane neuravnoteženo (razlika višin poddreves je 2), zato je zopet potrebno preurejanje drevesa. A tokrat je postopek nekoliko drugačen.

Problem nastane zopet pri korenu, kjer je ravnotežni faktor 2. Na spodnji sliki vidimo, da smo šli pri iskanju ustreznega mesta za vstavljanje novega vozlišča najprej levo in nato desno. Za preurejanje neuravnoteženega drevesa sedaj uporabimo dvojni zasuk in sicer levo-desno rotacijo.



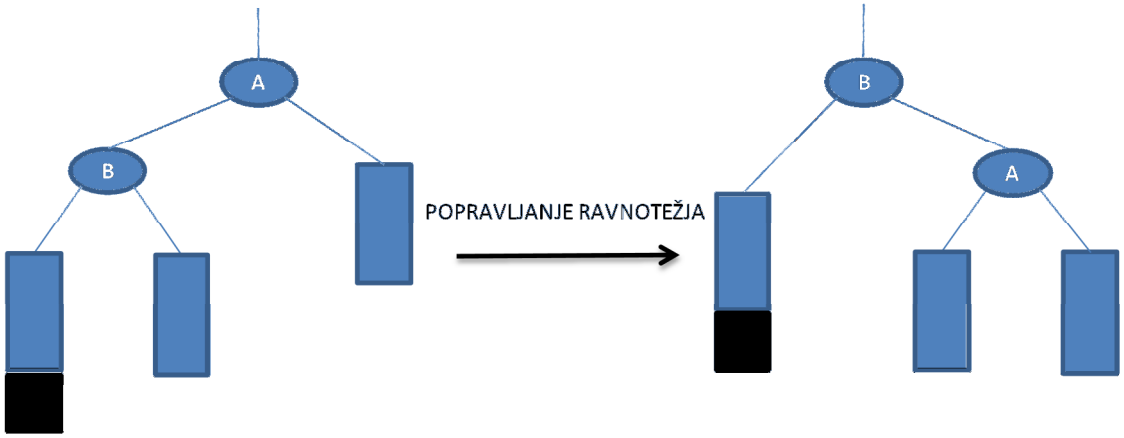
Slika 60: Vstavljanje vozlišča s podatkom 21

Najprej uporabimo levo rotacijo pri vozlišču s podatkom 19. Sledi desna rotacija pri vozlišču s prevelikim ravnotežnim faktorjem, torej pri korenu drevesa. Ta gre v desno poddrevo. Na njegovo mesto pride koren desnega poddrevesa njegovega levega poddrevesa. Tako dobimo drevo na sliki Slika 61.

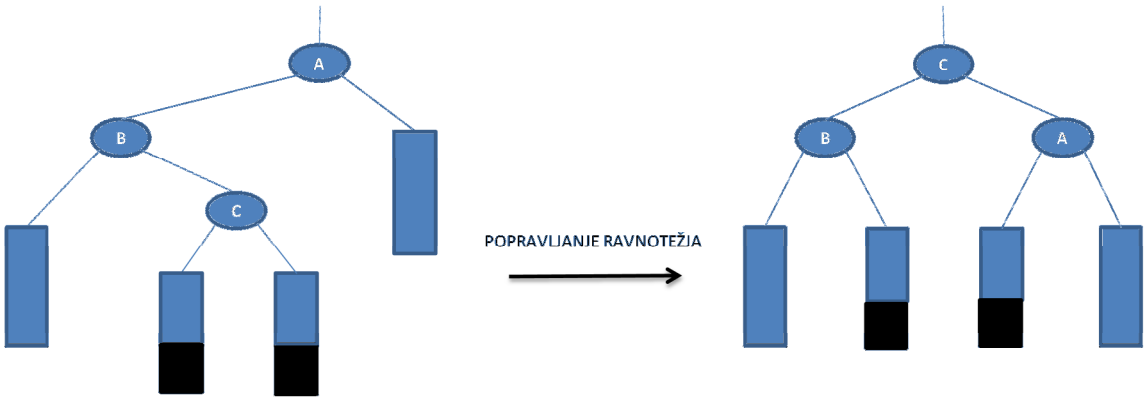


Slika 61: Uravnoteženo drevo po vstavljanju podatka 21

Dejansko obstajata le dve bistveno različni drevesni zgradbi, ki zahtevata posebno obravnavo. Ti dve zgradbi sta prikazani na slikah Slika 62 in Slika 63. Tu z modrimi pravokotniki označujemo poddrevesa, povečanje višine zaradi vstavljanja pa s črnimi pravokotniki. Preostale lahko izpeljemo iz teh dveh, sklicujoč se na določene simetrije.



Slika 62: Popravljanje ravnotežja 1



Slika 63: Popravljanje ravnotežja 2

Algoritem za vstavljanje in preurejanje uravnoteženega drevesa je bistveno odvisen od načina, kako hranimo informacijo o drevesnem ravnotežju. Ena od rešitev je hraniti informacijo o ravnotežju implicitno. To pomeni, da upoštevamo, da je ravnotežni faktor posameznega vozlišča določen z zgradbo drevesa. Vendar bi v takem primeru morali ravnotežni faktor nekega vozlišča vedno znova

določati, kadarkoli bi to vozlišče sodelovalo v postopku vstavljanja, torej kadar bi bilo na poti od korena do vstavljenega vozlišča. To bi povzročilo veliko dodatnega računskega dela. Druga rešitev je, da v vsakem vozlišču hranimo poleg podatkov in referenc na sinove še ravnotežni faktor.

Postopek vstavljanja vozlišča v uravnoteženo drevo je tako sestavljen iz treh delov:

- Poiščemo ustrezno mesto, kamor bomo dodali vozlišče in s tem vstavili podatek v drevo. Sočasno preverimo, da vozlišča s tem podatkom še ni v drevesu. Ena od lastnosti iskalnih dreves je, da v njem ni podvojenih elementov.
- Vstavimo novo vozlišče.
- Vrnemo se po iskalni poti do korena in pri vsakem vozlišču preverimo ravnotežni faktor. V primeru, ko kateri od ravnotežnih faktorjev vozlišč po vstavljanju novega vozlišča ne ustreza definiciji uravnoteženega drevesa, torej je večji od 1 ali manjši od -1, moramo drevo preurediti.

Ko na poti do korena pridemo do vozlišča z neustreznim ravnotežnim faktorjem, uporabimo ustrezno enojno ali dvojno rotacijo. Sedaj popravimo ravnotežni faktor vozlišč vse do korena. To pomeni, da na poti od »popravljenega« vozlišča do korena, pri vsakem vozlišču preverimo ravnotežni faktor. V primeru, da pridemo do vozlišča z neustreznim ravnotežnim faktorjem, zopet uporabimo ustrezno rotacijo. Ko drevo uravnotežimo, je postopek vstavljanja novega vozlišča v uravnoteženo drevo končan.

3.2.4.2. Brisanje

Podobno kot pri vstavljanju, bomo tudi pri brisanju elementa iz uravnoteženega drevesa najprej poiskali ta element, ga odstranili in nadomestili z ustreznim elementom v drevesu. Brisano vozlišče nadomestimo z najbolj desnim (največjim) elementom v levem poddrevesu (če ta obstaja) ali z najbolj levim (najmanjšim) elementom v desnem poddrevesu. Brisanje teh vozlišč je dokaj enostaven postopek (glej razdelek 2.5.4, kjer sta pomožna algoritma `brisanjeMinimum` in `brisanjeMaksimum`). Na koncu ustrezno popravimo ravnotežne faktorje na poti do korena in uporabimo rotacije, kadar je to potrebno.

Izkaže se, da je brisanje vozlišča iz uravnoteženega drevesa bolj zapleteno kot vstavljanje. Operacija preurejanja drevesa sicer ostane enaka, lahko pa se zgodi, da jo bomo morali tu uporabiti na več kot le enem mestu.

Pri brisanju elementa v uravnoteženem drevesu ločimo dva glavna primera:

1. Najprej si oglejmo primer, ko je element, ki ga želimo izbrisati, list. Vozlišče preprosto odstranimo, nato pa rekurzivno preverimo faktorje ravnotežja vseh vozlišč, ki so na poti do korena. Če so le-ta med -1 in 1, z brisanjem zaključimo, sicer uporabimo ustrezno rotacijo. Po opravljeni rotaciji moramo zopet preveriti faktor ravnotežja v vseh vozliščih na poti od vozlišča, kjer smo izvedli rotacijo, do korena. Ravnotežje se je namreč lahko porušilo na višjem nivoju. Po izbrisanem vozlišču (listu) torej postopek nadaljujemo na točki 2.

To je bistvena razlika med uravnoteževanjem drevesa pri postopku vstavljanja in brisanja. Pri vstavljanju lahko po opravljeni rotaciji postopek zaključimo.

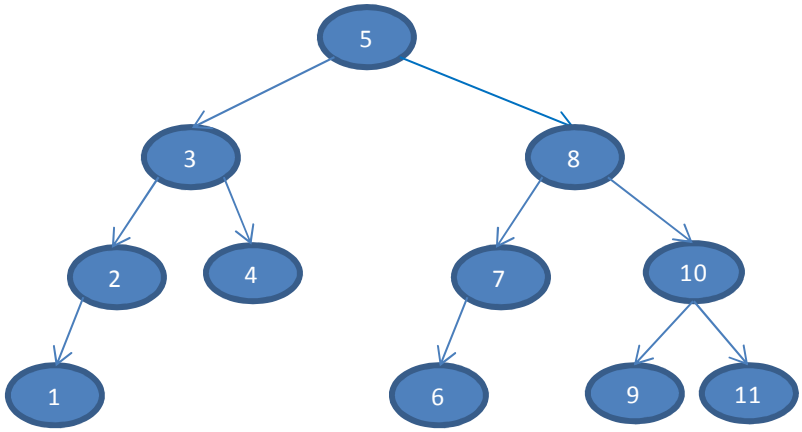
2. Sedaj pa želimo odstraniti element v notranjem vozlišču. Če ima vozlišče obe poddrevesi, izbrisani element nadomestimo z najbolj desnim elementom v levem poddrevesu. Če ima vozlišče le desno poddrevo, izbrisani element v tem vozlišču nadomestimo z najbolj levim

elementom v desnem poddrevesu. Najbolj desno vozlišče v levem poddrevesu, ali najbolj levo vozlišče v desnem poddrevesu nato izbrišemo. Od brisanega vozlišča na poti do korena popravimo ravnotežne faktorje. Sedaj lahko nastopijo tri situacije:

a. Faktor ravnotežja v nobenem vozlišču ni manjši od -1 in večji od 1. To pomeni, da je drevo po postopku brisanja še vedno uravnoteženo, zato je postopek končan.
b. Faktor ravnotežja trenutnega vozlišča je manjši od -1, kar pomeni, da je desno poddrevo višje od levega. V tem primeru za preureditev drevesa uporabimo levo rotacijo in preverimo ravnotežni faktor vseh vozlišč na poti od trenutnega vozlišča do korena. Če se je ravnotežje porušilo na višjem nivoju, zopet uporabimo ustrezno rotacijo. Postopek ponavljamo, dokler ni ravnotežni faktor vseh vozlišč med -1 in 1.
c. Faktor ravnotežja trenutnega vozlišča je večji od 1. Levo poddrevo vozlišča je višje od desnega, zato uporabimo rotacijo v desno. Drevo je po tej rotaciji lahko še vedno neuravnoteženo. Postopek nadaljujemo, dokler vsa vozlišča ne zadostujejo kriteriju uravnoteženega drevesa.

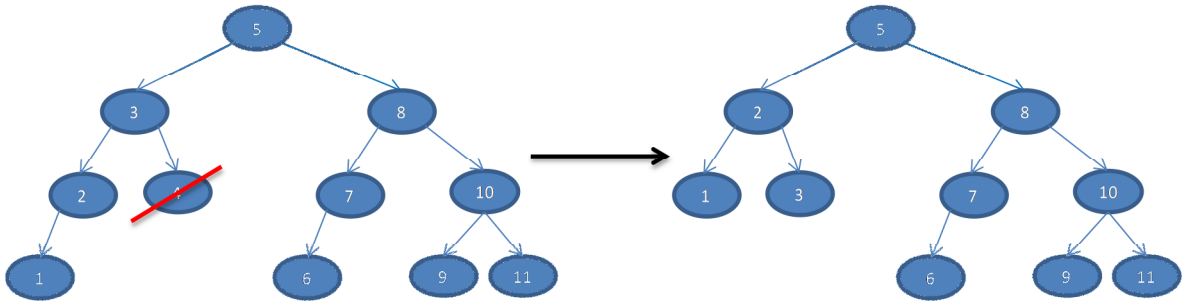
V najslabšem primeru moramo postopek rotacij izvajati pri vsakem vozlišču drevesa na poti od brisanega vozlišča do korena.

Postopek brisanja vozlišč iz uravnoteženega drevesa si poglejmo še na nekaj primerih. Iz drevesa na sliki Slika 64 zaporedno brišemo podatke.



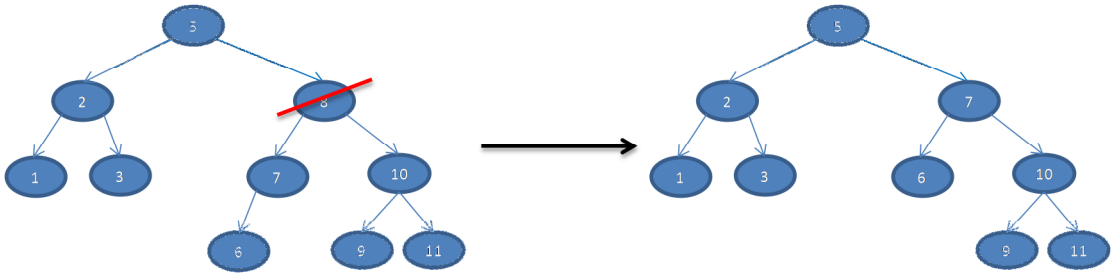
Slika 64: Uravnoteženo drevo, iz katerega brišemo podatke

1. Najprej želimo iz danega drevesa izbrisati vozlišče s podatkom 4. Samo izločanje je enostavno, ker imamo opravka s končnim vozliščem. Vendar pa se ravnotežje pri vozlišču s podatkom 3 poruši. Poddrevo moramo zato preurediti. Ker je višina levega poddrevesa vozlišča s podatkom 3 (njegova višina je 2) za 2 večja od višine levega poddrevesa (to je prazno), uporabimo enojno vrtenje v desno. Višina levega poddrevesa s korenom, ki vsebuje podatek 3, se je za 1 zmanjšala, zato moramo preveriti ravnotežni faktor vseh vozlišč na poti do korena. V našem primeru se je ravnotežni faktor korena spremenil iz 0 v -1. Višini levega in desnega poddrevesa korena drevesa se sedaj razlikujeta za 1. Preoblikovano drevo je torej uravnoteženo. Postopek je prikazan na sliki Slika 65.



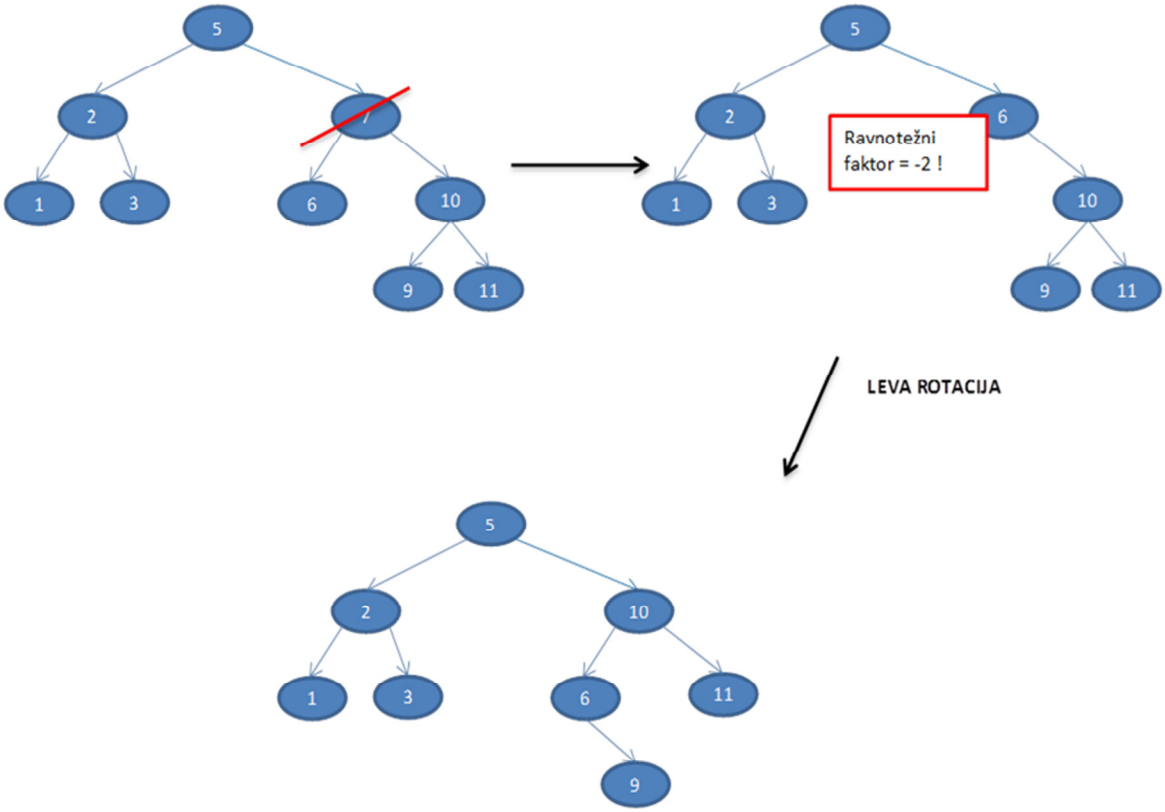
Slika 65: Brisanje podatka 4 iz uravnotežena drevesa

2. Naslednje vozlišče, ki ga želimo izbrisati iz novega drevesa je vozlišče s podatkom 8. To vozlišče je notranje vozlišče, ki ima tako levo kot desno poddrevo. Nadomestimo ga z najbolj desnim vozliščem v levem poddrevesu, torej z vozliščem, ki vsebuje podatek 7. Desno poddrevo zato ostane nespremenjeno, levo poddrevo korena s podatkom 7 pa je vozlišče s podatkom 6. S tem ostane ravnotežni faktor v korenu drevesa (vozlišče s podatkom 5) nespremenjen (še vedno je -1), zato lahko s postopkom brisanja vozlišča zaključimo (Slika 66).



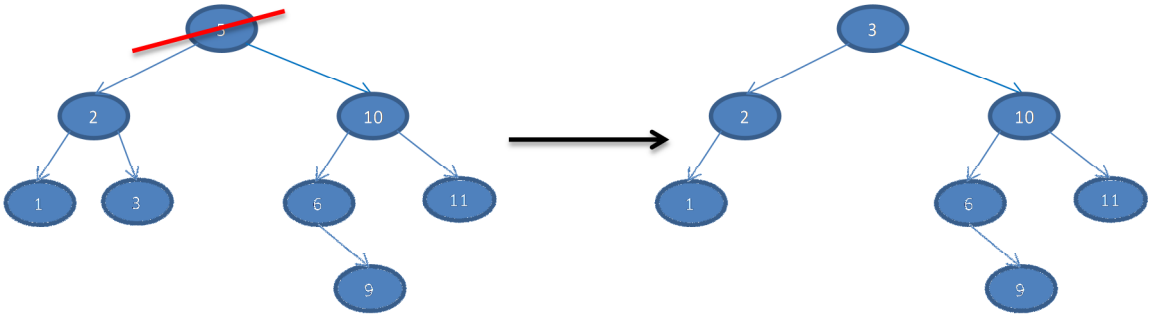
Slika 66: Brisanje podatka 8 iz uravnoteženega drevesa

3. V tem koraku odstranimo vozlišče s podatkom 7. Zopet ga nadomestimo z najbolj desnim vozliščem v levem poddrevesu - z vozliščem s podatkom 6. Kriterij ravnotežja se poruši. Koren desnega poddrevesa s podatkom 6 ima za svoje levo poddrevo prazno drevo (njegova višina je 0), za svoje desno poddrevo pa drevo višine 2. Ravnotežni faktor je zato -2. Pri vozlišču s podatkom 6 uporabimo enojno levo rotacijo. Preverimo ravnotežne faktorje po poti do korena. Vsa vozlišča imajo ravnotežni faktor med -1 in 1. Drevo na sliki Slika 67 je torej uravnoteženo.



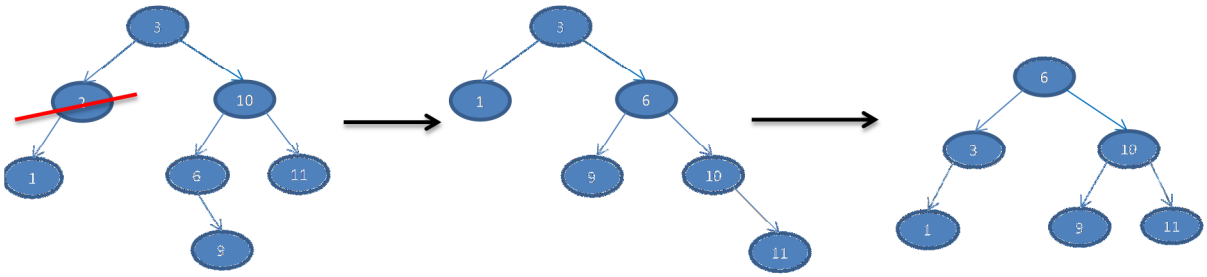
Slika 67: Brisanje podatka 7 iz uravnoteženega drevesa

4. Odstranimo vozlišče s podatkom 5, torej koren drevesa. Na njegovo mesto pride vozlišče s podatkom 3. S tem se spremeni ravnotežni faktor vozlišča s podatkom 2 v 1. Levo poddrevo je še vedno uravnoteženo, enako velja za desno poddrevo, ki ostane nespremenjeno. Novo drevo je torej uravnoteženo (Slika 68).



Slika 68: Brisanje podatka 5 iz uravnoteženega drevesa

5. Izločanje vozlišča s podatkom 2 je samo po sebi zelo preprosto, saj ima le enega naslednika. Vendar pa se ravnotežje v korenu poruši. Višina levega poddrevesa je za 2 manjša od višine desnega poddrevesa. Zato preoblikovanje drevesa zahteva dvojno vrtenje. V konkretnem primeru uporabimo desno-levo rotacijo (Slika 54). Pri vozlišču s podatkom 10 izvedemo najprej desno rotacijo. Vozlišče s podatkom 6 postane koren desnega poddrevesa, vozlišče 9 njegov levi sin in vozlišče 10 koren njegovega desnega poddrevesa. S tem se ravnotežje v korenu (vozlišče 3) poruši. Izvedemo še levo rotacijo pri vozlišču s podatkom 3. Sedaj je ravnotežni faktor korena (vozlišče 6) enak 0, njegovo levo in desno poddrevo sta uravnoteženi, zato s postopkom zaključimo. Drevo je uravnoteženo. Končno drevo je prikazano na sliki Slika 69.



Slika 69: Brisanje podatka 2 iz uravnoteženega drevesa

3.2.5. Časovna zahtevnost operacij nad uravnoteženimi drevesi

3.2.5.1. Iskanje

Operacija iskanja elementa z iskanim podatkom v uravnoteženem drevesu je enaka kot v iskalnem dvojiškem drevesu. Vendar je zaradi uravnoteženosti višina uravnoteženega drevesa vedno enaka $\log_2(n+1)$. Zato je časovna zahtevnost operacije iskanja tudi v najslabšem primeru $O(\log(n))$.

3.2.5.2. Vstavljanje

Pri operaciji vstavljanja moramo v drevesu najprej poiskati ustrezno mesto, kamor bomo vozlišče s podatkom vstavili. V najslabšem primeru moramo pri iskanju ustreznega mesta obiskati $\log_2(n+1)$ vozlišč. Ko se vračamo po iskalni poti, zopet obiščemo $\log_2(n+1)$ vozlišč, skupaj torej $2 * \log_2(n+1)$. Ob vrnitvi moramo preveriti faktor ravnotežja in po potrebi izvesti ustrezno rotacijo. Časovna zahtevnost ene rotacije je $O(1)$.

Časovna zahtevnost operacije vstavljanja elementa v uravnoteženo drevo je torej $O(\log(n))$.

3.2.5.3. Brisanje

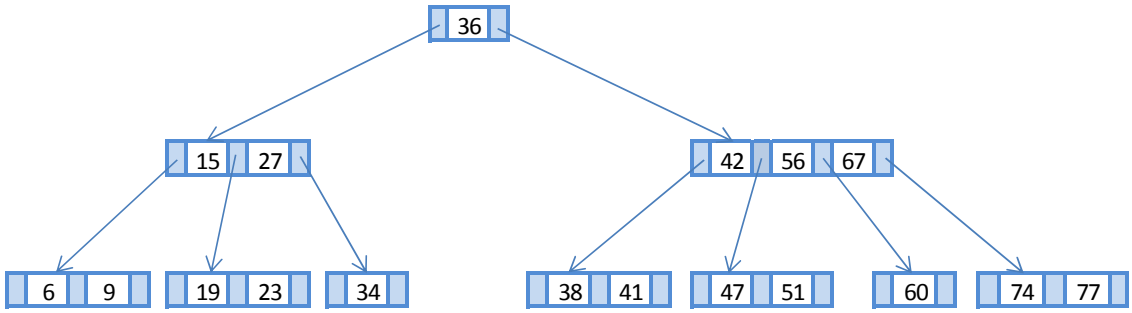
Tudi časovna zahtevnost operacije brisanja iz uravnoteženega drevesa je $O(\log(n))$. Postopek je precej podoben postopku vstavljanja. Bistvena razlika je le v tem, da lahko povzroči vstavljanje elementa eno samo vrtenje, brisanje pa lahko zahteva rotacije na vsakem vozlišču. Vendar je teh rotacij lahko največ log (n) in ker je vsaka izvedena v konstantni časovni zahtevnosti, rotacije prispevajo spet le $O(\log(n))$.

4. B-DREVESA

Veliko število podatkov navadno hranimo na disku. Pri takem shranjevanju podatkov je pomembno, da pri operacijah nad temi podatki (iskanje, vstavljanje, brisanje,..) čim manjkrat posegamo na disk, saj vsak poseg zahteva veliko časa. Rešitev za tako shranjevanje podatkov so večsmerna iskalna drevesa. Podobno kot pri dvojiških iskalnih drevesih (glej razdelek 2a), so tudi v večsmernih iskalnih drevesih podatki urejeni. Razlika je v številu elementov, ki jih vsebujejo vozlišča. Vsako vozlišče v večsmernem iskalnem drevesu lahko vsebuje večje število elementov (podatkov) in ima lahko več kot dva sinova. Število elementov in sinov posameznih vozlišč je odvisno od reda večsmernega iskalnega drevesa, ki ga bomo natančneje definirali v nadaljevanju. Pri teh drevesih je pri operacijah število posegov na disk odvisno od višine drevesa, zato je pomembno, da ta višina ne raste nenadzorovano, ampak je čim manjša. Tako pridemo do uravnoteženih dreves, kjer se višini levega in desnega poddrevesa razlikujeta kvečjemu za 1.

Če združimo ideji o večsmernih in uravnoteženih drevesih, pridemo do B-dreves.

Primer B-drevesa:



Slika 70: B-drevo reda 4

B-drevesa visokega reda se uporabljajo za shranjevanje velikih baz podatkov na zunanjem pomnilniku oziroma trdem disku. Podrobneje si bomo B-drevesa in osnovne operacije nad njimi ogledali v posebnem razdelku (glej razdelek 4.3).

4.1. Definicija B – drevesa

Literatura o B-drevesih ni enotna pri uporabi izrazov v zvezi z B-drevesi.

Bayer in McCreight (R. Bayer, 1972) , Comer (Comer, 1979) in ostali so določili red drevesa kot minimalno število elementov (in pripadajočih ključev) v notranjih vozliščih B-drevesa. Vendar pa je terminologija dvoumna, saj največje število ključev ni jasno. Vozlišče B-drevesa reda 3 ima lahko največ 6 ali največ 7 ključev. Knuth (Knuth, 1973) pa se je težavam z določanjem reda B-drevesa izognil tako, da je za red drevesa določil največje število otrok vozlišča (kar je za ena več, kot je največje število elementov in pripadajočih ključev tega vozlišča).

V nadaljevanju diplomske naloge bomo uporabljali naslednjo definicijo B-drevesa:

B-drevo reda *m* je iskalno drevo, za katerega veljajo naslednje lastnosti:

- Za število sinov (naslednikov) posameznih vozlišč velja:
  - Vsako vozlišče ima največ *m* sinov.

- Vsako notranje vozlišče, razen korena, ima najmanj $m/2$ sinov.
- Koren je bodisi list bodisi ima vsaj dva sinova.

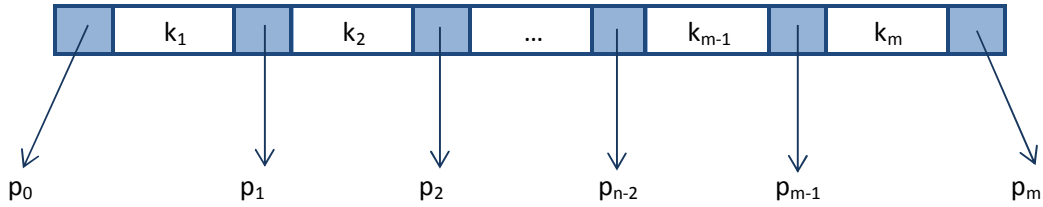
– Tudi število podatkov in pripadajočih ključev je odvisno od reda drevesa m:
  - Vsako vozlišče ima največ m-1 podatkov. Če ima vozlišče m-1 elementov, pravimo, da je vozlišče polno.
  - Vsako notranje vozlišče, razen korena ima, ima vsaj $m/2 - 1$ podatkov.
  - Notranje vozlišče, ki ima k sinov, vsebuje k-1 podatkov.

– Vsi listi B-drevesa so na istem nivoju. S tem nivojem je torej določena višina B-drevesa.

– Ključi so v posameznih vozliščih B-drevesa urejeni v naraščajočem vrstnem redu od leve proti desni.

Vidimo, da je število podatkov v posameznih vozliščih v B-drevesu in število naslednikov oz. sinov teh vozlišč navzgor in navzdol omejeno z redom drevesa.

Podatki notranjih vozlišč B-drevesa oziroma njihovi ključi delujejo kot nekakšne delitve svojih poddreves.

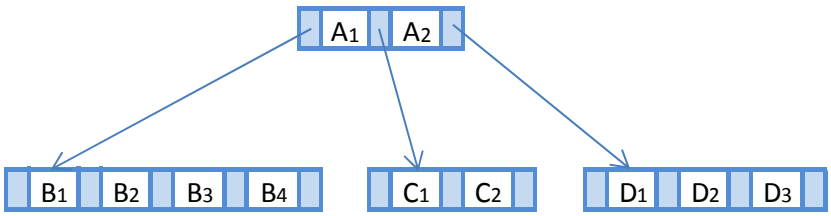


Slika 71: Vozlišče B-drevesa

B-drevo je nadgradnja iskalnega dvojiškega drevesa; je večsmerno iskalno drevo. Če označimo s k_1, k_2, k_3, …, k_i ključe posameznih podatkov vozlišča in s p_0, p_1, p_2, …, p_i kazalce na sinove, pri čemer je $m/2 \leq i \leq m$, potem velja:

- $k_1 < k_2 < k_3 < \ldots < k_i$.
- Vsi ključi iz poddrevesa, na katerega kaže p_i, so večji od k_i in manjši od k_{i+1}.

Če ima torej neko notranje vozlišče 3 sinove (ali 3 poddrevesa) in podatka s ključema A1 in A2, kjer je A1 < A2, potem so vsi ključi podatkov v levem poddrevesu manjši od A1, ključi podatkov v srednjem poddrevesu večji od A1 in manjši od A2 in vsi ključi podatkov v desnem poddrevesu večji od A2.



Slika 72: Vozlišče z elementoma A1 in A2 ter tremi sinovi

In ker so tudi ključi v posameznem vozlišču urejeni, za B-drevo oziroma njihove podatke (ključe) na sliki Slika 72 torej velja:

- $B_1 < B_2 < B_3 < B_4 < A_1$,
- $A_1 < C_1, < C_2 < A_2$,
- $A_2 < D_1 < D_2 < D_3$.

4.2. Terminologija pri B-drevesih

4.2.1. Notranja vozlišča

Notranja vozlišča so vsa vozlišča B-drevesa, razen listov in korena. Vsako vozlišče hrani nek nabor podatkov in kazalcev na poddrevesa. Vsako notranje vozlišče vsebuje največ U=m in najmanj L=m/2 sinov. Tako je število podatkov vedno za 1 manjše od števila sinov; število podatkov je med L-1 in U-1. Ker je U bodisi 2L bodisi 2L-1, je vsako notranje vozlišče vsaj pol polno.

Razmerje med U in L pomeni, da se lahko dve na pol polni vozlišči združita in tako dobimo polno vozlišče. Polno vozlišče pa je mogoče razdeliti na dve vozlišči, če obstaja prostor za en podatek v očetu. Te lastnosti omogočajo, da postopek vstavljanja in brisanja elementov iz drevesa ohranja lastnosti B-dreves.

4.2.2. Koren B-drevesa

Koren je posebno vozlišče B-drevesa, katerega število sinov je navzgor omejeno (kot pri vseh notranjih vozliščih). Spodnja meja glede števila sinov pa pri korenu ni določena. V primeru, ko je število vseh podatkov B-drevesa manjše od L-1, je koren edino vozlišče v drevesu, brez sinov.

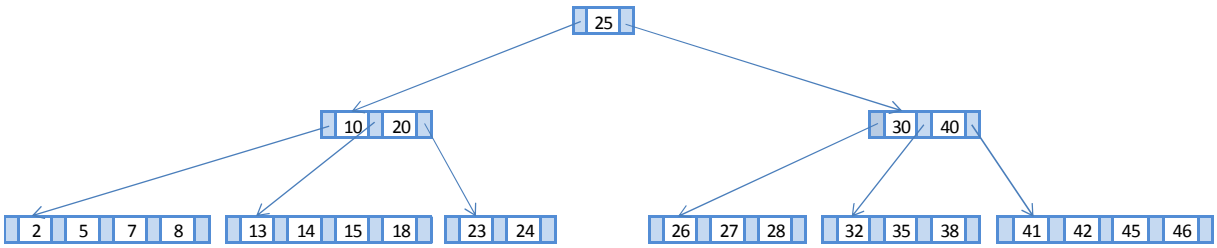
4.2.3. Listi B-drevesa

Izraz list je v različnih literaturah določen različno. V nekaterih primerih je nivo listov določen z najnižjim nivojem vozlišč, v drugih primerih pa je stopnja listov za eno stopnjo nižje od najnižjih vozlišč.

V nadaljevanju diplomske naloge bomo za liste šteli končna vozlišča B-drevesa, torej vozlišča brez sinov. Struktura listov v B-drevesu je enaka kot pri notranjih vozliščih. Torej imajo listi enake omejitve glede števila podatkov in pripadajočih ključev kot notranja vozlišča drevesa.

Časovna zahtevnost operacij iskanja, vstavljanja in brisanja raste z višino drevesa. Kot pri vseh uravnoteženih drevesih pa časovna zahtevnost algoritmov raste veliko počasneje kot število podatkov B-drevesa (glej razdelek 3.2.5).

Primer B-drevesa:



Slika 73: B-drevo reda 5

Na sliki Slika 73 je prikazano B-drevo reda 5. Po definiciji B-drevesa reda 5 vsako vozlišče vsebuje največ 4 podatke in ima največ 5 sinov (lahko se prepričamo, da to res velja za drevo na sliki). Koren drevesa vsebuje le en podatek in ima dva sinova. Ostala vozlišča imajo najmanj 2 in največ 4 podatke. Število podatkov v vozlišču je vedno za 1 manjše od števila sinov tega vozlišča. Vsi listi so na nivoju 3. Podatki v vozliščih drevesa so urejeni po velikosti od najmanjšega do največjega.

Največje število podatkov, ki jih hranimo v B-drevesu na določenem nivoju, je odvisno od reda drevesa *m*. Število podatkov na posameznem nivoju predstavimo v tabeli Tabela 2.

Tabela 2: Maksimalno število podatkov na nivoju B-drevesa

NIVO	MAKSIMALNO ŠTEVILO PODATKOV NA TEM NIVOJU
1	m-1
2	$m(m$-1)
3	$m^2(m$-1)
⋮	⋮
h	$m^{h-1}(m$-1)

Največje število podatkov e_{max}, ki jih lahko hranimo v B-drevesu reda m in višine h, je enako vsoti števil podatkov na posameznih nivojih:

$$e_{max} = (1+m+m^2+\ldots+m^{h-1})(m-1) = m^h-1.$$

V B-drevesu reda m=23 in višine h=3, je torej največje možno število podatkov, ki jih hranimo v tem drevesu, enako 12.166 (23^3-1 = 12.167 – 1).

4.3. Osnovne operacije nad B-drevesi

4.3.1. Iskanje v B-drevesu

Iskanje podatka v B-drevesu je podobno iskanju v iskalnem dvojiškem drevesu. Z iskanjem začnemo vedno v korenu drevesa, nato pa drevo rekurzivno prehodimo od vrha do dna.

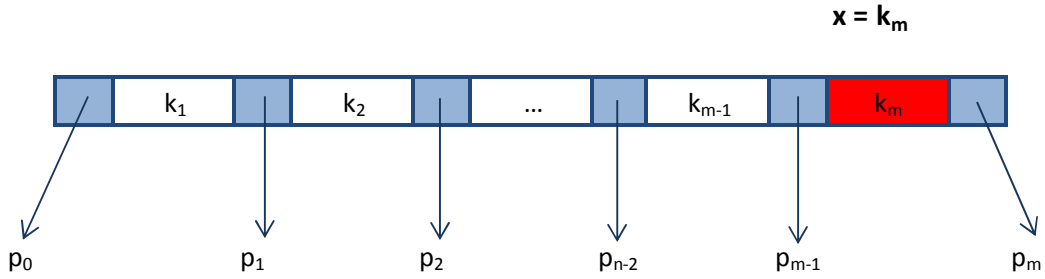
Naj bo podano neko B-drevo reda m, v katerem želimo poiskati določen podatek.

Pri pregledovanju drevesa bomo morali upoštevati, da je v vsakem vozlišču več podatkov, torej bomo morali iskati tudi v samem vozlišču. V primeru, ko je red m razmeroma majhen, zadostuje navadno zaporedno iskanje. Če pa je m zadosti velik, lahko uporabimo bisekcijo. Ta algoritem je namenjen

iskanju podatka v urejeni tabeli. Ker so podatki v vozliščih B-drevesa urejeni po velikosti od najmanjšega do največjega, si lahko vozlišča predstavljamo kot urejene tabele.

Recimo, da imamo vozlišče v obliki, ki jo kaže slika Slika 71: Vozlišče B-drevesa, in argument iskanja *x*.

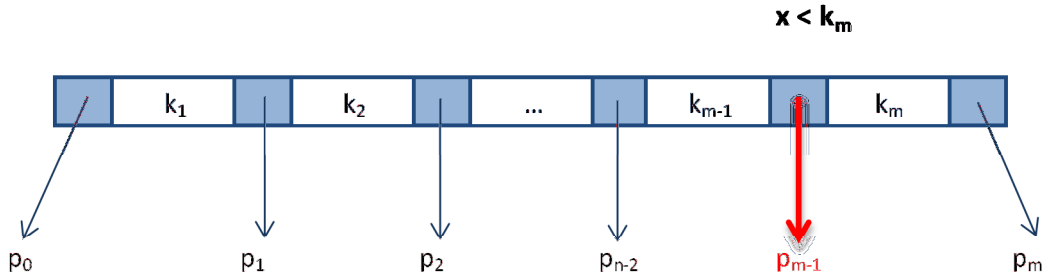
Ko v vozlišču pridemo do podatka, ki je enak iskanemu (k_m=x), z iskanjem zaključimo (Slika 74).



Slika 74: Iskanje ključa v vozlišču B-drevesa ($x = k_m$)

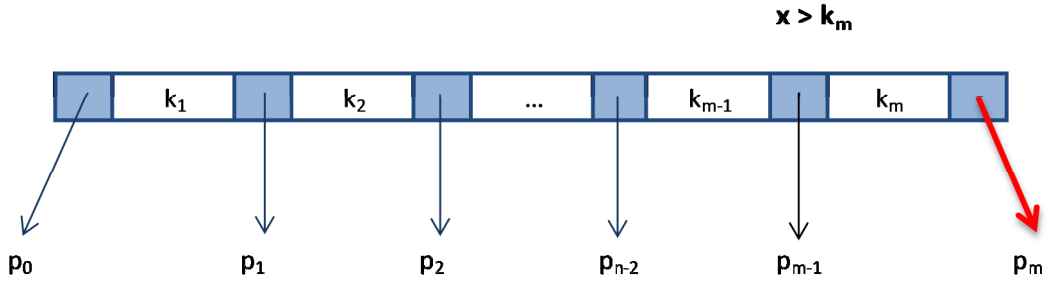
Če postopek iskanja ni uspešen, imamo opravka z enim od naslednjih primerov:

- Podatka z iskanim ključem ni v tem vozlišču, naletimo pa na ključ, ki je večji od iskanega (k_m>x). V tem primeru z iskanjem nadaljujemo v poddrevesu, na katerega kaže neposredni levi kazalec podatka s tem ključem (Slika 75). Če takega poddrevesa ni (takrat ima levi kazalec vrednost nil), potem podatka z iskanim ključem *x* ni v celotnem drevesu. Z iskanjem zaključimo.



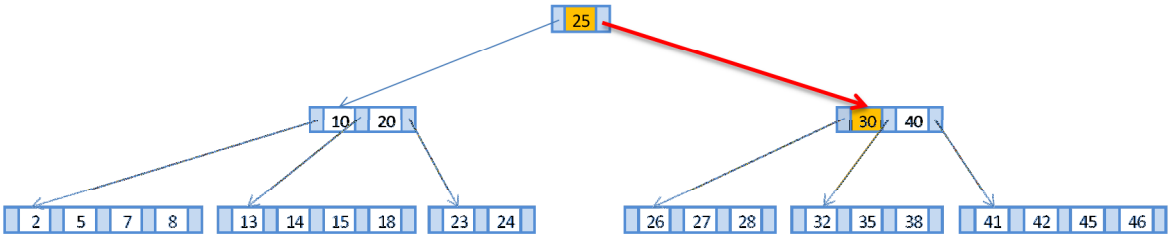
Slika 75: Iskanje ključa v vozlišču B-drevesa ($x < k_m$)

- Podatka z iskanim ključem ni v tem vozlišču, vsi ključi pa so manjši od iskanega ($k_m < x$). Z iskanjem nadaljujemo v vozlišču poddrevesa, na katerega kaže desni kazalec zadnjega podatka v tem vozlišču. Če takega poddrevesa ni, z iskanjem zaključimo. Podatka z iskanim ključem *x* ni v drevesu.



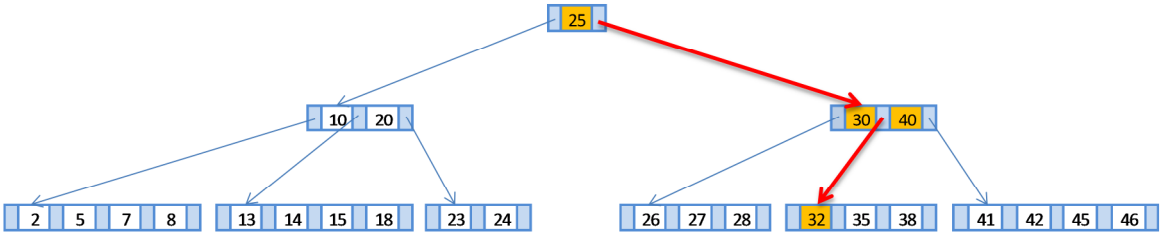
Slika 76: Iskanje ključa v vozlišču B-drevesa ($x > k_m$)

Postopek iskanja ključa v B-drevesu si oglejmo še na primeru. V B-drevesu na sliki Slika 73 iščemo podatek x=35. Z iskanjem začnemo v korenu drevesa. Ker tu podatka z iskanim ključem ni, ključ v korenu pa je manjši od iskanega, nadaljujemo v poddrevesu, na katerega kaže desni kazalec ključa 25. Prvi korak iskanja podatka s ključem 35 je prikazan na sliki Slika 77.



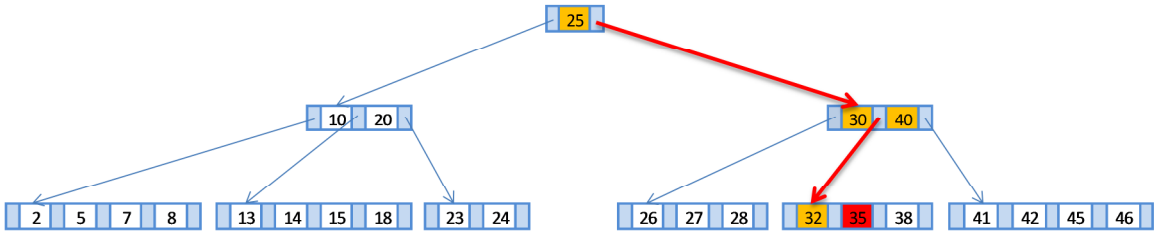
Slika 77: Iz korena se iskanje nadaljuje v desnem poddrevesu

Ker je red drevesa razmeroma majhen (m=5), nadaljujemo z zaporednim iskanjem. Ključ prvega podatka primerjamo z iskanim ključem. Ker ne ustreza iskanemu ključu, primerjamo še drugi ključ v tem vozlišču. Naletimo na podatek s ključem, ki je večji od iskanega (40>35), zato pot iskanja nadaljujemo v poddrevesu, na katerega kaže neposredni levi kazalec ključa 40.



Slika 78: Iskanje nadaljujemo levem poddrevesu

Zopet zaporedno primerjamo ključe v poddrevesu z iskanim. Pri drugem primerjanju naletimo na iskani ključ. Z iskanjem lahko zaključimo. Celotna pot iskanja ključa 35 je prikazana na sliki Slika 79.



Slika 79: Pot iskanja ključa 35 v B-drevesu

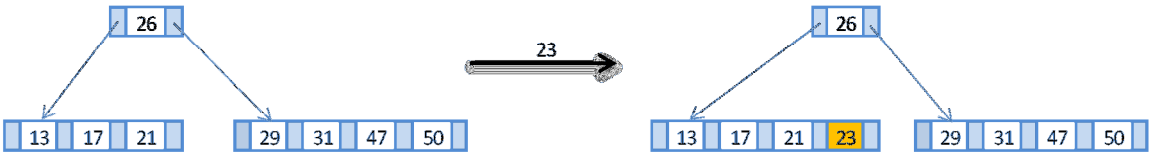
Če pa bi iskali npr. 37, bi opravili enake korake. Le po primerjanju s 35, bi pregledali še naslednji podatek. Ta je 40 in ker je njegov levi kazalec nil (saj smo v listu), bi vedeli, da podatka s ključem 37 v drevesu ni.

4.3.2. Vstavljanje v B-drevo

Podatke vedno vstavljamo v liste drevesa. Seveda moramo podatek vstaviti na ustrezno mesto, tako, da so ključi znotraj vozlišča v naraščajočem vrstnem redu. Najprej torej poiščemo ustrezni list in v njem mesto, kamor bomo vstavili nov podatek z danim ključem. Nadaljnji postopek pa je odvisen od števila podatkov v posameznem vozlišču.

Če je potrebno vstaviti podatek v vozlišče z $n < m$-1 podatki, potem je postopek vstavljanja omejen na tisto vozlišče. V njem je namreč še vedno prostor za nov podatek (vemo, da vsako vozlišče B-drevesa reda m, razen korena, vsebuje najmanj $m/2$ - 1 in največ m-1 podatkov).

Primer takšnega vstavljanja v B-drevo je prikazan na sliki Slika 80.



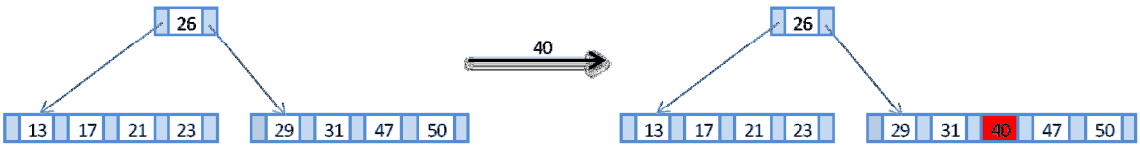
Slika 80: Vstavljanje podatka s ključem 23 v B-drevo

Na sliki Slika 80 je prikazan postopek vstavljanja podatka s ključem 23 v B-drevo reda 5. Podatek smo vstavili v list drevesa. Ker so v tem listu ključi vseh podatkov manjši od vstavljenega, je ta zasedel zadnje mesto v tem vozlišču.

Kadar pa vstavljamo podatek v že polno vozlišče (vozlišče z m-1 elementi), ima ta postopek posledice za celotno zgradbo B-drevesa. Dobimo nova vozlišča, saj moramo vozlišče s preveč podatki razdeliti.

Postopek je sledeč:

- Podatek vstavimo na ustrezno mesto. Sedaj je v tem vozlišču (listu) m podatkov.
- Vozlišče po vstavljanju podatka razpolovimo (dodamo eno vozlišče), tako da število podatkov enako porazdelimo med vozliščema. V vsakem od vozlišč je sedaj $m/2 - 1$ podatkov in pripadajočih ključev. V levem vozlišču so podatki, katerih ključi so manjši od ključa srednjega podatka, v desnem vozlišču pa podatki, katerih ključi so večji od ključa srednjega podatka. Kadar je v vozlišču sodo število elementov, je po delitvi vozlišča v levem vozlišču en element manj kot v desnem.
- Srednji ključ premaknemo en nivo višje- vstavimo ga na ustrezno mesto v očetu prvotnega vozlišča.



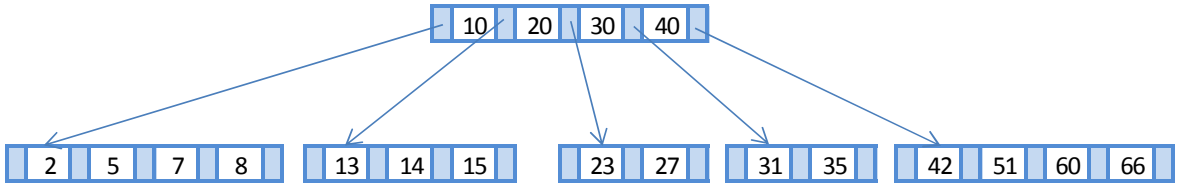
Slika 81: Vstavljanje podatka s ključem 40 v B-drevo

Ko vstavimo 40, se je lastnost B-drevesa porušila. V vozlišču, kamor smo 40 vstavili, je število podatkov preseglo zgornjo mejo. Vstavljeni podatek s ključem 40 je (slučajno) srednji podatek v vozlišču (Slika 81). Ključ prestavimo v očeta, v tem primeru v koren drevesa. Sedaj dobimo drevo, ki ima vse značilnosti B-drevesa (Slika 82).

Slika 82: Preoblikovanje drevesa v B-drevo

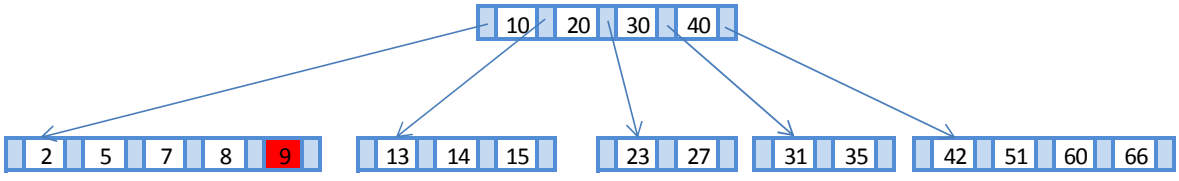
Vstavljanje podatka z danim ključem v predhodnika pa lahko povzroči, da je tudi v predhodniku preveč podatkov. Tako se lahko deljenje vozlišča širi navzgor. V skrajnem primeru doseže koren. Če se delitev vozlišča širi vse do korena, ustvarimo nov koren z enim podatkom in dvema sinovoma, saj spodnja meja glede velikosti notranjih vozlišč ne velja za koren. To pa je edini način, da B-drevo poveča svojo višino. S tem, ko smo ustvarili nov koren , se višina B-drevesa poveča za 1.

Povečanje višine B-drevesa si poglejmo na primeru. V B-drevo reda 5 s slike Slika 83 želimo vstaviti podatek s ključem 9.



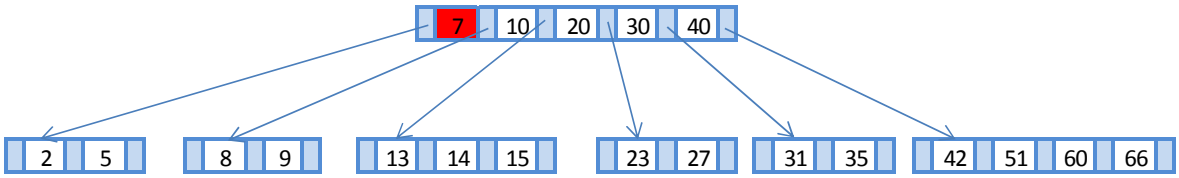
Slika 83: B-drevo reda 5 in višine 2

Podatek s ključem 9 vstavimo v najbolj levi list drevesa. Ta v njem zasede zadnje mesto, saj je ključ 9 največji med vsemi ključi v tem vozlišču (Slika 84).



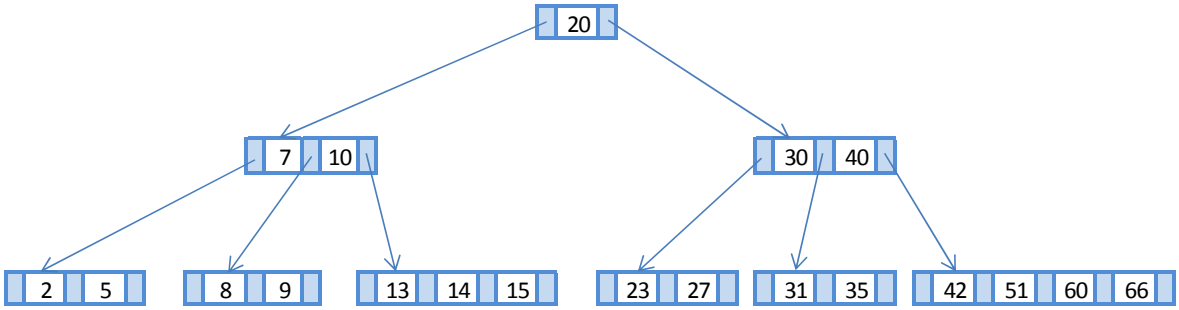
Slika 84: Vstavljanje podatka s ključem 9 v B-drevo

Ker je to vozlišče že polno, ga bomo razpolovili, srednji podatek s ključem 7 pa vstavili nivo višje, v koren drevesa (Slika 85).



Slika 85: Deljenje vozlišča in prestavljanje srednjega podatka

Ko smo podatek s ključem 7 vstavili v koren drevesa, smo porušili lastnost B-drevesa. Vsako vozlišče B-drevesa reda 5 ima največ 4 podatke. V našem primeru pa ima koren drevesa pet podatkov. Srednji podatek v korenu zato prestavimo nivo višje. Drevo dobi nov koren in postane višje (Slika 86).

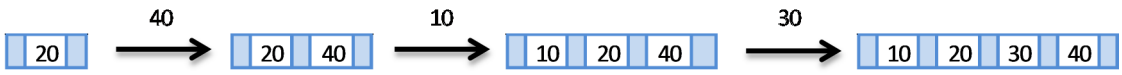


Slika 86: Povečanje višine B-drevesa

V primerjavi z dvojiškimi drevesi, kjer drevo raste od korena proti listom, B-drevo raste od listov navzgor proti korenu. B-drevo torej gradimo tako, da podatke vstavljamo v liste drevesa in nato iz njih nadaljujemo gradnjo notranjih vozlišč B-drevesa.

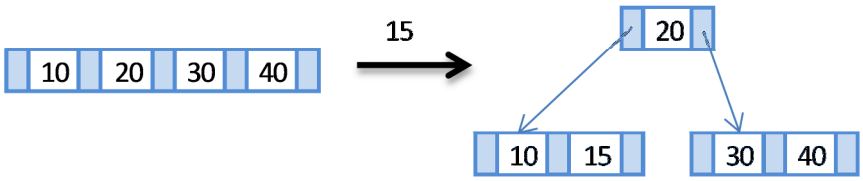
Vstavljanje v B-drevo si poglejmo še na primeru gradnje B-drevesa reda 5 pri naslednjem zaporedju ključev za vstavljanje: 20, 40, 10, 30, 15, 35, 7, 26, 18, 22, 5, 42, 13, 46, 27, 8, 32, 24, 45 in 25.

Vemo, da ima B-drevo reda 5 v vsakem vozlišču (razen v korenu) najmanj 2 in največ 4 podatke (ključe). Zato prve 4 podatke s ključi 20, 40, 10 in 30 vstavimo v koren drevesa v naraščajočem vrstnem redu.



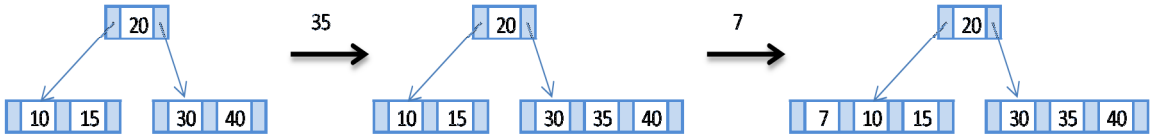
Slika 87: Vstavljanje podatkov s ključi 20, 40, 10 in 30 v prazno B-drevo

Vstavljanje naslednjega podatka (podatek s ključem 15) v koren drevesa poruši lastnost B-drevesa. V korenu je namreč 5 podatkov, zato moramo drevo preoblikovati. Vozlišče razdelimo na dva dela; torej ustvarimo novo vozlišče. Podatke enako porazdelimo med vozliščema (podatka s ključema 10 in 15 v eno vozlišče in podatka s ključema 30 in 40 v drugo vozlišče), srednjega (podatek s ključem 20) pa premaknemo en nivo višje. Drevo torej dobi nov koren s ključem 20 (Slika 88).

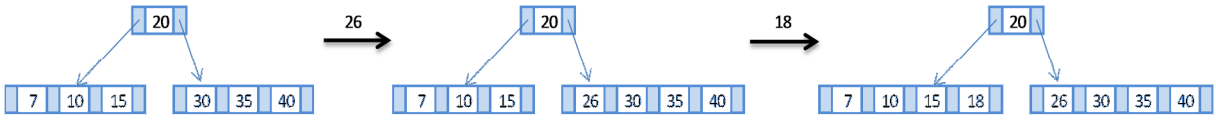


Slika 88: Gradnja B-drevesa (podatek s ključem 15)

Podatke s ključi 35, 7, 26 in 18 vstavimo v ustrezno vozlišče. Vsa vozlišča B-drevesa še vedno vsebujejo ustrezno število podatkov.

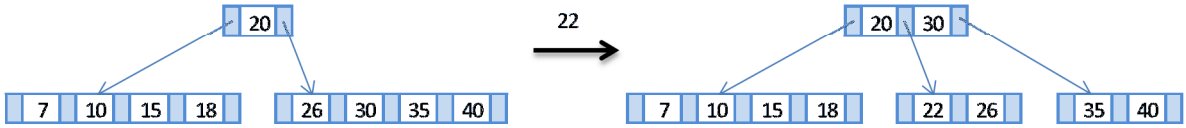


Slika 89: Gradnja B-drevesa (podatka s ključema 35 in 7)



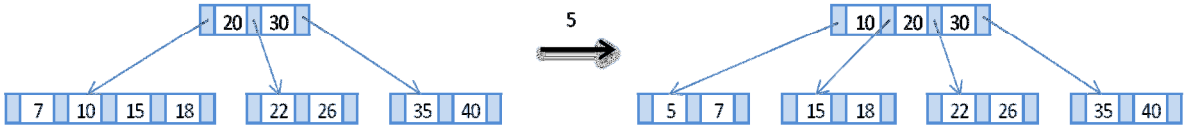
Slika 90: Gradnja B-drevesa (podatka s ključema 26 in 18)

Po vstavljanju podatka s ključem 22 v desni list drevesa, moramo drevo preoblikovati. List se razpolovi, pri čemer vsako vozlišče sedaj vsebuje dva podatka. Srednji ključ (30) vstavimo v očeta, v koren drevesa. Dobimo B-drevo na sliki Slika 91.



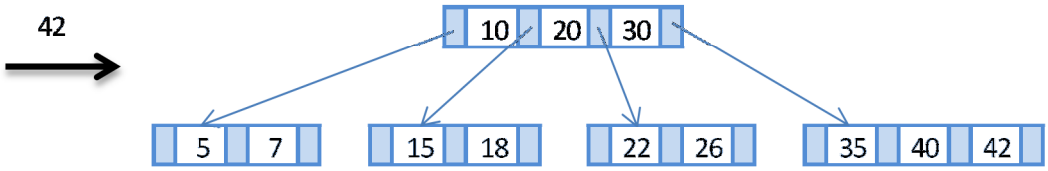
Slika 91: Gradnja B-drevesa (podatek s ključem 22)

Vstavljanje naslednjega ključa, 5, zopet poruši lastnost B-drevesa. Po vstavljanju podatka s ključem 5 v levi list, se ta razpolovi, srednji podatek s ključem 10 pa vstavimo v očeta. Postopek je prikazan na sliki Slika 92.

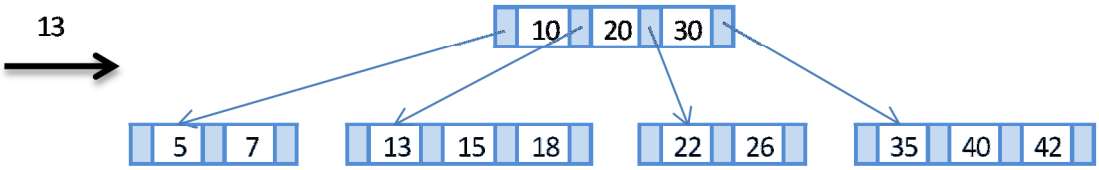


Slika 92: Gradnja B-drevesa (podatek s ključem 5)

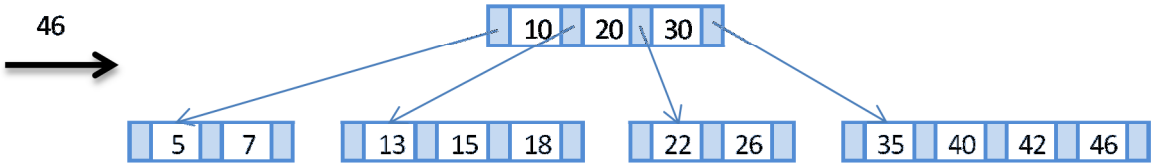
Podatke s ključi 42, 13, 46, 27, 8 vstavimo na ustrezno mesto v pripadajočih vozliščih brez dodatnega preurejanja.



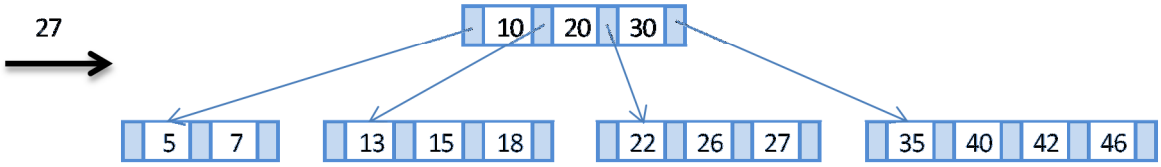
Slika 93: Gradnja B-drevesa (podatek s ključem 42)



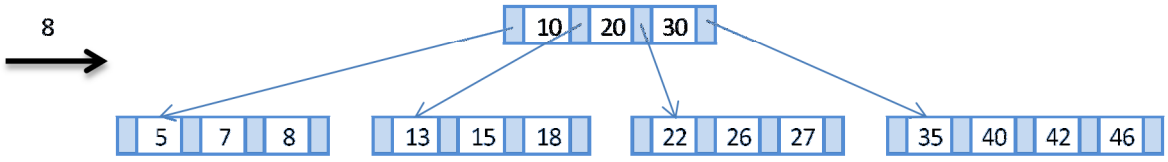
Slika 94: Gradnja B-drevesa (podatek s ključem 13)



Slika 95: Gradnja B-drevesa (podatek s ključem 46)

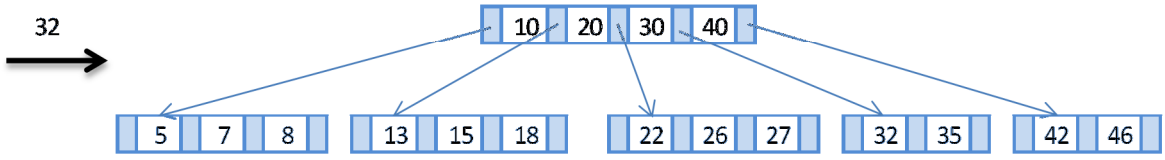


Slika 96: Gradnja B-drevesa (podatek s ključem 27)



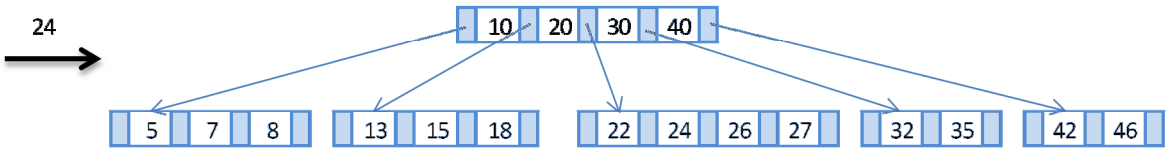
Slika 97: Gradnja B-drevesa (podatek s ključem 8)

Vstavljanje podatka s ključem 32 v B-drevo zahteva ponovno deljenje vozlišča, saj je le-to že pred vstavljanjem polno. Vozlišče z podatki in njihovimi ključi 32, 35, 40, 42 in 46 razpolovimo, srednji podatek (ključ 40) pa prestavimo en nivo višje, v koren.

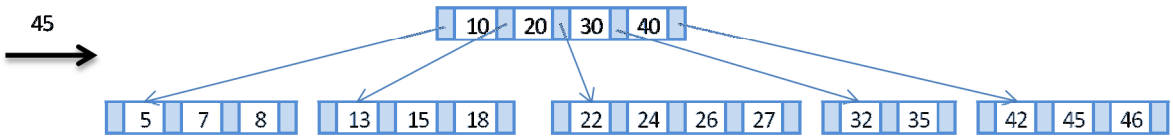


Slika 98: Gradnja B-drevesa (podatek s ključem 32)

Ključa 24 in 45 vstavimo na ustrezno mesto brez dodatnega preoblikovanja drevesa.

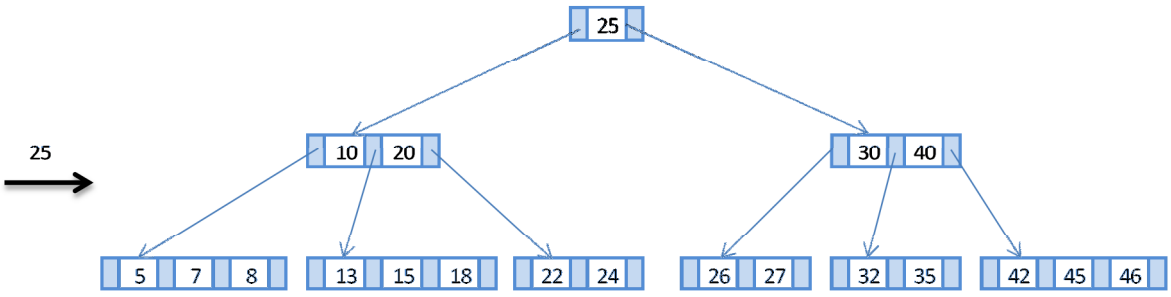


Slika 99: Gradnja B-drevesa (podatek s ključem 24)



Slika 100: Gradnja B-drevesa (podatek s ključem 45)

Z zadnjim vstavljenim podatkom (ključ 25) se spremeni višina B-drevesa. List, kamor ključ vstavimo, moramo razpoloviti in srednji podatek s ključem 25 (to je podatek, ki smo ga vstavili) prestavimo v koren. Ker je število podatkov v korenu prav tako preseglo zgornjo mejo, moramo postopek ponoviti. Drevo tako dobi nov koren in njegova višina se iz 2 spremeni v 3. Dobljeno drevo je na sliki Slika 101.

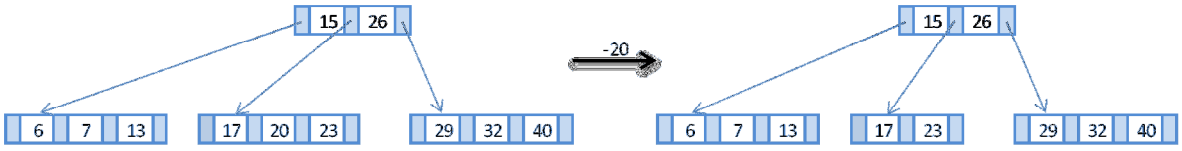


Slika 101: Končno B-drevo po zaporednem vstavljanju

4.3.3. Brisanje podatka iz B-drevesa

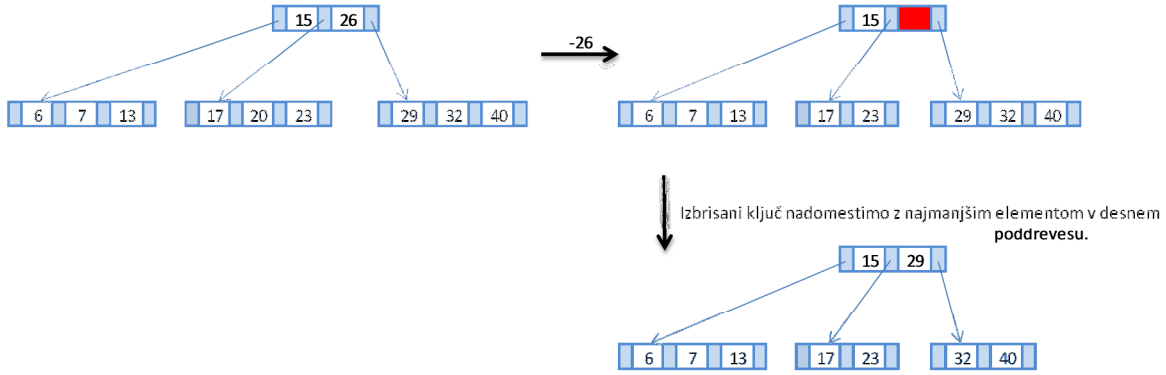
Podobno kot pri vstavljanju novega podatka, tudi pri brisanju obstoječega podatka iz B-drevesa, postopek izvajamo v listih drevesa. Pri tem lahko naletimo na dve situaciji:

– Podatek, ki ga izločamo, je v enem od listov B-drevesa. Takrat je ustrezni postopek enostaven in razumljiv. Podatek preprosto odstranimo. Če je v vozlišču, kjer smo podatek odstranili še vedno vsaj *m*/2 - 1 podatkov, je postopek brisanja končan.



Slika 102: Postopek brisanja podatka iz B-drevesa reda 6 (primer 1)

– Podatek, ki ga želimo izbrisati, ni v listu. Tedaj ga moramo zamenjati z enim od dveh najbližjih podatkov, ki sta v listih B-drevesa in jih torej ni težko izločiti. Izbrisani podatek zamenjamo z največjim podatkom ustreznega levega poddrevesa (torej s predhodnikom) ali minimalnim podatkom ustreznega desnega poddrevesa (z naslednikom). Opisan postopek brisanja podatka iz B-drevesa reda 6 je prikazan na sliki Slika 103.



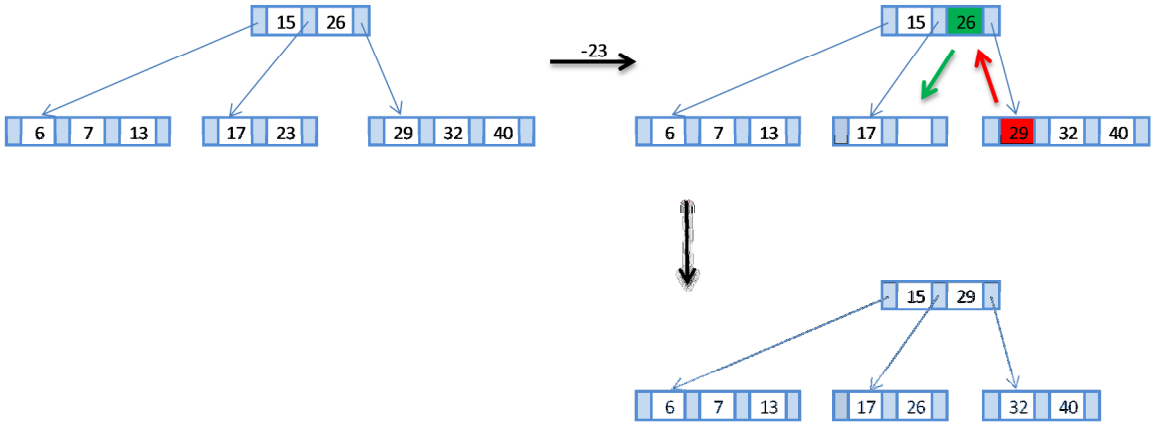
Slika 103: Postopek brisanja podatka iz B-drevesa reda 6 (primer 2)

V drugem primeru (Slika 103) je iskanje najbližjega podatka podobno iskanju, ki ga uporabljamo pri izločanju iz iskalnega dvojiškega drevesa (glej razdelek 2.5.4). Po desnih kazalih se spustimo do lista in zamenjamo podatek, ki ga izločamo, z levim podatkom v listu. V listu je sedaj en podatek manj.

V vsakem primeru moramo po zmanjšanju podatkov preveriti število podatkov v listu. Še vedno mora veljati, da vsako vozlišče vsebuje vsaj *m*/2 - 1 podatkov. Če to po brisanju podatka ne drži, se poruši osnovna lastnost B-drevesa in drevo moramo preurediti.

Preurejanje drevesa je odvisno od števila podatkov, ki jih imata neposredna brata tega vozlišča:

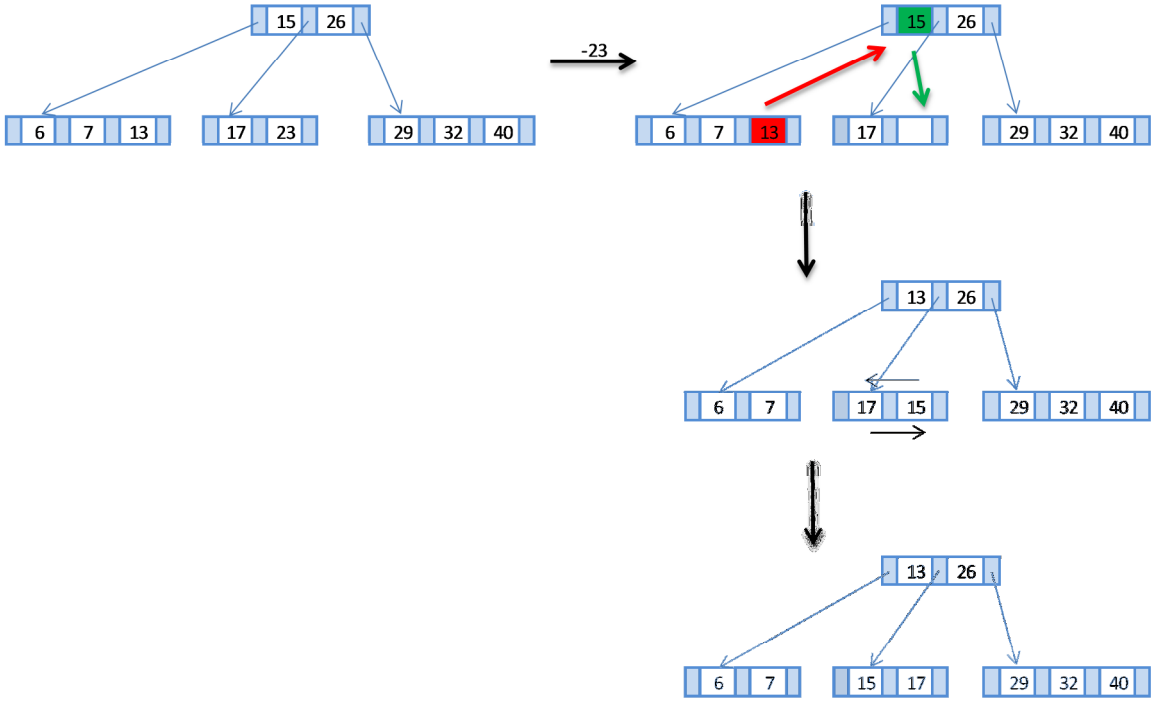
> Če vsaj eden od sosednjih bratov tega vozlišča (levi ali desni) vsebuje več kot minimalno število podatkov (vsaj *m*/2), prestavimo enega izmed podatkov v očeta, podatek iz očeta pa prestavimo v vozlišče, ki ima premalo podatkov.



Slika 104: Preurejanje B-drevesa reda 6 (1.a primer)

Na sliki Slika 104 je prikazan postopek brisanja ključa 23 iz B-drevesa reda 6. Podatek iz sosednjega brata (podatek s ključem 29) prestavimo v očeta, podatek iz očeta (ključ 26) pa v vozlišče, ki ima premalo podatkov- na mesto, kjer smo podatek odstranili.

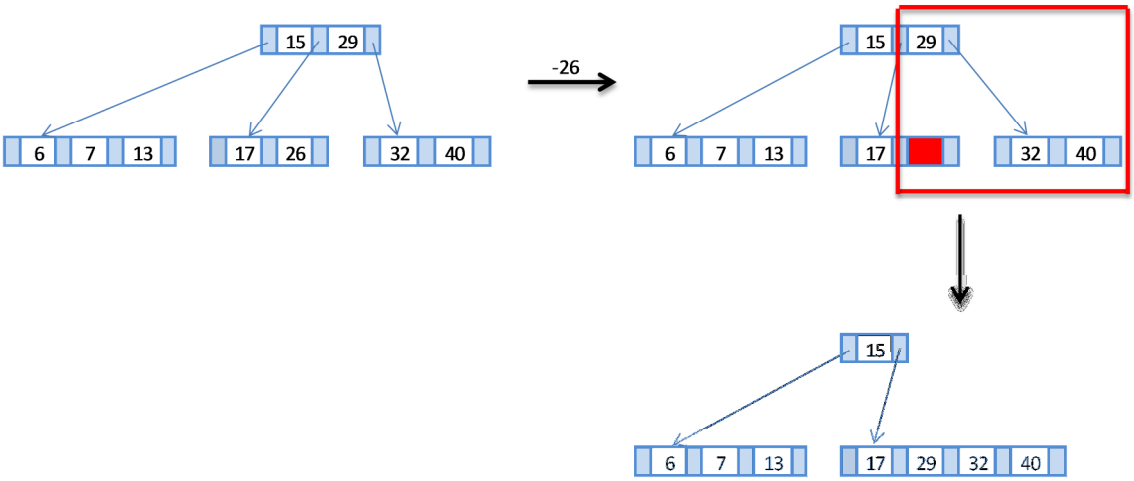
Poglejmo si še primer, ko za podatek, ki ga prestavimo v očeta, vzamemo podatek s ključem 13. To je zadnji podatek v levem bratu vozlišča s premalo podatki.



Slika 105: Preurejanje B-drevesa reda 6 (1.b primer)

Na sliki Slika 105 je prikazan postopek brisanja ključa 23 iz B-drevesa reda 6. Podatek iz levega brata (podatek s ključem 13) prestavimo v očeta, podatek iz očeta (ključ 15) pa v vozlišče, ki ima premalo podatkov- na mesto, kjer smo podatek odstranili. V listu, kjer je bilo premalo podatkov, sta sedaj podatka s ključema 17 in 15, kar ustreza zahtevam B-drevesa glede števila podatkov v vozlišču. Vendar pa podatka nista urejena po velikosti, zato ju moramo zamenjati. Podatek s ključem 15 je na prvem mestu in podatek s ključem 17 na drugem. Dobljeno drevo sedaj ustreza vsem zahtevam podatkovne strukture B-drevo.

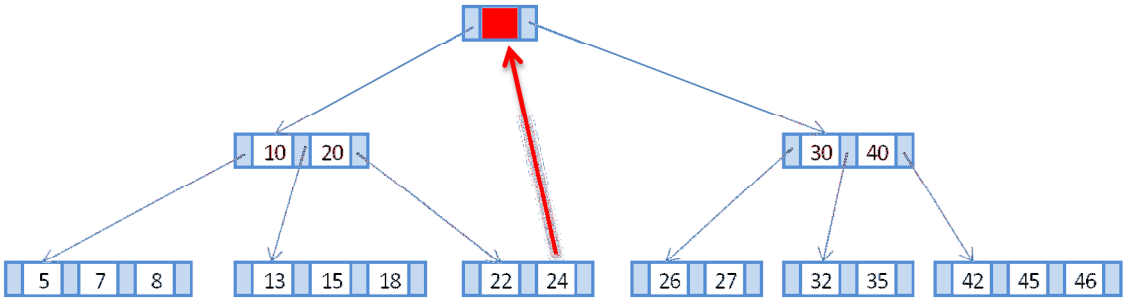
Če ima tako levi kot desni brat minimalno dovoljeno število podatkov (*m*/2 - 1), potem združimo vozlišče s premajhnim številom podatkov, podatek iz očeta in enega brata v skupno vozlišče. Če ima po tem postopku oče premajhno število podatkov, se postopek ponovi.



Slika 106: Preurejanje B-drevesa reda 6 (2. primer)

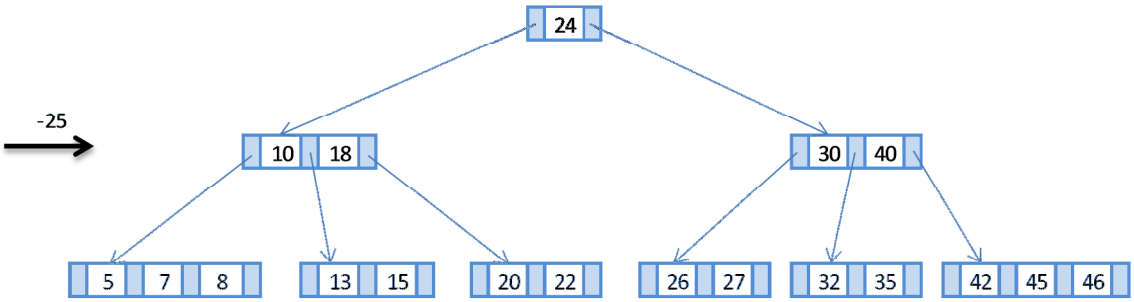
Postopek brisanja podatka iz B-drevesa si sedaj poglejmo še na primeru, ko iz B-drevesa na sliki Slika 101 zaporedno brišemo podatke s ključi 25, 45, 24 in 32.

Iz B-drevesa želimo izbrisati podatek s ključem 25. Ta je v korenu. Na sliki Slika 107 smo izbrisan podatek nadomestili z najbolj desnim (največjim) podatkom v ustreznem levem poddrevesu (naslednikom). To je podatek s ključem 24.



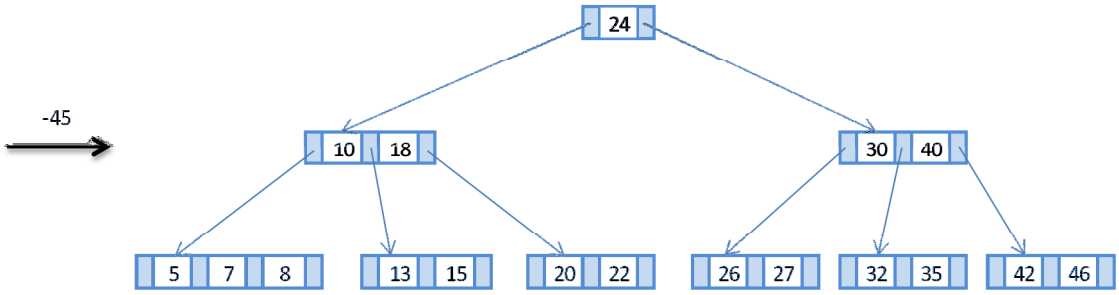
Slika 107: Podatek v korenu nadomestimo z ustreznim podatkom v listu

Število podatkov v vozlišču (listu), iz katerega smo podatek prestavili v koren, je padlo pod spodnjo mejo. Ker pa ima njegov levi brat več kot minimalno število podatkov, prestavimo podatek s ključem 18 v očeta, podatek iz očeta (ključ 20) pa prestavimo v vozlišče s premalo podatki. B-drevo, ki ga dobimo, je prikazano na sliki Slika 108.



Slika 108: Preurejeno B-drevo

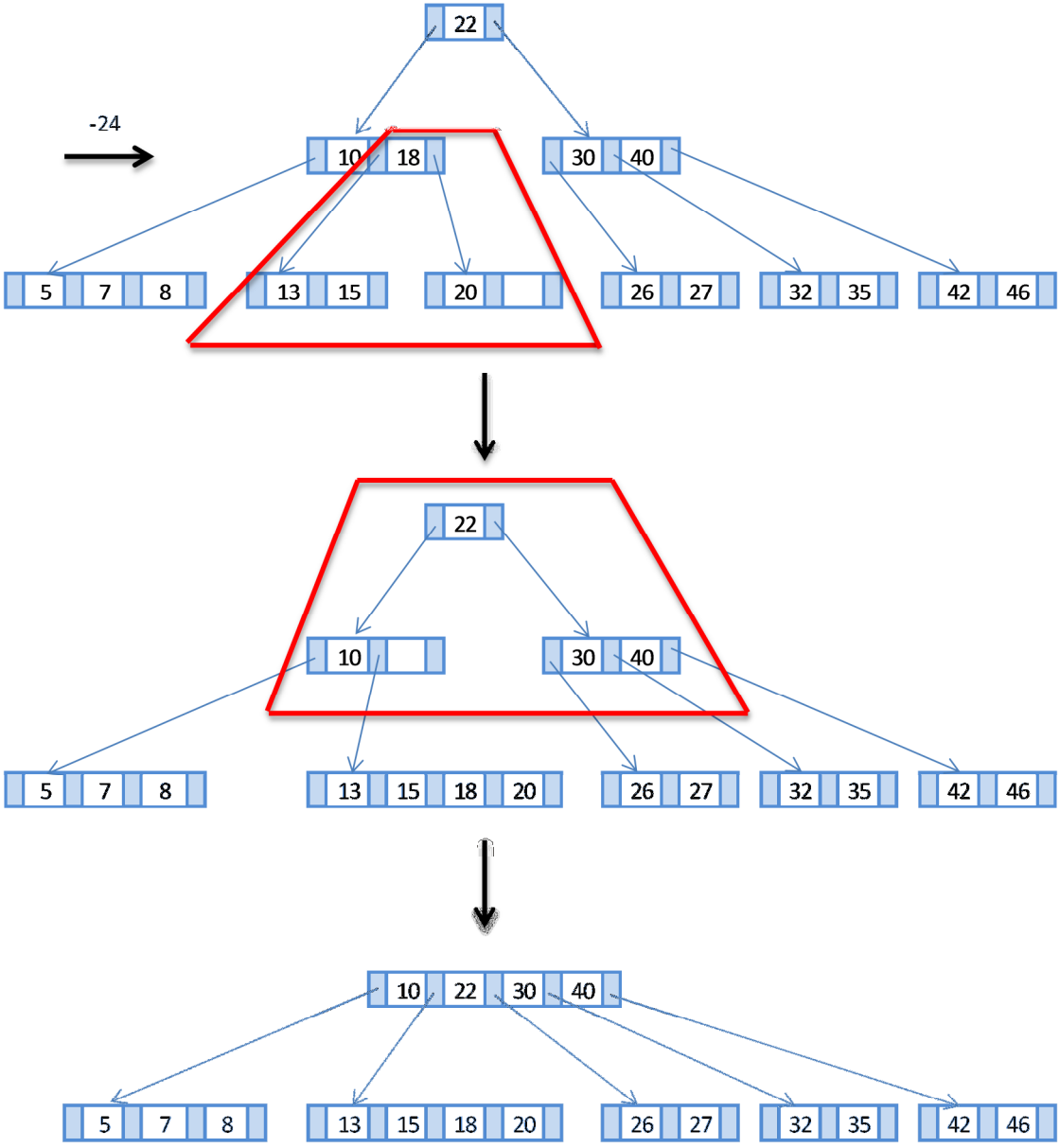
Brisanje podatka s ključem 45 ne zahteva posebnega preurejanja drevesa, saj sta po postopku brisanja v vozlišču, kjer smo podatek izbrisali, še vedno dva podatka (Slika 109).



Slika 109: Zaporedno brisanje iz B-drevesa (element s ključem 45)

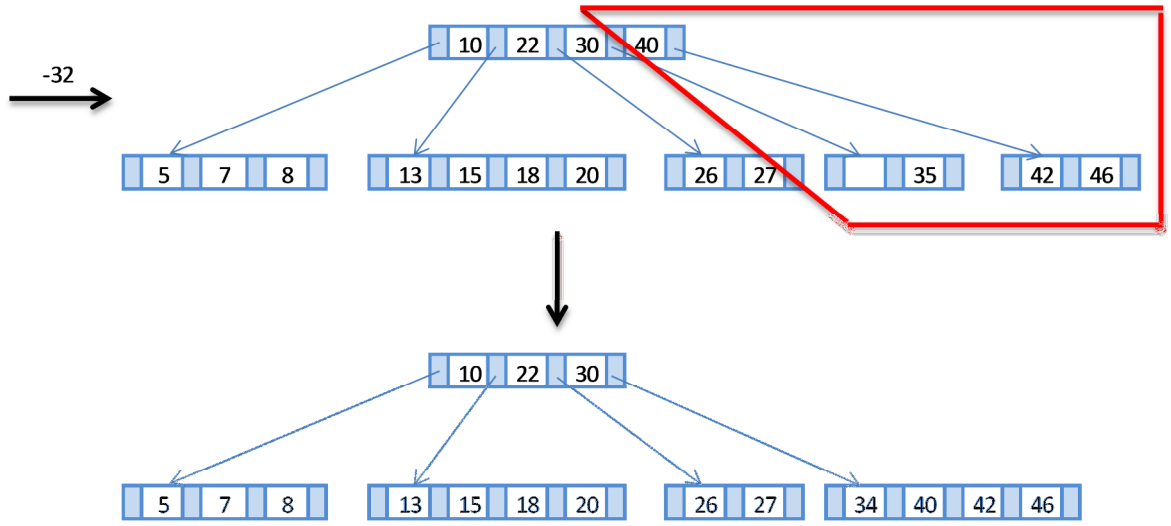
V naslednjem koraku želimo izbrisati podatek s ključem 24. Ker je ta podatek v notranjem vozlišču, ga moramo po brisanju nadomestiti s predhodnikom. V našem primeru je to podatek s ključem 22. Po prestavljanju podatka s ključem 22 v koren drevesa, je v vozlišču, v katerem je bil ta podatek, premalo podatkov. Ker ima neposredni sosed tega vozlišča le 2 podatka, drevo preuredimo tako, da vozlišče s podatkom 20 združimo s sosednjim vozliščem in podatkom iz očeta s ključem 18.

Ko vozlišča združimo, vidimo (Slika 110), da je v očetu le podatek s ključem 10. Zato postopek združevanja vozlišč ponovimo. Podatek iz korena se tako združi s svojima sinovoma v eno vozlišče. Korensko vozlišče zato izbrišemo. S tem se višina drevesa zmanjša.



Slika 110: Združevanje vozlišč

V naslednjem koraku iz dobljenega B-drevesa višine 2 izbrišemo podatek s ključem 32. Vozlišče s tem podatkom je po brisanju padlo pod spodnjo mejo števila podatkov, zato združimo to vozlišče z desnim bratom in očetom. Ker so v korenu štirje podatki, podatek s ključem 40 prenesemo nivo nižje brez dodatnega dela (Slika 111) .



Slika 111: Preurejanje B-drevesa po brisanju podatka s ključem 32

4.4. Časovna zahtevnost operacij nad B-drevesi

4.4.1. Iskanje

Vsako vozlišče B-drevesa reda *m*, razen korena, ima najmanj *m*/2 sinov in najmanj *m*/2 – 1 podatkov. Za korensko vozlišče B-drevesa vemo, da ima vsaj 2 sinova (kadar je notranje vozlišče) ali pa je brez otrok (kadar je celotno drevo sestavljeno samo iz korena). Denimo, da je višina B-drevesa enaka *h*. Takrat imamo v B-drevesu na nivoju 1 najmanj 2 vozlišči, na nivoju 2 najmanj $2(m/2) = m$ vozlišč, na nivoju 3 najmanj $2(m/2)^2 = m^2/2$,… Vsi listi B-drevesa so na nivoju *h* in teh je najmanj $2(m/2)^{h-1}$. Višina B-drevesa reda *m* z *n* podatki je:

$$h \leq \log_{(\frac{m}{2})} \frac{m+1}{2}.$$

Število posegov v disk pri iskanju podatka nad B-drevesom zahteva največ tako število operacij, ki je sorazmerno višini B-drevesa. Časovna zahtevnost operacije iskanja v B-drevesu je torej logaritemska ($O(\log(n))$.

4.4.2. Vstavljanje in brisanje

Operaciji vstavljanja novega podatka in brisanja obstoječega podatka v B-drevesu, v primerjavi z operacijo iskanja, zahtevata nekaj dodatnih stroškov. Kadar podatek vstavljamo ali brišemo iz drevesa, se lahko lastnost B-drevesa, poruši. Takrat moramo drevo preoblikovati. Vendar pa je časovna zahtevnost še vedno odvisna predvsem od višine B-drevesa.

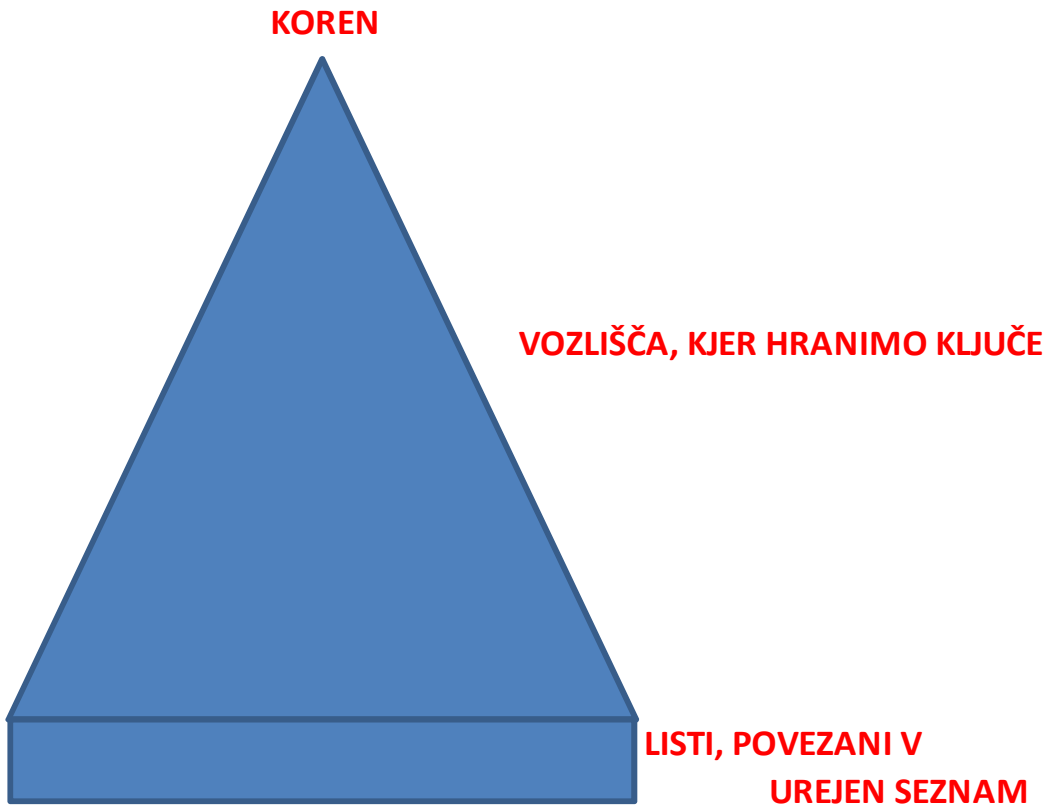
Časovna zahtevnost operacij vstavljanja in brisanja podatka nad B-drevesom reda *m*, ki vsebuje *n* podatkov, je enaka $O(\log_{\frac{m}{2}} n)$.

5. B^+-DREVESA

Osnovne lastnosti B-dreves in operacije nad njimi smo si natančneje ogledali v prejšnjem poglavju.

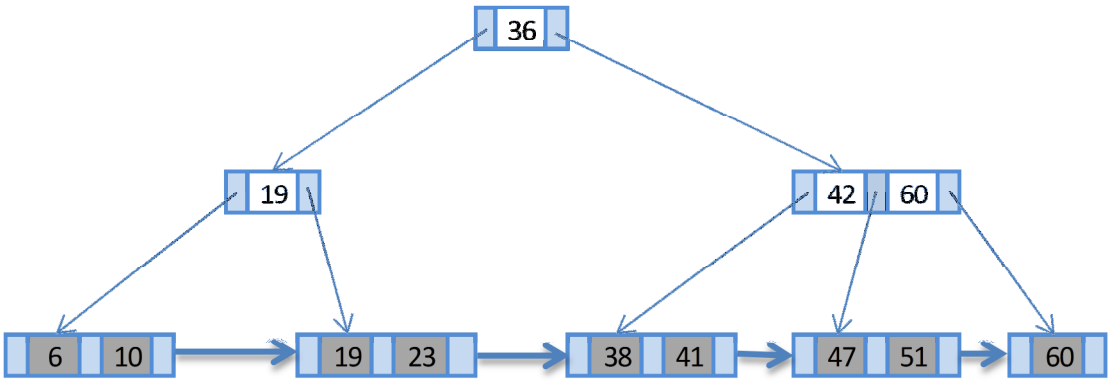
Ena od oblik B-drevesa je podatkovna struktura B^+-drevo, ki si jo bomo podrobneje pogledali v nadaljevanju. V B^+-drevesu so podatkovni zapisi (celotni podatki) shranjeni le v končnih vozliščih drevesa (v listih), v notranjih vozliščih pa hranimo le ključe, ki omogočajo (usmerjajo) postopek iskanja. Zato se ključi v B^+-drevesu lahko ponavljajo, kar pomeni, da se ključi, ki določajo podatke v listih drevesa lahko pojavijo tudi v (več) notranjih vozliščih.

Shema B^+-drevo je prikazana na sliki Slika 112.



Slika 112: Shema B^+-drevesa

Oglejmo si še konkreten primer B^+-drevesa.



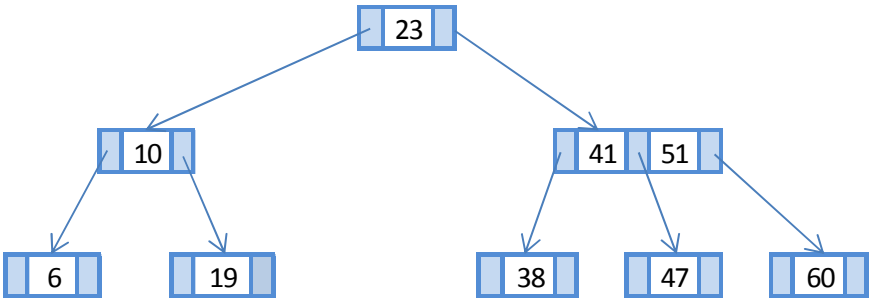
Slika 113: Primer B$^+$-drevesa

Listi B$^+$-drevesa so med seboj povezani s kazalci. Tako dobimo seznam podatkov, ki jih med seboj lahko primerjamo po velikosti. Govorimo torej o urejenem seznamu.

Na konkretnem primeru B$^+$-drevesa (Slika 113) vidimo, da se ključa 19 in 60 pojavita v notranjih vozliščih in v listih drevesa. V notranjih vozliščih nastopata zgolj kot ključa, v listih drevesa pa sta del podatkov (kot njuna enolična identifikatorja). Če bi v tem drevesu iskali podatek s ključem 19, bi se morali od korena spustiti do njegovega levega sina, kjer bi nas ključ 19 usmeril do lista, na katerega kaže njegov desni kazalec. Tako bi prišli do končnega podatka s ključem 19. Takrat lahko z iskanjem zaključimo.

Tako kot na sliki Slika 113 bomo tudi v nadaljevanju diplomske naloge notranja vozlišča in pripadajoče ključe označevali z belimi kvadratki, celotni zapisi v listih pa bodo označeni s sivimi kvadratki.

Če B$^+$-drevo primerjamo z »običajnim« B-drevesom, je očitno, da bomo za B$^+$-drevo potrebovali več prostora, saj so v B-drevesu podatki tudi v notranjih vozliščih. Pri B$^+$-drevesih notranja vozlišča uporabljamo le za shranjevanje ključev. To bomo najbolje videli na primeru. Denimo, da želimo sestaviti B-drevo reda 3, v katerem hranimo iste podatke kot v B$^+$-drevesu iz slike Slika 113. Takrat dobimo na primer tako B-drevo:



Slika 114: B-drevo s podatki iz slike Slika 113

Denimo, da imamo B$^+$-drevo reda *d*. Tu pod redom drevesa razumemo isti pojem kot pri B-drevesu. To pomeni, da ima vsako notranje vozlišče reda *d* najmanj *d*/2 in največ *d* sinov ter en podatek manj kot ima sinov. B$^+$-drevo je iskalno drevo, ki ima naslednje lastnosti:

- Za število sinov (naslednikov) posameznih vozlišč velja:
  - Vsako vozlišče ima največ *d* sinov.

- Vsako notranje vozlišče, razen korena, ima najmanj $d/2$ sinov. Torej so vsa vozlišča B^+-drevesa vsaj pol polna.
- Koren je list ali pa ima vsaj dva sinova.

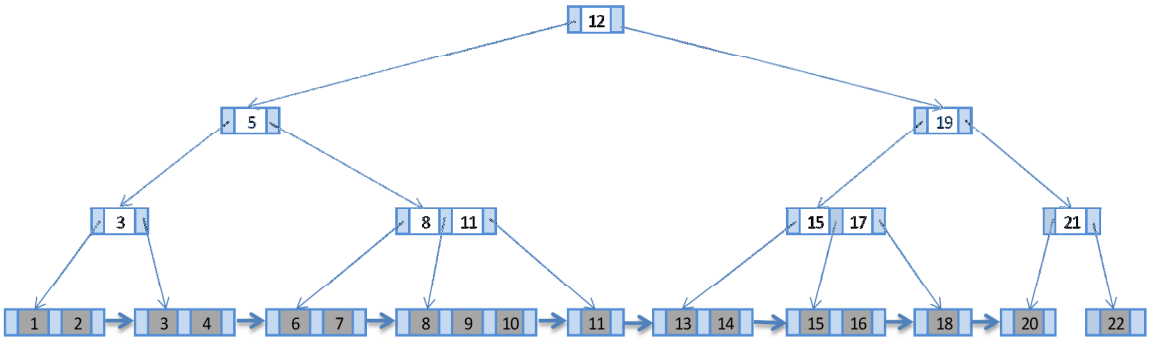
– Število podatkov in pripadajočih ključev je odvisno od reda drevesa d:
  - Vsako vozlišče ima največ d - 1 podatkov. Če ima vozlišče d - 1 podatkov, pravimo, da je vozlišče polno.
  - Vsako notranje vozlišče, razen korena ima, ima vsaj $d/2 - 1$ podatkov.
  - Koren je vozlišče, ki vsebuje m podatkov, pri čemer je $1 \leq m \leq d - 1$.
  - Notranje vozlišče, ki ima k sinov, vsebuje k-1 podatkov.

Naj bo dano neko vozlišče B^+-drevesa. Označimo ga z v in pripadajoče ključe tega vozlišča s k_i. Podobno kot v B-drevesu tudi v B^+-drevesu velja:

$$k_1 < k_2 < k_3 < \ldots < k_i.$$

Razlika med B-drevesom in B^+-drevesom je torej v sami zgradbi notranjih vozlišč. Ta pri B^+-drevesih vsebujejo le ključe. S tem usmerjajo postopek iskanja in posledično tudi postopek vstavljanja in brisanja podatkov. Celotni zapisi podatkov pa so shranjeni v listih B^+-drevesa. Posebna lastnost B^+-dreves je tudi ta, da liste s pomočjo dodatnih kazalcev povežemo v urejen seznam vozlišč.

Poglejmo si še en primer B^+-drevesa:



Slika 115: B^+-drevo reda 3 in višine 4

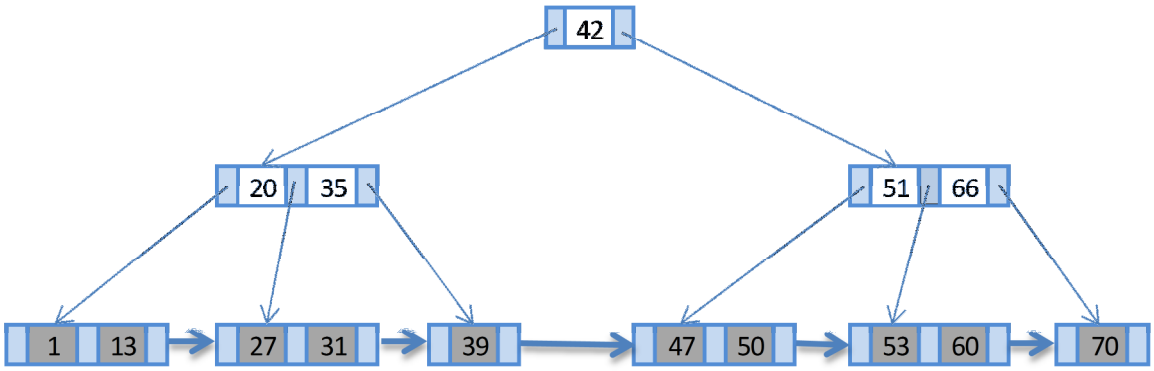
Na sliki Slika 115 je prikazano B^+-drevo reda 3 in višine 4. Vidimo, da se ključi 3, 8, 11 in 15 v drevesu pojavijo dvakrat (v notranjih vozliščih kot ključi, v listih pa kot del celotnega podatka s tem ključem). Hkrati vidimo, da so listi B^+-drevesa med seboj povezani s kazalci. Tako dobimo seznam, kjer so elementi urejeni po velikosti.

5.2. Osnovne operacije nad B^+-drevesi

5.2.1. Iskanje v B^+-drevesu

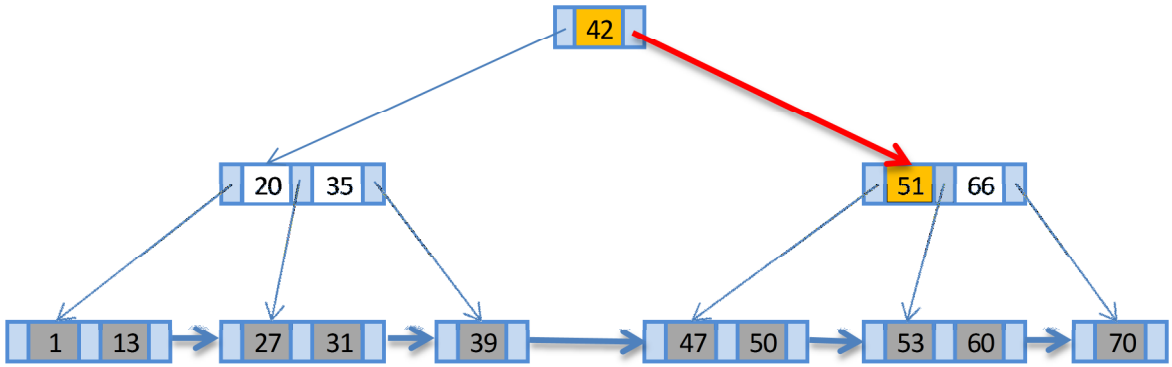
Postopek iskanja podatka (ključa) v B^+-drevesu zahteva, da prehodimo pot od korena do ustreznega lista. Dolžine poti do vseh listov so zaradi uravnoteženosti B^+-drevesa enake. Višina, od katere je odvisna dolžina poti od korena do kateregakoli lista v B^+-drevesu, je običajno majhna, saj je B^+-drevo navadno zelo razvejana drevesna struktura.

Pri postopku iskanja v B^+-drevesu moramo najprej poiskati ustrezni list, v katerem je podatkovni zapis z iskanim ključem. Postopek si podrobneje poglejmo na naslednjem primeru. Denimo, da imamo podano B^+-drevo s slike Slika 116. V njem želimo poiskati podatek s ključem 50.



Slika 116: B^+-drevo, v katerem iščemo podatek

Na začetku želimo poiskati list B^+-drevesa, kjer je podatek z iskanim ključem. Začnemo v korenu. Ker je ključ v korenu (42) manjši od iskanega, nadaljujemo v vozlišču, na katerega kaže desni kazalec.

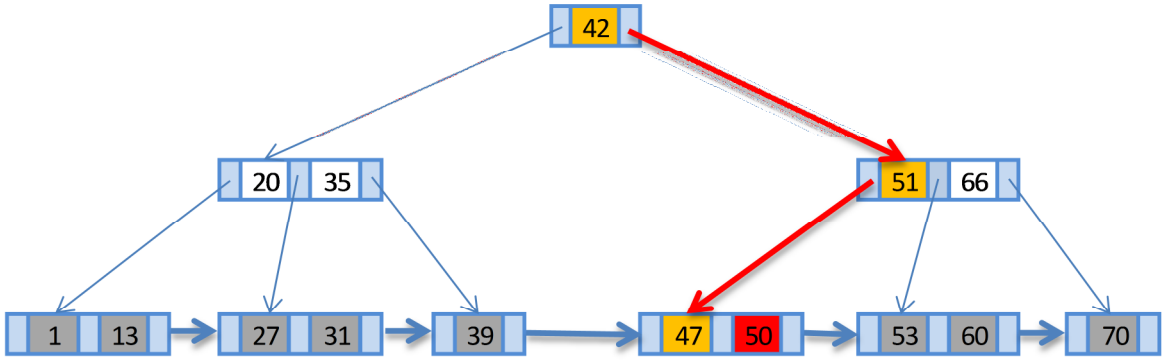


Slika 117: Prvi korak pri iskanju ključa 50

V tem vozlišču je prvi ključ 51. Ker je ta večji od 50, se »spustimo« do končnega vozlišča, na katerega kaže levi kazalec ključa 51.

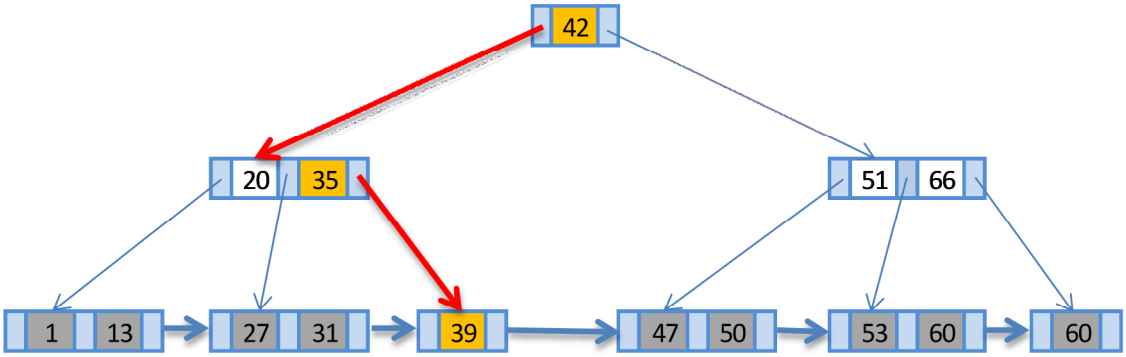
Pridemo do lista, kjer zaporedno primerjamo podatke (njihove ključe) z iskanim. Prvi ključ v ustreznem listu je manjši od iskanega (47<50), drugi (50) pa je enak iskanemu. Tako smo našli iskani podatek in z iskanjem zaključimo.

Na sliki Slika 118 sta podatka (in pripadajoča ključa) 47 in 50 označena z oranžno in rdečo barvo le zaradi nazornejšega prikaza poti iskanja.



Slika 118: Pot iskanja

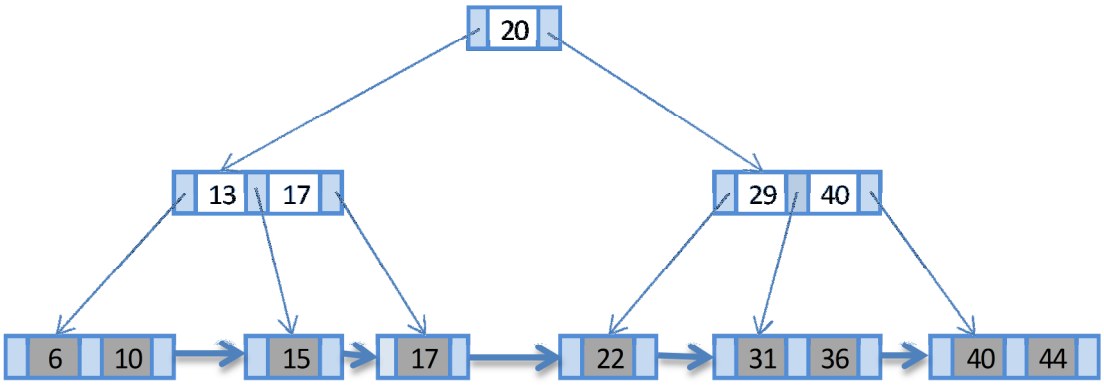
V primeru, ko v B^+-drevesu s slike iščemo ključ 36, iz korena iskano pot nadaljujemo v poddrevesu, na katerega kaže levi kazalec ključa 42 (36<42). Ključa v vozlišču (20 in 35) primerjamo z iskanim. Ker sta oba manjša, pot nadaljujemo v listu, na katerega kaže desni kazalec ključa 35. Pridemo do vozlišča, ki vsebuje podatek s ključem 39. Iskanega ključa ni v drevesu, zato z iskanjem zaključimo.



Slika 119: Iskanje podatka s ključem 36

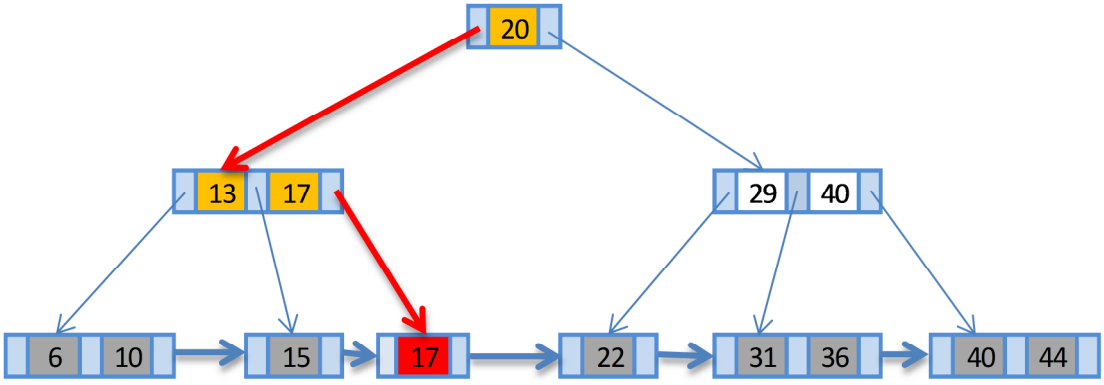
Postopek iskanja v B^+-drevesu reda d in višine h, ki smo ga opisali, zahteva $O(\log_d n)$ operacij, kjer je n število elementov v B^+-drevesu.

Sedaj si poglejmo še primer iskanja v B^+-drevesu, ko iščemo podatek, katerega ključ se ponovi v notranjem vozlišču drevesa. V B^+-drevesu na sliki Slika 120 iščemo podatek s ključem 17.



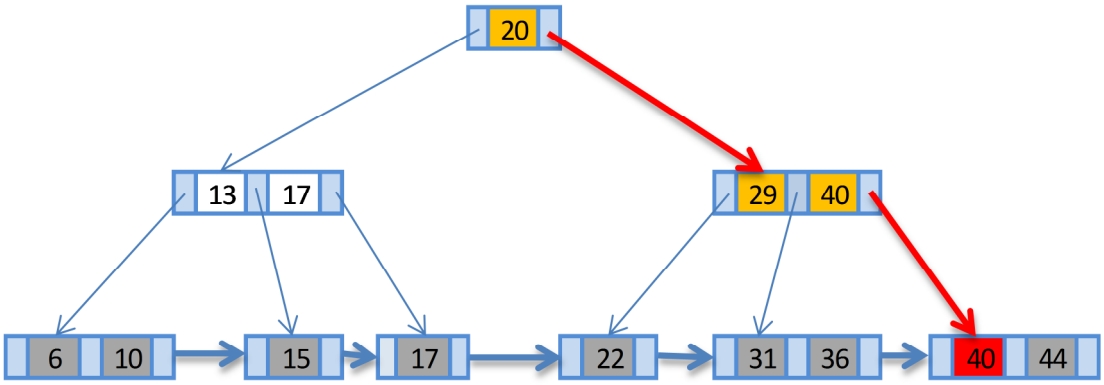
Slika 120: Iskanje v B$^+$-drevesu s podvojenimi ključi

Z iskanjem podatka s ključem 17 začnemo v korenu. Ker je tu ključ večji od iskanega se spustimo do levega sina. Prvi ključ v tem vozlišču (13) je manjši od iskanega, drugi pa je enak. Sedaj vemo, da bo podatek s tem ključem prvi podatek v desnem sinu tega vozlišča ali pa ga v celotnem B$^+$-drevesu ni. Po desnem kazalcu notranjega vozlišča se spustimo do desnega sina (lista), kjer je en sam podatek. Preverimo, ali je njegov ključ enak iskanemu. V tem primeru je. Z iskanjem zaključimo. Pot iskanja je prikazana na sliki Slika 121.



Slika 121: Pot iskanja podatka v B+-drevesu s podvojenimi ključi_primer 1

Postopek iskanje je podoben, kadar v drevesu na sliki Slika 120 iščemo podatek s ključem 40 (Slika 122). Takrat se od korena spustimo do njegovega desnega sina, kjer sta ključa 29 in 40. Drugi ključ v »trenutnem« vozlišču je torej enak iskanemu. Podatek s tem ključem je v danem B$^+$-drevesu na prvem mestu desnega sina »trenutnega« vozlišča. V nasprotnem primeru ga ni v celotnem B$^+$-drevesu.



Slika 122: Pot iskanja podatka v B+-drevesu s podvojenimi ključi_primer 2

Eden od načinov iskanja podatkov z določenim ključem v B^+-drevesu bi bil lahko tudi naslednji: od korena bi se po skrajno levih kazalcih spustili do skrajno levega lista B^+-drevesa in prišli do enojno povezanega urejenega verižnega seznama. Postopek iskanja v takem seznamu je naslednji: s pomočjo kazalcev se pomikamo po vozliščih (listih) drevesa in v vsakem listu preverimo, ali je dani ključ enak iskanemu. Časovna zahtevnost tega algoritma je O(*n*), kjer je *n* število podatkov v seznamu.

Glede na časovno zahtevnost prvo opisanega postopka za iskanje ključa v B^+-drevesu je jasno, da uporabljamo tega in ne ravnokar opisanega.

Zapišimo ga še v obliki algoritma.

Algoritem *poišči_podatek* nam vrne podatek za ključ *x*. Če takšnega ključa v B+ drevesu ni, sproži izjemo *»Danega ključa ni v drevesu.«*. Pri iskanju podatka si pomagamo s pomožnim algoritmom *poišči_indeks*.

```
algoritem poišči_podatek(d, k)
      i = 1;
      če d je prazno
            vrži izjemo »Drevo je prazno.«
      //če je vozlišče list, kličemo funkcijo, ki preveri, če obstaja
      //ključ k v listu in nam vrne podatke za ta ključ
      če d je list
            indeks = poišči_indeks(d, k);
            če indeks = -1
                  vrži izjemo »Danega ključa ni v drevesu.«;
            vrni podatek[indeks];
      //če vozlišče ni list, poiščemo v vozlišču ključ, ki ustreza
      //iskanemu ključu in se postavimo na vozlišče, na katerega
      //kaže za ključ
      če k < ključ_i  //ključ_i = ključ na i-tem mestu
            vrni  poišči_podatek(sin_i,k);
            //sin_i = vozlišče, an katerega kaže ključ na i-tem mestu
      dokler i ≤ št. indeksov v vozlišču d in
             k > ključ v vozlišču d na i-tem mestu
i = i + 1;
      vrni  poišči_podatek (sin_i,k);
vrži izjemo »Danega ključa ni v drevesu«;
```

Pomožni algoritem *poišči_indeks* v vozlišču *x* preveri, če vozlišče vsebuje podani ključ *k*. Če ga, algoritem vrne indeks *i*, sicer vrne -1.

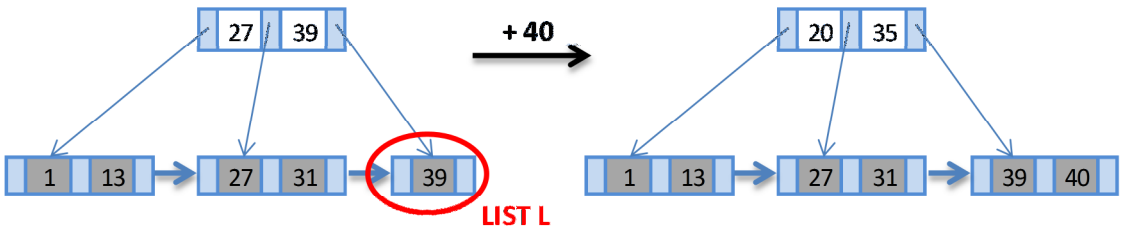
```
algoritem poišči_indeks(x,k);
začni
      i = 1;
      dokler  i <= št. indeksov v x
      če k == ključ_i
      // ključ_i je ključ v vozlišču x pri indeksu i
          vrni i;
      sicer
          i=i+1;
končaj
vrni -1;
```

5.2.2. Vstavljanje v B$^+$-drevo

Operacija vstavljanja nad B$^+$-drevesom ohranja drevo uravnoteženo.

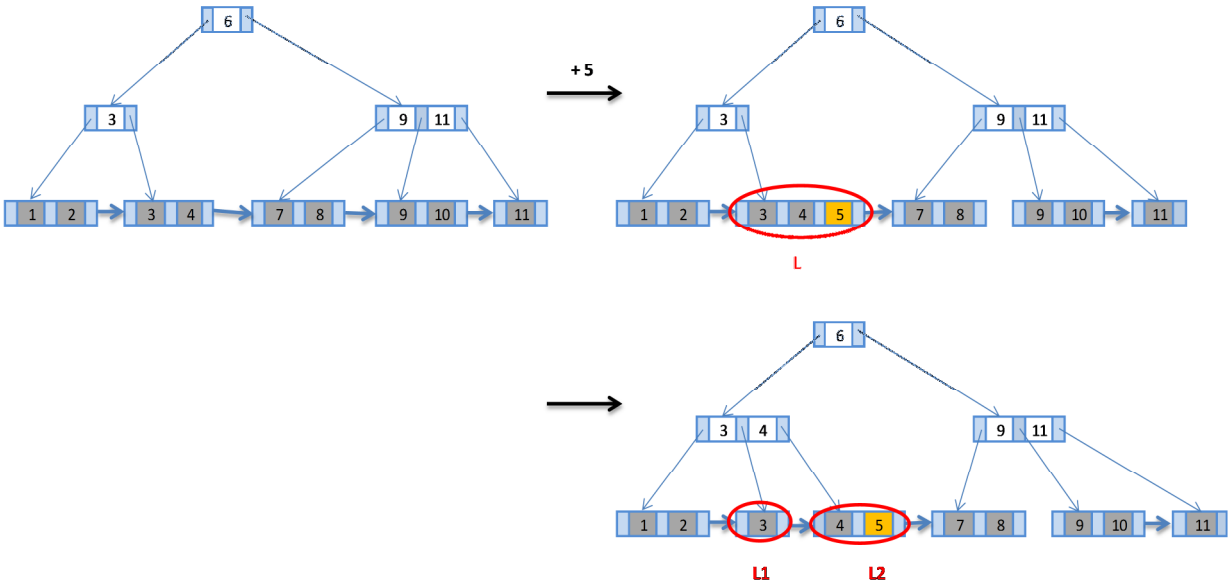
Tako kot v B-drevesu tudi tu podatke vstavljamo v liste drevesa. Najprej poiščemo ustrezen list, kamor bomo podatek vstavili. Naj bo to list L. Nadaljevanje postopka je odvisno od števila podatkov v ustreznem listu L.

Če je v L, kamor bomo vstavljali, manj kot *d* -1 podatkov (v njem še ni dosežena zgornja meja glede števila podatkov), podatek vstavimo med tista dva podatka, kjer je ključ prvega manjši in ključ drugega podatka večji od ključa vstavljenega podatka. Kadar je ključ vstavljenega podatka manjši od vseh ostalih ključev v L, vstavljeni podatek zasede prvo mesto v L. Če pa je ključ vstavljenega podatka večji od pripadajočih ključev v L, podatek vstavimo na zadnje mesto v L. Tako še vedno velja, da je ključ v vsakem vozlišču B$^+$-drevesa večji od vseh ključev v vozlišču, na katerega kaže levi kazalec in manjši od vseh ključev v vozlišču, na katerega kaže desni kazalec trenutnega vozlišča. S postopkom vstavljanja lahko (brez dodatnega preurejanja notranjih vozlišč) na tem mestu zaključimo (Slika 123).



Slika 123: Primer enostavnega vstavljanja v B$^+$-drevo

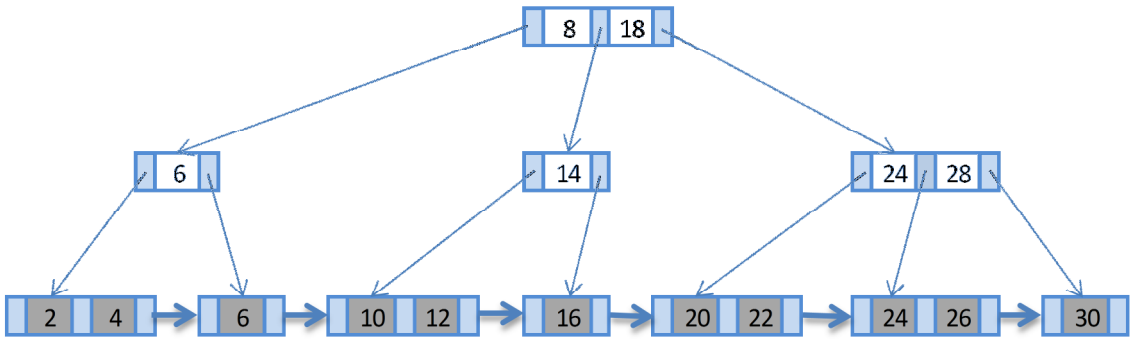
V nasprotnem primeru (ko je v listu L že dosežena zgornja meja glede števila podatkov in je list poln) moramo list po vstavljanju razdeliti. List razdelimo na dve vozlišči, ki ju tukaj označimo kot L1 in L2. Podatki se enakomerno porazdelijo med L1 in L2. Kadar je v listu L liho število podatkov, je po delitvi v L1 en podatek manj kot v L2. Če je v listu 7 podatkov, te razporedimo tako, da v levo vozlišče prenesemo 3 podatke, v desno pa 4. L2 nastopa kot desni brat lista L1 in sta med seboj povezana. Tako L1 kot L2 sta v celotno B$^+$-drevo povezana z očetom lista L. Najmanjši ključ v listu L2 kopiramo in vstavimo na zadnje mesto v očetu (Slika 124).



Slika 124: Delitev lista po vstavljanju podatka

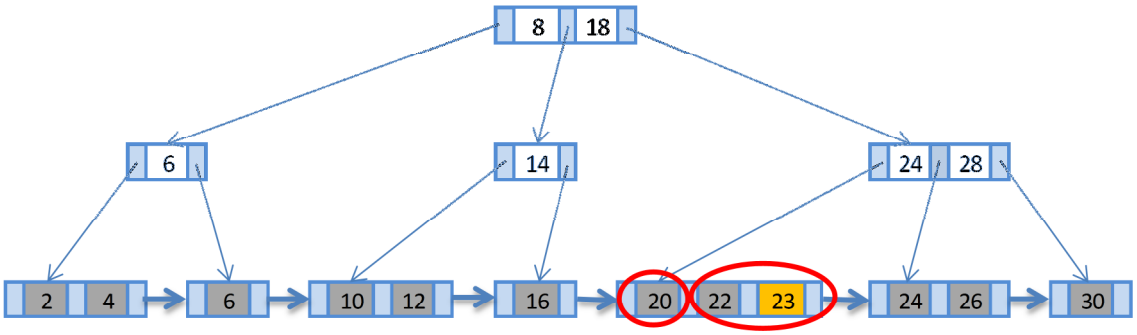
Lahko se zgodi, da je oče obeh listov pred vstavljanjem podvojenega ključa že poln (ko vsebuje $d - 1$ ključev in d kazalcev) . Takrat ga je treba razdeliti. Vendar pa je delitev notranjega vozlišča v B^+-drevesu nekoliko drugačna kot delitev vozlišča, ki je list. Že obstoječe ključe v očetu, ki usmerjajo postopek iskanja, in nov vstavljeni ključ zopet enakomerno porazdelimo med obe vozlišči. Srednji ključ pa prestavimo nivo višje.

Sedaj pa si poglejmo še primer, ko se delitev vozlišča širi vse do korena B^+-drevesa. V B^+-drevo reda 3 na sliki Slika 125 želimo vstaviti podatek s ključem 23.



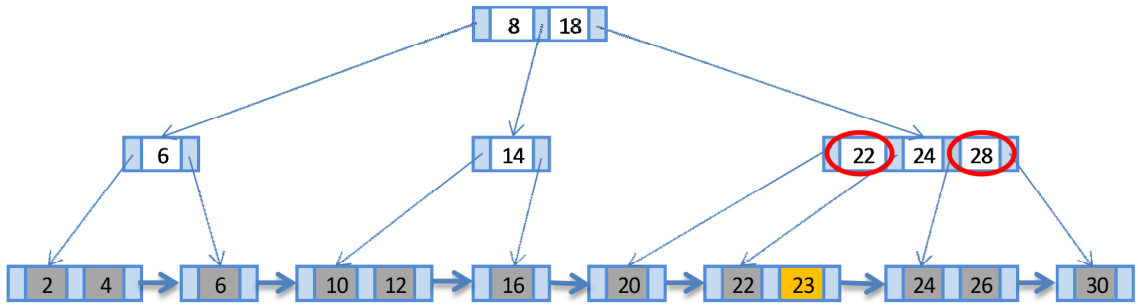
Slika 125: B^+-drevo reda in višine 3

Najprej poiščemo ustrezno mesto (list), kamor bomo podatek vstavili. To je list, ki ima podatka s ključema 20 in 22. Tu zasede zadnje mesto. Z vstavljenim podatkom pa se lastnost B^+-drevesa reda 3 poruši. V listu je sedaj en podatek preveč, kar zahteva delitev lista (Slika 126).



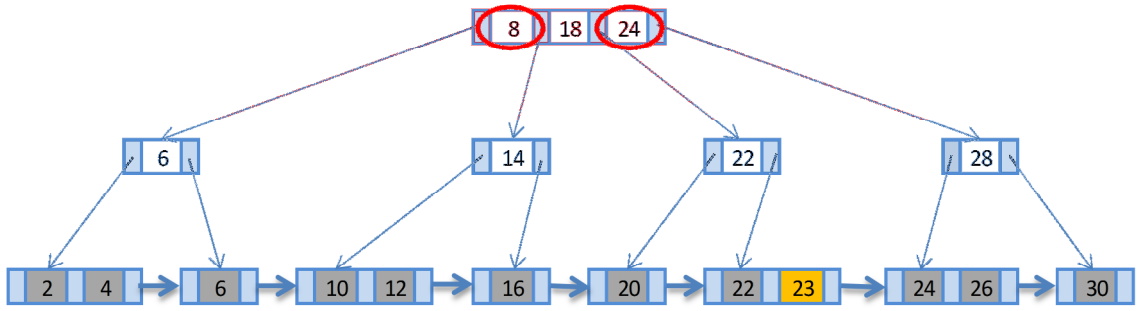
Slika 126: Porušena lastnost B⁺-drevesa

List razdelimo tako, da je sedaj v levem listu podatek s ključem 20, v desnem pa podatka s ključema 22 in 23. Srednji ključ (22) kopiramo v očeta. S tem je zopet presežena zgornja meja glede števila podatkov v vozlišču B⁺-drevesa reda 3 (Slika 127).



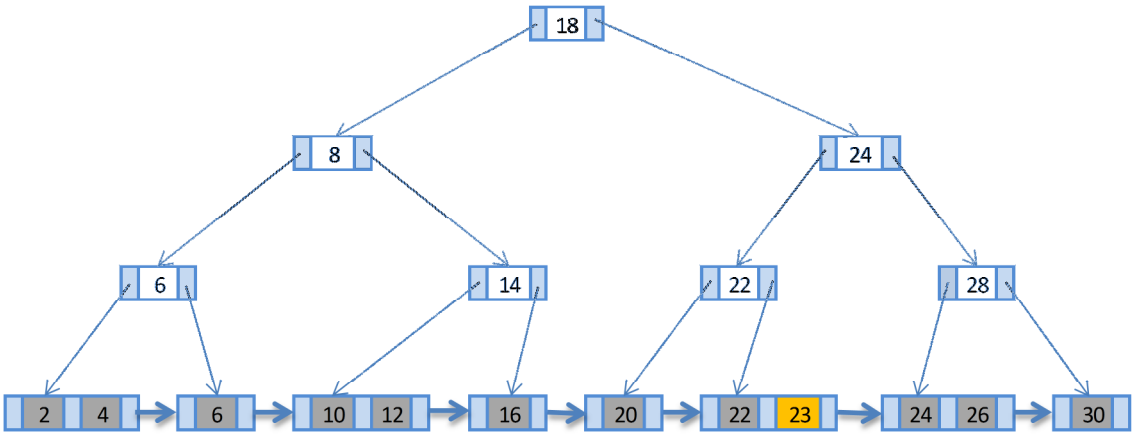
Slika 127: Delitev lista in kopiranje ključa v očeta

Delitev notranjega vozlišča zahteva, da srednji ključ (24) premaknemo v očeta (to je koren drevesa), ključa 24 in 28 pa se porazdelita med vozliščema. Delitev vozlišča se v tem primeru širi vse do korena. Ta ima sedaj 3 ključe, od katerih srednjega premaknemo nivo višje (Slika 128).



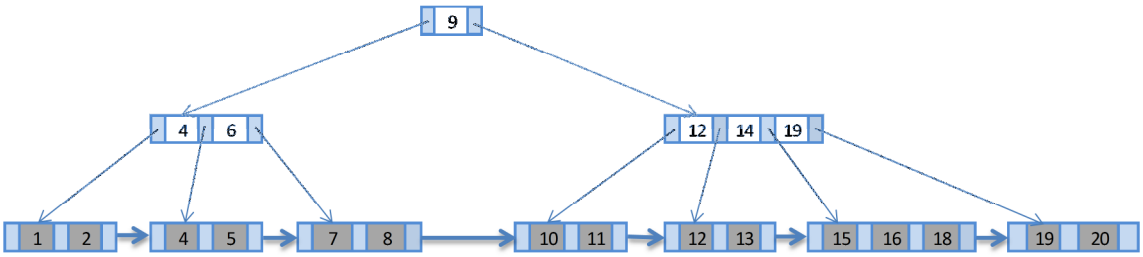
Slika 128: Delitev notranjega vozlišča v B⁺-drevesu

Tako B⁺-drevo dobi nov koren s ključem 18. Višina drevesa se iz 3 poveča na 4. Postopek vstavljanja je prikazan na spodnji sliki (Slika 129).



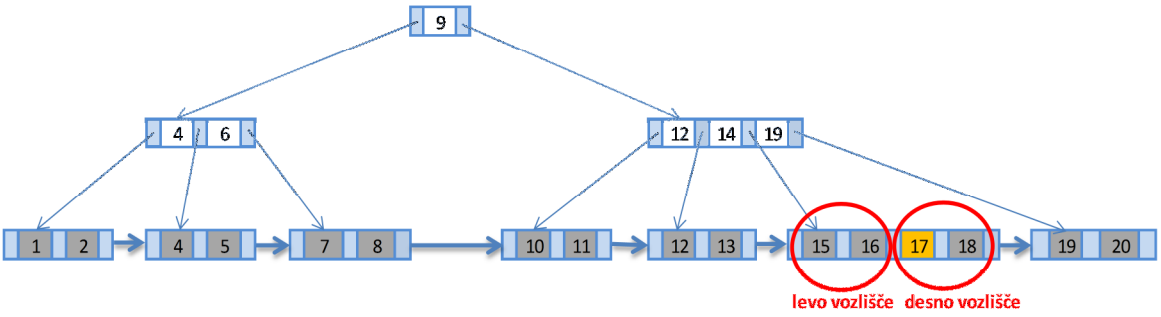
Slika 129: Širjenje delitve vozlišča vse do korena

Sedaj v B$^+$-drevo na sliki Slika 130 vstavimo podatek s ključem 17.



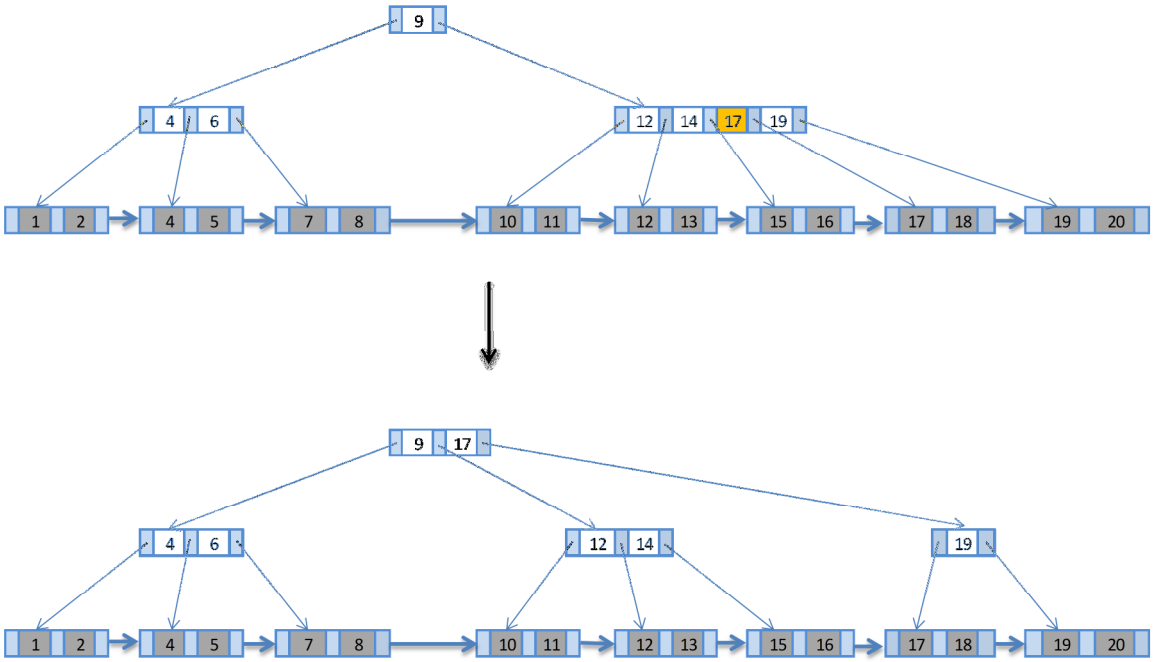
Slika 130: B$^+$-drevo, kamor vstavljamo podatek s ključem 17

Najprej poiščemo ustrezen list, kamor bomo podatek vstavili. Podatek s ključem 17 vstavimo v list, podatkov, ki jih lahko vsebuje vozlišče B$^+$-drevesa reda 4. Po vstavljanju so v listu štirje podatki (Slika 131), zato ga moramo razdeliti na dve vozlišči. V levem vozlišču sta sedaj podatka s ključema 15 in 16, v desnem pa podatka s ključema 17 in 18. Naslednji korak je kopiranje srednjega ključa v očeta prvotnega vozlišča (lista).



Slika 131: Presežena zgornja meja glede minimalnega števila podatkov v vozlišču B$^+$-drevesa

Za srednji ključ vzamemo ključ 17, torej najmanjšega v »desnem vozlišču« (Slika 131). Ko ga kopiramo v očeta, se lastnost B$^+$-drevesa reda 4 znova poruši. V notranjem vozlišču so sedaj štirje ključi, od katerih srednjega premaknemo nivo višje, v koren drevesa. Za srednji ključ zopet vzamemo najmanjšega v »desnem vozlišču«, torej ključ 17 (Slika 132).

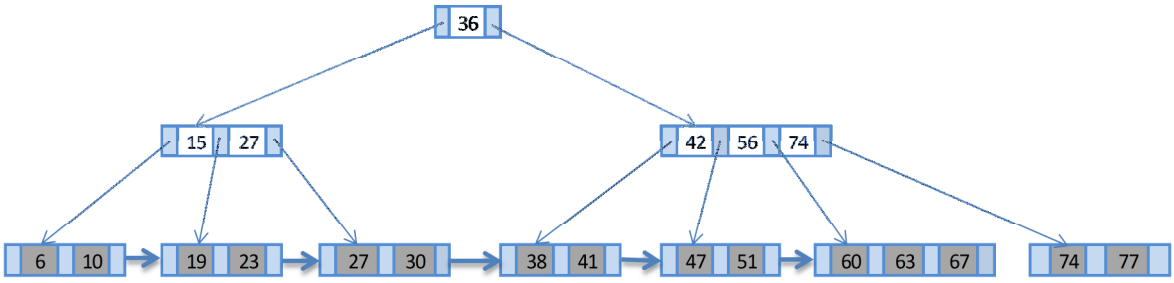


Slika 132: Preureditev B$^+$-drevesa, ko za srednji ključ vzamemo najmanjšega iz desnega vozlišča

Bistvena razlika med delitvijo notranjega vozlišča v B$^+$-drevesu in delitvijo lista je torej ta, da pri delitvi notranjega vozlišča srednji ključ premaknemo v očeta, pri delitvi lista pa srednji ključ kopiramo in vstavimo v očeta. Ta delitev se lahko širi navzgor vse do korena.

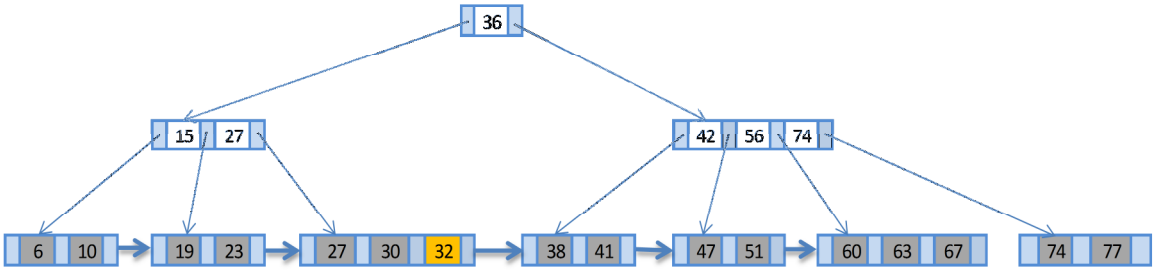
Kadar ključ dodajamo v koren, ki je že poln, to ponovno zahteva delitev. Koren razdelimo na dva dela, srednji ključ pa postane nov koren B$^+$-drevesa. Tako se višina B$^+$-drevesa poveča.

Postopek vstavljanja podatka v B$^+$-drevo si poglejmo na še enem primeru. V B$^+$-drevo reda 4 na sliki Slika 133 bomo zaporedoma vstavili podatke s ključi 32, 34, 35 in 33. Najprej vstavimo podatek s ključem 32.



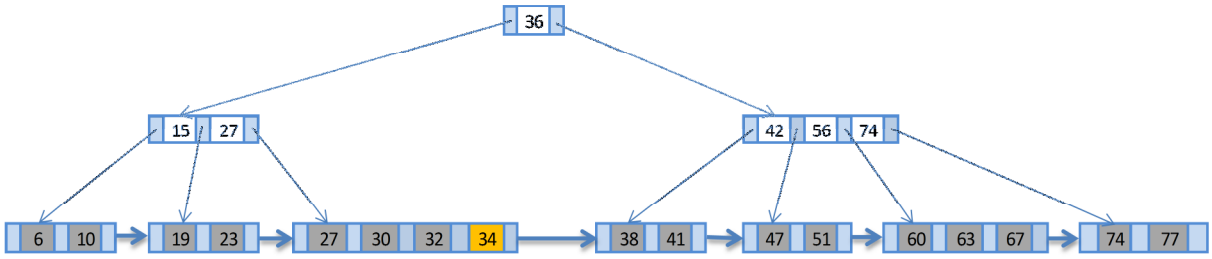
Slika 133: B$^+$-drevo, v katerega želimo vstaviti podatek s ključem 32

Najprej poiščemo ustrezno končno vozlišče B$^+$-drevesa, kamor bomo podatek vstavili. Iz korena se spustimo do njegovega levega sina, kjer sta oba ključa manjša od ključa podatka, ki ga vstavljamo. Zato se spustimo do lista, na katerega kaže desni kazalec ključa 27. Podatek vstavimo v list, kjer sta podatka s ključema 27 in 30. Vstavljeni podatek zasede zadnje mesto v tem listu (Slika 134).



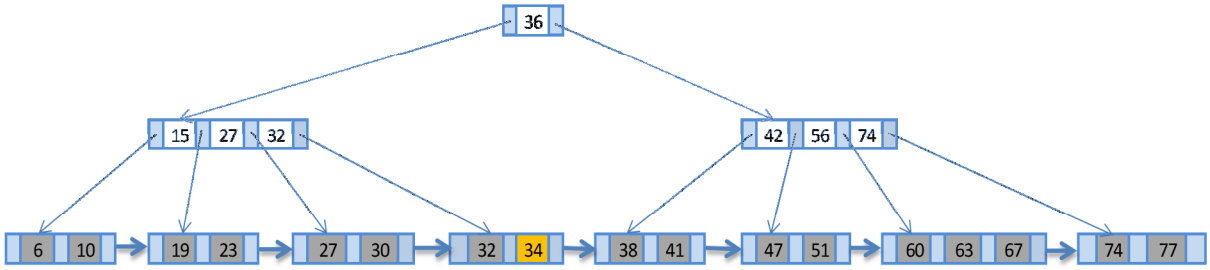
Slika 134: B$^+$-drevo po vstavljenem podatku s ključem 32

Sedaj v B$^+$-drevo na sliki Slika 134 vstavimo podatek s ključem 34. Zopet poiščemo ustrezno mesto (list), kamor bomo ta podatek vstavili. To je vozlišče, ki ima pred vstavljanjem 3 podatke, torej je polno. Ko vstavimo podatek s ključem 34, presežemo zgornjo mejo glede števila podatkov v posameznem vozlišču B$^+$-drevesa reda 4.



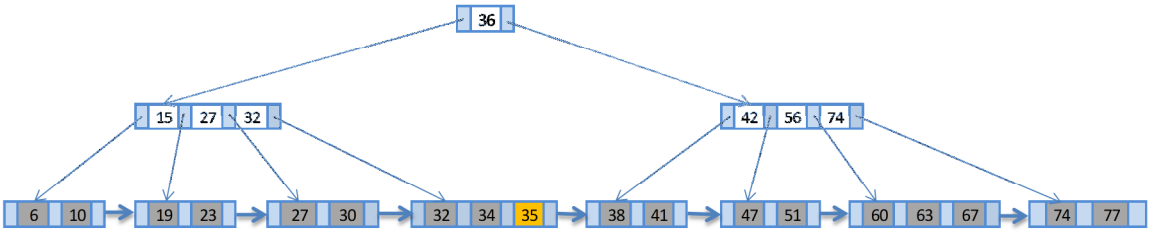
Slika 135: Porušena lastnost B$^+$-drevesa reda 4

V vozlišču, kamor smo podatek vstavili je sedaj en podatek preveč. Vozlišče zato razdelimo na dva dela. V prvem vozlišču bo prva polovica podatkov in pripadajočih ključev, v drugem pa druga. Ključ 32 kopiramo in vstavimo v očeta. Preurejeno drevo, ki zopet ustreza zahtevam B$^+$-drevesa, je prikazano na sliki Slika 136.



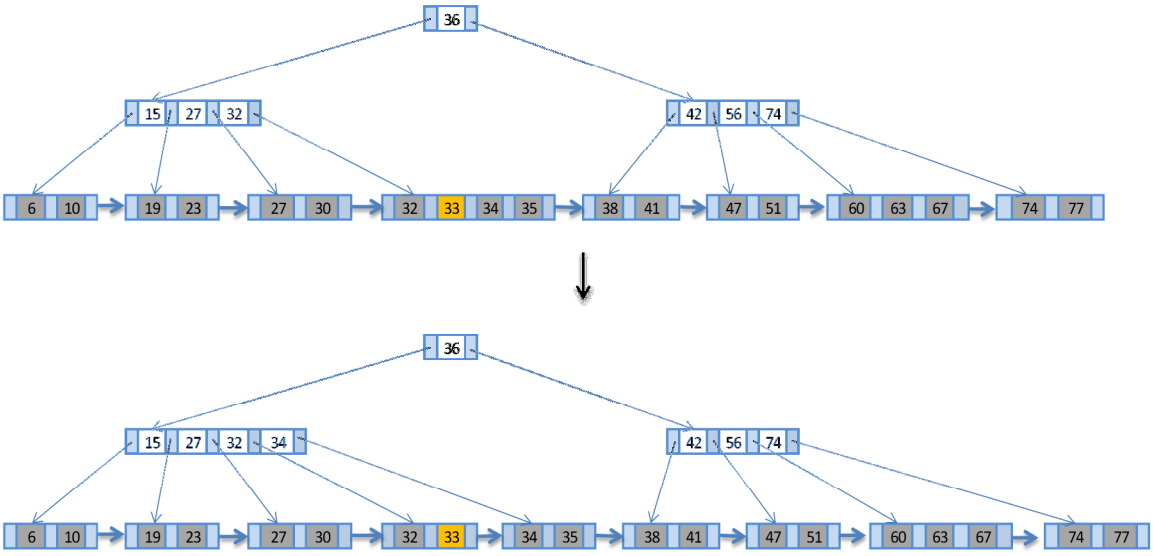
Slika 136: B$^+$-drevo po preureditvi

Naslednji podatek, ki ga vstavimo v B$^+$-drevo na sliki Slika 136, je podatek s ključem 35. Poiščemo ustrezni list (list, kjer sta podatka s ključema 32 in 34) in ga vstavimo na zadnje mesto v tem listu. V listu so sedaj trije podatki in s tem je dosežena zgornja meja glede števila podatkov v vozlišču B$^+$-drevesa reda 4. S postopkom vstavljanja podatka s ključem 35 na tem mestu zaključimo (Slika 137).



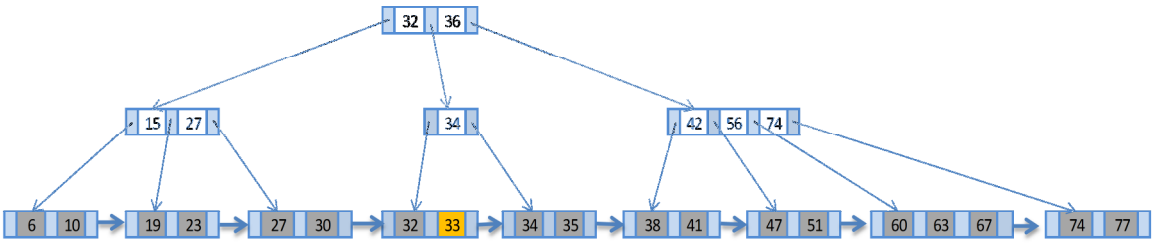
Slika 137: Vstavljanje podatka s ključem 35

Zadnji podatek, ki ga zaporedno vstavljamo v dano drevo je podatek s ključem 33. Ustrezno mesto za vstavljanje tega podatka je list, ki je pred vstavljanjem poln. Zato sledi delitev lista in kopiranje srednje ključa v očeta (Slika 138). Ko srednji ključ (34) kopiramo v očeta, se lastnost B^+-drevesa ponovno poruši.



Slika 138: Ponovna delitev lista B^+-drevesa

V notranjem vozlišču imamo sedaj 4 ključe, kar ne ustreza zahteva B^+-drevesa reda 4. To vozlišče razdelimo na dva dela (levo vozlišče ima ključa 15 in 27, desno pa ključ 34), srednji ključ (32) pa premaknemo (in ne kopiramo) v koren drevesa. S postopkom vstavljanja zaključimo in dobimo B^+-drevo na sliki Slika 139.



Slika 139: Končno B^+-drevo

Primer vstavljanja podatkov v B^+-drevo si poglejmo še na primeru, ko je začetno B^+-drevo prazno. Vanj želimo vstaviti podatke s ključi 6, 9, 3, 7, 5, 13, 10 in 8. Predpostavimo, da potrebujemo B^+-drevo reda 3.

Na začetku B^+-drevo sestavlja le prazen list z enim ali več podatkovnih kazalcev in kazalcem, ki kaže na njegovega desnega soseda. Vanj vstavimo podatek s ključem 6 (Slika 140).

Slika 140: B^+-drevo z enim samim podatkom

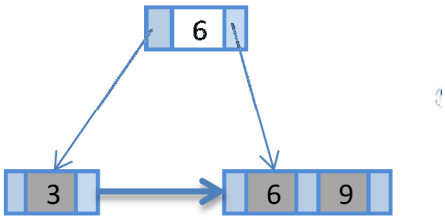
Naslednji podatek, ki ga vstavimo v drevo, je podatek s ključem 9. Vstavimo ga v list, kjer že imamo podatek s ključem 6. Ker je ključ podatka, ki ga vstavljamo večji od 6, zasede drugo mesto v listu.

Slika 141: B^+-drevo, ki ima podatka s ključema 6 in 9

Ko v B^+-drevo s slike Slika 141 vstavimo podatek s ključem 3 se lastnost B^+-drevesa poruši. V vozlišču B^+-drevesa reda 3 sta lahko največ dva podatka. Drevo moramo zato preoblikovati. Vozlišče (list) razdelimo na dve vozlišči, od katerih prvo vsebuje podatek s ključem 3 ter drugo podatka s ključema 6 in 9. Obe vozlišči sta tako vsaj pol polni.

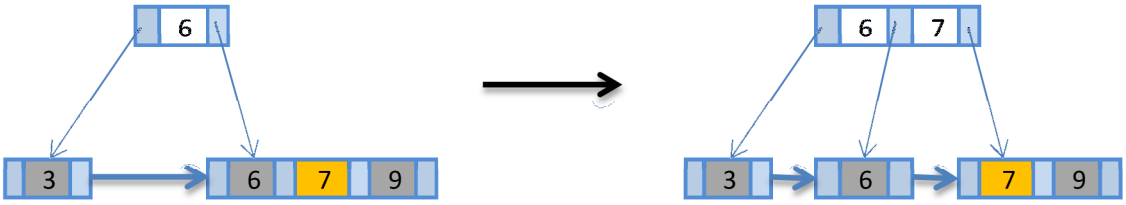
Slika 142: B^+-drevo po treh zaporednih vstavljanjih

Oblikovati pa moramo še korensko vozlišče s kazalcema na obe vozlišči (Slika 143). V koren kopiramo srednji ključ, ključ 6.



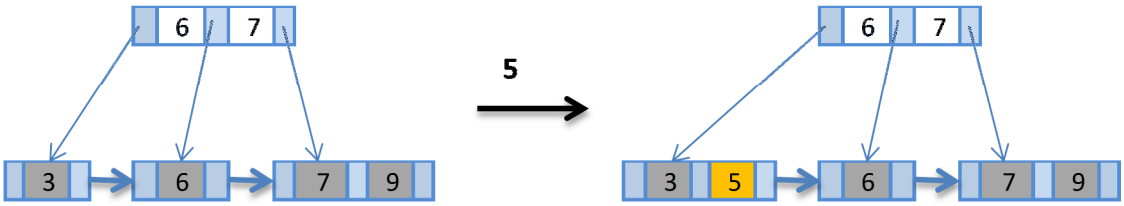
Slika 143: B^+-drevo reda 3 s ključi 3, 6 in 9

Podatek s ključem 7 vstavimo na ustrezno mesto. Ker je ključ 7 večji od ključa v korenu drevesa, ga vstavimo v list, na katerega kaže desni kazalec korena. Ta list je pred vstavljanjem že poln, zato ga moramo po vstavljanju razdeliti. V prvem listu je podatek s ključem 6, v drugem pa podatka s ključema 7 in 9. Srednji ključ (ključ 7) kopiramo v koren drevesa. Dobljeno B^+-drevo je prikazano na sliki Slika 144.



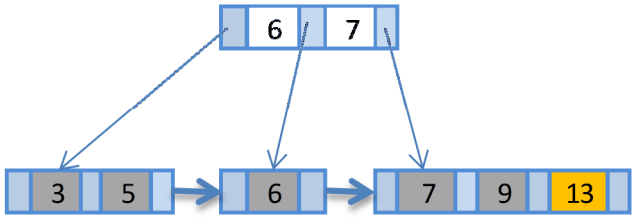
Slika 144: B$^+$- drevo po vstavljanju podatka s ključem 7

Naslednji podatek, ki ga vstavljamo v drevo, je podatek s ključem 5. Vstavimo ga v list drevesa, kjer je podatek s ključem 3. Dobljeno drevo ustreza vsem zahtevam B$^+$-drevesa, zato lahko s postopkom vstavljanja zaključimo.



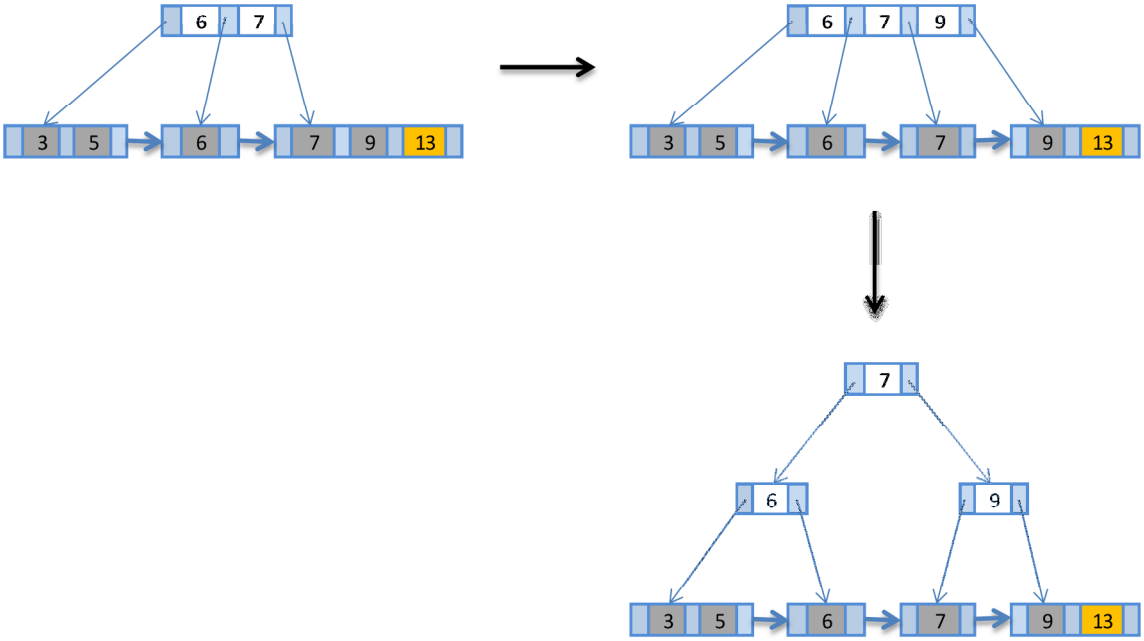
Slika 145: B$^+$-drevo po vstavljanju podatka s ključem 5

Sedaj želimo vstaviti podatek s ključem 13. Najprej poiščemo ustrezno mesto. Od korena se spustimo po desnih kazalcih do skrajno desnega lista, kjer imamo podatka s ključema 7 in 9. Ker je ključ, ki ga vstavljamo večji od vseh ključev, ki so že v B$^+$-drevesu, zasede ta zadnje mesto v tem listu.



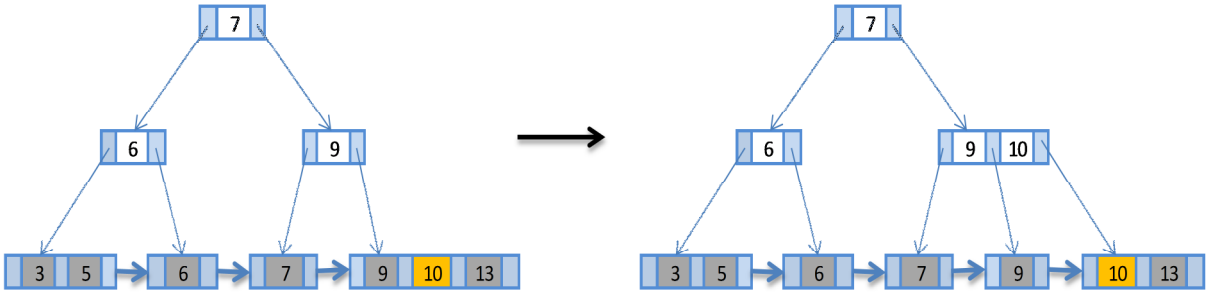
Slika 146: Porušena lastnost B$^+$-drevesa

Z vstavljenim podatkom (Slika 146) se je lastnost B$^+$-drevesa porušila. V listu, kamor smo vstavili podatek s ključem 13, je sedaj en podatek preveč. List ponovno razdelimo na dva dela. V prvem listu bo podatek s ključem 7, v drugem pa podatka s ključema 9 in 13. Ko srednji ključ (ključ 9) kopiramo v koren drevesa, je zgornja meja glede števila podatkov v vozliščih B$^+$-drevesa reda 3 zopet presežena. Koren zato razdelimo na dve vozlišči, kjer eno vozlišče vsebuje ključ 6 in drugo 9. Ključ 7 pa premaknemo nivo višje. B$^+$-drevo tako dobi nov koren in višina drevesa se poveča za 1 (Slika 147).



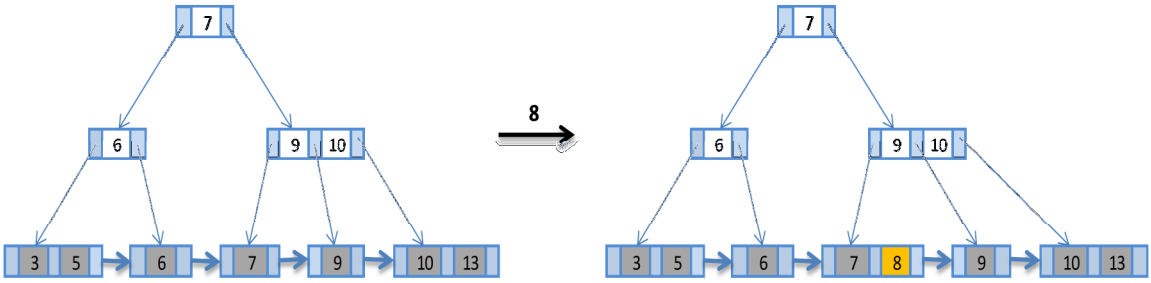
Slika 147: Povečanje višine B$^+$-drevesa

Naslednji podatek, ki ga vstavljamo v B$^+$-drevo, je podatek s ključem 10. To vstavljanje zopet zahteva preurejanje drevesa.



Slika 148: Vstavljanje podatka s ključem 10

Zadnji podatek, ki ga vstavljamo v naše B$^+$-drevo, je podatek s ključem 8. Ustrezno mesto, kamor ga bomo vstavili, je list s podatkom 7. V tem listu sta sedaj 2 podatka. Lastnosti, ki jih mora imeti B$^+$-drevo, so se ohranile in zato lahko z vstavljanjem podatka zaključimo. Tako dobimo končno B$^+$-drevo reda 3.

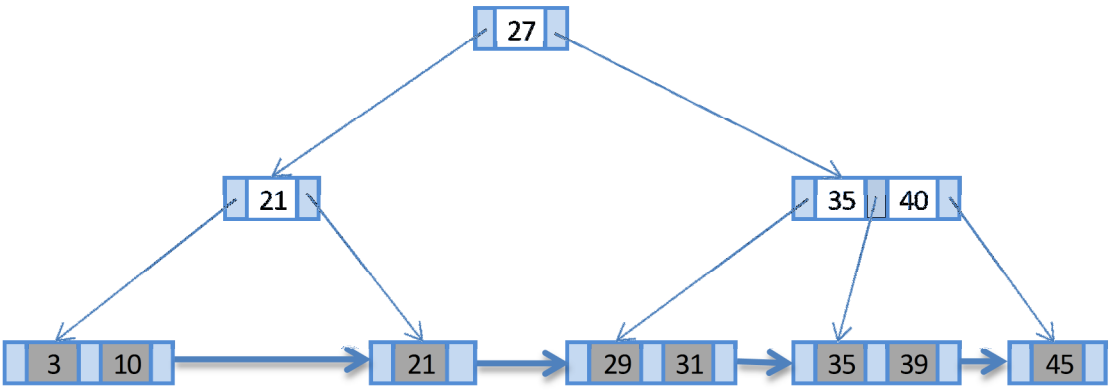


Slika 149: Končno B$^+$-drevo

5.2.3. Brisanje podatka iz B$^+$-drevesa

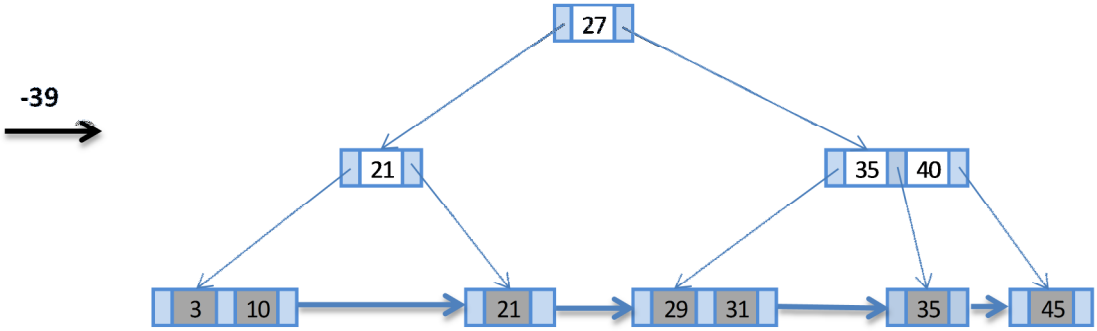
Postopek brisanje podatka iz B$^+$-drevesa se, tako kot v B-drevesu, začne s postopkom iskanja. Seveda moramo tukaj upoštevati, da so podatki samo v listih. Z iskanjem začnemo v korenu in se pomikamo po drevesu navzdol, dokler ne pridemo do ustreznega lista, kjer podatek je. Po izbrisu podatka je treba zagotoviti, da je vozlišče (list) vsaj pol polno. Tako bo B$^+$-drevo še vedno ustrezalo zahtevam. Če je list po izbrisu podatka še vedno pol poln, lahko s postopkom brisanja zaključimo. Sicer moramo drevo preoblikovati. To storimo tako, da preverimo, koliko podatkov vsebuje levi ali/in desni brat lista, kjer smo podatek izbrisali.

Postopek »enostavnega« brisanja podatka iz B$^+$-drevesa si poglejmo na primeru. Iz B$^+$-drevesa reda 3 na sliki Slika 150 želimo izbrisati podatek s ključem 39.



Slika 150: B$^+$-drevo, iz katerega želimo izbrisati podatek s ključem 39

Podatek s ključem 39 je v listu, kjer sta 2 podatka. Po izbrisu je list še vedno pol poln, zato lahko postopek brisanja zaključimo brez nadaljnjega preoblikovanja drevesa. Dobimo B$^+$-drevo na sliki Slika 151, ki ustreza vsem zahtevam B$^+$-drevesa reda 3 (vsako vozlišče ima vsaj 1 ključ in največ 2 ter vsako vozlišče, razen korena, ima najmanj 2 in največ 3 sinove).

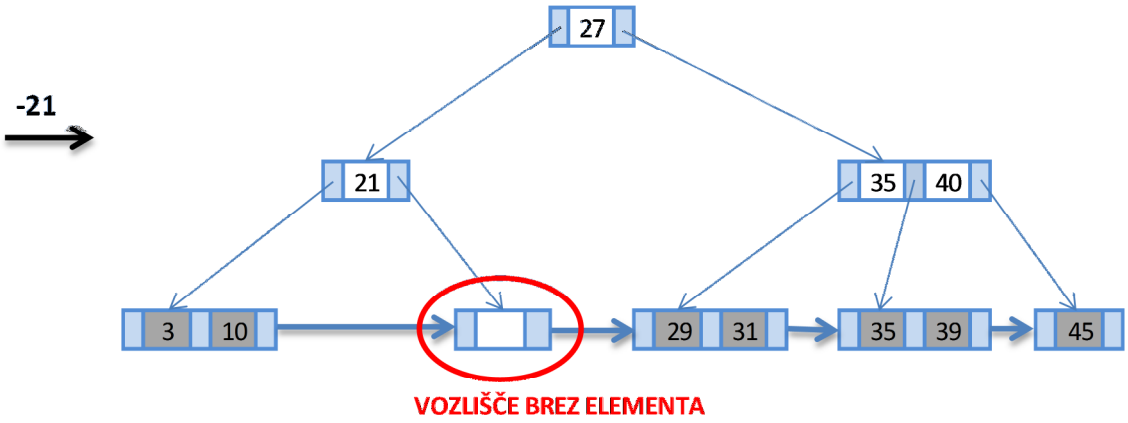


Slika 151: B$^+$-drevo po izbrisu podatka s ključem 39

Če list L pred brisanjem vsebuje minimalno število podatkov, moramo izbrisani podatek nadomestiti z drugim podatkom, ki zasede njegovo mesto, hkrati pa ohraniti pravilni vrstni red. Če želimo poiskati ta podatek, pregledamo vozlišči, ki sta na levi in desni strani vozlišča, iz katerega brišemo podatek. Naj bo njegov levi brat L_{levi} in njegov desni brat L_{desni}. Vemo, da vsaj eden od njiju zagotovo obstaja.

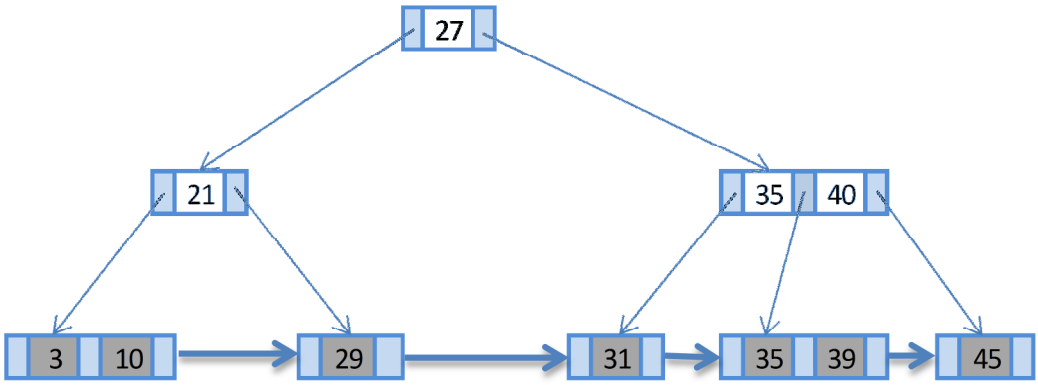
Če ima vsaj eden od bratov več kot minimalno število podatkov, en podatek prestavimo na ustrezno mesto v list s premalo podatki. Kadar ima tako L_{levi} kot L_{desni} več kot minimalno število elementov, bomo podatek prestavili iz desnega brata. Listi B$^+$-drevesa so povezani v urejen seznam, zato je dostop do desnega brata nekoliko enostavnejši kot dostop do levega brata.

Za primer vzemimo B$^+$-drevo s slike Slika 150, iz katerega izbrišemo podatek s ključem 21. Po izbrisu v listu ni nobenega podatka, zato se lastnost B$^+$-drevesa poruši.



Slika 152: Preurejanje B$^+$-drevesa po izbrisu podatka s ključem 21

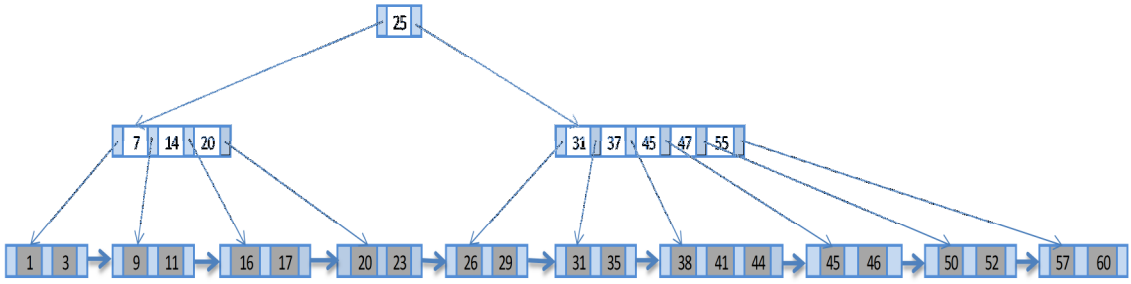
Preverimo, ali je njegov desni brat več kot pol poln. L_{desni} ima 2 podatka, zato lahko enega prestavimo v list, kjer smo brisali. V prazen list prestavimo podatek s ključem 29. Ker je B$^+$-drevo iskalno drevo, morajo biti vrednosti v levem poddrevesu manjše in vrednosti v desnem poddrevesu večje ali enake od vrednosti v očetu. To mora veljati za vsako vozlišče drevesa. Po prestavljanju podatka iz desnega brata je ta lastnost še vedno izpolnjena (21<29). Tako dobimo B$^+$-drevo na sliki Slika 153.



Slika 153: Preurejeno B$^+$-drevo po izbrisu

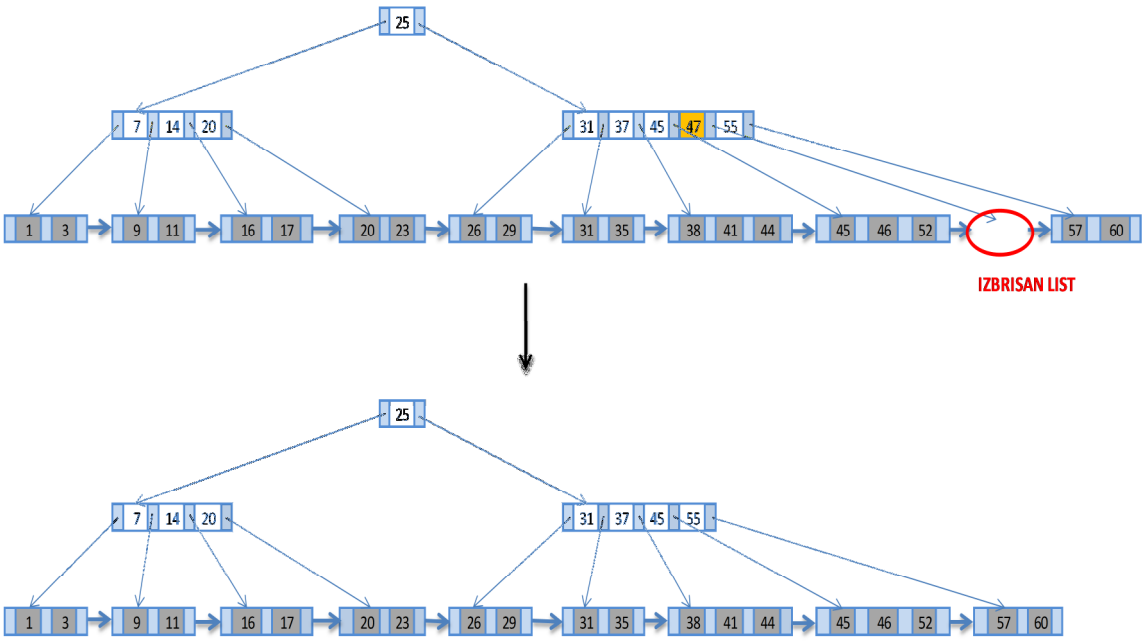
V primeru, ko imata tako L_{levi} kot L_{desni} minimalno število podatkov, list L združimo z enim od bratov. Podatke iz L torej prestavimo v L_{levi} ali L_{desni}, L pa odstranimo iz drevesa. Vozlišče, kamor smo vstavili podatke, ne presega zgornje meje glede števila podatkov. To spajanje vozlišč združi dve poddrevesi, kar zahteva brisanje vmesnega ključa iz očeta. V očetu moramo izbrisati ključ, katerega levi kazalec kaže na L_{levi}, desni kazalec pa na list L oziroma ključ, katerega levi kazalec kaže na L, desni kazalec pa na L_{desni}.

Tako preurejanje drevesa si poglejmo na primeru. Podano imamo B$^+$-drevo reda 6 na sliki Slika 154. Iz njega želimo izbrisati podatek s ključem 50.



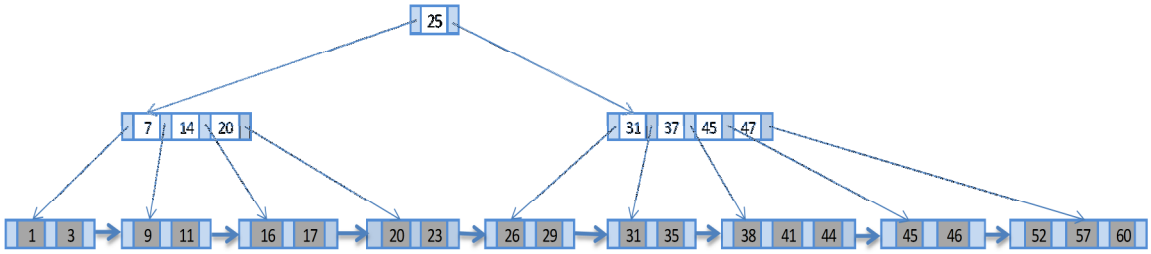
Slika 154: B$^+$-drevo reda 6

Potem, ko podatek s ključem 50 izbrišemo iz lista L, število podatkov v njem pade pod spodnjo mejo (v listu ostane en sam podatek, podatek s ključem 52). Preverimo, ali ima kateri od bratov tega lista več kot minimalno število podatkov. Ker imata tako L_{levi} kot L_{desni} le 2 podatka (spodnja meja), list, v katerem smo izbrisali podatek, združimo z levim bratom $L_{levi.}$ Podatek s ključem 52 prenesemo v list, kjer sta podatka s ključema 45 in 46. Tu zasede zadnje mesto. List L izbrišemo. To zahteva brisanje ključa 47 v očetu. Vsa vozlišča so sedaj vsaj pol polna, zato lahko postopek brisanja podatka zaključimo. Dobimo B$^+$-drevo reda 6 na sliki Slika 155.



Slika 155: Združevanje lista z levim bratom

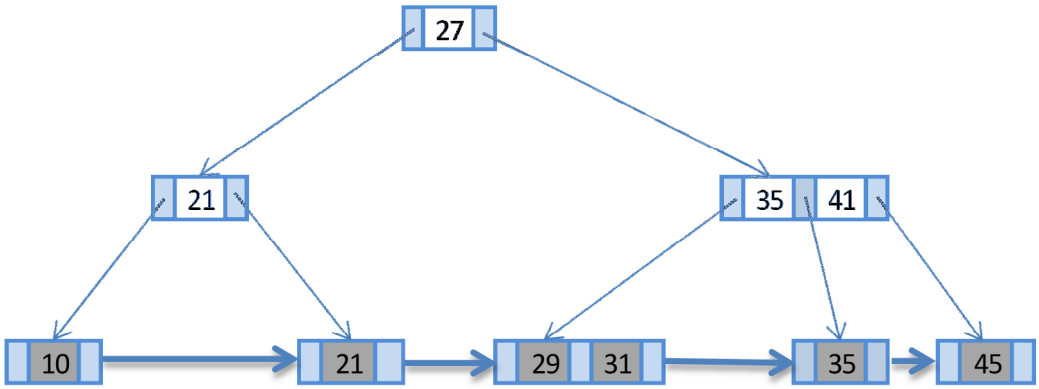
List s premalo podatki bi lahko združili tudi z desnim bratom $L_{desni.}$ V tem primeru podatek s ključem 52 prenesemo v list, kjer sta podatka s ključema 57 in 60. Tu podatek s ključem 52 zasede prvo mesto. List L ponovno izbrišemo. To v očetu zahteva brisanje ključa 55. Vsa vozlišča so sedaj vsaj pol polna, zato lahko postopek brisanja podatka zaključimo. Dobimo B^+-drevo reda 6 na sliki Slika 156.



Slika 156: Združevanje lista z desnim bratom

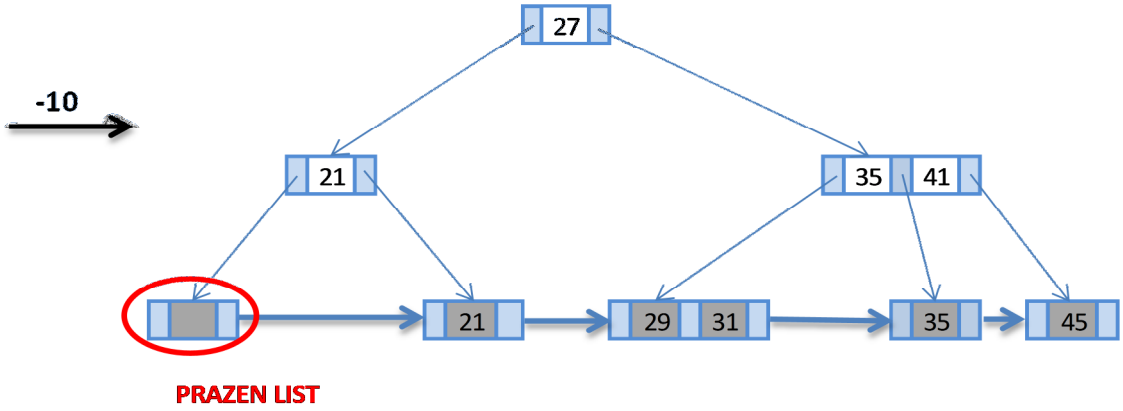
Združevanje vozlišč in brisanje ključa iz očeta se lahko širi vse do korena. To lahko povzroči brisanje korena in s tem zmanjšanje višine B^+-drevesa. Tak primer brisanja podatka si poglejmo na primeru.

Iz B^+-drevesa reda in višine 3 na sliki Slika 157 želimo izbrisati podatek s ključem 10.



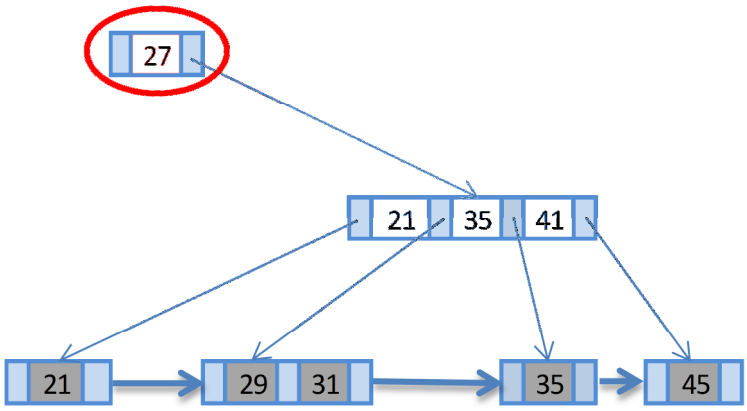
Slika 157: B⁺-drevo, iz katerega izbrišemo podatek s ključem 10

Ker je podatek s ključem 10 edini podatek v levem listu drevesa, je ta po izbrisu prazen. Ker je to skrajno levi list, levega brata nima, v njegovem desnem bratu pa je en sam podatek (podatek s ključem 21). Torej ni podatka, ki bi ga lahko premaknili v prazen list. Prazen list zato izbrišemo iz drevesa (Slika 158).



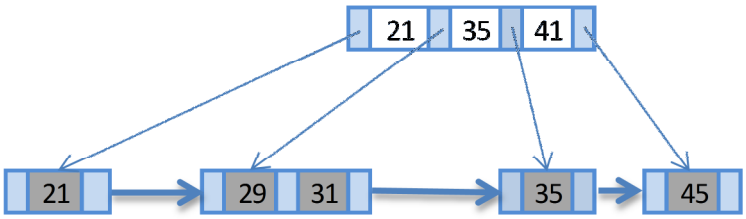
Slika 158: B⁺-drevo s praznim listom

Brisanje praznega lista povzroči združevanje notranjih vozlišč. Ključ 21 prenesemo v vozlišče s ključema 35 in 41. Prazno vozlišče (vozlišče, kjer je bil prej ključ 21) izbrišemo.



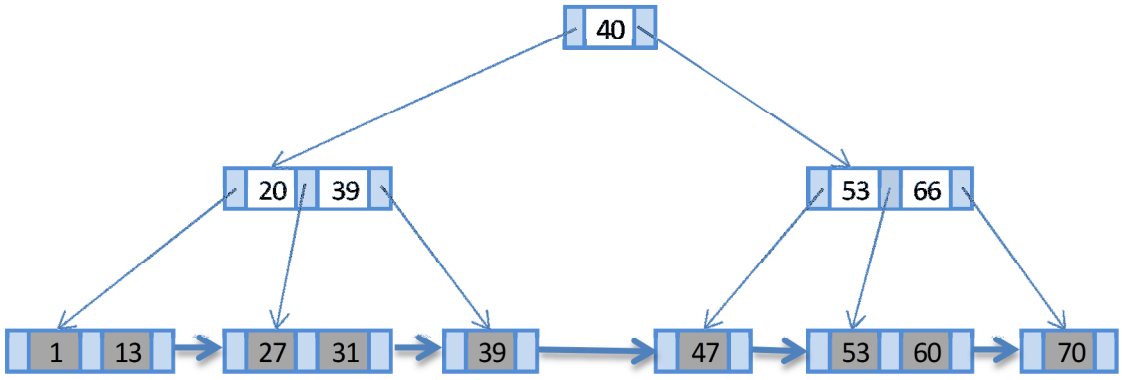
Slika 159: Brisanje korena B⁺-drevesa

Brisanje vozlišča se tako širi vse do korena. Dobimo B^+-drevo reda 3 in višine 2 na sliki Slika 160.



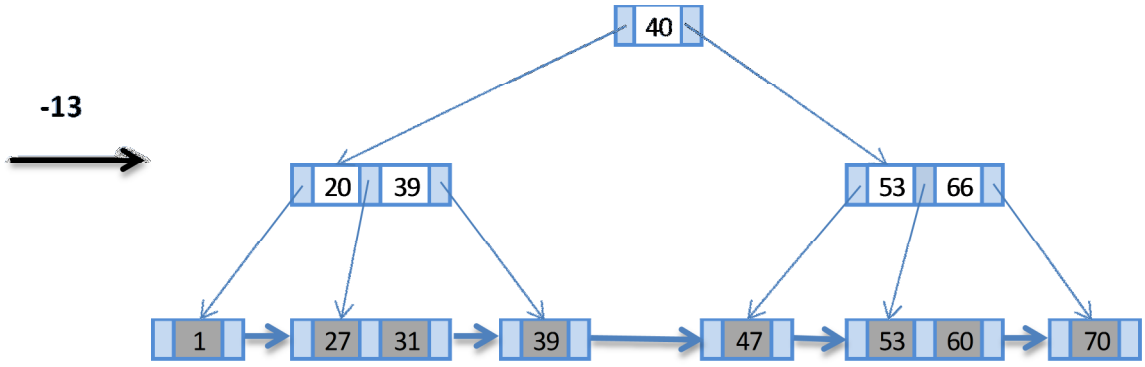
Slika 160: Zmanjšanje višine B^+-drevesa

Poglejmo si še en primer brisanja ključev iz B^+-drevesa. Iz B^+-drevesa na sliki Slika 161 želimo zaporedoma izbrisati podatke s ključi 13, 39, 47 in 70.



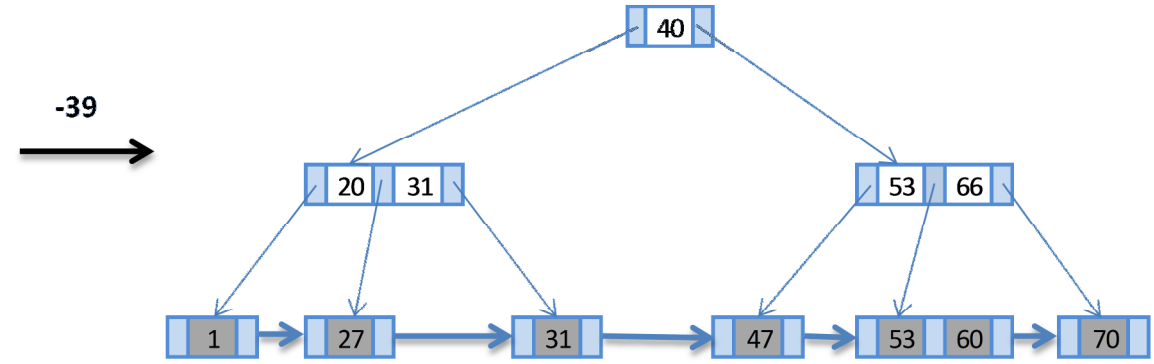
Slika 161: B^+-drevo, iz katerega postopno brišemo podatke

Podatek s ključem 13 izbrišemo iz B^+-drevesa brez dodatnega preurejanja, saj so tudi po brisanju vsa vozlišča v drevesu vsaj pol polna. Dobimo B^+-drevo na sliki Slika 162.



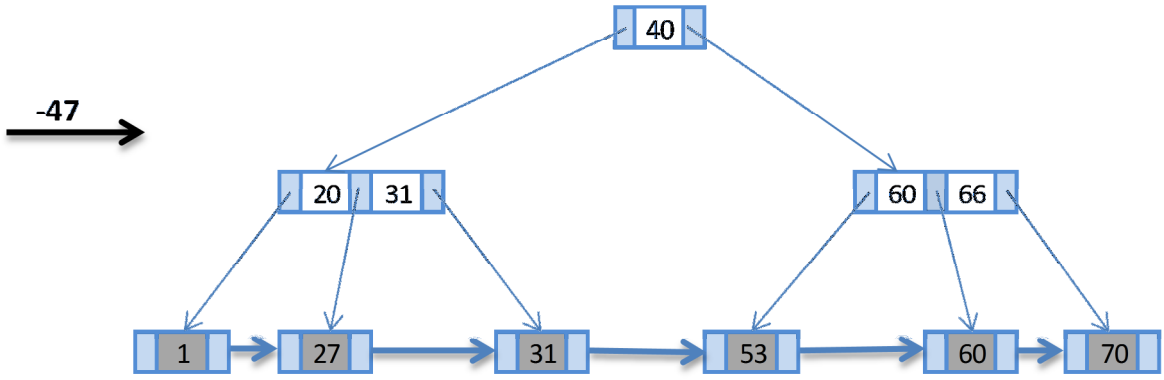
Slika 162: B^+-drevo po izbrisu podatka s ključem 13

Sedaj iz drevesa izbrišemo podatek s ključem 39. Ko podatek izbrišemo, število podatkov v tem listu pade pod spodnjo mejo (vozlišče je prazno). Zato na njegovo mesto vstavimo podatek s ključem 31 iz njegovega levega brata. V očetu listov ključ 39 nadomestimo s ključem 31 in postopek brisanja zaključimo.



Slika 163: B$^+$-drevo po izbrisu podatka s ključem 39

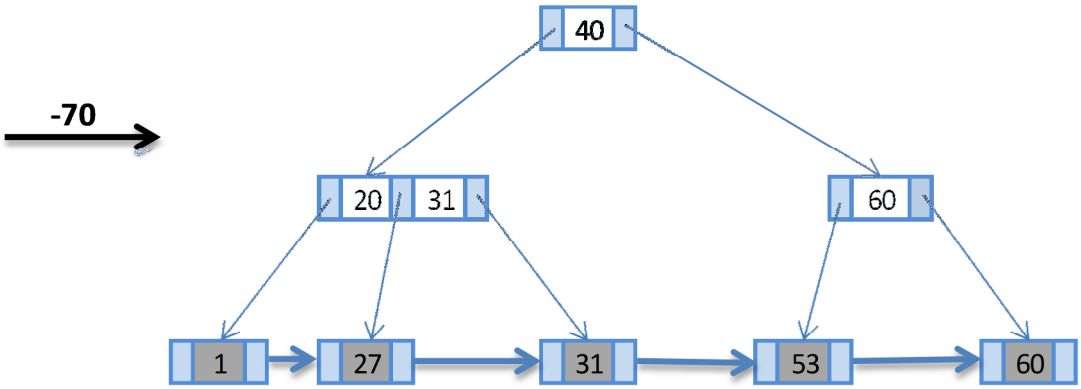
Naslednji podatek, ki ga brišemo, je podatek s ključem 47. Postopek brisanja je podoben prejšnjemu. Tu podatek zamenjamo s prvim podatkom iz desnega brata lista, v katerem smo podatek izbrisali. Sedaj imajo vsi listi v B$^+$-drevesu na sliki Slika 164 minimalno število podatkov.



Slika 164: B$^+$-drevo po izbrisu podatka s ključem 39

Po izbrisu podatka s ključem 70, moramo drevo ponovno preoblikovati. Najprej preverimo, ali je v levem bratu lista, kjer smo brisali, več podatkov kot je spodnja meja. Ker je v njem le en podatek, ni podatka, s katerim bi nadomestili izbrisanega. List zato izbrišemo iz B$^+$-drevesa. Izbrisati pa moramo tudi ključ v očetu tega lista. Dobljeno drevo ustreza vsem zahtevam B$^+$-drevesa, zato s postopkom brisanja končamo.

Končno B$^+$-drevo reda in višine 3 je pokazano na sliki Slika 165.



Slika 165: B$^+$-drevo po izbrisu podatka s ključem 70

6. ZAKLJUČEK

Osrednja tema diplomske naloge so B-drevesa. Na začetku sem predstavila podatkovno strukturo drevo in opisala pomen posameznih pojmov, ki jih potrebujemo za njeno razumevanje. Kot posebno obliko dreves sem obravnavala dvojiška drevesa in nato iskalna dvojiška drevesa. Opisala sem postopke iskanja, vstavljanja in brisanja nad iskalnimi dvojiškimi drevesi in ugotovila, da je le ta odvisna od višine danega drevesa. Izkaže se, da se lahko z vstavljanjem in brisanjem podatkov v iskalnem dvojiškem drevesu, njegova struktura hitro spremeni. V najslabšem primeru dobimo izrojeno ali izrojenemu podobno drevo. Takrat časovna zahtevnost algoritmov ni več logaritemska, temveč linearna.

Da ohranimo učinkovitost iskanja, vstavljanja in brisanja, uporabimo večsmerna drevesa. Večsmerna drevesa imajo v vozliščih večje število podatkov (in ne več samo dva, kot dvojiška drevesa). Vozlišča večsmernih dreves so zgrajena tako, da se pot iskanja lahko nadaljuje v več smereh. Pri tem pa je pomembno, da naše drevo raste nadzorovano. V ta namen vpeljemo pojem uravnoteženega faktorja in z njim povezana uravnotežena drevesa. Uravnoteženo dvojiško drevo je bodisi prazno bodisi se višini njegovega levega in desnega poddrevesa razlikujeta kvečjemu za 1. Kadar združimo ideji o večsmernem in uravnoteženem iskalnem drevesu, pridemo do B-drevesa.

B-drevo reda m je m-smerno iskalno drevo, ki je bodisi prazno, bodisi višine vsaj 2. Vsako vozlišče B-drevesa reda m ima najmanj m/2 sinov in m/2-1 podatkov in največ m sinov in m-1 podatkov. V diplomski nalogi sem podrobneje predstavila postopke iskanja, vstavljanja in brisanja nad B-drevesi. Časovna zahtevnost teh postopkov je O(log n). Posebna oblika B-dreves so B+-drevesa. Tu so celotni podatki shranjeni le v listih drevesa, v notranjih vozliščih B+-drevesa pa so le ključi, ki usmerjajo postopek iskanja. Listi so med seboj povezani v urejen seznam. Posebna lastnost B+-dreves je ta, da se ključi v drevesu lahko podvojijo.

LITERATURA IN VIRI

Comer, D. (1979). *The Ubiquitous B-Tree 11(2):123.* Computing Surveys.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1999). *Introduction to algorithms.* Cambridge: The MIT Press.

Knuth, D. E. (1973). *The art of computer programming. Vol. 3, Sorting and searching.* Boston: Addison-Wesley.

Kononenko, I. R. (2004). *Algoritmi in podatkovne strukture I in II.* Ljubljana: Založba FE in FRI Ljubljana.

Kozak, J. (1997). *Podatkovne strukture in algoritmi.* Ljubljana: Društvo matematikov, fizikov in astronomov Slovenije.

R. Bayer, E. M. (1972). *Organization and Maintenance of Large Ordered Indexes, Vol. 1, Fasc. 3.* Acta Informatica.

Wirth, N. (1984). *Računalniško programiranje. Del 1* . Ljubljana: Društvo matematikov, fizikov in astronomov SRS.

KAZALO PONAZORITEV

KAZALO SLIK

KAZALO TABEL