Strukture podataka

Balansirana uređena binarna drveta

Uređena binarna drveta su pogodna struktura podataka za implementaciju asocijativnih struktura podataka poput uređenih mapa i uređenih skupova[1]. Međutim, treba imati na umu da su ona efikasna samo kada su balansirana. Naime, ukoliko uređeno binarno drvo nije balansirano, operacije pretrage, umetanja i brisanja mogu zahtevati linearno vreme u odnosu na broj čvorova u drvetu. Nizovi umetanja elemenata mogu dovesti do veoma nebalansiranih drveta sa lošom asimptotskom složenošću osnovnih operacija. Na primer, ako vršimo redom umetanje n elemenata čije su vrednosti ključeva strogo rastuće ili strogo opadajuće, ukupna složenost n umetanja biće $O(n^2)$. S druge strane, ako bismo mogli da održavamo visinu drveta tako da bude reda $O(\log n)$, kao što je to slučaj sa savršeno balansiranim drvetom, ukupna složenost umetanja n operacija umetanja bila bi $O(n \log n)$. Šta više, balansirana binarna drveta garantuju da se to ne može dogoditi i da je vremenska složenost najgoreg slučaja operacija pretrage, umetanja i brisanja elementa logaritamska u odnosu na broj čvorova u drvetu.

Da bi se drvo održavalo balasiranim prilikom izvršvanja operacija koje menjaju strukturu drveta kao što su umetanje ili brisanje elementa, potrebno je sprovesti dodatne operacije balansiranja drveta. Pritom, moramo biti veoma pažljivi da prilikom balansiranja drveta ne narušimo uslov uređenosti drveta.

Zbog velikog značaja balansiranih uređenih binarnih drveta, razvijene su različite varijante uređenih binarnih drveta koje koriste različite algoritme balansiranja. One se međusobno razlikuju i po načinu na koji definišu balansiranost, tj. invarijantama koje održavaju, kao i u tome kada i kako vrše balansiranje.

U nastavku ćemo razmotriti dve najčešće korišćene vrste balansiranih uređenih binarnih drveta:

- Adeljson-Veljski Landisova drveta (AVL drveta) i
- crveno-crna drveta.

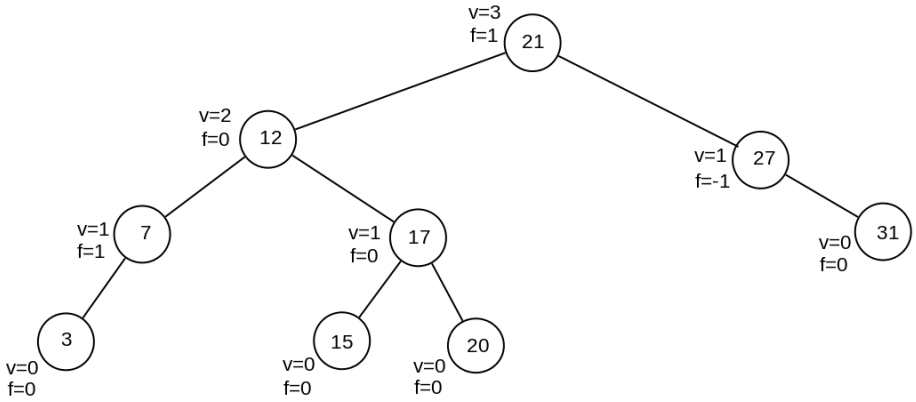
AVL drveta

AVL drveta su struktura podataka koja garantuje da složenost ni jedne od operacija traženja, umetanja i brisanja u najgorem slučaju nije veća od $O(\log n)$, gde je n broj elemenata u strukturi. Naziv je dobila po prvim slovima prezimena dva ruska matematičara: Georgii Adelson-Velskii i Evgenii Mikhailovich Landis koji su predložili ovu strukturu podataka. Posle svake od operacija umetanja i brisanja elemenata ulaže se dodatni napor da se drvo izbalansira, tako da visina

[1] Asocijativne strukture podataka podrazumevaju pristup elementima ne na osnovu indeksa, odnosno pozicije elementa u strukturi, već na osnovu vrednosti ključa.

drveta uvek bude $O(\log n)$. Pri tome se balansiranost drveta definiše tako da se može lako održavati.

Visina drveta je najveće rastojanje korena do nekog lista. Ona se može razmatrati u terminima broja čvorova ili broja grana na najdužem putu od korena do lista. Mi ćemo ovde usvojiti drugu konvenciju, odnosno gledati broj grana na putu, te će drvo koje sadrži jedan čvor biti visine 0, drvo koje sadrži dva čvora biti visine 1, a prazno drvo visine -1. Definišimo pojam *faktora ravnoteže čvora*, kao razlike visina levog i desnog poddrveta tog čvora. Pritom ako čvor nema levo (desno) dete, visina levog (desnog) poddrveta je podrazumevano -1. AVL drvo se definiše kao uređeno binarno drvo kod koga je *za svaki čvor apsolutna vrednost faktora ravnoteže manja ili jednaka od jedan*, odnosno za svaki čvor važi da visine levog i desnog poddrveta mogu razlikovati najviše za jedan.

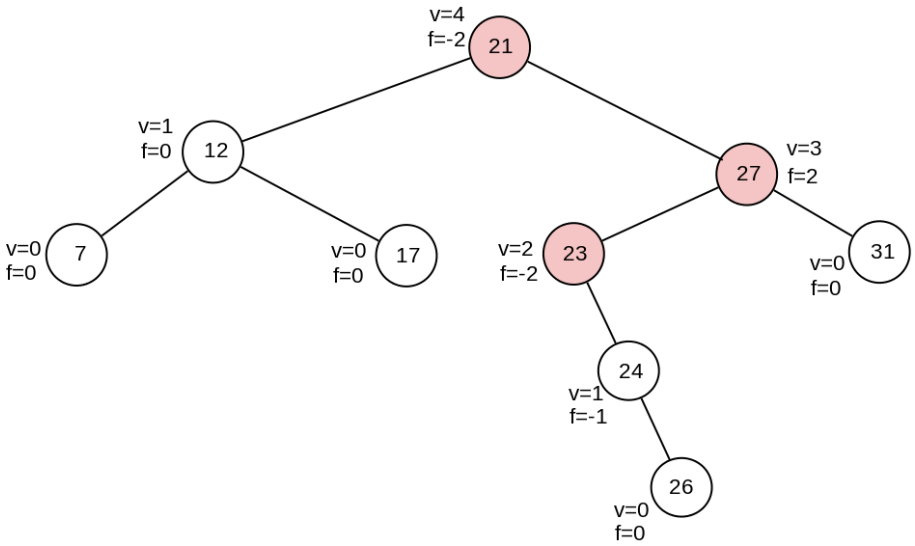


Slika 1: Primer AVL drveta. Uz svaki čvor drveta prikazana je njegova visina i faktor ravnoteže tog čvora.

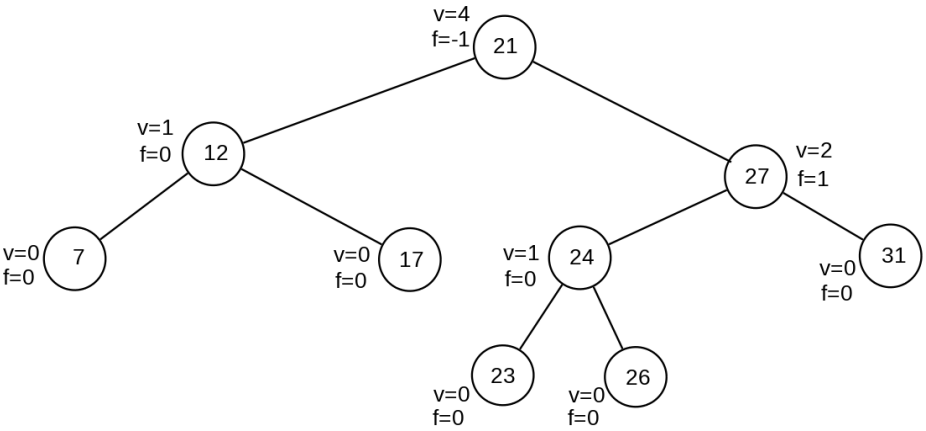
Na slici 1 dat je primer jednog AVL drveta. Uz svaki čvor prikazana je visina drveta sa korenom u tom čvoru i faktor ravnoteže čvora. Možemo primetiti da su faktori ravnoteže svih čvorova datog drveta iz skupa $\{-1, 0, 1\}$. Na primer, faktor ravnoteže čvora čija je vrednost ključa 7 je 1 jer je visina njegovog levog poddrveta 0, a desnog -1, dok je faktor ravnoteže čvora sa vrednošću ključa 21 jednak 1, jer je visina njegovog levog poddrveta 2, a desnog 1.

Na slici 2 prikazano je jedno uređeno binarno drvo koje nije AVL drvo. Naime, faktori ravnoteže čvorova sa vrednostima ključa 21, 27 i 23 nisu iz skupa $\{-1, 0, 1\}$. Međutim, malim korekcijama ovog drveta možemo od njega dobiti balansirano binarno uređeno drvo prikazano na slici 3.

Primetimo da prema definiciji AVL drveta uslov za faktor ravnoteže treba da važi za svaki čvor u drvetu, odakle sledi da je svako poddrvo AVL drveta takođe AVL drvo.



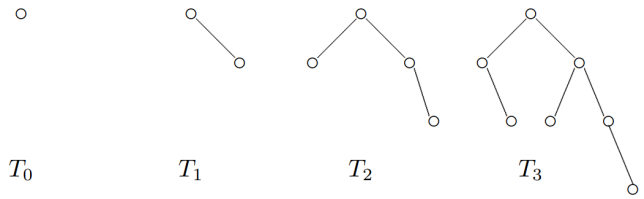
Slika 2: Primer uređenog binarnog drveta koje nije AVL drvo.



Slika 3: Balansiranje prethodno prikazanog ne-AVL drveta tako da postane AVL drvo.

Teorema: Visina h AVL drveta sa n čvorova zadovoljava uslov $h \leq 2\log_2 n$.

Dokaz: Dokažimo ovo tvrđenje indukcijom po visini drveta h. Prvo, primetimo da visina drveta ne određuje jednoznačno broj čvorova u drvetu. Za dokazivanje navedene nejednakosti kritična su drveta sa najmanjim brojem čvorova za datu visinu (jer ako ona ne krše nejednakost, ne krše je ni drveta sa više čvorova zbog monotonosti logaritma). Stoga se u dokazu bavimo najmanjim drvetima date visine. Neka je T_h AVL drvo visine $h \geq 0$ sa najmanjim mogućim brojem čvorova. Drvo T_0 sadrži samo koren, a drvo T_1 koren i jedno dete, recimo desno. Za $h \geq 2$ drvo T_h može se formirati na sledeći način: njegovo poddrvo manje visine $h-2$ takođe treba da bude AVL drvo sa minimalnim brojem čvorova, dakle T_{h-2}; slično, njegovo drugo poddrvo treba da bude T_{h-1}. Drveta opisana ovakvom rekurentnom jednačinom zovu se *Fibonačijeva drveta*. Nekoliko prvih Fibonačijevih drveta, takvih da je u svakom unutrašnjem čvoru visina levog poddrveta manja, prikazano je na slici 4.



Slika 4: Fibonačijeva AVL drveta.

Označimo sa n_h minimalni broj čvorova AVL drveta visine h, tj. broj čvorova drveta T_h. Važi $n_0 = 1$, $n_1 = 2$, $n_2 = 4, \ldots$ Prema definiciji Fibonačijevih drveta važi $n_h = n_{h-1} + n_{h-2} + 1$, za $h \geq 2$. S obzirom na to da je $n_h > n_{h-1}$ za svako $h \geq 1$ važi:

$$n_h = n_{h-1} + n_{h-2} + 1 > 2n_{h-2} + 1 > 2n_{h-2}$$

Dokažimo indukcijom da važi tvrđenje

$$n_h \geq 2^{h/2}$$

Za $h = 0$ i $h = 1$ tvrđenje važi. Pretpostavimo da tvrđenje važi za $h-2$, odnosno da važi $n_{h-2} \geq 2^{(h-2)/2}$. Na osnovu veze $n_h > 2n_{h-2}$ važi

$$n_h \geq 2 \cdot 2^{(h-2)/2} = 2^{h/2-1+1} = 2^{h/2}$$

te tvrđenje važi. Nakon logaritmovanja obe strane dobijamo

$$h \leq 2\log_2 n_h \leq 2\log_2 n$$

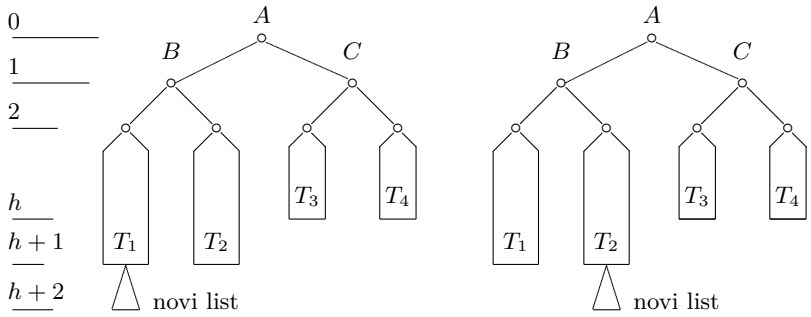
jer je po pretpostavci za sva drveta visine h broj čvorova n veći ili jednak od n_h.

Dakle, visina AVL drveta sa n čvorova je $O(\log n)$.□

Razmotrimo sada kako se realizuju osnovne operacije nad AVL drvetom, odnosno na koji način se nakon umetanja, odnosno brisanja elementa iz AVL drveta ono može „popraviti“ tako da i dalje ostane AVL drvo.

Umetanje elementa u AVL drvo Prilikom umetanja novog elementa u AVL drvo postupa se najpre na način uobičajen za uređeno binarno drvo: pronalazi se mesto čvoru, pa se u drvo dodaje novi list sa ključem jednakim zadatom broju. Čvorovima na putu koji se tom prilikom prelaze odgovaraju faktori ravnoteže iz skupa $\{0, 1, -1\}$. Posebno je interesantan poslednji čvor na tom putu koji ima faktor ravnoteže različit od nule, tzv. *kritični čvor*. Ispostavlja se da je prilikom umetanja elementa u AVL drvo *dovoljno izbalansirati poddrvo sa korenom u kritičnom čvoru*.

Primetimo da za svaki čvor važi da se visine levog i desnog poddrveta pre umetanja mogu razlikovati najviše za jedan. Nakon umetanja elementa u jedno od poddrveta, visine se mogu razlikovati najviše za dva.



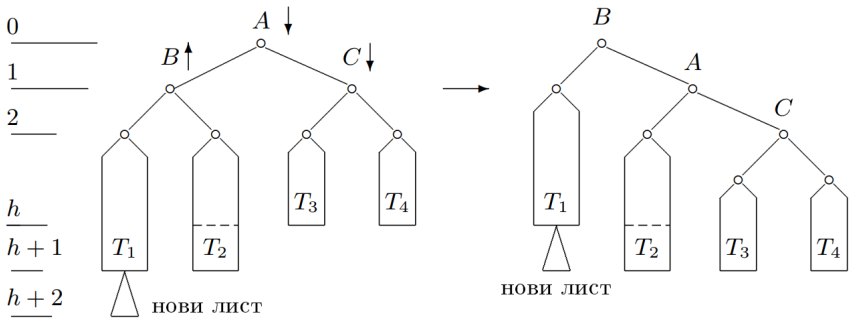
Slika 5: Umetanja koja remete AVL svojstvo drveta.

Pretpostavimo da je faktor ravnoteže u kritičnom čvoru jednak 1 (slika 5), odnosno da levo poddrvo ima za jedan veću visinu od desnog. Novi čvor N može da završi u:

- desnom poddrvetu – u tom slučaju poddrvo kome je koren kritični čvor ostaje AVL
- levom poddrvetu – u tom slučaju drvo prestaje da bude AVL jer je visina levog drveta za 2 veća od visine desnog i potrebno je intervenisati. Prema tome da li je novi čvor dodat levom ili desnom poddrvetu levog poddrveta, razlikujemo dva slučaja.
  - u slučaju kada je novi čvor dodat levom poddrvetu levog poddrveta na drvo se primenjuje tzv. *LL rotacija* (ilustrovana na slici 6): koren levog poddrveta B drveta sa korenom u kritičnom čvoru A „podiže“ se i postaje koren poddrveta kome je koren bio kritičan čvor, a ostatak drveta preuređuje se tako da drvo i dalje ostane uređeno binarno drvo. Drvo T_1 se podiže za jedan nivo ostajući i dalje levo poddrvo

čvora B; drvo T_2 ostaje na istom nivou, ali umesto desnog poddrveta čvora B postaje levo poddrvo čvora A (primetimo da se na ovaj način uslov uređenosti drveta ne remeti); desno poddrvo čvora A spušta se za jedan nivo. Pošto je A kritičan čvor, faktor ravnoteže čvora B je nužno 0, pa drveta T_1 i T_2 imaju istu visinu. Drveta T_3 i T_4 ne moraju imati istu visinu, jer čvor C nije na putu od kritičnog čvora do mesta umetanja. Novi koren poddrveta postaje čvor B, sa faktorom ravnoteže 0. Čitaocu ostavljamo da se uveri da se ovom rotacijom održavaju svojstva uređenosti drveta.

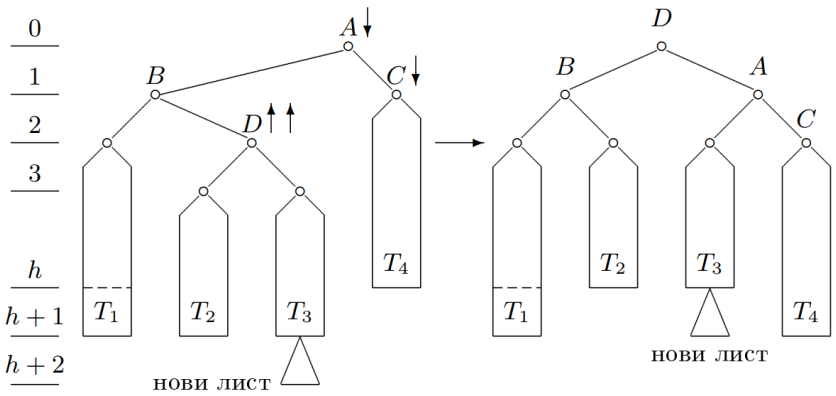
– slučaj kada je novi čvor dodat desnom poddrvetu levog poddrveta je komplikovaniji; tada se drvo može uravnotežiti *LR rotacijom*, odnosno *dvostrukom rotacijom* (ilustrovanom na slici 7). Novi čvor poddrveta umesto kritičnog čvora postaje desno dete levog deteta kritičnog čvora. Primetimo opet da pošto je A kritični čvor, drveta T_1, T_2 i T_3 moraju pre umetanja imati istu visinu. Nakon LR rotacije, faktori ravnoteže čvorova B, A i D biće redom jednaki 1, 0 i 0, odnosno drvo postaje AVL. Čitaocu ostavljamo da se uveri da se i dvostrukom rotacijom održavaju svojstva uređenosti drveta.



Slika 6: LL rotacija.

Analogno, u slučaju kada je faktor ravnoteže u kritičnom čvoru jednak -1 (odnosno, kada je visina desnog poddrveta za jedan veća od visine levog), i kada se vrši umetanje čvora u desno poddrvo desnog poddrveta, potrebno je primeniti *RR rotaciju*, a ako se vrši umetanje u levo poddrvo desnog poddrveta uravnotežavanje se vrši *RL rotacijom*, odnosno dvostrukom rotacijom. Sve pomenute vrste rotacija ilustrovane su na slici 8. Primetimo da kod LL i RR rotacije pre vršenja rotacije, a i nakon vršenja rotacija važi naredni odnos vrednosti čvorova $A < B < C < D < E$, dok kod LR i RL rotacija se takođe nakon rotacije čuva odnos vrednosti čvorova i on iznosi: $A < B < C < D < E < F < G$.

Zapazimo da u svakom od razmatranih slučajeva *visina poddrveta kome je koren kritični čvor posle balansiranja ostaje nepromenjena.* Balansiranje poddrveta kome je koren kritičan čvor zbog toga ne utiče na ostatak drveta.

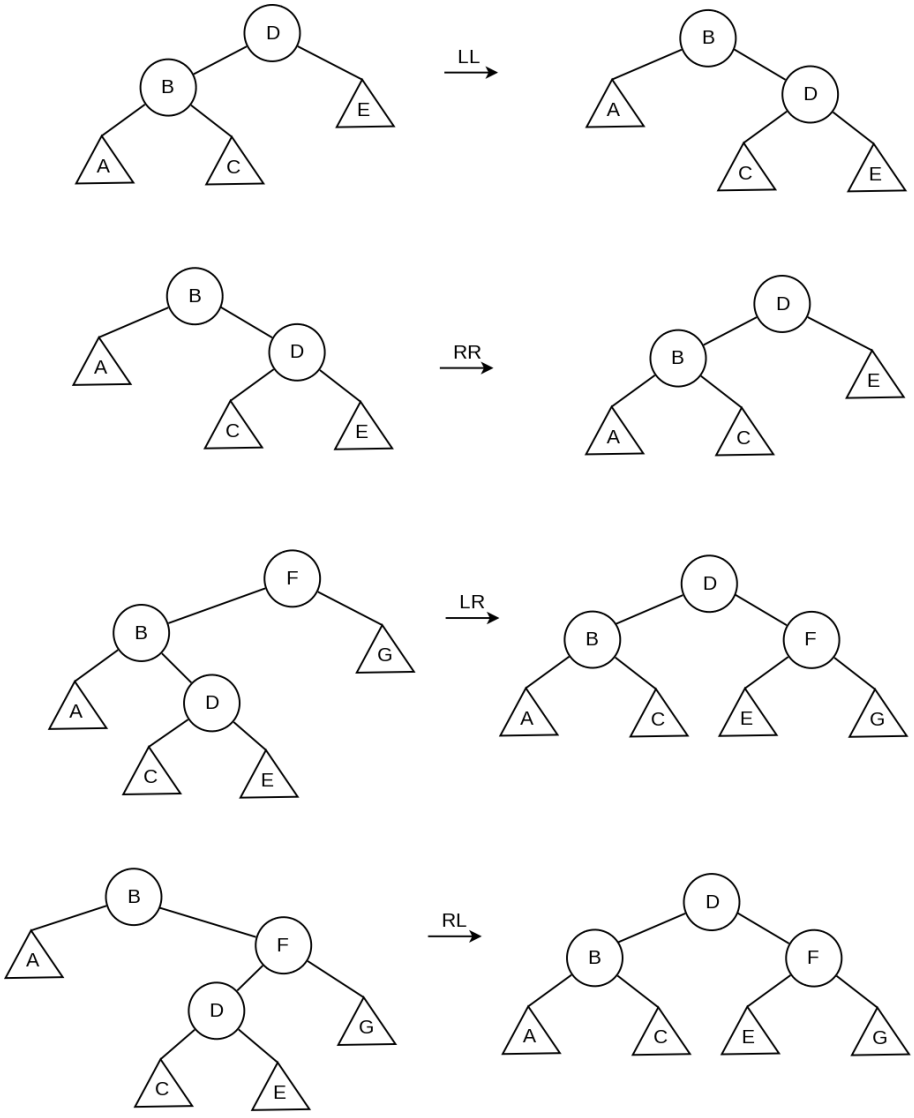


Slika 7: LR rotacija, odnosno dvostruka rotacija.

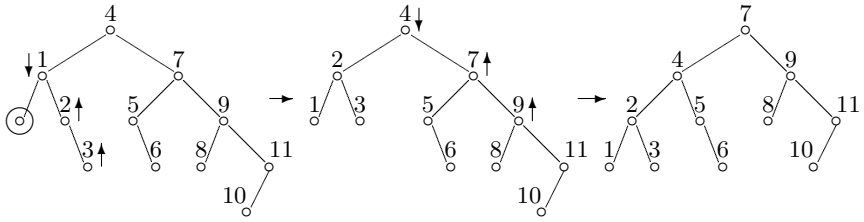
U implementaciji AVL drveta uz svaki čvor drveta čuva se njegova visina i njegov faktor ravnoteže, jednak razlici visina njegovog levog i desnog poddrveta; za AVL drvo su te razlike elementi skupa $\{-1, 0, 1\}$. Uravnotežavanje postaje neophodno ako je faktor nekog čvora iz skupa $\{-1, 1\}$, a novi čvor se umeće na pogrešnu stranu.

Brisanje elementa iz AVL drveta Brisanje iz AVL drveta je komplikovanije, kao i kod običnog uređenog binarnog drveta. U opštem slučaju se balansiranje drveta ne može izvesti pomoću samo jedne ili dve rotacije. Na primer, da bi se Fibonačijevo drvo F_h sa n čvorova uravnotežilo posle brisanja „loše“ izabranog čvora, potrebno je izvršiti $h - 2$, odnosno $O(\log n)$ rotacija (slika 9). U opštem slučaju je granica za potreban broj rotacija $O(\log n)$. Na sreću, svaka rotacija zahteva konstantni broj koraka, pa je vreme izvršavanja brisanja takođe ograničeno odozgo sa $O(\log n)$.

Primene AVL drveta Prednost AVL drveta je to što je samobalansirajuće, čime se njegova visina održava da bude reda $O(\log n)$. Na ovaj način se za operacije pretrage, umetanja i brisanja garantuje složenost od $O(\log n)$ u najgorem slučaju. AVL drveta su najstarija samobalansirajuća uređena binarna drvet. Nastala su iz potrebe da se za indeksiranje baza podataka koriste balansirana binarna drveta, koja omogućavaju efikasnu pretragu i dohvatanje podataka. AVL drveta ima smisla koristiti u aplikacijama koje rade sa bazama podataka kod kojih se često sprovode operacije pretrage, dok su umetanja i brisanja ređa. Na primer, baza podataka o postojećim linijama metroa može koristiti AVL drvo jer se nove linije metroa dodaju jako retko, dok je broj operacija pretrage relativno visok jer veliki broj ljudi svakodnevno pretražuje bazu kako bi rezervisao kartu za metro. Koriste se takođe i u svim drugim primenama koje zahtevaju efikasnu pretragu. AVL drveta se mogu koristiti i za internu implementaciju raznih tipova



Slika 8: Ilustracija četiri vrste rotacija: LL rotacije, RR rotacije, LR rotacije i RL rotacije. A, C, E i G su balansirana uređena binarna drveta.

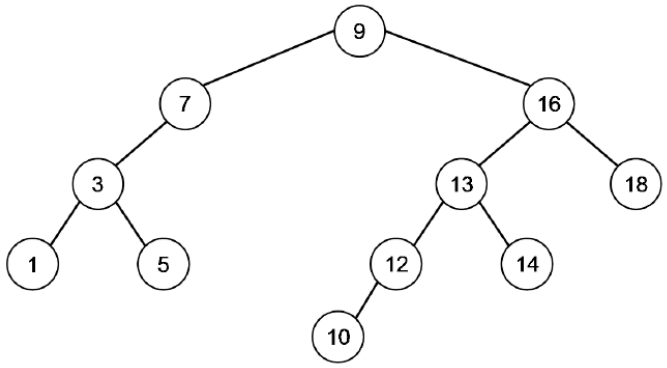


Slika 9: Uravnotežavanje Fibonačijevog drveta T_4 posle brisanja čvora, pomoću dve rotacije.

internih kolekcija, kao što su uređeni skupovi i rečnici. Takođe, mogu imati primenu u sistemima koji rade u realnom vremenu, kao što su sistemi za kontrolu letova ili u video igrama, kada je potrebno brzo sprovesti operacije nad podacima, u logaritamskom vremenu u najgorem slučaju.

Zadaci za vežbu

1. Za koje čvorove datog drveta ne važi uslov balansiranosti?



2. Nacrtati sve rotacije koje je potrebno izvršiti tokom umetanja elemenata $1, 10, 5, 7, 3, 13, 6, 4, 8, 9$ u datom redosledu u prazno AVL drvo kako bi ono ostalo AVL.

3. Pokazati da je dvostruka rotacija (LR i RL) kompozicija dve rotacije tipa LL ili RR.

4. U AVL drvo se unosi pet ključeva $6, 20, 45, 80$ i 96. Utvrditi redosled umetanja koji će u nekom trenutku prouzrokovati dvostruku rotaciju (LR ili RL).

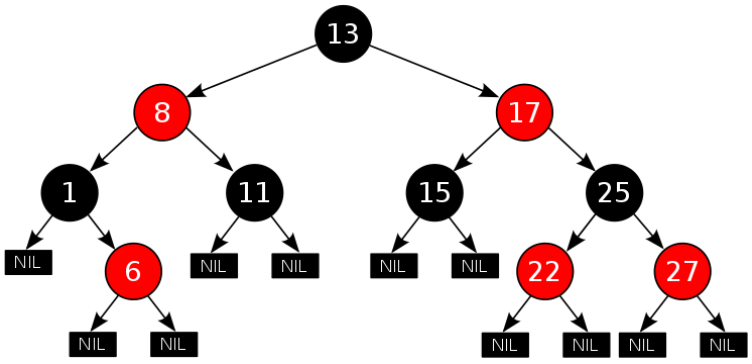
5. Neka su a, c i e proizvoljni čvorovi u poddrvetima A, C i E sa slike 8. Kako se menjaju visine ovih čvorova nakon svake od rotacija?

Crveno-crna drveta

Crveno-crno drvo (eng. red-black tree, RBT) je uređeno binarno drvo koje dodatno mora da zadovolji sledeće invarijante:

1. Svaki čvor je ili crven ili crn.
2. Koren je crn.
3. Svi listovi su crni i ne sadrže vrednosti (označavamo ih sa NIL).
4. Svi crveni čvorovi imaju tačno dva crna deteta.
5. Sve putanje od nekog čvora do listova u njegovom poddrvetu sadrže isti broj crnih čvorova.

Primer jednog crveno-crnog drveta je prikazan na slici 10. Primetimo da je koren drveta crne boje, a da su svi listovi crne boje i sadrže vrednost NIL. Svaki crveni čvor ima tačno dva crna deteta, dok za crne čvorove važi da mogu imati dva crvena, dva crna, ili crveno i crno dete. Razmotrimo sve putanje od čvora sa vrednošću 13 do listova – sve putanje sadrže po 2 crna čvora (ako ne računamo čvor 13), ali različit broj crvenih čvorova.



Slika 10: Primer crveno-crnog drveta.

Naravno, NIL čvorovi ne moraju nužno biti kodirani kao čvorovi drveta, već bi mogli da odgovaraju vrednostima pokazivača. Međutim, ovakav pogled na njih olakšava operacije sa crveno-crnim drvetom jer NIL čvorovi mogu imati roditelja, brata i mogu biti deca, ali nikad roditeljski čvor.

Navedena svojstva nam garantuju da će *svaka putanja od korena do njemu najdaljeg lista biti najviše duplo duža nego putanja do njegovog najbližeg lista.* Zaista, najkraća putanja do nekog lista će se sastojati samo od crnih čvorova, dok će se se u najdužoj putanji naizmenično smenjivati crveni i crni listovi (jer na osnovu uslova 4. posle crvenog čvora mora doći crni, a na osnovu uslova 5. sve putanje imaju isti broj crnih čvorova). Ovo svojstvo nam garantuje određeni vid balansiranosti drveta i logaritamsku složenost operacija (pod uslovom da svako umetanje u drvo i brisanje elementa iz drveta održava nabrojanih pet invarijanti).

Dokažimo da su prethodno navedeni uslovi dovoljni da bi se garantovalo da visina drveta logaritamski zavisi od broja čvorova. Neka je:

- $h(v)$ – *visina poddrveta* čiji je koren čvor v, tj. broj čvorova od čvora v do najdaljeg lista (ne računajući čvor v);
- $h_b(v)$ – *crna visina poddrveta* čiji je koren čvor v, tj. broj crnih čvorova od čvora v do proizvoljnog lista (ne računajući čvor v ako je on crn)[2].

Na primer, visina $h(v)$ čvora 13 sa slike 10 je 4, a crna visina $h_b(v) = 2$, dok je visina čvora 8 jednaka 3, a crna visina 2.

Pod *unutrašnjim čvorovima* drveta podrazumevaćemo sve čvorove sem listova, odnosno NIL čvorova i mi ćemo se u daljem tekstu uglavnom fokusirati na unutrašnje čvorove drveta jer oni sadrže vrednosti ključeva.

Lema: Crveno-crno drvo sa korenom u čvoru v ima bar $2^{h_b(v)} - 1$ unutrašnjih čvorova.

Dokaz: Označimo sa n broj unutrašnjih čvorova datog drveta. Dokaz teče indukcijom po visini drveta tj. po vrednosti $h(v)$.

- Bazu čini slučaj $h(v) = 0$. Visinu nula ima jedino NIL čvor za koji važi $n = 0$, pa je tada i $h_b(v) = 0$, odnosno $2^{h_b(v)} - 1 = 0$, pa je zahtev tvrđenja trivijalno ispunjen.
- Pretpostavimo kao induktivnu hipotezu da svako drvo sa korenom u čvoru w čija je visina $h(w) < k$ ima bar $2^{h_b(w)} - 1$ unutrašnjih čvorova. Neka drvo ima koren u čvoru v i neka je $h(v) = k$. Pošto je $h(v) > 0$ čvor v je unutrašnji i ima dva deteta v_l i v_d koji su koreni drveta visine manje od k te za njih važi induktivna hipoteza. Za čvor v_l važi da mu je crna visina ili $h_b(v)$ (ako je čvor v_l crven) ili $h_b(v) - 1$ (ako je čvor v_l crn). Analogno važi i za čvor v_d. Na osnovu induktivne hipoteze za broj unutrašnjih čvorova n_l poddrveta sa korenom u čvoru v_l važi:

  $$n_l \geq 2^{h_b(v_l)} - 1 \geq 2^{h_b(v)-1} - 1,$$

  i slično za broj unutrašnjih čvorova n_d poddrveta sa korenom u čvoru v_d:

  $$n_d \geq 2^{h_b(v)-1} - 1,$$

  pa je broj unutrašnjih čvorova drveta sa korenom u čvoru v bar

  $$n = n_l + n_d + 1 = 2(2^{h_b(v)-1} - 1) + 1 = 2^{h_b(v)} - 1$$

  (objedini smo unutrašnje čvorove u oba poddrveta i čvor v). □

Teorema: Visina h crveno-crnog drveta koje sadrži n unutrašnjih čvorova zadovoljava uslov $h \leq 2\log_2(n+1)$.

[2]Prema svojstvu 5. pojam crne visine je dobro definisan jer sve putanje od nekog čvora do listova sadrže isti broj crnih čvorova.

Dokaz: Na osnovu prethodne leme važi da je broj unutrašnjih čvorova u crveno-crnom drvetu sa korenom u čvoru v bar $2^{h_b(v)} - 1$. Pošto je bar pola čvorova na svakoj putanji od korena v do listova crno, važi da je $h_b(v) \geq h(v)/2$. Zato je na osnovu leme broj unutrašnjih čvorova n veći ili jednak $2^{h(v)/2} - 1$. Zato je $n + 1 \geq 2^{h(v)/2}$, pa je $\log_2(n+1) \geq h(v)/2$ i važi da je $h(v) \leq 2\log_2(n+1)$. □

Iz tvrđenja prethodne teoreme direktno sledi da je visina crveno-crnog drveta $O(\log n)$.

Crveno-crna drveta je 1972. godine izumeo Rudolf Bayer, dok su se kasnije 1978. godine Leonidas Guibas i Robert Sedgewick bavili analizom njihovih svojstava i uveli crveno-crnu konvenciju bojenja čvorova[3]. 1993. godine Arne Andersson je naknadno osmislio jednostavniju varijantu crveno-crnih drveta kojima se pojednostavljuju operacije umetanja i brisanja.

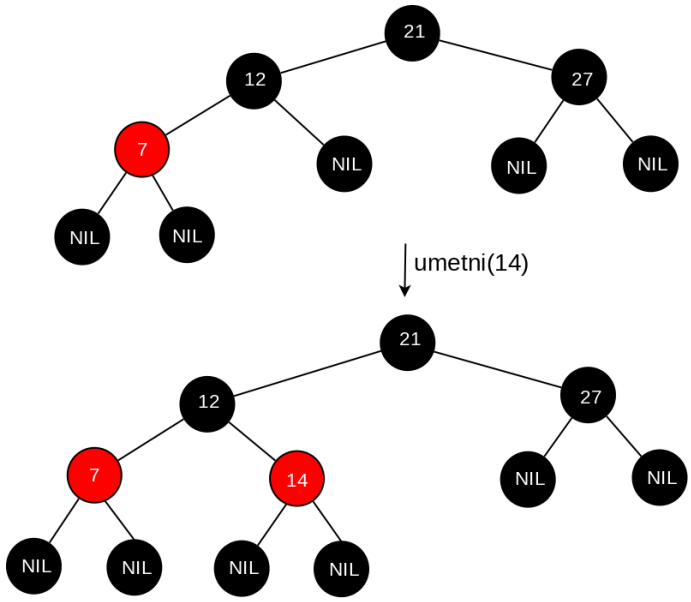
U nastavku teksta razmotrićemo na koji način se realizuju osnovne operacije nad crveno-crnim drvetom. Poseban akcenat je kao i kod AVL drveta stavljen na to da se nakon umetanja, odnosno brisanja elementa iz crveno-crnog drveta mogu pokvariti neka od svojstava crveno-crnog drveta i onda je potrebno izmeniti boje čvorova i/ili preusmeriti neke od pokazivača, tako da dobijeno drvo i dalje ostane crveno-crno drvo. Preusmeravanje pokazivača se vrši operacijama rotacije drveta. Postoje leva i desna rotacija i one odgovaraju LL i RR rotaciji u AVL drvetu (vidi sliku 8).

Umetanje elementa u crveno-crno drvo Umetanje novog čvora u crveno-crno drvo se sprovodi na način uobičajen za uređena binarna drveta: na osnovu vrednosti ključa pronalazi se mesto gde treba dodati novi čvor i na mesto NIL lista se postavlja novi čvor sa zadatom vrednošću ključa. Pritom se novi čvor boji crveno i dodaju mu se dva crna NIL deteta. Na slici 11 ilustrovan je primer umetanja elementa u crveno-crno drvo. Primetimo da je drvo i nakon umetanja elementa ostalo crveno-crno.

Razmotrimo koja svojstva crveno-crnih drveta su pri umetanju elementa i njegovog bojenja u crveno mogla biti narušena. Svojstvo 1, kojim se tvrdi da su svi čvorovi obojeni crveno ili crno, i svojstvo 3 kojim se tvrdi da je svaki list crn trivijalno ostaju očuvana. Svojstvo 5, kojim se tvrdi da je broj crnih čvorova isti na svakoj putanji od fiksiranog čvora do proizvoljnog lista je zadovoljen jer je novi čvor obojen crveno i sa svoja dva crna deteta (lista) je zamenio prethodni crni list. Stoga, jedina dva svojstva koja su mogla biti narušena operacijom umetanja elementa su svojstva 2 i 4. Svojstvo 2, kojim se zahteva da je koren drveta crn je mogao biti narušen samo u slučaju da je novododati čvor koren drveta, dok je svojstvo 4 kojim se zahteva da crveni čvor ne može imati crveno dete mogao biti narušen ako je roditelj novododatog čvora bio crven.

Važno je primetiti da je tačno jedno od svojstava 2 i 4 moglo biti narušeno: naime, svojstvo 2 je moglo biti narušeno samo u situaciji da je čvor koji se umeće

[3]Uz crnu, crvena boja je izabrana zato što je davala najbolji prikaz na laserskom štampaču koji je autorima bio dostupan.



Slika 11: Primer umetanja elementa u crveno-crno drvo kojim se ne narušava nijedan od uslova.

koren drveta, ali on u ovom slučaju nema roditelja, a ima dva crna NIL deteta, te svojstvo 4 nije moglo biti istovremeno narušeno.

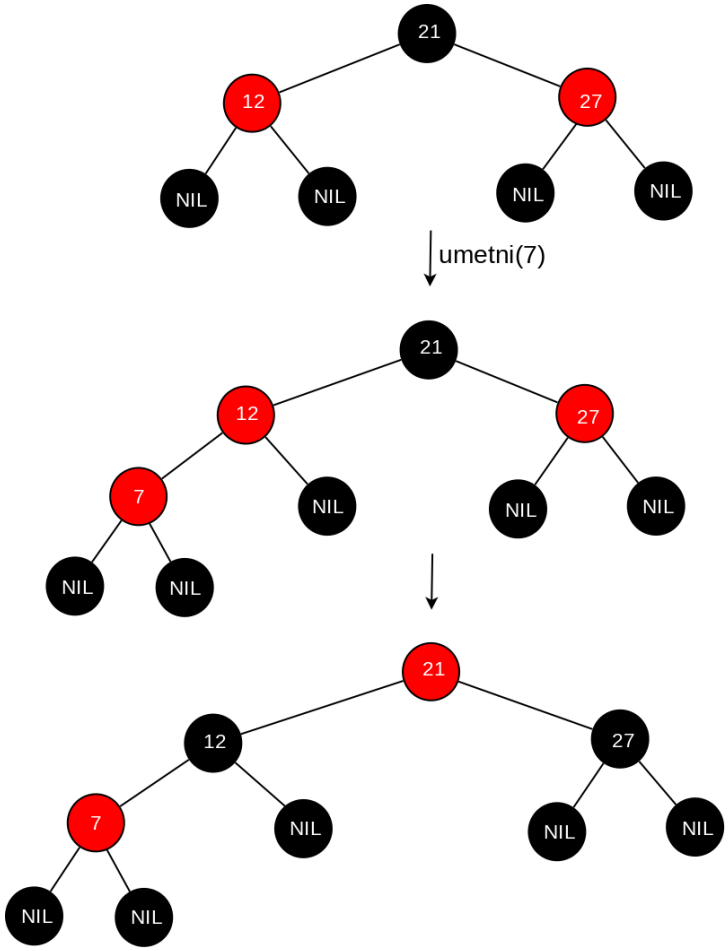
U daljem razmatranju ćemo često referisati na čvor koji je brat roditelja razmatranog čvora i na njega ćemo referisati kao na *ujaka* datog čvora.

Razmotrimo kako održati sva svojstva crveno-crnih drveta nakon umetanja novog elementa z. Najpre, ukoliko je narušen uslov 2, odnosno ako se čvor umeće u prazno drvo i postaje koren drveta, potrebno je čvoru koji se umeće boju promeniti iz crvene u crnu.

Inače, pretpostavićemo da je roditelj čvora z levo dete svog oca (slučaj kada je z desno dete svog oca se analogno razmatra). Odnos koji je narušen nakon umetanja jeste odnos novog čvora z i njegovog roditelja, jer su oba ova čvora crvena. Razlikujemo tri različita slučaja:

1. Prvi slučaj nastupa kada su istovremeno i roditelj i ujak čvora z crvene boje. S obzirom na to da je deda čvora z crn, možemo njegovoj deci (roditelju i ujaku čvora z) promeniti boju iz crvene u crnu, a njega obojiti u crveno (slika 12). Na ovaj način smo razrešili problem jer su istovremeno i čvor z i njegov roditelj bili crvene boje, a održali smo i svojstvo 5 (broj crnih čvorova na putanjama ostaje isti). Primetimo da smo na ovaj način mogli poremetiti svojstvo između dede čvora z i njegovog roditelja, te isti postupak treba sada sprovesti za čvor dva nivoa iznad čvora z, odnosno za

dedu čvora z. Međutim, s obzirom na to da je visina drveta logaritamske složenosti u funkciji broja čvorova drveta, broj ovakvih popravki je odozgo ograničen sa $O(\log n)$.



Slika 12: Primer umetanja elementa u crveno-crno drvo kada su i roditelj i ujak novog čvora crveni.

2. Drugi slučaj se odnosi na situaciju kada je ujak čvora z crn, a z je desno dete svoga roditelja (slika 13 (a)). U ovom scenariju se primenom leve rotacije drvo transformiše u situaciju kada je z levo dete svog roditelja i na taj način se svodi na treći slučaj. S obzirom na to da su i čvor z i njegov roditelj crvene boje, rotacija ne utiče na crne visine čvorova, te stoga ni na svojstvo 5.

3. U trećem slučaju koji nastupa kada je ujak čvora z crn, a z je levo dete svoga roditelja, vrši se izmena boja čvorova i desna rotacija, kojom se

čuva svojstvo 5 (slika 13 (b)). Primetimo da nakon ovoga ne postoje više dva crvena čvora koja su u odnosu roditelj-dete, niti se neki od problema prolongira uz drvo. Dakle, u ovom slučaju je jedna rotacija ovog tipa bila dovoljna da drvo ponovo zadovoljava svih pet navedenih svojstava.

Primetimo da nijedan od tri razmatrana scenarija ne narušava svojstva $1, 3$ i 5, a ni svojstvo 2 jer čvor z nije koren. Primetimo takođe, da se samo u slučaju prvog od tri različita slučaja, čvor z pomera dva nivoa iznad i ponovo se razmatra da li je došlo do narušavanja nekog svojstva crveno-crnih drveta. S obzirom na to da se može izvesti najviše $O(\log n)$ korekcija, ukupna složenost popravki je u najgorem slučaju $O(\log n)$.

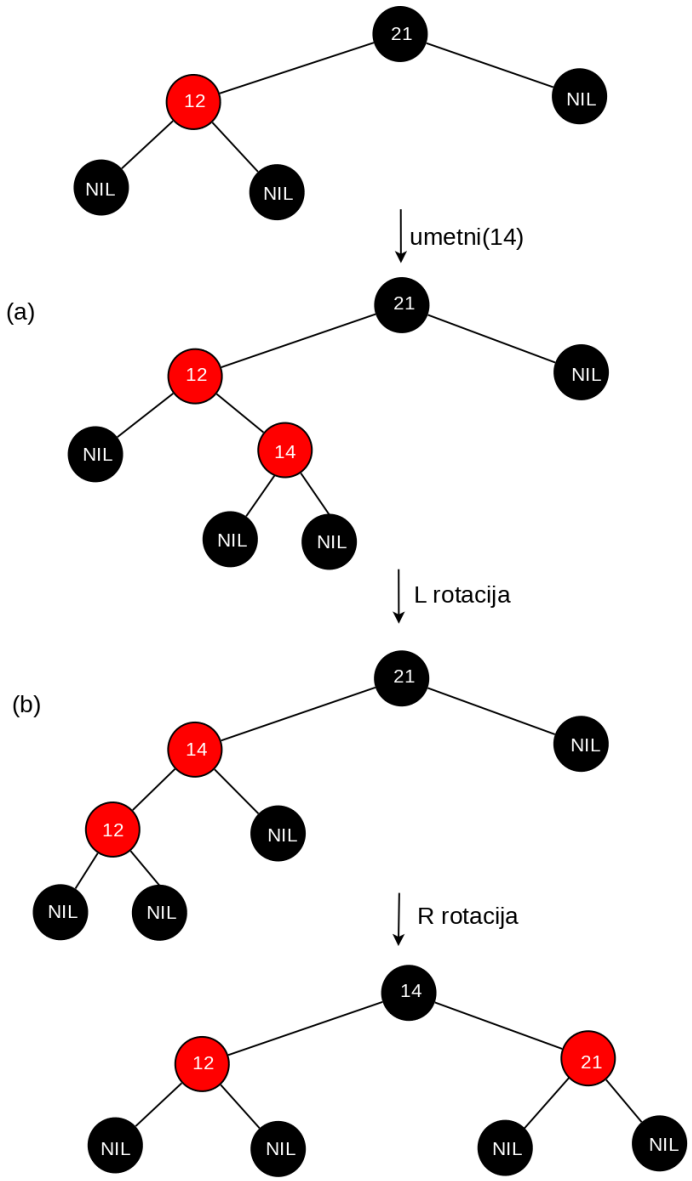
Brisanje elementa iz crveno-crnog drveta Kao i u slučaju AVL drveta, operacija brisanja je složenija i nećemo je detaljno razmatrati u ovom materijalu.

Poređenje AVL drveta sa crveno-crnim drvetima AVL drveta su strožije izbalansirana od crveno-crnih drveta. U opštem slučaju visina AVL drveta je manja, pa se pretraga u AVL drvetu izvodi efikasnije nego u crveno-crnim drvetima. Stoga se AVL drveta pokazuju kao bolja opcija kada je očekivani broj operacija pretraživanja značajno veći od broja operacija umetanja. Međutim, bojenje čvorova u crveno-crnim drvetima omogućava manji broj potrebnih operacija rebalansiranja, jer nam bojenje čvorova nekada omogućava da izbegnemo ili makar smanjimo broj operacija rebalansiranja. AVL drveta sobom nose veće troškove rotacije nego crveno-crna drveta, što će, u situacijama kada se očekuje veliki broj umetanja, dovesti do velikog broja rotacija i tada neka druga struktura podataka poput crveno-crnih drveta predstavlja bolji izbor.

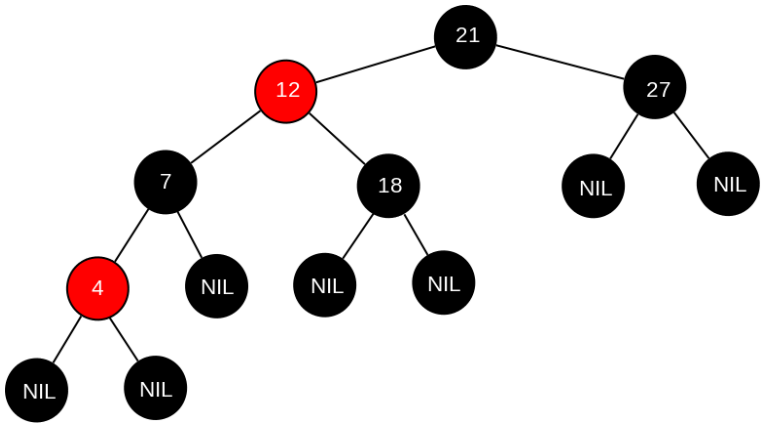
Primetimo i to da se svako AVL drvo može transformisati u crveno-crno bojenjem čvorova redom u crveno i crno, dok postoje crveno-crna drveta koja nisu AVL (slika 14).

Zadaci za vežbu

1. Prikazati crveno-crno drvo koje se dobija uzastopnim umetanjima ključeva $41, 38, 31, 12, 19, 8$ u inicijalno prazno crveno-crno drvo.
2. Nacrtati potpuno uređeno binarno drvo visine 3 koje sadrži vrednosti ključeva $\{1, 2, \ldots, 15\}$. Dodati NIL čvorove i obojiti čvorove na tri različita načina tako da je visina drveta u terminima broja crnih čvorova 2, 3 i 4.
3. Koliki je maksimalni, a koliki minimalni mogući broj unutrašnjih čvorova u crveno-crnom drvetu čija je crna visina k?
4. Opisati crveno-crno drvo sa n ključeva koji ima maksimalni mogući odnos crvenih unutrašnjih čvorova u odnosu na crne unutrašnje čvorove. Koliki je taj odnos?



Slika 13: Primer umetanja elementa u crveno-crno drvo kada je roditelj novog čvora crven, a ujak novog čvora crn.



Slika 14: Primer crveno-crnog drveta koje nije AVL.

5. Opisati crveno-crno drvo sa n ključeva koji ima minimalni mogući odnos crvenih unutrašnjih čvorova u odnosu na crne unutrašnje čvorove. Koliki je taj odnos?
6. Objasniti zašto u crveno-crnom drvetu crveni čvor ne može imati tačno jedan čvor koji nije NIL.
7. Ukoliko bi se čvor čije se umetanje vrši bojio u crno umesto u crveno, onda se svojstvo 4 ne bi moglo narušiti. Zašto ne radimo na ovaj način?

Skip liste

Kao što smo već pomenuli, balansirana uređena binarna drveta se često koriste za realizaciju uređenih asocijativnih struktura podataka. Algoritmi za rad sa balansiranim drvetima tokom izvršavanja osnovnih operacija (umetanja i brisanja elemenata) menjaju strukturu drveta da bi se održao uslov balansiranosti i osigurale dobre performanse.

Skip lista (brza lista ili lista sa prečicama) je probabilistička alternativa balansiranim drvetima. Kako samo ime sugeriše, ona predstavlja uopštenje uređene povezane liste, ali ima dobru očekivanu složenost operacija pretrage, umetanja i brisanja elemenata. Interesantno je da se očekivana složenost ovde ne odnosi na raspodelu ključeva na ulazu, već na generator slučajnih brojeva koji se koristi prilikom umetanja novog elementa u skip listu.

Skip liste su randomizovana (probabilistička) struktura podataka: isti niz operacija umetanja i brisanja može da proizvede različite strukture. Ovo je zato što skip liste koriste tehniku poznatu kao nasumično bacanje novčića za generisanje slučajnih brojeva potrebnih za izgradnju strukture podataka. Tehnika bacanja novčića je ono što skip listama daje probabilističku prirodu i što omogućava efikasnu očekivanu složenost.

Razmotrimo običnu povezanu listu koja sadrži elemente u rastućem poretku. Naime, ključevi se koriste za uređenje elemenata u listi, dok vrednosti u čvorovima mogu biti proizvoljne. Primetimo da i pored sortiranosti ključeva nije moguće vršiti binarnu pretragu nad njom jer nemamo mogućnost indeksnog pristupa, već je prilikom pretrage elementa potrebno u najgorem slučaju pregledati svaki čvor liste (slika 15a)). Ukoliko lista čuva elemente u sortiranom redosledu i svaki drugi čvor ima i pokazivač na čvor dva mesta ispred njega u listi onda je maksimalni broj čvorova koje treba ispitati $\lceil n/2 \rceil + 1$ (gde je sa n označena ukupna dužina liste); za traženje se koriste proređeni pokazivači dok se ne naiđe na čvor sa ključem većim od traženog: tada se pretraga nastavlja putem osnovnih pokazivača (slika 15b)). Opišimo postupak pretrage u slučaju kada lista čuva pet elemenata u sortiranom poretku, a mi tražimo element koji se nalazi na četvrtom mestu u listi. Najpre bismo ispitali prvi čvor, zaključili da je vrednost elementa koji tražimo veća od vrednosti u ovom čvoru i pokazivačem koji preskače po dva čvora liste ispitali treći čvor liste, pošto je i njegova vrednost manja od tražene, naredni bismo ispitali peti čvor liste. Međutim, vrednost petog čvora liste je veća od tražene, te bismo iz trećeg čvora liste idući pokazivačem koji povezuje susedne čvorove liste ispitali četvrti čvor liste i zaključili da se tražena vrednost nalazi na tom mestu. Dakle ukupno bismo analizirali četiri čvora liste što je jednako $\lceil 5/2 \rceil + 1$.

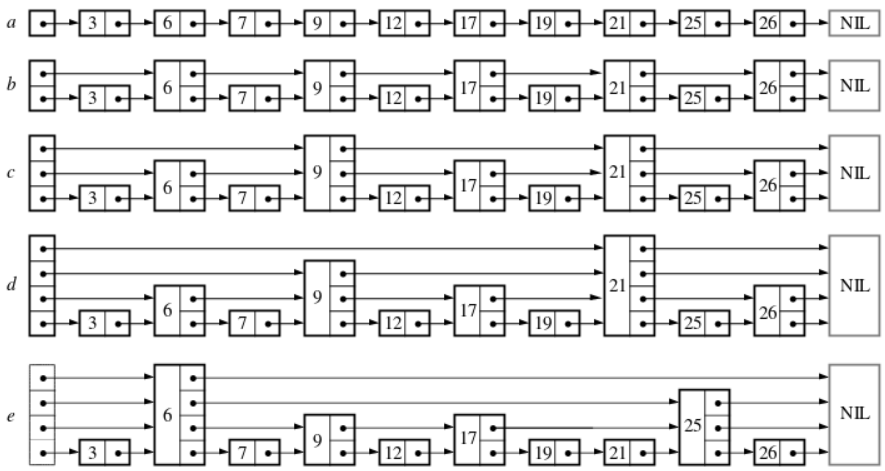
Ako bi svaki četvrti čvor pored toga imao i pokazivač ka čvoru četiri mesta ispred njega, maksimalni ukupan broj čvorova koje treba analizirati bio bi $\lceil n/4 \rceil + 2$ (slika 15c)). Ako svaki 2^i-ti čvor u listi ima pokazivače na čvorove $2^0, 2^1, \ldots, 2^i$ mesta ispred njega u listi, maksimalni broj čvorova koje treba analizirati se smanjuje na $O(\log_2 n)$ (slika 15d)).

Razmotrimo koliki bi bio ukupan broj pokazivača u skip listi. Neka lista sadrži n čvorova. Svih n čvorova ima pokazivač na najnižem nivou, $n/2$ čvorova ima pokazivač na drugom nivou, $n/4$ njih pokazivač na trećem nivou itd. Dakle ukupan broj pokazivača jednak je:

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + \frac{n}{2^{\log_2 n}} = n \sum_{i=0}^{\log n} \frac{1}{2^i} < 2n$$

Zaključujemo da bi ukupan broj pokazivača u skip listi bio dvostruko veći od broja pokazivača u klasičnoj jednostruko povezanoj listi.

Ovako definisana struktura podataka bi bila efikasna za pretragu, ali bi brisanja i umetanja bila potpuno nepraktična. Naime, nakon brisanja ili umetanja elementa, potencijalno bismo morali da za veliki broj čvorova ažuriramo njihove nivoe, tj. broj pokazivača koji iz njih vodi, kao i da pravilno povežemo čvorove pokazivačima. Poželjno je da izmene budu samo lokalne. To možemo postići ako ne fiksiramo unapred informacije koje pozicije u listi imaju koliki broj pokazivača unapred.



Slika 15: Povezana lista sa dodatnim pokazivačima.

Nazovimo čvor koji ima k pokazivača unapred *čvorom nivoa k*. Ukoliko svaki 2^i-ti čvor ima pokazivač na čvor 2^i mesta ispred njega, onda su nivoi čvorova raspodeljeni na sledeći način: 1/2 čvorova je nivoa 1, 1/4 čvorova je nivoa 2, 1/8 njih je nivoa 3, itd. U idealnom slučaju skip lista na svakom narednom nivou sadrži tačno pola elemenata sa prethodnog nivoa. Međutim, kao što smo već napomenuli, ovakav način organizacije nije pogodan za operacije umetanja i brisanja, te se ovaj zahtev mora relaksirati tako da je očekivani broj elemenata na narednom nivou polovina broja elemenata sa prethodnog.

Šta bi se desilo ako bi nivoi čvorova bili birani na slučajan način, ali u istoj proporciji (slika 15e))? Čvorov i-ti pokazivač bi umesto da pokazuje na čvor 2^{i-1} mesta ispred njega, pokazivao na naredni čvor nivoa i ili većeg. Umetanja i brisanja bi zahtevala samo lokalne izmene. Nivo čvora izabran na slučajan način prilikom kreiranja ne bi morao nikada da se menja. Ovakvu strukturu podataka nazivamo skip listom.

U skip listi, elementi su organizovani po nivoima, tako da svaki naredni nivo ima manji broj elemenata od prethodnog nivoa. Krajnji donji nivo predstavlja regularnu povezanu listu, dok nivo iznad njega sadrži pokazivače koji omogućavaju brzi prelaz između elemenata koji su međusobno udaljeni na donjem nivou. Ideja je omogućiti brz prelazak do željenog elementa. Otud i potiče ime strukture jer viši nivoi liste omogućavaju da se „preskoči“ određen broj elemenata liste. Skip liste je 1989. godine izumeo Vilijem Pu (William Pugh).

Očekivani broj čvorova nivoa 1 je $n/2$, nivoa 2 je $n/4$, nivoa 3 je $n/8, \ldots$ nivoa $\log n$ je 1. Dakle, očekivani broj nivoa skip liste je $O(\log n)$.

Operacije sa skip listom

Razmotrimo kako se nad skip listom izvode osnovne operacije: pretraga, umetanje i brisanje.

Svaki element je predstavljen čvorom čiji se nivo bira na slučajan način prilikom umetanja u listu. Čvor nivoa i ima i pokazivača unapred, sa indeksima od 1 do i. Nivoi su ograničeni konstantom $MaxNivo$. *Nivo liste* je maksimalni nivo čvora liste (ili 1 ako je lista prazna). Glava liste (početni element, koji postoji i u praznoj listi) ima pokazivače na nivoima od 1 do $MaxNivo$. Pokazivači unapred glave liste na nivoima većim od trenutno maksimalnog nivoa liste pokazuju na element NIL.

Inicijalizacija skip liste Inicijalizacija skip liste se sastoji od dva koraka:

- alocira se element NIL i daje mu se vrednost veća od svih dozvoljenih vrednosti za ključ (kako bi uvek bio poslednji),
- inicijalizuje se nova lista tako da je nivo liste jednak 1 i svi pokazivači glave liste pokazuju na NIL.

Traženje elementa u skip listi Traženje elementa u skip listi sa ključem jednakim zadatom broju a počinje na najvišem nivou liste. Na tom nivou prate se pokazivači dok se ne naiđe na čvor čija je vrednost ključa veća ili jednaka a. U prethodnom čvoru na tom nivou prelazi se na sledeći „niži“ pokazivač. Postupak se ponavlja na tom i svim nižim nivoima i završava se na osnovnom nivou. Ako se na osnovnom nivou naiđe na čvor sa ključem jednakim a, pretraga je uspešno završena, inače se vraća da se element sa datom vrednošću ključa ne nalazi u listi.

Pod pretpostavkom da svaki član liste ima polja k (ključ), *vrednost* i niz pokazivača *naredni*, a da lista ima član *nivo*, postupak traženja se može opisati narednim kodom.

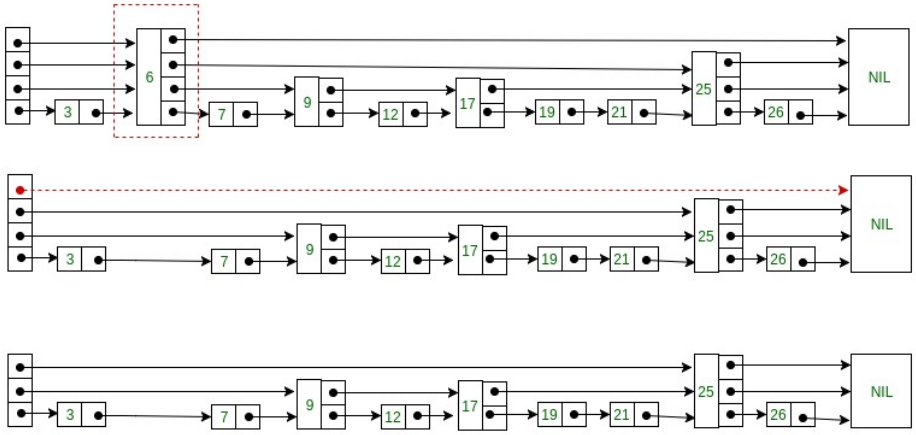
Algoritam Search$(L, kljuc)$;
Ulaz: L (skip lista) i $kljuc$ (ključ elementa koji se traži).
Izlaz: vrednost elementa sa datim ključem ili *neuspeh* ako se element ne nalazi u listi.
begin
    $tekuci := L \rightarrow glava$
    for $i := L \rightarrow nivo$ **downto** 1 **do**
        while $tekuci \rightarrow naredni[i] \rightarrow k < kljuc$ **do**
            $tekuci := tekuci \rightarrow naredni[i]$
    $tekuci := tekuci \rightarrow naredni[1]$
    if $tekuci \rightarrow k = kljuc$ **then return** $tekuci \rightarrow vrednost$
    else return *neuspeh*
end

Na primer, ako se u skip listi prikazanoj na slici 15 traži čvor sa vrednošću ključa 12, polazi se od čvora 6 i nivoa 4, pa se nastavlja sa istim čvorom na nivou 3 i nivou 2, dolazi se do čvora 9 na nivou 2, čvora 9 na nivou 1. Tu se iskače iz petlje, pri čemu je u tom trenutku *tekuci* uperen na čvor 9 na nivou 1. Naredno premeštanje vodi nas u čvor sa traženim ključem 12.

Brisanje elementa iz skip liste Da bismo obrisali čvor iz skip liste, potrebno je da pretražimo skip listu, nađemo dati čvor i da prevežemo pokazivače. Prilikom potrage za elementom koji treba obrisati podaci neophodni za brisanje pamte se u pomoćnom vektoru *pom* dužine jednake broju nivoa liste. Preciznije, $pom[i]$ sadrži pokazivač na najdesniji čvor nivoa i ili višeg koji se nalazi levo od lokacije gde treba izvesti brisanje.

Na primer, ako iz liste sa slike 16 želimo da obrišemo čvor sa ključem 6, prvo pronalazimo taj čvor. Pritom vrednosti $pom[4]$, $pom[3]$ i $pom[2]$ sadrže pokazivač na glavu liste, a $pom[1]$ pokazivač na čvor sa vrednošću 3. Čvor sa ključem 6 briše se tako što se prevezuju pokazivači na nivoima manjim ili jednakim od njegovog nivoa (koji je jednak 4): pokazivači u čvorovima na koje pokazuju elementi niza *pom* dobijaju nove vrednosti, tako da pokazuju na čvorove na koje su pokazivali redom pokazivači istog nivoa obrisanog čvora: u ovom slučaju na NIL, 25, 9 i 7.

Nakon svakog brisanja, vrši se provera da li je obrisani čvor bio jedini čvor sa maksimalnim nivoom u listi; ako jeste (kao što je to slučaj u primeru ilustrovanom na slici 16), smanjuje se maksimalni nivo liste. To možemo uraditi tako što proverimo da li nakon brisanja elementa liste pokazivač glave liste na nekom nivou pokazuje na NIL.



Slika 16: Ilustracija brisanja elementa iz skip liste.

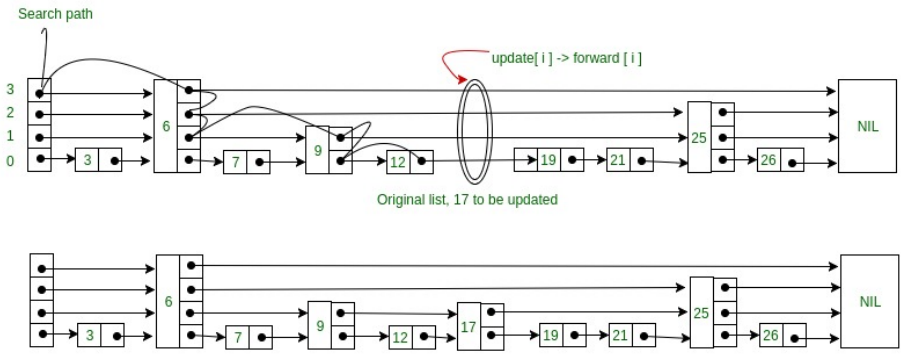
Algoritam Delete$(L, kljuc)$;
Ulaz: L (skip lista) i $kljuc$ (ključ elementa koji se briše).
begin
    $tekuci := L \rightarrow glava$
    for $i := L \rightarrow nivo$ **downto** 1 **do**
        while $tekuci \rightarrow naredni[i] \rightarrow k < kljuc$ **do**
            $tekuci := tekuci \rightarrow naredni[i]$
        $pom[i] := tekuci$
    $tekuci := tekuci \rightarrow naredni[1]$
    if $tekuci \rightarrow k = kljuc$ **then**
        for $i := 1$ **to** $L \rightarrow nivo$ **do**
            if $pom[i] \rightarrow naredni[i] \neq tekuci$ **then break** {premašili smo nivo čvora $tekuci$}
            $pom[i] \rightarrow naredni[i] := tekuci \rightarrow naredni[i]$
        $free(tekuci)$
        while $L \rightarrow nivo > 1$ **and** $L \rightarrow glava \rightarrow naredni[L \rightarrow nivo] =$ **NULL do**
            $L \rightarrow nivo := L \rightarrow nivo - 1$
end

Umetanje elementa u skip listu Da bismo umetnuli čvor u skip listu, kao i kod brisanja, potrebno je da pretražimo listu i da prevežemo pokazivače (slika 17). Za prevezivanje pokazivača nam je opet potreban vektor *pom* koji, kao i u slučaju brisanja elementa iz liste, sadrži pokazivač na najdesniji čvor nivoa i ili višeg koji se nalazi levo od lokacije gde treba izvesti umetanje elementa.

Razmotrimo skip listu ilustrovanu na slici 17 i operaciju umetanja elementa sa ključem 17. U ovom slučaju $pom[4]$ pokazuje na 6, $pom[3]$ na 6, $pom[2]$ na 9, a $pom[1]$ na 12. Nivo novog čvora bira se na slučajan način. Nakon toga vrši se

prevezivanje pokazivača tako da ovi čvorovi sada pokazuju redom na čvorove sa vrednostima NIL, 25, 25 i 19.

Ako u skip listi već postoji element sa datom vrednošću ključa, onda se samo ažurira vrednost elementa sa datim ključem.



Slika 17: Ilustracija dva koraka od kojih se sastoji operacija umetanja vrednosti 17 u datu skip listu.

Ukoliko umetanje generiše čvor većeg nivoa nego što je prethodni nivo liste, ažuriramo vrednost nivoa liste i inicijalizujemo odgovarajuće delove niza *pom*.

Biranje nivoa čvora na slučajan način Kako na slučajan način izabrati nivo novog čvora? Pretpostavimo da procenat p čvorova nivoa i ima pokazivače nivoa $i+1$ (najčešće se bira $p = 1/2$). Da bi se odredio nivo novog čvora koristi se generator slučajnih brojeva, koji na izlazu daje slučajni broj iz intervala $[0, 1)$ sa ravnomernom raspodelom verovatnoća. Svaki put kada se na izlazu iz generatora dobije broj manji od p (što se dešava sa verovatnoćom jednakom upravo p), nivo se povećava za 1, sem ako već nije dostignut maksimalni nivo.

Primetimo da ova funkcija nije konstantne vremenske složenosti. Naime, ovakav način generisanja nivoa, koji prati ideje predstavljene u originalnom radu o skip listama, nekada može biti neefikasan jer podrazumeva potencijalno veći broj poziva funkcije *random*(), doduše dva poziva u proseku.

Može se desiti da u skip listi od 16 elemenata generisanih pomoću vrednosti $p = 1/2$ imamo 9 elemenata nivoa 1, 3 elementa nivoa 2, 3 elementa nivoa 3 i jedan element nivoa 14. Ako bismo pretragu vršili korišćenjem prethodno navedenog algoritma, imali bismo puno praznog hoda. Gde bi bilo pogodnije započeti pretragu? Sprovedena analiza sugeriše da bi idealno bilo započeti pretragu na nivou $L(n)$ na kome očekujemo $1/p = 2$ čvora. Ovo se dešava za nivo $L(n) = \log_{1/p} n = \log_2 n$. Dakle, umesto nivoa liste, u algoritmu ćemo koristiti vrednost $L(n)$.

```
Algoritam Insert(L, kljuc, nova_vrednost);
Ulaz: L (skip lista), kljuc (ključ elementa koji se dodaje) i nova_vrednost (vrednost elementa koji se dodaje).
begin
    tekuci := L → glava
    for i := L → nivo downto 1 do
        while tekuci → naredni[i] → k < kljuc do
            tekuci := tekuci → naredni[i]
        pom[i] := tekuci
    tekuci := tekuci → naredni[1]
    if tekuci → k = kljuc then {već postoji ta vrednost ključa, samo ažuriramo vrednost}
        tekuci → vrednost := nova_vrednost
    else
        nivo := SlučajniNivo()   { na slučajan način bira se nivo novog čvora }
        if nivo > L → nivo then    { ako je nivo novog čvora veći od tekućeg nivoa liste}
            for i := L → nivo + 1 to nivo do
                pom[i] := L → glava
            L → nivo := nivo
        tekuci := makeNode(nivo, kljuc, nova_vrednost)
        for i := 1 to nivo do
            tekuci → naredni[i] := pom[i] → naredni[i]
            pom[i] → naredni[i] := tekuci
end
```

```
Algoritam SlučajniNivo();
Izlaz: nivo čvora.
begin
    nivo := 1
    while random() < p and nivo < MaxNivo do
        nivo := nivo + 1
    return nivo
end
```

Primetimo da ni za jednu od navedenih operacija nije potrebno da u samom čvoru čuvamo informaciju o njegovom nivou.

Složenost osnovnih operacija sa skip listom Vremenima potrebnim za izvršavanje operacija pretrage, umetanja i brisanja dominira vreme potrebno za nalaženje odgovarajućeg elementa u skip listi; kod operacija umetanja i brisanja postoji dodatna cena proporcionalna nivou čvora koji se umeće ili briše. Vreme potrebno za pronalaženje elementa je proporcionalno dužini puta pretrage. Podsetimo se da je očekivani broj nivoa skip liste $O(\log n)$, ali je i dalje potrebno proveriti da je broj koraka na svakom nivou mali.

Pokazuje se da je lakše analizirati dužinu puta pretrage, ako ga posmatramo unazad, počev od čvora sa vrednošću ključa k koji smo tražili, napredujući ka višim nivoima i nalevo.

Označimo sa $C(k)$ očekivanu cenu (tj. dužinu) puta ako smo na k-tom nivou računato odozgo. Tada je $C(0) = 0$, jer se pri traženju polazi od najvišeg nivoa, nivoa 0. Primetimo da se pri pretrazi penjemo naviše uvek kada možemo jer se u čvor uvek dolazi putem njegovog najvišeg nivoa; samo ukoliko ne možemo da se krećemo naviše idemo ulevo. Prilikom procene očekivane dužine puta $C(k)$ pretpostavljamo da je skip lista beskonačne dužine.

- Sa verovatnoćom $p = 1/2$ u čvor gde se sada nalazimo dospeli smo iz istog čvora sa višeg nivoa (ako postoji viši nivo čvora, došli smo iz njega), a tome je prethodio put čija je očekivana dužina $C(k-1)$.
- Sa verovatnoćom $1 - p = 1/2$ u čvor gde se sada nalazimo dospeli smo iz čvora sa istog nivoa, a tome je prethodio put čija je očekivana dužina $C(k)$.

Prema tome, vrednosti $C(k)$ zadovoljavaju diferencnu jednačinu:

$$C(k) = 1/2(1 + C(k)) + 1/2(1 + C(k-1))$$

Sređivanjem ovog izraza dobijamo:

$$C(k) = 2 + C(k-1)$$

odnosno u slučaju kada je $p = 1/2$ očekivani broj koraka na svakom nivou je 2. Daljim raspisivanjem izraza za $C(k)$ dobijamo $C(k) = 2k$. Podsetimo se još jednom da je sa k označen redni broj nivoa sa koga kreće pretraga, računato odozgo.

S obzirom na to da skip lista nije beskonačna, u nekom momentu pri kretanju ulevo naići ćemo na glavu liste iz koje više nisu moguće kretnje ulevo već samo naviše. Ako sa $L(n) = \log_2 n$ označimo očekivani nivo skip liste dužine n, maksimalna vrednost za k je $L(n) - 1$ jer se inicijalno već nalazimo na nivou 1 (na kome smo našli čvor). Dakle, očekivano vreme traženja je $C(k) = O(\log n)$. Povratak na najviši nivo liste nas ne mora nužno direktno dovesti do glave skip liste, ali očekivani broj preostalih koraka je mali (jer se na tom nivou očekuje konstantan broj čvorova) te ne utiče na ukupnu složenost.

Poređenje skip listi i balansiranih uređenih drveta Iako u najgorem slučaju skip liste imaju loše performanse, ni jedan ulaz ne odgovara konzistentno najgorem slučaju (slično algoritmu brzog sortiranja kada se pivot bira na slučajan način). Malo je verovatno da skip lista bude značajno nebalansirana. Dakle, skip lista ima svojstvo balansiranosti nalik uređenim binarnim drvetima izgrađenim slučajnim umetanjima elemenata, ali ne zahteva da umetanja budu slučajna.

Kod skip listi se sa veoma malom verovatnoćom dešavaju degenerisani slučajevi: da postane obična povezana lista ili pak da svaki čvor bude istog maksimalnog nivoa.

Skip liste ne zahtevaju da se uz čvor liste čuvaju informacije o balansiranosti (kao kod balansiranih drveta), ali zato zahtevaju čuvanje dodatnih pokazivača – u proseku dva pokazivača po čvoru.

Za mnoge primene skip liste su prirodnija reprezentacija od drveta i vode jednostavnijim algoritmima. Dosta su jednostavnije i za implementaciju od balansiranih binarnih drveta. Takođe, prilikom operacija umetanja i brisanja sve izmene su lokalne, sa izuzetkom promene maksimalnog nivoa liste, te su pogodnije za paralelizaciju i u situacijama kada je potrebno omogućiti konkurentni pristup strukturi podataka. Npr. u crveno-crnom drvetu, je nakon umetanja potrebno rebalansirati potencijalno čitavo drvo i dok traje ažuriranje ne bi bio moguć pristup drugim elementima strukture, dok bi umetanje u skip listu uticalo samo na susedne čvorove, te bi tokom ovih izmena velikom delu liste bio moguć pristup.

Skip liste, kao i klasične povezane liste, ne mogu da iskoriste lokalnost referenci jer susedni elementi u listi često neće biti u istom regionu u memoriji, što znači da skip lista ne može da iskoristi prednosti keširanja.

Primene skip listi Skip liste se koriste za distribuirane primene, npr. za implementaciju reda sa prioritetom koji dobro radi u višenitnom okruženju jer ne zahtevaju zaključavanje čitave strukture podataka, dok se izvodi pisanje, već samo zaključavanje čvorova susednih čvoru nad kojim radi. One se mogu koristiti i za implementaciju struktura podataka potrebnih za grafovske algoritme, poput algoritama za najkraće puteve. Mogu se koristiti i za indeksiranje u bazama podataka, kao i za sortiranje velikih skupova podataka. One imaju primenu i u geografskim informacionim sistemima za indeksiranje i pretragu geografskih podataka, poput mapa i satelitskih slika. Koriste se i za efikasna statistička izračunavanja, npr. za računanje pokretne medijane (eng. running median).

Zadaci za vežbu

1. Kolika je očekivana složenost operacije pretraživanja elemenata u skip listi, a kolika složenost u najgorem slučaju?
2. Simulirati umetanje elementa sa ključem 20 u skip listu sa slike 15 ako je nivo novog čvora 4. Na koje čvorove pokazuju elementi niza pom?
3. Simulirati brisanje elementa sa ključem 25 iz skip liste sa slike 15. Na koje čvorove pokazuju elementi niza pom?
4. U kakvom su odnosu vrednosti ključeva čvorova na koje pokazuju pokazivači $pom[1]$, $pom[2]$, $pom[3], \ldots$ pre umetanja elementa?
5. Neka skip lista sadrži n elemenata. Dokazati da je očekivani ukupan broj pokazivača $O(n)$.

6. Koliko je unapređenje vremenske složenosti umetanja i brisanja postignuto prelaskom sa jednostruko povezane liste na skip listu?

   a. sa $O(n)$ na $O(\log n)$

   b. sa $O(n)$ na $O(1)$

   c. sa $O(n)$ na $O(n^2)$

   d. nije postignuto nikakvo unapređenje

7. Pretpostavimo da imamo dve skip liste na raspolaganju: skip listu A koja sadrži m elemenata i skip listu B koja sadrži n elemenata. Opisati algoritam za spajanje ove dve liste u jedinstvenu skip listu koja sadrži $m+n$ elemenata. Ne pretpostavljati da su svi ključevi jedne liste manji od svih ključeva druge liste. Očekivana vremenska složenost algoritma treba da bude $O(n+m)$.