Stabla pretrage

© Goodrich, Tamassia, Goldwasser

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2019.

Mape sa poretkom

- postoji relacija poretka nad ključevima
- elementi se skladište prema vrednosti ključa
- pretrage „najbliži sused“ (nearest neighbor):
  - nađi element sa najvećim ključem manjim ili jednakim k
  - nađi element sa najmanjim ključem većim ili jednakim k

Binarna pretraga

- binarna pretraga može da pronađe „najbližeg suseda“ za mapu sa poretkom implementiranu pomoću niza koji je sortiran po ključu
  - u svakom koraku prepolovi se broj kandidata
  - radi u $O(\log n)$ vremenu
- primer: nađi 7

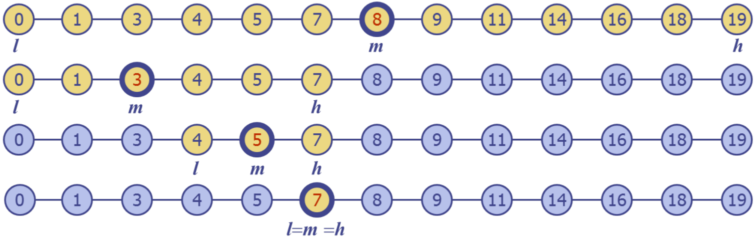


Tabela pretrage

- tabela pretrage je mapa sa poretkom implementirana pomoću sortiranog niza
  - eksterni komparator za ključeve
- performanse:
  - binarna pretraga je $O(\log n)$
  - dodavanje je $O(n)$
  - uklanjanje je $O(n)$
- radi efikasno samo za mali broj elemenata ili tamo gde je pretraga česta a izmene retke (npr. provera kreditne kartice)

Sortirana mapa ATP

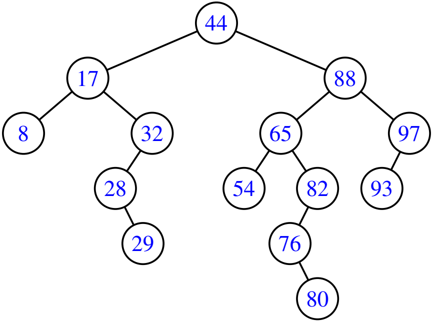
- standardne operacije mape

`M[k]`	vraća vrednost v za ključ k u mapi M; implementira je `__getitem__`
`M[k]=v`	dodaje novi element (k, v) u M ili menja postojeći; implementira je `__setitem__`
`del M[k]`	uklanja element sa ključem k iz M; implementira je `__delitem__`

- dodatne funkcionalnosti
  - sortiran redosled prilikom iteracije
  - nađi veće: find_gt(k)
  - nađi u opsegu: find_range($start, stop$)

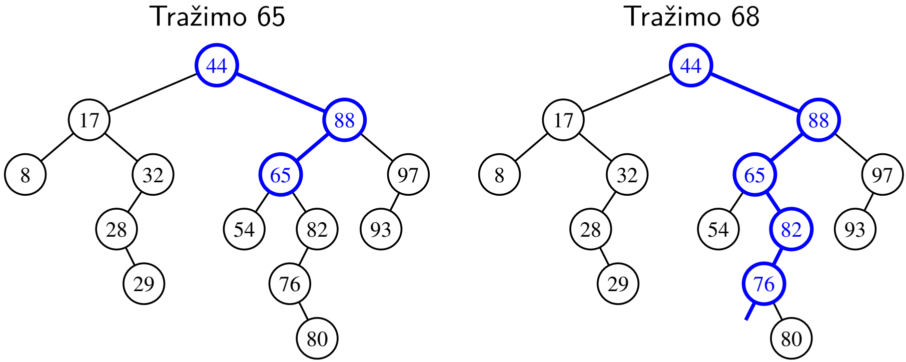
Binarno stablo pretrage

- binarno stablo pretrage je binarno stablo koje čuva (k, v) parove u čvorovima p tako da važi:
  - ključevi koji se nalaze u **levom** podstablu od p su **manji** od k
  - ključevi koji se nalaze u **desnom** podstablu od p su **veći** od k
- listovi ne čuvaju elemente, reference na listove mogu biti None
- inorder obilazak: ključevi u rastućem redosledu



Pretraga u binarnom stablu

- tražimo ključ k polazeći od korena
- idemo levo ako je k manji od tekućeg čvora
- idemo desno ako je k veći od tekućeg čvora
- ako dođemo do lista, k nije nađen

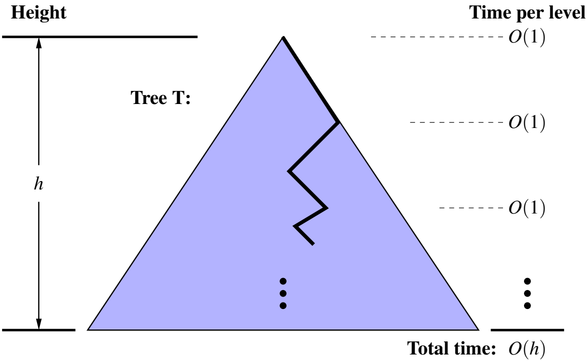


Pretraga u binarnom stablu

TreeSearch(T, p, k)
  if $k = p.key$ **then**
    return p {pronađen}
  else if $k < p.key \wedge T.left(p) \neq None$ **then**
    return TreeSearch($T, T.left(p), k$) {levo podstablo}
  else if $k > p.key \wedge T.right(p) \neq None$ **then**
    return TreeSearch($T, T.right(p), k$) {desno podstablo}
  return None {nije pronađen}

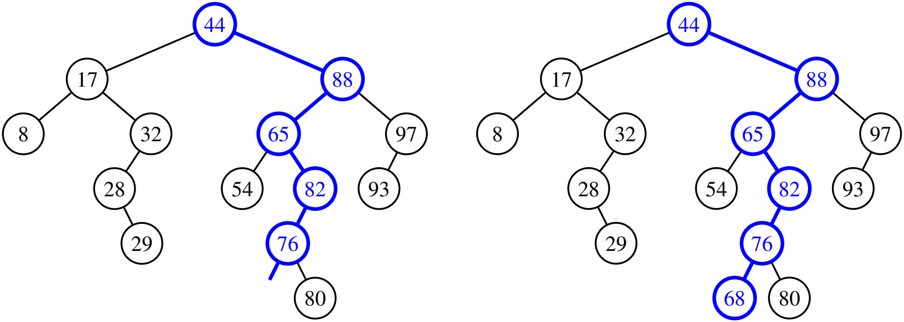
Performanse pretrage u binarnom stablu

- u svakom rekurzivnom pozivu spuštamo se za jedan nivo u stablu
- testiranje u okviru jednog nivoa je $O(1)$
- ukupan broj testova je $O(h)$, gde je h visina stabla



Dodavanje u stablo

- dodajemo element (k, v)
- prvo tražimo k
- ako k nije u stablu, došli smo do lista gde treba dodati čvor
- primer: dodajemo 68



Dodavanje u stablo

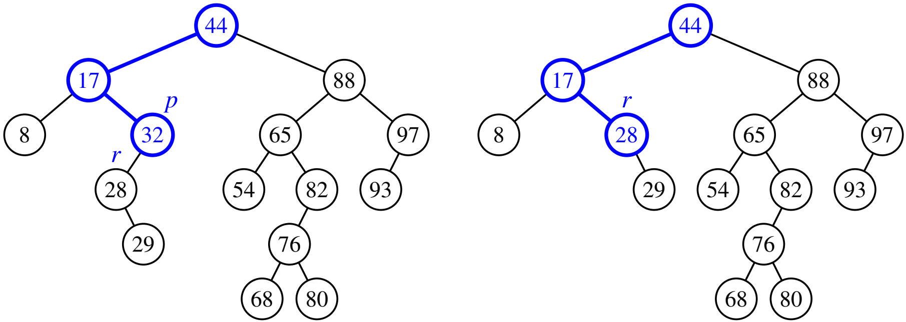
TreeInsert(T, k, v)

$p \leftarrow$ TreeSearch$(T, T.root, k)$
if $k = p.key$ **then**
  $p.value \leftarrow v$ {ako već postoji zameni vrednost}
else if $k < p.key$ **then**
  p.add_left(k, v) {dodaj levo dete}
else
  p.add_right(k, v) {dodaj desno dete}

- dodaje se uvek u list

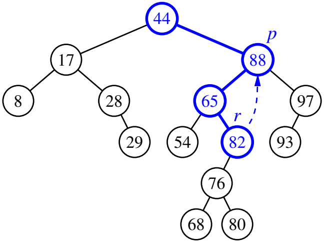
Uklanjanje iz stabla

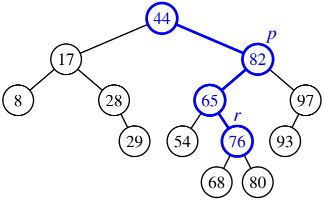
- uklanjamo element sa ključem k
- prvo nađemo p koji sadrži k
- ako p ima **najviše jedno** dete
- njegovo dete r vežemo u stablo umesto njega
- primer: uklanjamo 32



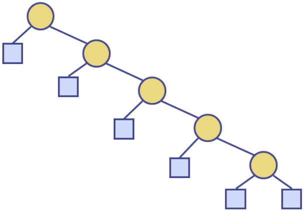
Uklanjanje iz stabla

- ako p ima **dva** deteta
  - nađemo čvor r čiji ključ neposredno prethodi p – to je „najdesniji" čvor u njegovom levom podstablu
  - vežemo r na mesto p; pošto r neposredno prethodi p po vrednosti ključa, svi elementi u desnom podstablu od p su veći od r i svi elementi u levom podstablu od p su manji od r
  - treba još obrisati stari r – pošto je to „najdesniji" element, on nema desno dete, pa se može obrisati po prethodnom algoritmu
- primer: uklanjamo 88

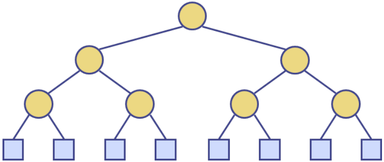




Performanse binarnog stabla pretrage



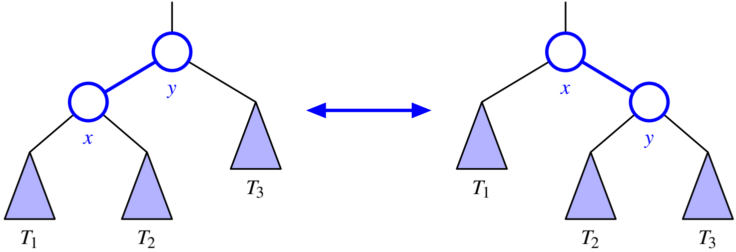
- zauzeće memorije je $O(n)$
- pretraga, dodavanje i uklanjanje su $O(h)$
- visina stabla h je $O(\log n) \leq h \leq O(n)$



balansirano stablo ima bolje performanse

Balansiranje binarnog stabla

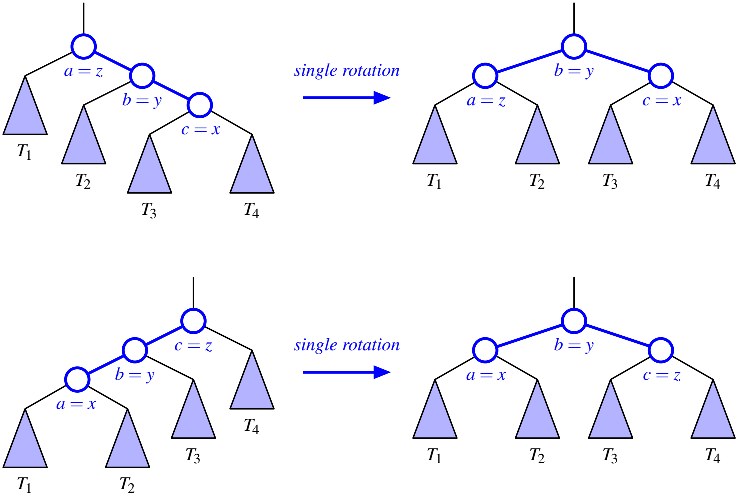
- osnovna operacija za balansiranje je rotacija
- „rotiramo" dete i njegovog roditelja
- tom prilikom i podstabla menjaju mesta
- jedna rotacija traje $O(1)$



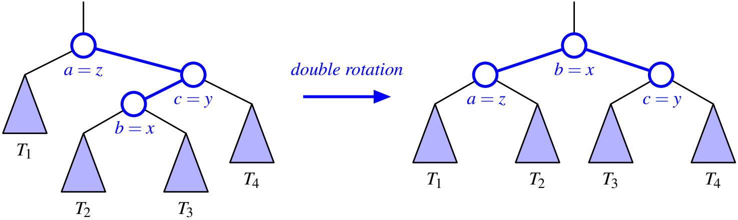
Balansiranje binarnog stabla

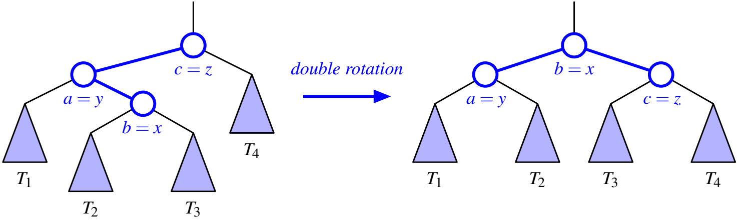
- kompozitna operacija „restrukturiranje tri čvora“ (tri-node restructuring)
- posmatraju se čvor, njegovo dete i unuče
- cilj je da se skrati putanja od čvora do unučeta
- četiri moguća rasporeda čvorova
  - prva dva traže jednu rotaaciju
  - druga dva traže dve rotacije

Restrukturiranje sa jednom rotacijom



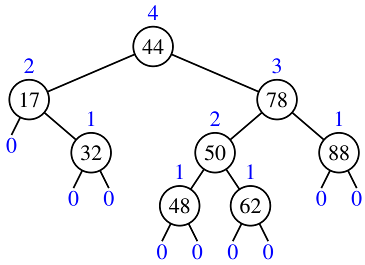
Restrukturiranje sa dve rotacije





AVL stablo

- autori: G.M. Adelson-Velskii i E. Landis
- visina podstabla: broj čvorova na najdužoj putanji od korena do lista
- visina čvora = visina podstabla sa njim kao korenom
- AVL stablo je binarno stablo koje ima dodatnu osobinu:
  - za svaki čvor u stablu, visine njegove dece razlikuju se najviše za 1



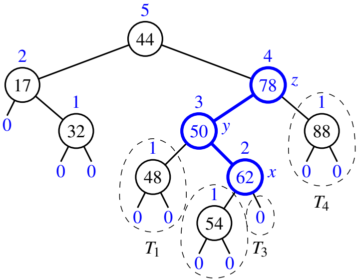
Visina AVL stabla

- **teorema**: visina AVL stabla sa n čvorova je $O(\log n)$
- **dokaz**: $n(h)$ – najmanji broj unutrašnjih čvorova u AVL stablu visine h
- očevidno: $n(1) = 1$, $n(2) = 2$
- za $n > 2$, AVL stablo visine h sadrži koren, podstablo visine $h - 1$ i podstablo visine $h - 2$
- tj. $n(h) = 1 + n(h-1) + n(h-2)$
- kako je $n(h-1) > n(h-2)$, važi $n(h) > 2n(h-2)$
    - indukcijom: $n(h) > 2n(h-2)$, $n(h) > 4n(h-4)$, $n(h) > 8(h-6)$, ...
    - $n(h) > 2^i n(h-2i)$
- bazni slučaj: $n(h) > 2^{h/2-1}$
- odnosno: $h < 2 \log n(h) + 2$
- tj. visina AVL stabla h je $O(\log n)$

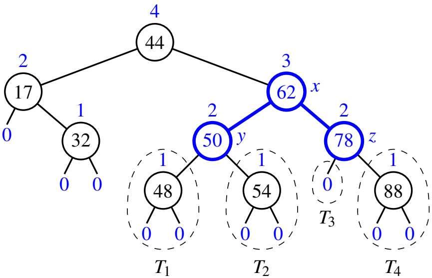
AVL stablo: dodavanje

- stablo u koje dodajemo novi čvor je AVL stablo
- dodavanje se vrši isto kao kod binarnog stabla – u list
- dodavanje može da naruši balansiranost
- čvorovi koji mogu biti disbalansirani su samo preci novog čvora
- primer: dodajemo čvor 54

pre balansiranja



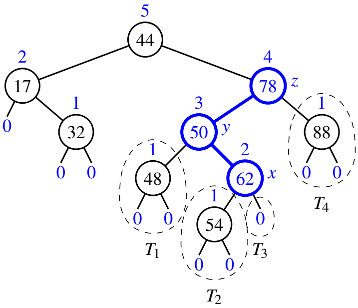
posle balansiranja



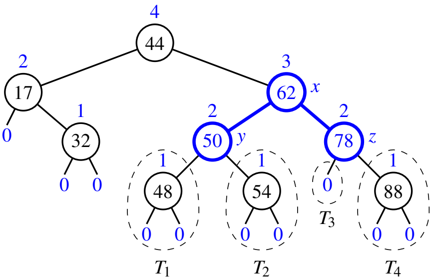
AVL stablo: dodavanje

- „search-and-repair“ strategija
- z – prvi nebalansirani čvor na polazeći od p koji smo naišli
- radimo **trinode restructuring** za z

pre balansiranja

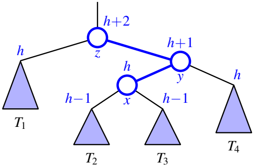


posle balansiranja

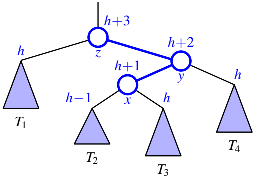


AVL stablo: dodavanje

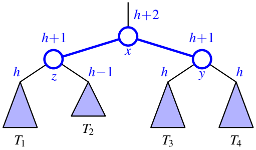
pre dodavanja:



dodavanje u T_3 remeti balans u z:



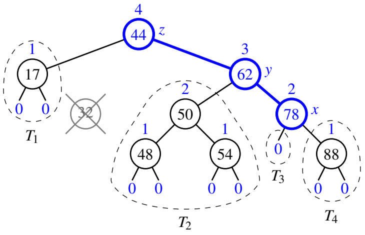
nakon restrukturiranja:



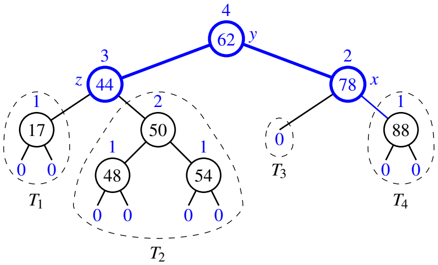
AVL stablo: uklanjanje

- uklanjanjem se može narušiti balans AVL stabla
- i ovde radimo restrukturiranje posle uklanjanja
- primer: uklanjamo 32

pre balansiranja, koren nije balansiran



posle balansiranja (jedna rotacija)



AVL stablo: performanse

- jedno restrukturiranje je $O(1)$
- pretraga je $O(\log n)$ – visina stabla je $O(\log n)$
- dodavanje je $O(\log n)$
  - pronalaženje mesta je $O(\log n)$
  - restrukturiranje uz stablo je $O(\log n)$
- uklanjanje je $O(\log n)$
  - pronalaženje mesta je $O(\log n)$
  - restrukturiranje uz stablo je $O(\log n)$

Splay stablo

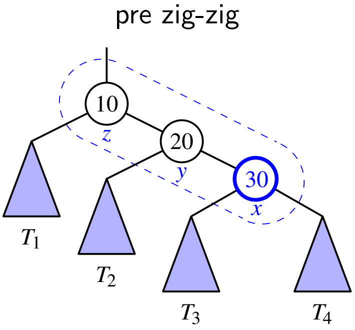
- splay: „rašireno“
- ne nameće logaritamsko ograničenje na visinu
- splaying: „širenje“ stabla prilikom dodavanja, uklanjanja **i pretrage**
- ideja: da češće korišćeni elementi budu bliže korenu

Splaying

- čvor x se premešta u koren nizom restrukturiranja
- operacije restrukturiranja zavise od položaja x, y (roditelja) i z (dede, ako postoji)
- postoje tri slučaja:
  - zig-zig
  - zig-zag
  - zig

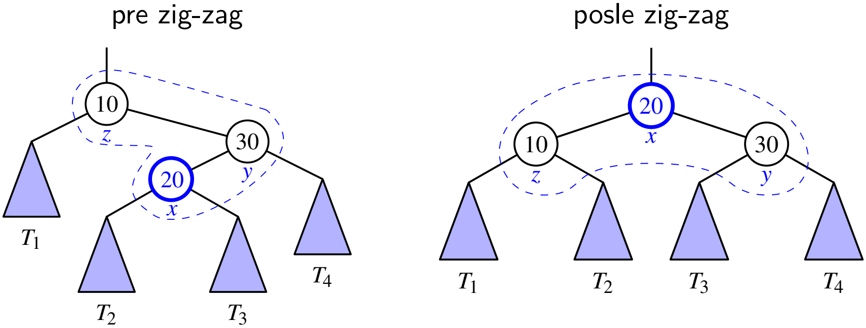
Splaying: zig-zig

- x i y su
  - obojica levo dete svog roditelja ili
  - obojica desno dete svog roditelja
- x postaje koren, y njegovo dete, z njegovo unuče



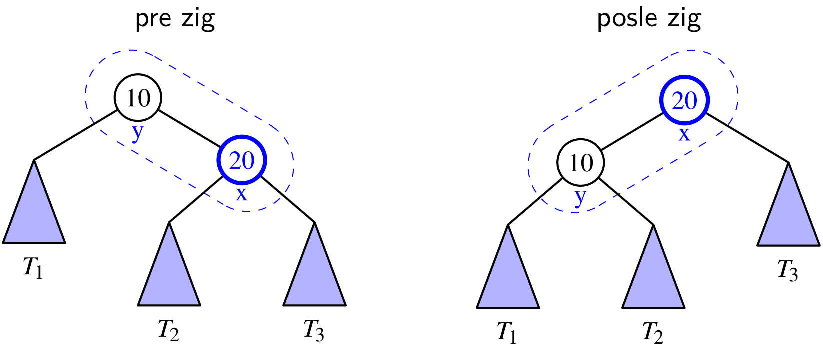
Splaying: zig-zag

- x i y
  - prvi je levo dete a drugi je desno dete, ili
  - prvi je desno dete a drugi je levo dete
- x postaje koren, y i z njegova deca



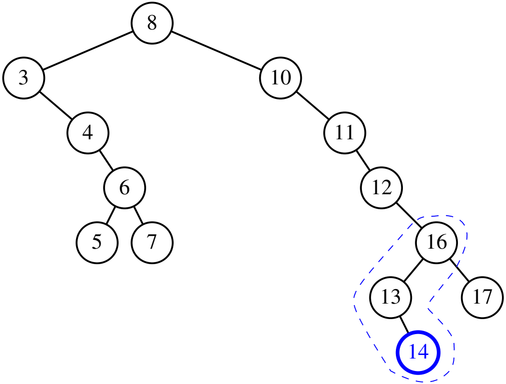
Splaying: zig

- x ima roditelja y ali nema dedu z :(
- x postaje koren, y njegovo dete



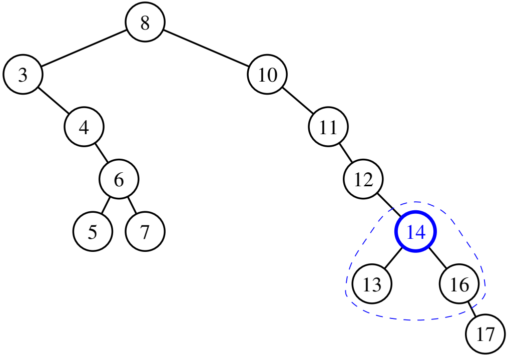
Splaying: primer $_1$

- zig-zig, zig-zag i zig primenjujemo sve dok x ne postane koren
- primer: dodajemo 14
- na 14 se primenjuje zig-zag (jer 14 je desno dete a 13 je levo dete)



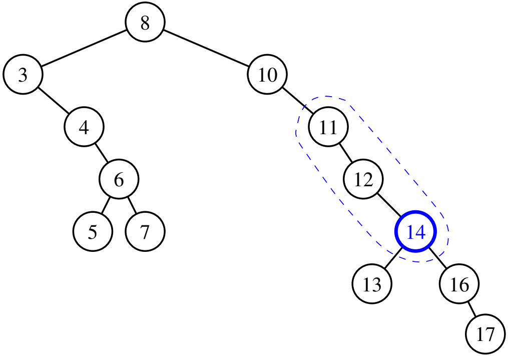
Splaying: primer $_2$

- posle primenjenog zig-zag stanje je



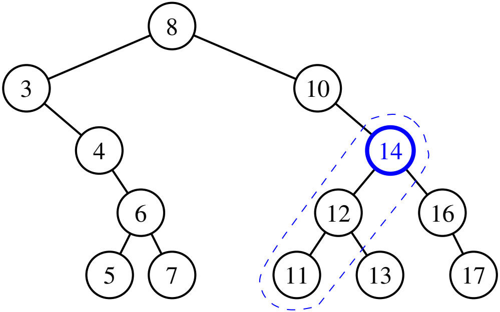
Splaying: primer $_3$

- sada može da se primeni zig-zig



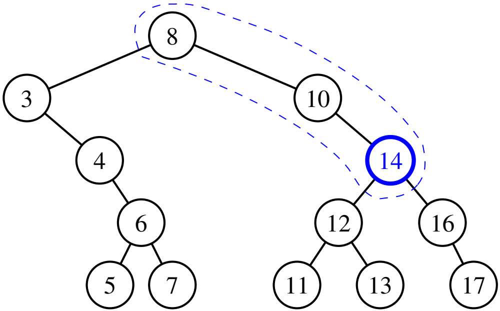
Splaying: primer $_4$

- nakon primene zig-zig



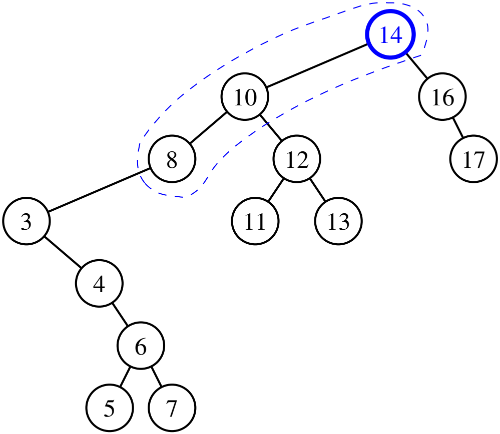
Splaying: primer $_5$

- sada može ponovo zig-zig



Splaying: primer $_6$

- nakon drugog zig-zig



Splay stablo: performanse

- zig-zig, zig-zag i zig su $O(1)$
- splaying čvora p je $O(d)$ gde je d dubina čvora p
- tj. isto koliko je potrebno i za navigaciju od korena do p

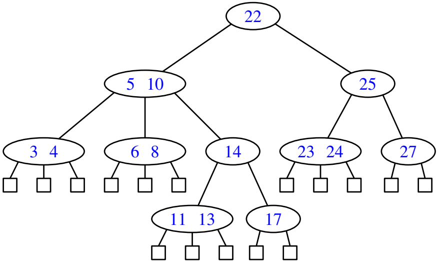
- u najgorem slučaju, pretraga, dodavanje i uklanjanje su $O(h)$ gde je h visina stabla
- stablo nije balansirano $\Rightarrow$ može biti $h = n$
- $\Rightarrow$ slabe performanse u najgorem slučaju

Splay stablo: performanse

- za **amortizovane** operacije vreme je $O(\log n)$
- a za često tražene podatke pretraga je i **brža od** $O(\log n)$

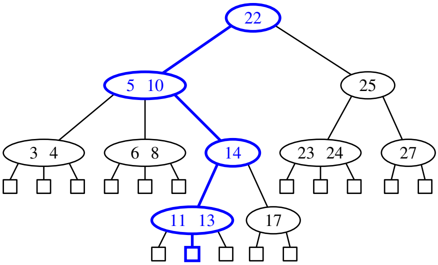
n-arno stablo

- neka je w čvor stabla; ako w ima d dece zovemo ga d-čvor
- n-arno stablo pretrage ima sledeće osobine:
  - svaki unutrašnji čvor ima bar dva deteta, tj. svaki je d-čvor za $d \geq 2$
  - svaki unutrašnji d-čvor sa decom $c_1, c_2, \ldots, c_d$ čuva $d - 1$ parova $(k_1, v_1), \ldots, (k_{d-1}, v_{d-1})$
  - za $k_0 = -\infty$, $k_d = +\infty$ važi: za svaki element (k, v) iz podstabla od w kome je koren c_i važi da je $k_{i-1} \leq k \leq k_i$



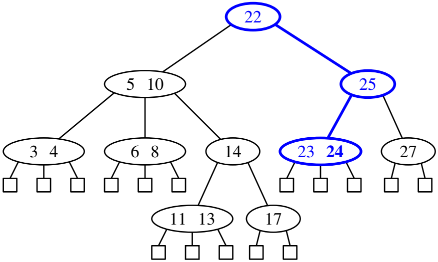
Pretraga u n-arnom stablu

- tražimo ključ k polazeći od korena
- u čvoru w poredimo k sa ključevima $k_1, \dots, k_{d-1}$
  - ako je $k = k_i$ za neko $1 \leq i \leq d-1$ pronašli smo ključ
  - inače nastavljamo pretragu od deteta c_i tako da je $k_{i-1} < k < k_i$
- ako smo došli do lista pretraga je neuspešna
- primer: tražimo $k = 12$ (neuspešna)



Pretraga u n-arnom stablu

- primer: tražimo $k = 24$ (uspešna)

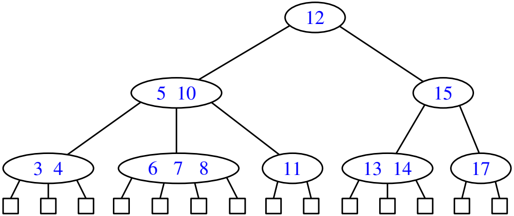


Pretraga u n-arnom stablu

- pretraga unutar čvora?
- treba nam **sekundarna struktura podataka**
  - binarna pretraga po nizu je $O(\log d)$
  - sortirana mapa
- pretraga u stablu je $O(h \log d_{\max})$

(2,4) stablo

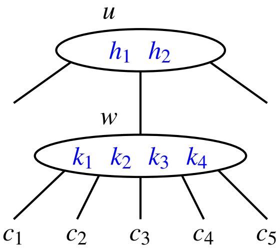
- (2,4) stablo je n-arno stablo sa dve osobine
  - unutrašnji čvor ima najviše 4 deteta
  - svi listovi imaju istu dubinu



- svaki čvor ima 2, 3 ili 4 deteta
- visina stabla od n elemenata je $O(\log n)$

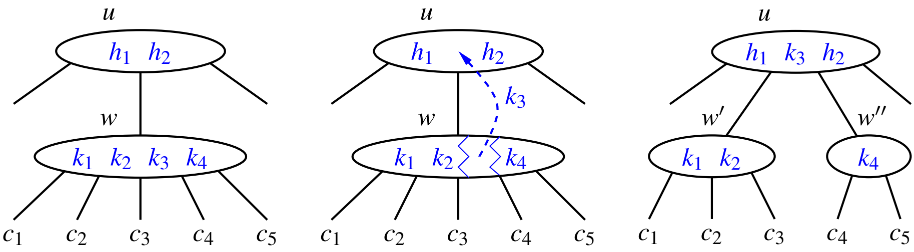
(2,4) stablo: dodavanje

- prvo tražimo ključ k
- neuspešna pretraga se završava u listu
- dodamo k u roditelja w tog lista
- (prelivanje, **overflow**): ako je taj roditelj bio 4-čvor, sada je 5-čvor; moramo ga podeliti (**split**):



(2,4) stablo: dodavanje

- podela čvora w prilikom prelivanja na w' i w''
- w' je 3-čvor sa decom c_1, c_2, c_3 i ključevima k_1, k_2
- w'' je 2-čvor sa decom c_4, c_5 i ključem k_4
- ključ k_3 se penje u roditelja od w; ako je w koren, napravi novi čvor



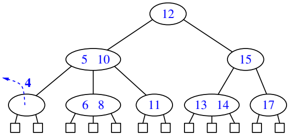
podela može biti kaskadna!

(2,4) stablo: uklanjanje

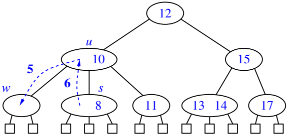
- prvi slučaj: uklanjanjem ključa ne narušavaju se osobine (2,4) stabla
- primer: uklanjamo 13

(2,4) stablo: uklanjanje

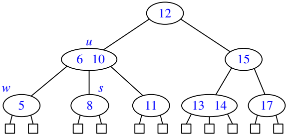
- drugi slučaj: uklanjanje iz w izaziva **underflow**
- da li je jedan od najbliže braće 3-čvor ili 4-čvor?
- radimo **transfer**:
  - premeštamo ključ iz brata u roditelja
  - ključ iz roditelja u w
- primer: uklanjamo 4



uklanjamo 4



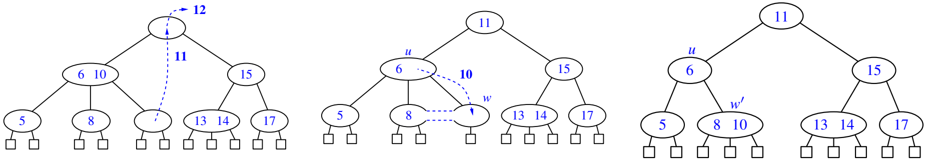
transfer



rezultat

(2,4) stablo: uklanjanje

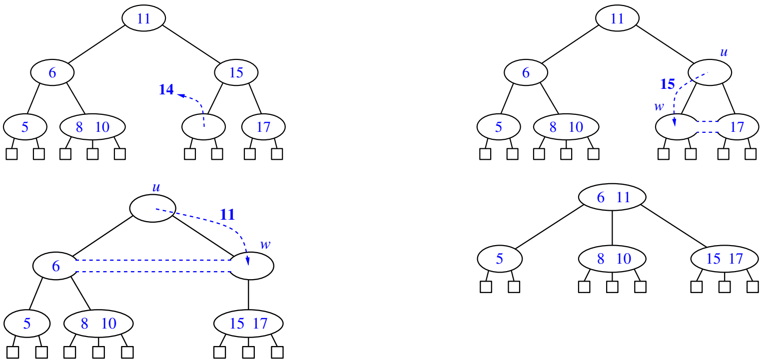
- treći slučaj: uklanjanje izaziva **underflow**
- nijedan od najbliže braće nije 3-čvor ili 4-čvor
- radimo **fuziju**:
  - spajamo w sa bratom
  - ključ iz roditelja spuštamo u spojeni čvor
- primer: uklanjamo 12



uklanjamo 12 | fuzija | rezultat

(2,4) stablo: uklanjanje

- fuzija može da propagira underflow
- ako se koren isprazni fuzijom, prosto se obriše



(2,4) stablo: performanse

- **dodavanje** u (2,4) stablu sa n elemenata
  - visina stabla je $O(\log n)$
  - traženje ključa je $O(\log n)$
  - dodavanje novog ključa u čvor je $O(1)$
  - svaka podela čvora je $O(1)$
  - ukupan broj podela je $O(\log n)$
- $\Rightarrow$ dodavanje je $O(\log n)$

(2,4) stablo: performanse

- **uklanjanje** u (2,4) stablu sa n elemenata
  - visina stabla je $O(\log n)$
  - traženje ključa je $O(\log n)$
  - uklanjanje ključa je $O(1)$
  - može da usledi $O(\log n)$ iza kojih je max 1 transfer
  - fuzija i transfer su $O(1)$
- $\Rightarrow$ uklanjanje je $O(\log n)$

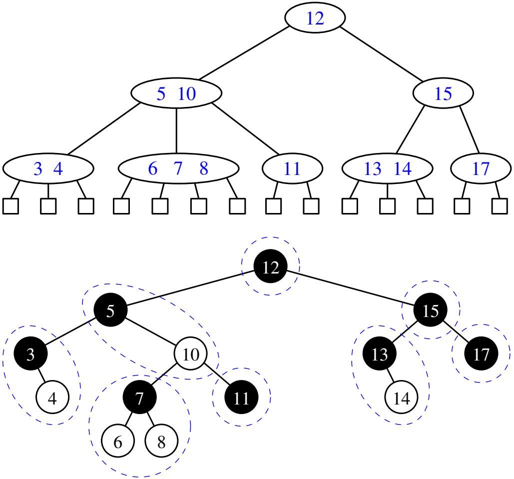
Različite implementacije mape

	search	**insert**	**delete**	**napomene**
hash tabela	1 očekivano	1 očekivano	1 očekivano	nema metode za sortiranu mapu; jednostavna implementacija
skip lista	$\log n$ verovatno	$\log n$ verovatno	$\log n$ verovatno	randomized insert; jednostavna implementacija
AVL i (2,4)	$\log n$ najgore	$\log n$ najgore	$\log n$ najgore	komplikovana implementacija

Crveno-crno stablo

- **red-black** stablo je reprezentacija (2,4) stabla pomoću binarnog stabla čiji čvorovi su obojeni **crveno** ili **crno**
- u poređenju sa (2,4) stablom, RB stablo ima
  - jednake $O(\log n)$ performanse
  - jednostavniju implementaciju

(2,4) stablo i crveno-crno stablo



Crveno-crno stablo: definicija

- crveno-crno stablo je binarno stablo pretrage sa osobinama
  - koren je **crn**
  - deca **crvenog** čvora su **crna**
    - deca **crnog** čvora ne moraju biti **crvena**!
  - sve putanje od korena do lista sadrže isti broj **crnih** čvorova
    - **crna dubina** čvora: broj crnih predaka (računajući i dati čvor)

Crveno-crno stablo: posledice

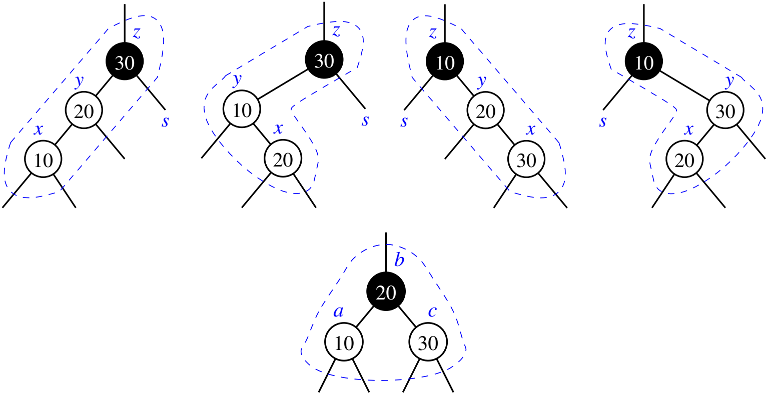
- putanja od korena do najdaljeg lista nije više od duplo duža od putanje do najbližeg lista
- RB-stablo je *prilično dobro* balansirano
- visina RB stabla sa n elemenata je $O(\log n)$
- pretraga na isti način kao za binarno stablo
- pretraga je takođe $O(\log n)$

Crveno-crno stablo: dodavanje

- dodavanje na isti način kao za binarno stablo
- novi čvor se boji u crveno ako nije koren; koren je uvek crn
- ako je roditelj novog čvora crven tada imamo duplo crveno – potrebna je reorganizacija stabla

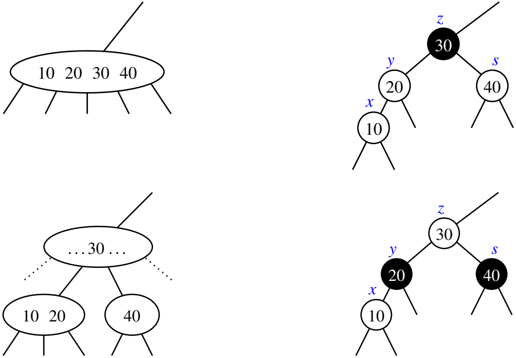
[add] Korekcija duplog crvenog: stric je crn

- slučaj 1: brat roditelja novog čvora je crn ili ne postoji
- radimo tri-node restructuring
- četiri moguće situacije, sa istim rezultatom:



[add] Korekcija duplog crvenog: stric je crven

- slučaj 2: brat roditelja novog čvora je crven
- radimo **recoloring**:
  - obojimo roditelja i strica u crno a dedu u crveno (ako je deda koren, ostaje crn)
- moguća propagacija uz stablo

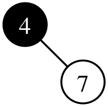


[add] Primer: korak 1

- stablo je na početku prazno
- dodajemo 4
- on će biti koren
- bojimo ga u crno

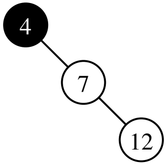
[add] Primer: korak 2

- dodajemo 7 kao desno dete od 4
- bojimo ga u crveno
- nije potrebno restrukturiranje



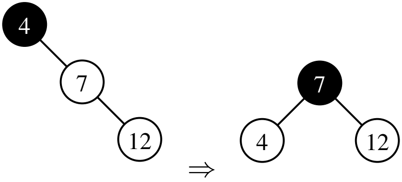
[add] Primer: korak 3

- dodajemo 12 kao desno dete od 7
- bojimo ga u crveno
- imamo duplo crveno



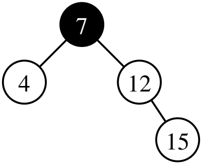
[add] Primer: korak 3a

- 12 nema strica: slučaj 1
- radimo tri-node restructuring



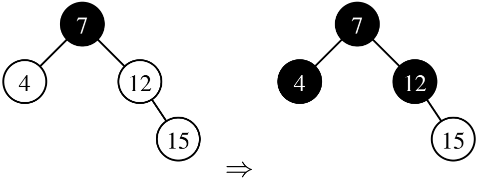
[add] Primer: korak 4

- dodajemo 15 kao desno dete od 12
- bojimo ga u crveno
- imamo duplo crveno



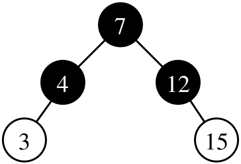
[add] Primer: korak 4a

- 15 ima crvenog strica: slučaj 2
- radimo recoloring: obojimo oca i strica u crno a dedu u crveno osim ako je koren



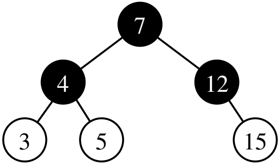
[add] Primer: korak 5

- dodajemo 3 kao levo dete od 4
- bojimo ga u crveno
- sve je u redu



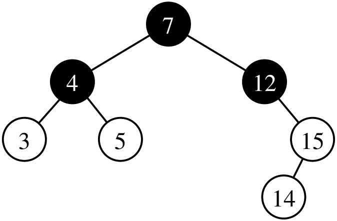
[add] Primer: korak 6

- dodajemo 5 kao desno dete od 4
- bojimo ga u crveno
- sve je u redu



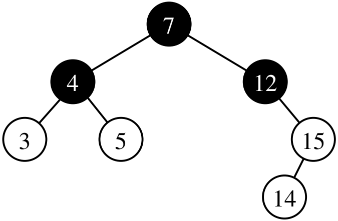
[add] Primer: korak 7

- dodajemo 14 kao levo dete od 15
- bojimo ga u crveno
- imamo duplo crveno

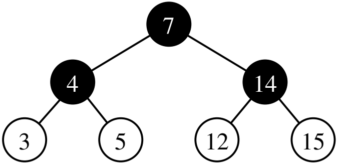


[add] Primer: korak 7a

- 14 nema strica: slučaj 1
- radimo tri-node restructuring za 12-15-14

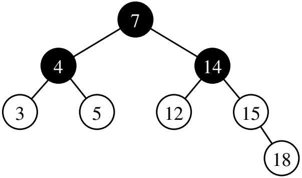


⇒



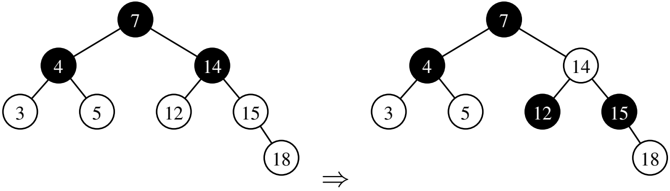
[add] Primer: korak 8

- dodajemo 18 kao desno dete od 15
- bojimo ga u crveno
- imamo duplo crveno



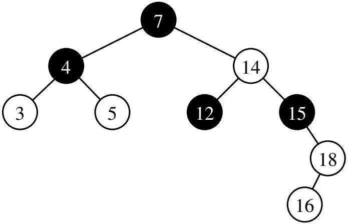
[add] Primer: korak 8a

- 18 ima crvenog strica: slučaj 2
- radimo recoloring: oca i strica u crno a dedu u crveno



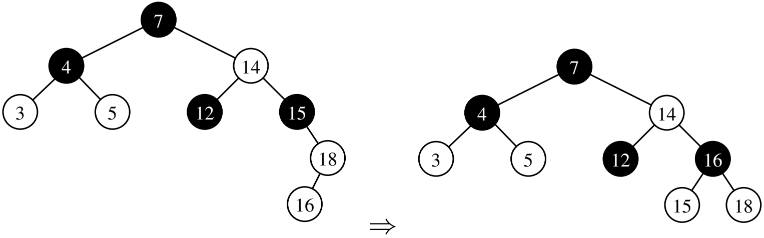
[add] Primer: korak 9

- dodajemo 16 kao levo dete od 18
- bojimo ga u crveno
- imamo duplo crveno



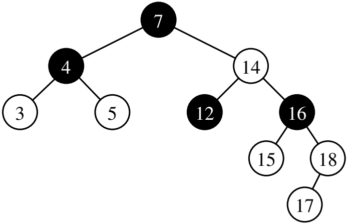
[add] Primer: korak 9a

- 16 nema strica: slučaj 1
- radimo tri-node restructuring za 15-18-16



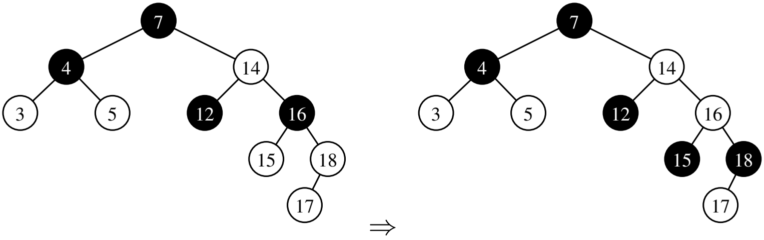
[add] Primer: korak 10

- dodajemo 17 kao levo dete od 18
- bojimo ga u crveno
- imamo duplo crveno



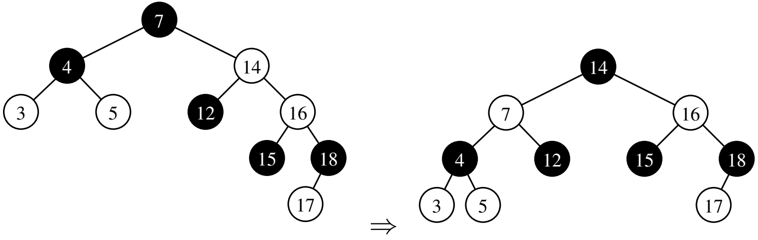
[add] Primer: korak 10a

- 17 ima crvenog strica: slučaj 2
- radimo recoloring: oca i strica u crno a dedu u crveno



[add] Primer: korak 10b

- sada je deda (16) u problemu - imamo duplo crveno
- 16 ima crnog strica: slučaj 1
- radimo tri-node restructuring za 7-14-16

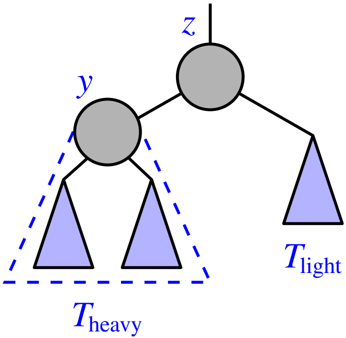


Crveno-crno stablo: uklanjanje

- uklanjanje na isti način kao za binarno stablo
- $\Rightarrow$ uklanja se čvor koji ima najviše jedno dete
- ako je uklonjeni čvor bio crven sve je OK (ne menja se crna dubina)
- ako je uklonjeni čvor bio crn:
    - ako je uklonjeni čvor list - njegov brat je koren podstabla sa crnom dubinom 1
    - ako mu je dete crveno: pomeramo dete na mesto uklonjenog roditelja i bojimo crno
    - ako mu je dete crno: gubimo jedan crni čvor na putanji, narušava se crna dubina u tom podstablu

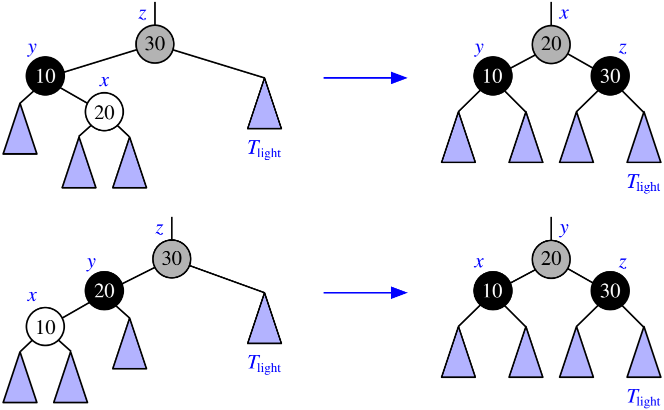
Korekcija crne dubine

- uklonjeni crni čvor je bio koren podstabla T_{light}
- gledamo šta se nalazi u podstablu njegovog brata y
- (brat mora da postoji jer bi inače bila narušena crna dubina)



Korekcija crne dubine 1: brat je crn i ima crveno dete

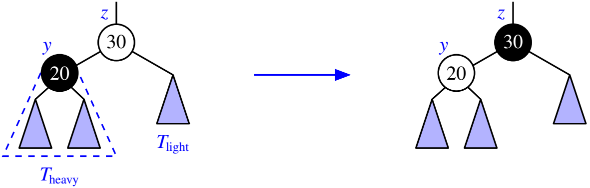
- **slučaj 1**: brat y je crn i ima crveno dete x
- radimo **tri-node restructuring**

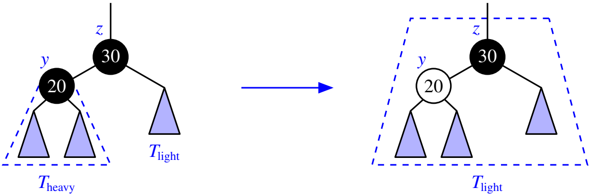


(druga dva slučaja su simetrična)

Korekcija crne dubine 2: brat je crn i ima dva crna deteta

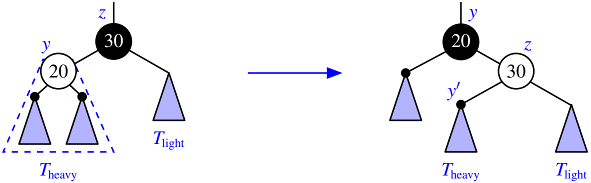
- **slučaj 2**: brat y je crn i oba deteta su mu crna ili ih nema
- radimo **recoloring**
- ako je z bio crven, tu je kraj; ako je bio crn, recoloring propagira prema gore





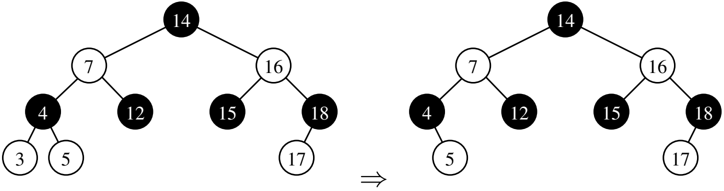
Korekcija crne dubine 3: brat je crven

- **slučaj 3**: brat y je crven
- radimo **rotaciju** pa **recoloring**
- čvor y' je crn pa treba primeniti slučaj 1 ili slučaj 2 na njega



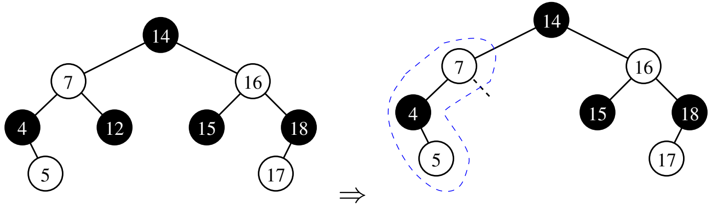
[remove] Primer: korak 1

- uklanjamo 3
- 3 je crven: ne menja se crna dubina



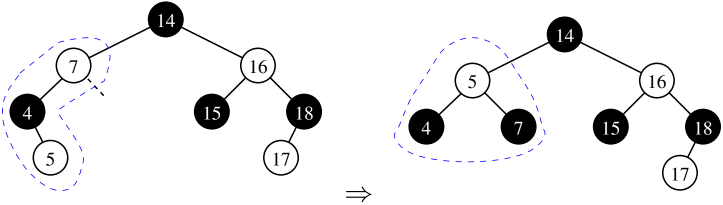
[remove] Primer: korak 2

- uklanjamo 12 - to je crni list
- njegov brat je crn i ima crveno dete
- slučaj 1: radimo tri-node restructuring



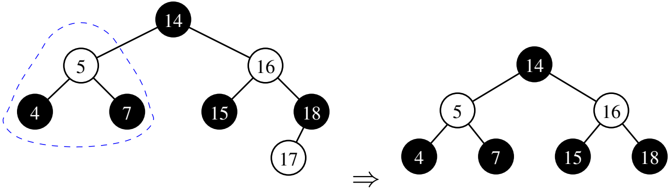
[remove] Primer: korak 2a

- slučaj 1: radimo tri-node restructuring za 7-4-5



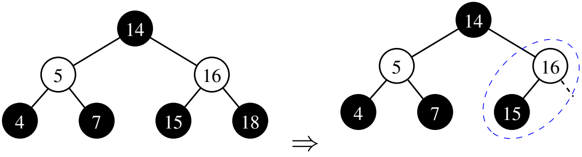
[remove] Primer: korak 3

- uklanjamo 17 - to je crveni list



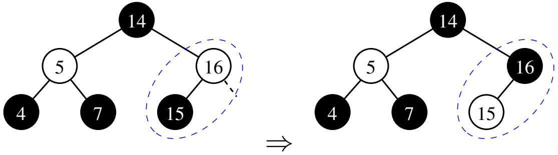
[remove] Primer: korak 4

- uklanjamo 18 - crni list
- njegov crni brat nema dece
- slučaj 2: recoloring - menjamo boju brata i roditelja



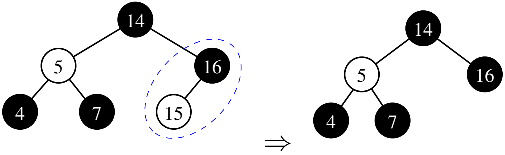
[remove] Primer: korak 4a

- slučaj 2: recoloring - menjamo boju brata i roditelja



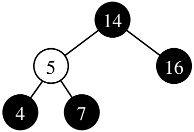
[remove] Primer: korak 5

- uklanjamo 15 - crveni list



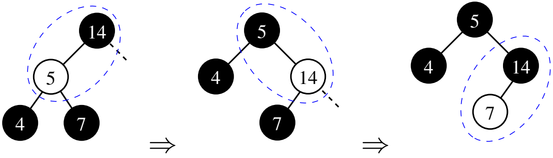
[remove] Primer: korak 6

- uklanjamo 16 - crni list
- njegov brat 5 je crven: slučaj 3



[remove] Primer: korak 6a

- slučaj 3: radimo rotaciju pa recoloring



Crveno-crno stablo: performanse

- jednake kao i za (2,4) stablo
- sve operacije su $O(\log n)$
- dodavanje i uklanjanje zahtevaju konstantan broj operacija restrukturiranja

- odličan interaktivni demo za RB stabla: `http://gauss.ececs.uc.edu/RedBlack/redblack.html`