Fakultet elektrotehnike i računarstva
Zavod za primjenjeno računarstvo

Napredni algoritmi i strukture podataka

1. laboratorijska vježba

Josip Milić
0036456339

Zagreb, 02.11.2015.

1. Zadatak

Zadatci za 17 bodova
Napisati program koji učitava ili briše niz prirodnih brojeva iz ASCII datoteke (po pretpostavci, datoteka nije prazna) i upisuje ili briše ih u (inicijalno prazno) **crveno-crno (RB) stablo** istim redoslijedom kao u datoteci. Program može biti konzolni ili s grafičkim sučeljem, po vlastitom izboru. Konzolni program naziv ulazne datoteke treba primiti prilikom pokretanja kao (jedini) argument s komandne linije, a grafički iz odgovarajućeg sučelja po pokretanju programa. Nakon upisa svih podataka, ispisati izgrađeno stablo na standardni izlaz (monitor) pri čemu treba nekako naznačiti boje čvorova. Program zatim treba omogućiti dodavanje novih čvorova te nakon svake promjene treba ponovo ispisati stablo.

2. Rješenje zadatka

2.1. Teorijski uvod

Crveno-crna stabla (RB-stabla, *Red-Black Trees*) su posebna vrsta samo-balansirajućeg binarnog pretražujućeg stabla (*Binary Search Tree*). Svaki čvor osim vrijednosti (obično cijeli broj) i pokazivača na lijevo i desno dijete (i na roditelja) sadrži dodatnu informaciju koja predstavlja boju čvora: crvena ili crna. Na temelju te informacije, nakon svakog dodavanja ili brisanja vrijednosti, stablo se rekonstruira kako bi se poštivala definicijska pravila crveno-crnog stabla:

1. Svaki čvor je crven ili crn
2. Korijen je crn (neobavezno, ali uobičajeno)
3. Svaki list (čvor bez vrijednosti) je crn
4. Oba potomka crvenog čvora su crna
5. Svaka staza od nekog čvora do (bilo kojeg) lista koji je njegov potomak prolazi istim brojem crnih čvorova

Posljedica toga je konstantno zadržavanje (približne) uravnoteženosti stabla čime se postiže brzo pretraživanje:

Kao posljedica 4. i 5. pravila, visina *h* RB-stabla*[1]* s *n* unutarnjih čvorova je

$$h \leq 2 \cdot \log_2(n+1)$$

Pretraživanje binarnog stabla je složenosti $O(h)$ pa je složenost pretraživanje RB-stabla $O(\log_2 n)$. Dodavanje i brisanje čvorova su također $O(\log_2 n)$ operacije.

Kod RB-stabla posebni su postupci za dodavanje vrijednosti u stablo ili brisanje vrijednosti iz stabla. Također, razlikujemo posebno algoritme rekonstrukcije za pojedini postupak. Ti postupci odnosno algoritmi rekonstrukcije su pojašnjeni u sljedećem poglavlju.

2.2. Implementacija

Postupci i algoritmi su implementirani u programskom jeziku C#. Implementacija je izvršena koristeći standardne biblioteke tog jezika. Za vizualni prikaz tj. grafičko sučelje programa koristila se biblioteka Windows Forms (*WinForms*), prikaz je naveden u podpoglavlju 2.3. Kao pomoć pri pisanju implementacije i otklanjanju grešaka koristilo se razvojno okruženje MS Visual Studio 2015.

2.2.1. Čvor stabla

Čvor RB-stabla (*RBTreeNode*) sadrži vrijednost (cijeli broj, *int*), referencu na lijevo dijete (*RBTreeNode*), referencu na desno dijete (*RBTreeNode*), referencu na roditelja čvora (*RBTreeNode*) i bit informacije je li čvor crvene boje (*1* za crvenu i *0* za crnu boju) (Boolean). Dodatno kako bi se uklonila potreba za dodatnim razredom NULL lista, čvor sadrži bit informacije je li čvor NULL list (*1* ako je, *0* inače). Čvor se prikazuje na sljedeći način:

- ako je crvene boje: *[vrijednost]*
- ako je crne boje: *(vrijednost)*

Čvor je temeljna struktura RB-stabla.

2.2.2. Dodavanje vrijednosti (novog čvora)

Novi čvor (crvene boje) se u RB-stablo dodaje kao u svako drugo binarno stablo (BST):

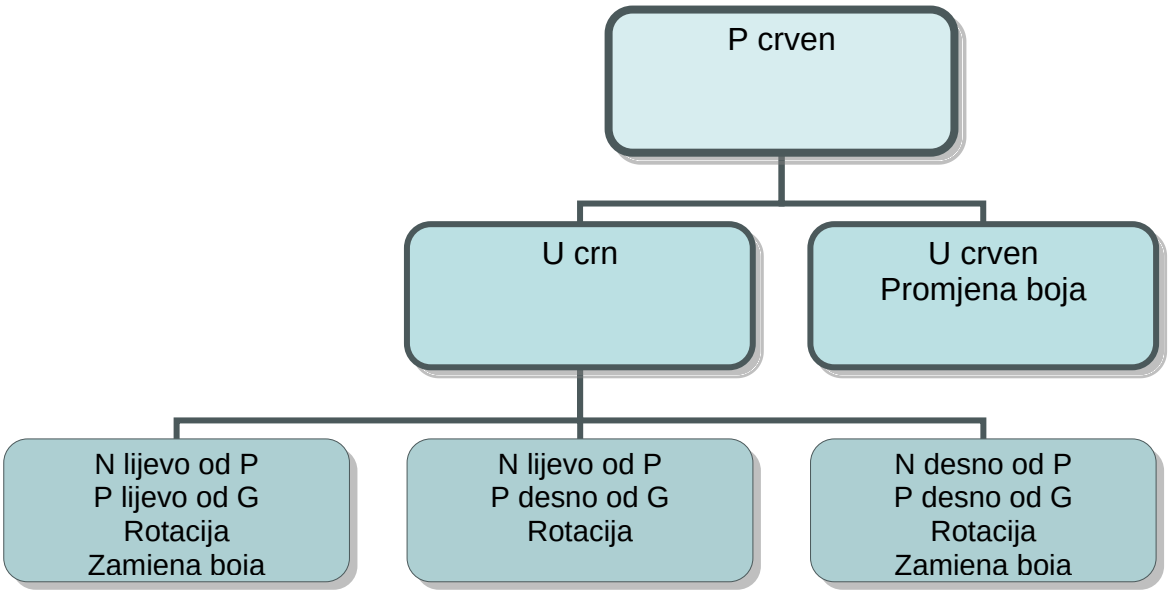
```
insertIntoTree(newNode)

node = root
while node is not null
      if newNode.value < node.value
            if node.left is null
                  node.left = newNode
                  return
            node = node.left
      else
            if node.right is null
                  node.right = newNode
                  return
            node = node.right
```

Nakon dodavanja novog čvora postoji mogućnost da je narušeno 2. ili 4. definicijsko pravilo. U tom slučaju vrši se rekonstrukcija stabla:

Ako je novi čvor korijen, on je crvene boje i time je narušeno 2. pravilo. Novi čvor odnosno korijen se oboja u crnu boju i postupak rekonstrukcije je završen.

Ako novi čvor nije korijen onda može biti narušeno samo 4. pravilo zato što je roditelj novog čvora također crven. Zbog boje roditelja zaključujemo da on ne može biti korijen pa zaključujemo da sigurno postoji i djed, a budući da ima crveno dijete, djed je crn.

Rekonstrukcija za slučaj narušavanja 4. pravila se može sažeto opisati sljedećim stablom odlučivanja *[1]*:



Gdje su: N - novi čvor, P – roditelj novog čvora, G – djed novog čvora, U – ujak novog čvora (drugo dijete od G).

Algoritam rekonstrukcije nakon dodavanja čvora se može kompaktno prikazati *[1]*:

```
insertFixup(N)

// N = new node
// p(N) = parent of N
// g(N) = grandparent of N
// U = other child of grandparent of N
while colour(p(N)) is red
      if p(N) is left child of g(N)
            U = right child of g(N)
            if colour(U) = red
            {  colour(p(N)) = black
               colour(U) = black
               colour(g(N)) = red
               N = g(N)
            }
            else
            {  if N is right child of P
               {  left-rotate N about P
                  N = P
               }
               right-rotate p(N) about g(N)
               colour(P) = black
               colour(G) = red
      else
            // same as above, with left and right exchanged
colour(root) = black
```

2.2.3. Brisanje vrijednosti (čvora)

Brisanje čvora u RB-stablu se izvodi uz pomoć kopiranja vrijednosti (*Deletion by Copying*). Umjesto brisanja zadanog čvora, iz stabla se zapravo uklanja zamjenski čvor. Postupak se svodi na pronalazak zamjenskog čvora, koji je najbliži prethodnik ili sljedbenik čvora koji se briše (najdesniji u lijevom,

najljeviji u desnom podstablu), prepisivanje vrijednosti iz zamjenskog čvora u čvor 'koji se briše', usmjeravanje pokazivača roditelja zamjenskog čvora na dijete zamjenskog čvora i konačno uklanjanje zamjenskog čvora.

deleteNodeTree(*node*)

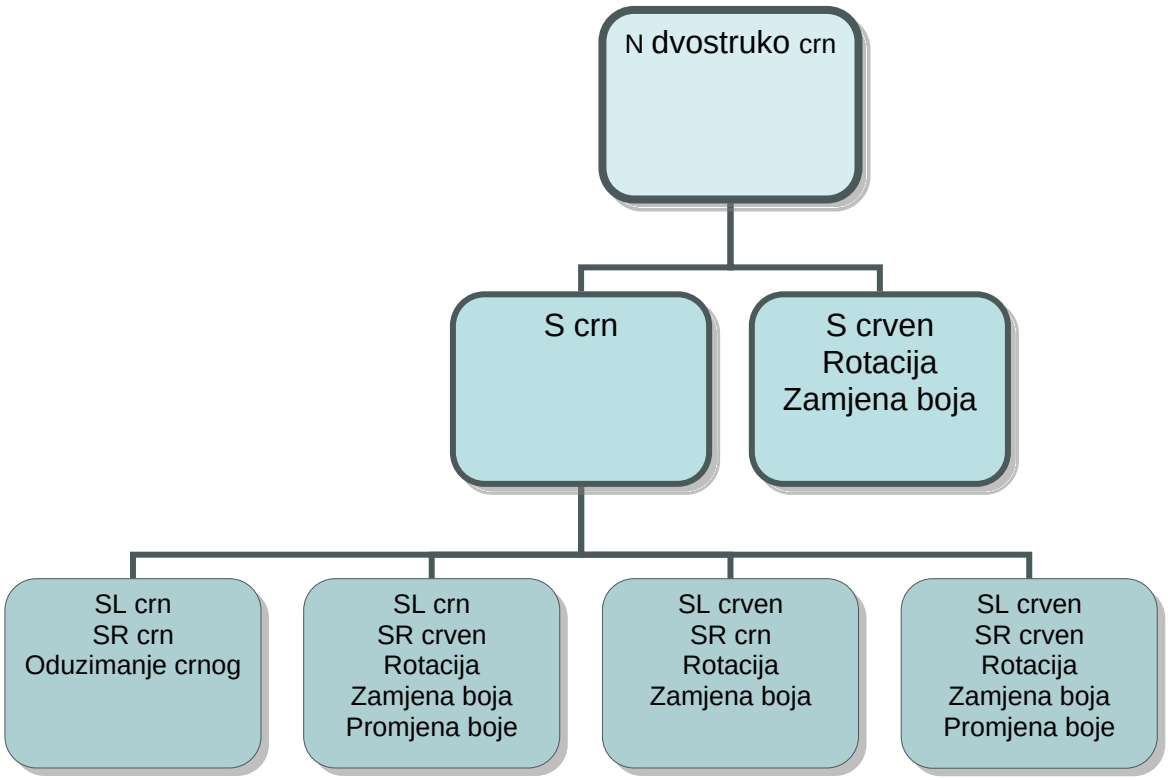
```
if node.right is null
      node = node.left
else-if node.right is null
      node = node.right
else
      tmp = node.left
      previous = node
      while tmp.right is not null
      {     previous = tmp
            tmp = tmp.right
      }
      node.value = tmp.value
      if previous is node
            previous.left = temp.left
      else
            previous.right = tmp.left
delete tmp
```

Prije samog brisanja zamjenskog čvora X potrebno je zapamtiti koje je boje zamjenski čvor i koje je njegovo dijete N.
Ako je on crvene boje, nema potrebe za rekonstrukcijom jer pravila nisu narušena, inače nastaju tri moguća problema:

- Ako je uklonjen korijen, mogao je imati samo jedno dijete koje postaje novi korijen, a ono može biti i crveno čime se narušava 2. pravilo.
- Nakon uklanjanja X, njegovo dijete N i roditelj P su u odnosu dijete-roditelj i ako su oboje crveni, prekršeno je 4. pravilo.
- Uklanjanje crnog X znači smanjenje crne visine svih njegovih prethodnika čime se narušava 5. pravilo.

Prvi problem se rješava bojanjem N u crno i postupak rekonstrukcije time završava.

Zamislimo da možemo nekako prenijeti crninu X-a na N. Tada ju uklanjanjem X-a ne bismo izgubili i RB pravila ne bi bila prekršena. Recimo da je to moguće i da tako postupimo. Ako je N prethodno bio crven, postat će crveno-crn i crnoj visini doprinositi =1. Ako je N prethodno bio crn, postat će dvostruko crn i crnoj visini doprinositi =2. Dakle, N može biti ili crveno-crn ili dvostruko crn. Ako je crveno-crn, dovoljno je prebojati ga u čisto crno i gotovo; stablo će ponovno biti pravilno RB stablo.
Ostaje nam riješiti (ukloniti) dvostruku crninu (“dvostruko crno”). Ideja je proslijediti višak crnog prethodniku i tako taj višak podizati sve dok ne dođe na mjesto gdje ga možemo trajno ugraditi u stablo ili dok ne dođe u korijen, gdje se jednostavno može odbaciti, tj. zanemariti[1]:



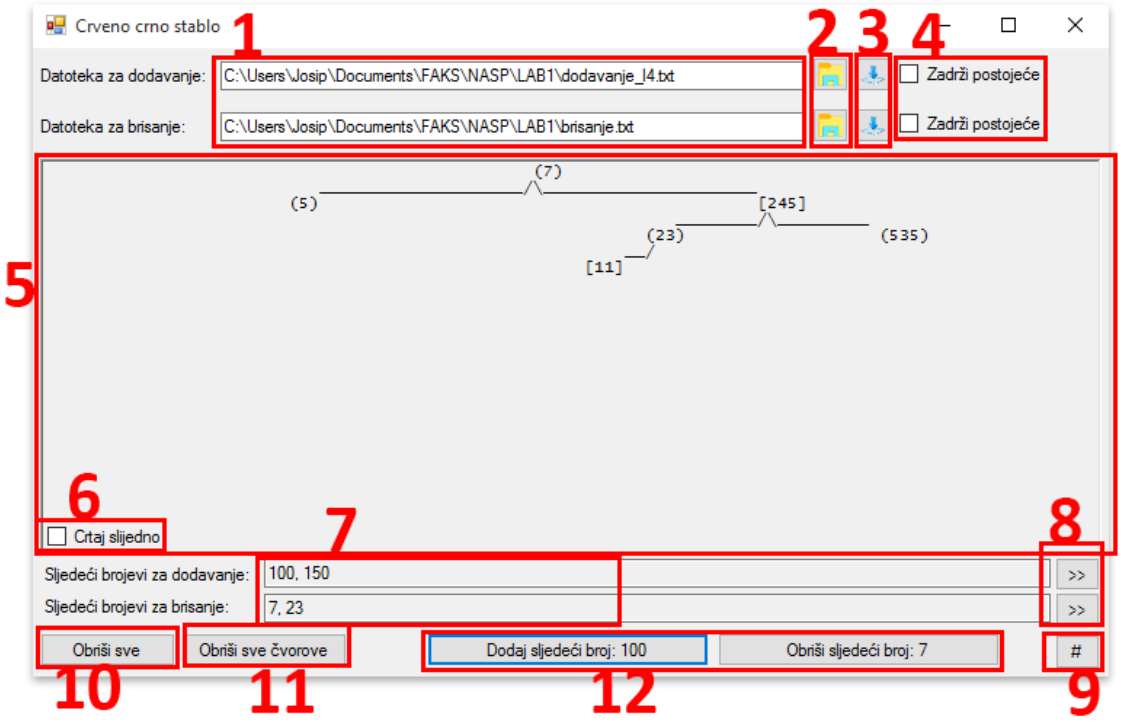
Gdje su: N – dijete zamjenskog čvora X, P – roditelj od N, G – djed od N, S – drugo dijete od P, SL – lijevo dijete od S, SR – desno dijete od S.

Algoritam rekonstrukcije nakon brisanja čvora se može kompaktno prikazati *[1]*:

deleteFixup(*N*)

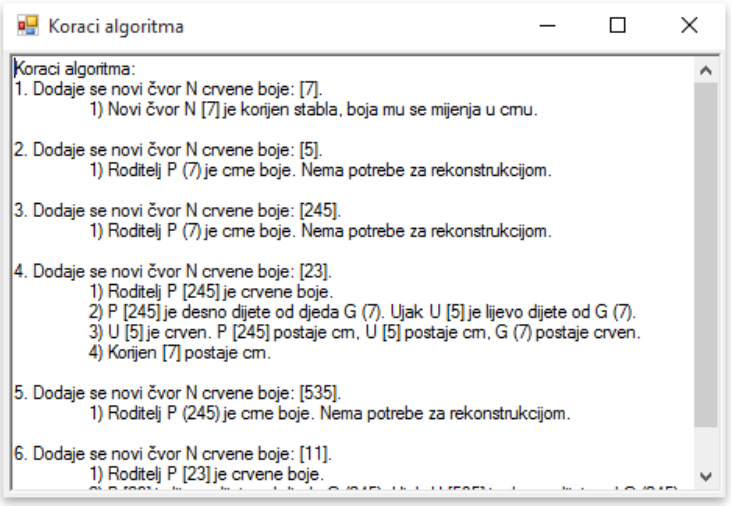
```
while N is not root and colour(N) is black
      if N is left child of P
            S = right child of P
            if colour(S)= red
                  colour(S) = black
                  colour(p(N)) = red
                  left-rotate S about P
            if colour(SL) = black and colour(SR) = black
                  colour(S) = red
                  N = p(N)
            else
                  if colour(SR) = black
                  {  colour(SL) = black
                     colour(S) = red
                     right-rotate SL about S
                  }
                  colour(S) = colour(P)
                  colour(P) = black
                  colour(SR) = black
                  left-rotate S about P
                  N = root
      else
            // same as above, with left and right exchanged
colour(N) = black
```

2.3. Grafičko sučelje programa



Slika 1.

1. Putanje datoteke za dodavanje/brisanje brojeva.
2. Datotečni prikaz za odabiranje datoteke za dodavanje/brisanje brojeva.
3. Učitavanje datoteke za dodavanje/brisanje brojeva.
4. Ako se označi trenutno preostali učitani brojevi neće biti obrisani.
5. Tekstualni prikaz stabla. [vrijednost] – crveni čvor, (vrijednost) – crni čvor.
6. Ako se označi prethodno prikazano stablo se neće obrisati iz prikaza.
7. Sljedeći brojevi na redu za dodavanje/brisanje.
8. Dodavanje/brisanje svih preostalih brojeva.
9. Prikaz/skrivanje prikaza koraka algoritma (Slika 2.).
10. Brisanje svih čvorova stabla i prikaza stabla odnosno koraka algoritma.
11. Brisanje svih čvorova stabla.
12. Gumb za dodavanje/brisanje sljedećeg broja.



Slika 2.

3. Zaključak

RB-stablo je zanimljiva struktura koja omogućuje brzo pretraživanje pohranjene vrijednosti. Iako joj je temeljna struktura običan čvor, potrebno je biti jako pažljiv prilikom pisanja implementacije rekonstrukcije jer su potrebne mnogobrojne promjene referenci ovisno o pojedinom slučaju i lako se slučajno napravi neki krivi korak koji uzrokuje pogrešku. Zato je prilikom pisanja bitna i kvaliteta IDE-a s kojim se piše jer omogućava pregled stabla (i njegovih komponenti) uz pomoć *debuggera*. Također je potreban vizualan prikaz svakog koraka algoritma dodavanja ili brisanja čvora kako bi se moglo usporediti funkcioniranje vlastitog programa s tim prikazom*[2]*.

4. Literatura

[1] Nikica Hlupić, Damir Kalpić, Napredne strukture podataka (prezentacija predavanja), FER, Zagreb, 2009.
[2] David Galles, University of San Francisco, Department of Computer Science, https://www.cs.usfca.edu/~galles/visualization/RedBlack.html, 2011