Day Stout Warren algoritam

Seminarski rad u okviru kursa
Konstrukcija i analiza algoritama 2
Matematički fakultet

Gorana Vučić, 1095/2017
goranavucic94@gmail.com

20. februar 2019

Sadržaj

1 Uvod

Day Stout Warren algoritam (u daljem tekstu DSW) je algoritam koji služi za efikasno balansiranje binarnih stabala pretrage. DSW algoritam za razliku od uravnoteženih binarnih stabala pretrage koja se balansiraju nakon svake operacije (ukoliko je to neophodno), radi periodično.

Algoritam se sastoji iz dve faze:

- U prvoj fazi se zadato stablo ispravlja u povezanu listu tako da se transformacijama od početnog stabla dolazi do degenerisanog stabla.
- U drugoj fazi algoritma vrši se balansiranje tako da se transformacijama od degenerisanog stabla dolazi do AVL stabla.

Kako bi se navedene transformacije izvršile postoje dve operacije u algoritmu koje predstavljaju rotacije potomka oko pretka.

Desna rotacija potomka oko pretka podrazumeva da se koren stabla koji u ovom slučaju predstavlja C, spušta za jedan nivo ispod, pri čemu desni pokazivač čvora C i dalje pokazuje na svoje desno podstablo D. B ostaje na istom nivou, ali umesto desnog podstabla A postaje levo podstablo čvora C. Sa druge strane A se podiže za jedan nivo iznad, tako da postaje koren stabla, dok njegov levi pokazivač i dalje pokazuje na svoje levo podstablo, a desni pokazivač pokazuje na C (Slika 1).

Slika 1: Desna rotacija potomka oko pretka

Leva rotacija potomka oko pretka predstavlja transformaciju u kojoj se D kao potomak rotira oko B. Čvor D podiže se za jedan nivo iznad i postaje koren stabla, pri čemu njegov desni pokazivač nastavlja da pokazuje na svoje desno podstablo, a levi pokazuje na B. Čvor B koji na početku predstavlja koren stabla, spušta se za jedan nivo ispod pri čemu njegov levi pokazivač i dalje nastavlja da pokazuje na svoje levo podstablo. Stablo C ostaje na istom nivou ali umesto levog podstabla D postaje desno podstablo od B (Slika 2).

2 Implementacija

Za prvu fazu algoritma koja zadato stablo ispravlja u degenerisano stablo implementirana je funkcija *Node* DSW::treeToArray(Node* root)* (Slika 3). U okviru nje uvodi se novi pomoćni čvor -1 čiji desni pokazivač pokazuje na binarno stablo pretrage koje je prethodno formirano. U funkciji se prolazi kroz desne grane i za svaki čvor proverava se da li sadrži levo podstablo. Ukoliko ne sadrži prelazi se na sledeći čvor koji predstavlja desnog sina od prethodnog čvora, a

Slika 2: Leva rotacija potomka oko pretka

ukoliko sadrži onda se izvršava desna rotacija za taj čvor koja je implementirana funkcijom *Node *DSW::rightRotation(Node *root)* (Slika 4).

```
Node* DSW::treeToArray(Node* root)
{

    Node* p, *r;
    r = new Node;
    r->key = -1;
    r->left = nullptr;
    r->right = root;

    p = r;
    while(p->right){
        if(p->right->left == nullptr){
            p = p->right;
        }
        else{
            p->right = rightRotation(p->right);
        }
    }

    return r;
}
```

Slika 3: Prva faza algoritma

```
Node *DSW::rightRotation(Node *root)
{
    Node* left, *leftRight;
    if((root==nullptr) || (root->left == nullptr))
        return root;

    left = root->left;
    leftRight = left->right;
    left->right = root;
    root->left = leftRight;

    return left;

}
```

Slika 4: Desna rotacija

Za drugu fazu algoritma gde je potrebno zadato degenerisano stablo koje je predstavljeno u vidu niza sortiranih elemenata transformisati u AVL stablo, implementirana je funkcija *Node *DSW::arrayToTree(Node *root)*. U okviru nje prvo se u promenljivoj *lastC* izračunava koliki broj čvorova treba da bude na poslednjem nivou stabla što se određuje kao $n+1-2^{\log_2(n+1)}$, gde n predstavlja broj čvorova. Potom se toliki broj puta primenjuje leva rotacija pozivom funkcije *Node *DSW::rotation(Node *root, int numberOfRotations)* (Slika 6). Leva

rotacija primenjuje se na svaki parni čvor, gde treba imati u vidu da je prvi čvor pomoćni čvor -1. Potom se izračunava novi broj potrebnih rotacija koje treba da se izvrše tako što se od ukupnog broja čvorova degenerisanog stabla oduzme *lastC* vrednost i onda se u while petlji poziva funckija *rotation* sve dok važi uslov da je $n > 1$ pri čemu se n deli sa 2. Transformacija se izvršava sve dok stablo ne postane balansirano binarno stablo pretrage.

```
Node *DSW::rotation(Node *root, int numberOfRotations)
{
    Node* tmp;
    tmp = root;
    while(numberOfRotations){
        tmp->right = leftRotation(tmp->right);
        tmp = tmp->right;
        --numberOfRotations;
    }
    return root;
}
```

Slika 5: Funkcija rotacije

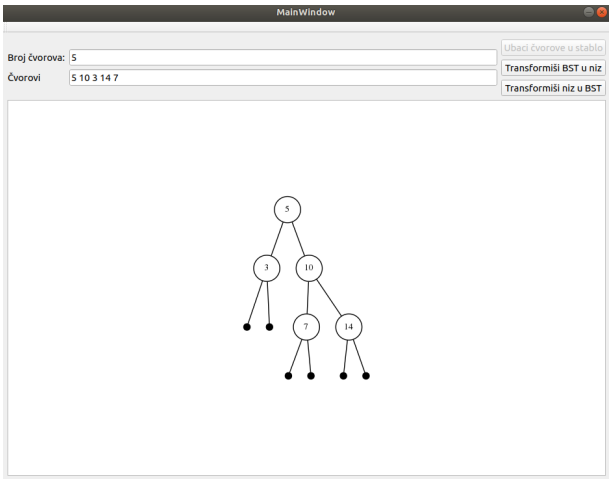
```
Node *DSW::leftRotation(Node *root)
{
    Node* right, *rightLeft;
    if((root == nullptr) || (root->right == nullptr))
        return root;

    right = root->right;
    rightLeft = right->left;
    right->left = root;
    root->right = rightLeft;
    return right;

}
```

Slika 6: Leva rotacija potomka oko pretka

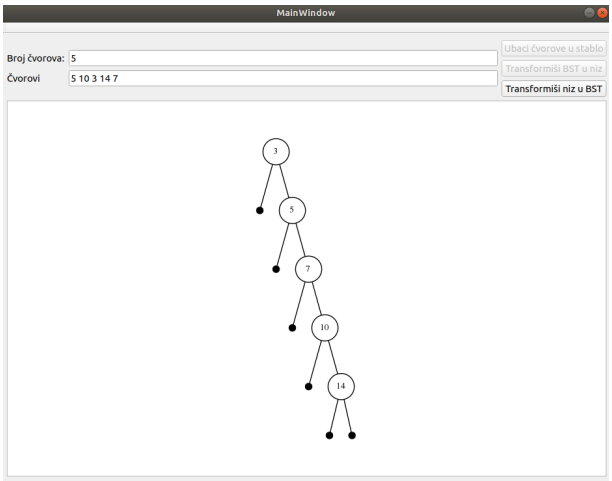
3 Izgled programa

Napravljena je aplikacija u $Qt - u$ koja prikazuje zadate transformacije. Za predstavljanje stabla korišćen je paket *Graphviz* koji služi za prikazivanje grafova. Kako bi program mogao da se pokrene potrebno je instalirati ovaj paket sa *sudo apt-get install graphviz*. Neophodno je uneti broj čvorova za dato stablo i niz zadatih čvorova (Napomena - aplikacija koja služi za vizuelni prikaz je napravljena za unos manjeg broja čvorova, a zasebno je dat kod za algoritam za velike ulaze kao što je na primer 1000 elemenata).

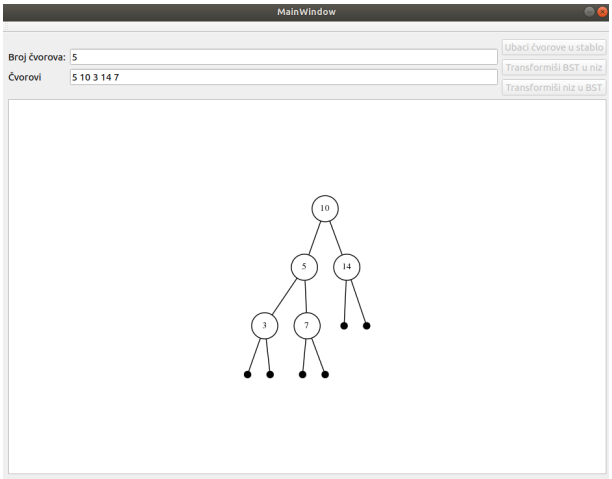
Na narednim slikama dat je jedan kratak primer gde se za ulaz od 5 čvorova, čije vrednosti su 5, 10, 3, 14, 7 može videti kako izgleda sama aplikacija i niz primenjenih transformacija.



Slika 7: Ubacivanje čvorova u stablo



Slika 8: Transformacija stabla u niz



Slika 9: Transformacija niza u stablo

4 Složenost

Pored samog programa koji je napravljen u $Qt-u$, zasebno postoje *dsw.cpp* i *dsw.hpp* datoteke u kojima je izdvojen sam algoritam kako bi se testiralo vreme izvršavanja za velike ulaze. Složenost prve faze algoritma u najgorem slučaju iznosi $O(n)$, kao i složenost druge faze. Ukupna vremenska složenost je $O(n)$.

Vreme izvršavanja je izraženo u nanosekundama. Pri pokretanju programa prosleđuje se broj željenih čvorova koji se random generišu i ubacuju u stablo pomoću funkcije *insert*. Potom se meri vreme izvršavanja transformacija. Za različite ulaze program je svaki put pozvan 10 puta i izračunato je prosečno vreme izvršavanja, na osnovu kog je kasnije napravljen dijagram zavisnosti vremena izvršavanja od broja čvorova u stablu (Slika 10).

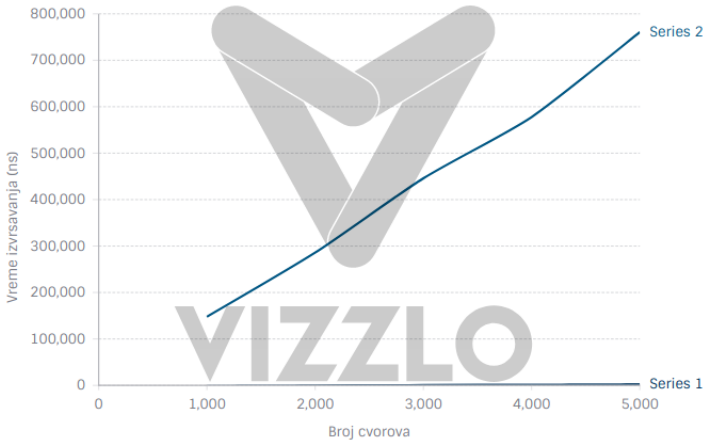
Za ulaz od 1000 elemenata dobijena su sledeća vremena izvršavanja: 143058 147370, 139255, 142724, 160909, 150014, 164182, 140528, 149412, 140375, gde je prosečno vreme 147782,7.

Za ulaz od 2000 elemenata dobijena su sledeća vremena izvršavanja: 296262, 152035, 288250, 292316, 309987, 310599, 313809, 306168, 295724, 293526 gde je prosečno vreme 285867,6.

Za ulaz od 3000 elemenata dobijena su sledeća vremena izvršavanja: 445688, 455483, 448000, 498349, 473053, 444709, 454438, 336774, 460108, 442461, gde je prosečno vreme 445906,3.

Za ulaz od 4000 elemenata dobijena su sledeća vremena izvršavanja: 576471, 607108, 624053, 630411, 596583, 433244, 674994, 678778, 342321, 614761 gde je prosečno vreme 577873,4.

Za ulaz od 5000 elemenata dobijena su sledeća vremena izvršavanja: 904459, 797626, 779860, 416847, 810975, 816176, 801443, 803010, 697200, 779244 gde je prosečno vreme 760684.



Slika 10: Vreme izvršavanja/broj čvorova

5 Literatura

https://web.eecs.umich.edu/~qstout/pap/CACM86.pdf