Univerzitet u Sarajevu
Prirodno-matematički fakultet
Odsjek za matematiku

Dokumentacija za Projekat 2 iz predmeta Strukture podataka i algoritmi

Student:

Edib Šupić

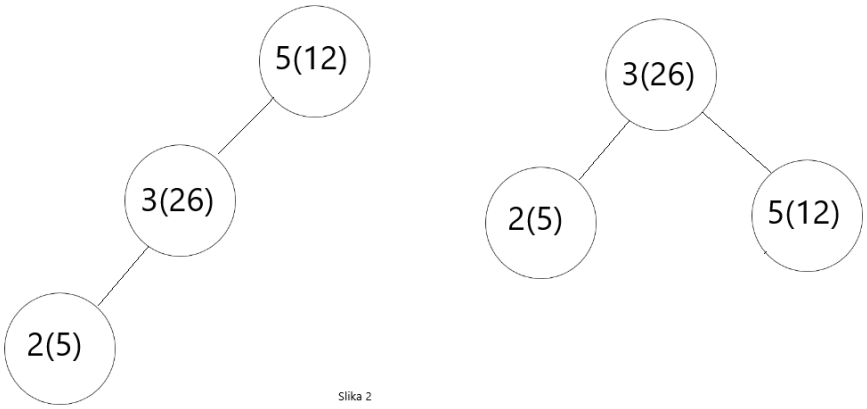
Interfejs klase **Stablo** dat je na slici 1. Klasa je definisana kao *class template* i može se koristiti za sve tipove podataka za koje je definisan *operator <. Za* implementaciju traženih funkcionalnosti klase dodane su i funckije: *void* **checkProperty**(*Node* ***currentRoot**), *void* **checkUnite**(*Node* ***currentRoot**), *void* **rotateRight**(*Node* ***currentRoot**), *void* **rotateLeft**(*Node* ***currentRoot**), *T* **findMin**(*Node* ***currentRoot**) i *T* **findMax**(*Node* ***currentRoot**).

```
template <typename T>
class Stablo {
private:
   // MEMBERS
   struct Node {
      T element;
      int priority;
      Node *leftChild, *rightChild, *parent;
   };
   int length;
   Node *root;
   // CONSTRUCTOR
   Stablo(Node *n1, int length = 100) : length(length), root(n1) { if (root) root->parent = nullptr;  }
   // ADDITIONS
   void myAdd(T x, Node *currentRoot);
   void myDivide(T x, Node *currentRoot);
   // REMOVE
   void myRemove(T x, Node *&currentRoot);
   // PROPERTY CHECKS
   void checkProperty(Node *currentRoot);
   void checkUnite(Node *currentRoot);
   // ROTATIONS
   void rotateRight(Node *currentRoot);
   void rotateLeft(Node *currentRoot);
   // SEARCHES
   T findMin(Node *currentRoot) const;
   T findMax(Node *currentRoot) const;
   bool myFind(T x, const Node *const& currentRoot) const;
   // OUTPUT
   void preOrder(Node *currentRoot, int d = 0) const;
public:
   Stablo(int length=100) : length(length), root(nullptr) {}
   void myAdd(T x) { myAdd(x, root); }
   // REMOVE
   void myRemove(T x) { myRemove(x,root); }
   // SEARCHES
   T findMin() const { return findMin(root); }
   T findMax() const { return findMax(root); }
   bool myFind(T x) const { return myFind(x, root); }
   // OUTPUT
   void preOrder() const { preOrder(root); }
   // DIVIDE, UNITE AND UNION
   pair<Stablo, Stablo> myDivide(T x);
   template <typename T1>
   friend Stablo<T1> myUnite(Stablo<T1> s1, Stablo<T1> s2);
   template <typename T1>
   friend Stablo<T1> myUnion(Stablo<T1> s1, Stablo<T1> s2);
};
```

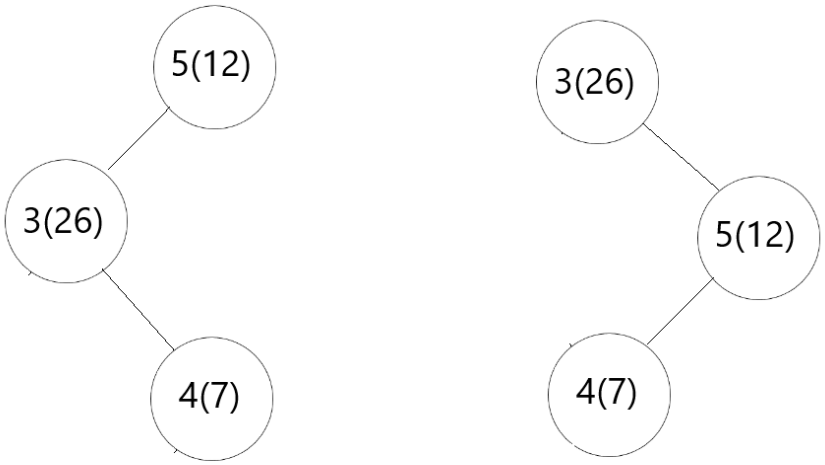
Slika 1.

Funkcija **checkProperty** poredi prioritete datog čvora sa prioritetima njegove djece i ukoliko lijevo dijete ima veći prioritet poziva funkciju **rotateRight**, a ako desno ima veći prioritet poziva **rotateLeft**, nakon čega se funkcija poziva rekurzivno za početnog roditelja datog čvora. Funckija **rotateLeft** i **rotateRight** su identične tako da ćemo ovdje objasniti samo **rotateRight**. Ukoliko lijevo dijete čvora ima veći prioritet od samog čvora provjeravmo da li lijevo dijete ima djecu. Uvedimo neke oznake: CR(currentRoot) je čvor koji prima funkcija, LC(leftChild) je lijevo dijete od CR, RC(rightChild) je desno dijete od CR. Ako LC nema desno dijete dovoljno je CR postaviti kao desno dijete od LC, a LC na mjesto CR(slika 2). Ukoliko LC ima desno dijete onda CR postavljamo kao desno dijete od LC, a kao lijevo dijete od CR postavljamo desno

dijete od LC. LC zatim stavljamo na početno mjesto od CR(slika 3).



Slika 2



Slika 3

findMin, **findMax** i **myFind**, koja je ujedno tražena funkcija za pretragu, su trivijalne, a funkcija **checkUnite** će biti objašnjena u nastavku. Tražena funkcija za umetanje **myAdd** definisana je kao u binarnom stablu pretrage uz razliku što se prilikom umetanja čvora generiše nasumičan broj koji pretstavlja prioritet i zapisuje u čvor. Funkcija za brisanje **myRemove** radi na način da ako je čvor u pitanju, list samo se obriše, ako čvor za brisanje ima jedno dijete ono se prepiše na njegovo mjesto a čvor obriše. Ukoliko čvor ima dvoje djece u desnom djetetu nađemo najmanji elemenat i prepišemo ga na mjesto čvora a zatim **myRemove** rekurzivno pozovemo za desno podstablo. Budući da ne diramo prioritete stabla i čvor koji na kraju brišemo je ili list ili ima samo desno dijete, nećemo narušiti osobinu heapa, pa nema potrebe za dodatnim rotacijama. Funkcija za razdvajanje **myDivide** funkcioniše tako što specijalnom verzijom funckije **myAdd**, koja se zove također **myDivide**, ubacuje elemenat, sa prioritetom za 1 većim od najvećeg mogućeg, u stablo i zatim ga uz pomoć funkcije **checkProperty** postavlja kao korijen. Stablo se zatim dijeli na dva nova stabla čiji su korijeni lijevo dijete i desno dijete od dodanog čvora. Funkcija za spajanje **myUnite** prvo nađe broj veći od najvećeg u lijevom stablu i manji od najmanjeg u desnom i postavi ga kao korijen sa prioritetom -1. Zatim uz pomoć funkcije **checkUnite** novi čvor postavljamo kao list i na kraju brišemo. Sada ćemo objasniti funkciju **checkUnite**. Kada čvor nema djece brišemo ga, ako ima samo lijevo dijete rotiramo ga na desno uz pomoć funkcije **rotateRight**, za čvorove sa samo desnim djetetom radimo suprotno. A ako čvor ima dvoje djece biramo ono sa većim prioritetom i rotiramo na suprotnu stranu.

Reference:

http://www.pmf.unsa.ba/nastava/pluginfile.php/8523/mod_resource/content/3/Mark%20A.%20Weiss-Data%20Structures%20and%20Algorithm%20Analysis%20in%20C%2B%2B-Pearson%20%282014%29.pdf