\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DinamickoStablo.h \\\neparni

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <iostream>

using namespace std;

int max(int a, int b)

{

return a>b?a:b;

}

template<class T>

class Element

{

private:

T info;

Element<T>\* levi;

Element<T>\* desni;

public:

Element()

{

levi = desni = NULL;

}

Element(T info)

{

this->info = info;

levi = desni = NULL;

}

template<class T> friend class DinamickoStablo;

};

template<class T>

class DinamickoStablo

{

private:

Element<T>\* koren;

public:

DinamickoStablo()

{

koren = NULL;

}

~DinamickoStablo()

{

obrisiStablo(koren);

}

void obrisiStablo(Element<T>\* p)

{

if(p!=NULL)

{

obrisiStablo(p->levi);

obrisiStablo(p->desni);

delete p;

}

}

void ubaci(T element)

{

Element<T>\* tekuci = koren;

Element<T>\* prethodni = NULL;

while(tekuci!=NULL)

{

prethodni = tekuci;

if(tekuci->info > element)

tekuci = tekuci->levi;

else

if(tekuci->info < element)

tekuci = tekuci->desni;

else

return;

}

if(koren==NULL)

koren = new Element<T>(element);

else

if(prethodni->info > element)

prethodni->levi = new Element<T>(element);

else

if(prethodni->info < element)

prethodni->desni = new Element<T>(element);

}

void prikazi()

{

prikazi(koren);

cout << endl;

}

void prikazi(Element<T>\* p)

{

if(p!=NULL)

{

prikazi(p->levi);

cout << p->info << " ";

prikazi(p->desni);

}

}

3. ***Projektovati klasu za rad sa dinamičkim binarnim stablom i implementirati metod za određivanje visine stabla.***

int visinaStabla() // 3. zadatak

{

return visinaStabla(koren);

}

int visinaStabla(Element<T>\* p) // 3. zadatak

{

if(p==NULL)

return 0;

if(p->levi==NULL && p->desni==NULL)

return 1;

return 1 + max(visinaStabla(p->levi), visinaStabla(p->desni));

}

*5****. Projektovati klasu za rad sa dinamičkim binarnim stablom i implementirati metod za određivanje momenta stabla.***

int momentStabla() // 5. zadatak

{

return momentStabla(koren);

}

int momentStabla(Element<T>\* p) // 5. zadatak

{

if(p==NULL)

return 0;

if(p->levi==NULL && p->desni==NULL)

return 1;

return 1 + momentStabla(p->levi) + momentStabla(p->desni);

}

***7. Projektovati klasu za rad sa dinamičkim binarnim stablom i implementirati metod za određivanje težine stabla.***

int tezinaStabla() // 7. zadatak

{

return tezinaStabla(koren);

}

int tezinaStabla(Element<T>\* p) // 7. zadatak

{

if(p==NULL)

return 0;

if(p->levi==NULL && p->desni==NULL)

return 1;

return tezinaStabla(p->levi) + tezinaStabla(p->desni);

}

***9. Projektovati klasu za rad sa dinamičkim binarnim stablom i implementirati metod za brisanje svih listova***.

void brisanjeListova() // 9. zadatak

{

if(koren!=NULL)

if(koren->levi==NULL && koren->desni==NULL)

{

delete koren;

koren = NULL;

}

brisanjeListova(koren);

}

void brisanjeListova(Element<T>\* p) // 9. zadatak

{

if(p==NULL)

return;

if(p->levi!=NULL)

if(p->levi->levi==NULL && p->levi->desni==NULL)

{

delete p->levi;

p->levi = NULL;

}

if(p->desni!=NULL)

if(p->desni->levi==NULL && p->desni->desni==NULL)

{

delete p->desni;

p->desni = NULL;

}

brisanjeListova(p->levi);

brisanjeListova(p->desni);

}

***11. Projektovati klasu za rad sa dinamičkim binarnim stablom i implementirati metod za formiranje mirror kopije datog stabla (slika u ogledalu).***

void mirrorKopija() // 11. zadatak

{

mirrorKopija(koren);

}

void mirrorKopija(Element<T>\* p) // 11. zadatak

{

if(p==NULL)

return;

Element<T>\* pom = p->levi;

p->levi = p->desni;

p->desni = pom;

mirrorKopija(p->levi);

mirrorKopija(p->desni);

}

***13. Projektovati klasu za rad sa dinamičkim binarnim stablom i implementirati metod za spajanje dva uređena stabla. Rezultujuće stablo takođe mora biti uređeno.***

void spajanjeStabala(DinamickoStablo<T>& ds) // 13. zadatak

{

spajanjeStabala(ds.koren);

}

void spajanjeStabala(Element<T>\* p) // 13. zadatak

{

if(p==NULL)

return;

spajanjeStabala(p->levi);

this->ubaci(p->info);

spajanjeStabala(p->desni);

}

***15. Projektovati klasu za rad sa dinamičkim binarnim stablom i implementirati metod za određivanje nivoa sa najviše čvorova.***

int nivoSaNajviseCvorova() // 15. zadatak

{

if(koren==NULL || (koren->levi==NULL && koren->desni==NULL))

return 0;

int v = this->visinaStabla();

int\* a = new int[v];

for(int i=0; i<v; i++)

a[i] = 0;

nivoSaNajviseCvorova(a, koren);

int imax = 0;

for(int i=1; i<v; i++)

{

if(a[imax]<a[i])

imax = i; // i = imax - opasna greska!!!

}

return imax;

}

void nivoSaNajviseCvorova(int\* a, Element<T>\* p) // 15. zadatak

{

if(p!=NULL)

{

nivoSaNajviseCvorova(a, p->levi);

a[dubinaCvora(p)]++;

nivoSaNajviseCvorova(a, p->desni);

}

}

int dubinaCvora(Element<T>\* p) // 15. zadatak

{

if(p==koren)

return 0;

return 1+dubinaCvora(roditeljCvora(p, koren));

}

Element<T>\* roditeljCvora(Element<T>\*p, Element<T>\* q) // 15. zadatak

{

if(q==NULL)

return NULL;

if(q->levi==p || q->desni==p)

return q;

Element<T>\* pom;

pom = roditeljCvora(p, q->levi);

if(pom!=NULL)

return pom;

pom = roditeljCvora(p, q->desni);

if(pom!=NULL)

return pom;

return NULL;

}

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

main.cpp \\\\\\\ neparni

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "DinamickoStablo.h"

int main()

{

DinamickoStablo<int> ds1;

ds1.ubaci(5);

ds1.ubaci(4);

ds1.ubaci(2);

ds1.ubaci(1);

ds1.ubaci(7);

ds1.ubaci(3);

ds1.ubaci(6);

cout << "Inorder stabla: ";

ds1.prikazi();

cout << "Visina stabla je: " << ds1.visinaStabla() << endl;

cout << "Moment stabla je: " << ds1.momentStabla() << endl;

cout << "Tezina stabla je: " << ds1.tezinaStabla() << endl;

ds1.brisanjeListova();

cout << "Inorder stabla posle brisanja listova: ";

ds1.prikazi();

ds1.mirrorKopija();

cout << "Inorder mirror kopije stabla : ";

ds1.prikazi();

DinamickoStablo<int> prvoStablo;

DinamickoStablo<int>\* drugoStablo = new DinamickoStablo<int>();

prvoStablo.ubaci(5);

prvoStablo.ubaci(4);

prvoStablo.ubaci(2);

prvoStablo.ubaci(1);

prvoStablo.ubaci(7);

prvoStablo.ubaci(3);

prvoStablo.ubaci(6);

drugoStablo->ubaci(8);

drugoStablo->ubaci(1);

drugoStablo->ubaci(10);

drugoStablo->ubaci(9);

cout << "Inorder prvog stabla: ";

prvoStablo.prikazi();

cout << "Inorder drugog stabla: ";

drugoStablo->prikazi();

prvoStablo.spajanjeStabala(\*drugoStablo);

delete drugoStablo;

cout << "Inorder spojenih stabala: ";

prvoStablo.prikazi();

DinamickoStablo<int> ds2;

ds2.ubaci(5);

ds2.ubaci(4);

ds2.ubaci(2);

ds2.ubaci(1);

ds2.ubaci(7);

ds2.ubaci(3);

ds2.ubaci(6);

ds2.ubaci(8);

cout << "Inorder stabla: ";

ds2.prikazi();

cout << "Nivo sa najvise cvorova je: " << ds2.nivoSaNajviseCvorova() << endl;

return 0;

}