Лабораторная работа № 6
Пример задания

Тема: *Деревья. Сбалансированные по высоте деревья (АВЛ-деревья). 2-3 деревья. Б-деревья. Красно-черные деревья. Практическое применение.*

Цель: *ознакомиться с понятиями «Деревья», «АВЛ-деревья», «Б-деревья», «Красно-черные деревья», изучить основные алгоритмы их обработки, научиться применять полученные знания на практике.*

Написать программу, которая создает АВЛ-дерево и реализует следующие операции:

- добавление нового узла
- поиск минимального элемента
- поиск максимального элемента
- поиск по значению
- удаление элемента
- обход в глубину (pre-order)
- обход в глубину (in-order)
- обход в глубину (post-order)
- вывод высоты дерева

```
#include <iostream.h>
#include<ctype.h>
#include <stdlib.h>
#include <conio.h>

//using namespace std;

struct node
{
        int element;
        node *left;
        node *right;
        int height;
};

 typedef struct node *nodeptr;

class bstree
{
        public:
                void insert(int,nodeptr &);
                void del(int, nodeptr &);
                int deletemin(nodeptr &);
                void find(int,nodeptr &);
                nodeptr findmin(nodeptr);
                nodeptr findmax(nodeptr);
                void makeempty(nodeptr &);
                void copy(nodeptr &,nodeptr &);
                nodeptr nodecopy(nodeptr &);
                void preorder(nodeptr);
                void inorder(nodeptr);
                void postorder(nodeptr);
```

```
                int bsheight(nodeptr);
                nodeptr srl(nodeptr &);
                nodeptr drl(nodeptr &);
                nodeptr srr(nodeptr &);
                nodeptr drr(nodeptr &);
                int max(int,int);
                int nonodes(nodeptr);
};
// Inserting a node
void bstree::insert(int x,nodeptr &p)
{
        if (p == NULL)
        {
                p = new node;
                p->element = x;
                p->left=NULL;
                p->right = NULL;
                p->height=0;
                if (p==NULL)
                {
                        cout<<"Out of Space\n"<<endl;
                }
        }
        else
        {
                if (x<p->element)
                {
                        insert(x,p->left);
                        if ((bsheight(p->left) - bsheight(p->right))==2)
                        {
                                if (x < p->left->element)
                                {
                                        p=srl(p);
                                }
                                else
                                {
                                        p = drl(p);
                                }
                        }
                }
                else if (x>p->element)
                {
                        insert(x,p->right);
                        if ((bsheight(p->right) - bsheight(p->left))==2)
                        {
                                if (x > p->right->element)
                                {
                                        p=srr(p);
                                }
                                else
                                {
                                        p = drr(p);
                                }
                        }
                }
                else
                {
                        cout<<"Элемет существует\n"<<endl;
```

```
                }
}
int m,n,d;
m=bsheight(p->left);
n=bsheight(p->right);
d=max(m,n);
p->height = d + 1;
}
// Finding the Smallest
nodeptr bstree::findmin(nodeptr p)
{
        if (p==NULL)
        {
                cout<<"В дереве нет элементов\n"<<endl;
                return p;
        }
        else
        {
                while(p->left !=NULL)
                {
                        p=p->left;
                        //return p;
                }
                return p;
        }
}
// Finding the Largest node
nodeptr bstree::findmax(nodeptr p)
{
        if (p==NULL)
        {
                cout<<"В дереве нет элементов\n"<<endl;
                return p;
        }
        else
        {
                while(p->right !=NULL)
                {
                        p=p->right;
                        //return p;
                }
                return p;
        }
}
// Finding an element
void bstree::find(int x,nodeptr &p)
{
        if (p==NULL)
        {
                cout<<"Простите, но такого элемента нет\n"<<endl;
        }
        else
        {
                if (x < p->element)
                {
                        find(x,p->left);
                }
                else
```

```
                {
                        if (x>p->element)
                        {
                                find(x,p->right);
                        }
                        else
                        {
                                cout<<"Элемент, который вы искали есть в дереве!\n"<<endl;
                        }
                }
        }
}
// Copy a tree
void bstree::copy(nodeptr &p,nodeptr &p1)
{
        makeempty(p1);
        p1 = nodecopy(p);
}
// Make a tree empty
void bstree::makeempty(nodeptr &p)
{
        nodeptr d;
        if (p != NULL)
        {
                makeempty(p->left);
                makeempty(p->right);
                d=p;
                free(d);
                p=NULL;
        }
}
// Copy the nodes
nodeptr bstree::nodecopy(nodeptr &p)
{
        nodeptr temp;
        if (p==NULL)
        {
                return p;
        }
        else
        {
                temp = new node;
                temp->element = p->element;
                temp->left = nodecopy(p->left);
                temp->right = nodecopy(p->right);
                return temp;
        }
}

// Deleting a node
void bstree::del(int x,nodeptr &p)
{
        nodeptr d;
        if (p==NULL)
        {
                cout<<"Простите, но такого элемента нет\n"<<endl;
        }
        else if ( x < p->element)
```

```
        {
                del(x,p->left);
        }
        else if (x > p->element)
        {
                del(x,p->right);
        }
        else if ((p->left == NULL) && (p->right == NULL))
        {
                d=p;
                free(d);
                p=NULL;
                cout<<"Элемент удален\n"<<endl;
        }
        else if (p->left == NULL)
        {
                d=p;
                free(d);
                p=p->right;
                cout<<"Элемент удален\n"<<endl;
        }
        else if (p->right == NULL)
        {
                d=p;
                p=p->left;
                free(d);
                cout<<"Элемент удален\n"<<endl;
        }
        else
        {
                p->element = deletemin(p->right);
        }
}

int bstree::deletemin(nodeptr &p)
{
        int c;
        cout<<"Выбрано удаление минимального значения\n"<<endl;
        if (p->left == NULL)
        {
                c=p->element;
                p=p->right;
                return c;
        }
        else
        {
                c=deletemin(p->left);
                return c;
        }
}
void bstree::preorder(nodeptr p)
{
        if (p!=NULL)
        {
                cout<<p->element<<"\t";
                preorder(p->left);
                preorder(p->right);
        }
```

```
}

// Inorder Printing
void bstree::inorder(nodeptr p)
{
	if (p!=NULL)
	{
		inorder(p->left);
		cout<<p->element<<"\t";
		inorder(p->right);
	}
}

// PostOrder Printing
void bstree::postorder(nodeptr p)
{
	if (p!=NULL)
	{
		postorder(p->left);
		postorder(p->right);
		cout<<p->element<<"\t";
	}
}

int bstree::max(int value1, int value2)
{
	return ((value1 > value2) ? value1 : value2);
}
int bstree::bsheight(nodeptr p)
{
	int t;
	if (p == NULL)
	{
		return -1;
	}
	else
	{
		t = p->height;
		return t;
	}
}

nodeptr bstree:: srl(nodeptr &p1)
{
	nodeptr p2;
	p2 = p1->left;
	p1->left = p2->right;
	p2->right = p1;
	p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;
	p2->height = max(bsheight(p2->left),p1->height) + 1;
	return p2;
}
nodeptr bstree:: srr(nodeptr &p1)
{
	nodeptr p2;
	p2 = p1->right;
	p1->right = p2->left;
	p2->left = p1;
```

```
        p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;
        p2->height = max(p1->height,bsheight(p2->right)) + 1;
        return p2;
}
nodeptr bstree:: drl(nodeptr &p1)
{
        p1->left=srr(p1->left);
        return srl(p1);
}
nodeptr bstree::drr(nodeptr &p1)
{
        p1->right = srl(p1->right);
        return srr(p1);
}

int bstree::nonodes(nodeptr p)
{
        int count=0;
        if (p!=NULL)
        {
                nonodes(p->left);
                nonodes(p->right);
                count++;
        }
        return count;
}

int main()
{
        //clrscr();
        nodeptr root,root1,min,max;//,flag;
        int a,choice,findele,delele;
        char ch='y';
        bstree bst;

        //system("clear");
        root = NULL;
        root1=NULL;
        cout<<"\n\t\t\t\tАВЛ Дерево"<<endl;
        cout<<"\t\t\t\t:::::::::::::::::::\n"<<endl;

        do
        {
                cout<<"\t\t::::::::::::::::::::::::::::::::::::::::::::::::"<<endl;
                cout<<"\t\t::::1 Вставить новый узел::::::::::::::::"<<endl;
                cout<<"\t\t::::2 Найти минимальный элемент:::::::::::"<<endl;
                cout<<"\t\t::::3 Найти максимальный элемент::::::::::::::"<<endl;
                cout<<"\t\t::::4 Поиск по значению:::::::::::::::::::"<<endl;
                cout<<"\t\t::::5 Удалить элемент::::::::::::::::::"<<endl;
                cout<<"\t\t::::6 Вариант обхода1::::::::::::::::"<<endl;
                cout<<"\t\t::::7 Вариант обхода2:::::::::::::::::"<<endl;
                cout<<"\t\t::::8 Вариант обхода3::::::::::::::::"<<endl;
                cout<<"\t\t::::9 Показать высоту дерева:::"<<endl;
                cout<<"\t\t::::0 Выход::::::::::::::::::::::::::::::"<<endl;
                cout<<"\t\t::::::::::::::::::::::::::::::::::::::::::::::::\n"<<endl;

                cout<<"\nВыберите нужное действие и нажмите Enter: ";
                cin>>choice;
```

```
		switch(choice)
		{
			case 1:
				cout<<"\n\t\tДобавление нового узла"<<endl;
				cout<<"\t\t::::::::::::\n"<<endl;
				cout<<"Введите элемент: ";
				cin>>a;
				bst.insert(a,root);
				cout<<"\nНовый элемент добавлен успешно\n"<<endl;
				break;
			case 2:
				if (root !=NULL)
				{
					min=bst.findmin(root);
					cout<<"\nМинимальный элемент в дереве: "<<min-
>element<<endl;
				}
				break;
			case 3:
				if (root !=NULL)
				{
					max=bst.findmax(root);
					cout<<"\nМаксимальный элемент в дереве: "<<max-
>element<<endl;
				}
				break;
			case 4:
				cout<<"\nВведите искомый элемент: ";
				cin>>findele;
				if (root != NULL)
				{
					bst.find(findele,root);
				}
				break;
			case 5:
				cout<<"\nКакой узел удалять? : ";
				cin>>delele;
				bst.del(delele,root);
				bst.inorder(root);
				cout<<endl;
				break;
			case 6:
				cout<<"\n\t\tВариант обхода1"<<endl;
				bst.preorder(root);
				cout<<endl;
				break;
			case 7:
				cout<<"\n\t\tВариант обхода2"<<endl;
				bst.inorder(root);
				cout<<endl;
				break;
			case 8:
				cout<<"\n\t\tВарант обхода3"<<endl;
				bst.postorder(root);
				cout<<endl;
				break;
			case 9:
```

```
                cout<<"\n\t\tВЫСОТА\n"<<endl;
                cout<<"ТДерево имеет высоту: "<<bst.bsheight(root)<<endl;

                break;
            case 0:
                cout<<"\n\tБлагодарим вас за использование програмы\n"<<endl;
                break;
            default:
                cout<<"Sorry! wrong input\n"<<endl;
                break;
        }
        system("pause");
        system("cls");
    }while(choice != 0);
    return 0;
}
```