Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное учреждение высшего образования  
Московский Авиационный Институт  
(национальный исследовательский университет)

Институт №3  
Системы управления, информатика и электроэнергетика  
Кафедра 304

Отчет по лабораторной работе  
По учебной дисциплине  
«**Структуры и алгоритмы обработки данных**»

На тему  
**«Бинарные деревья поиска»**

Группа M30-224 Б-18

Выполнили:

Мариненко Г. В.  
Рогачев В. Н.

Проверил:  
[Силаев Андрей Владимирович](https://mai.ru/education/schedule/ppc.php?guid=70916f74-1d9a-11e0-9baf-1c6f65450efa)

Москва 2020

**Код программы:**

#include <iostream>

#include <fstream>

using namespace std;

int NumKeys = 25;

struct Node

{

int x;

Node \*Left,\*Right;

~Node(){cout << this->x << " deleted" << endl;};

};

void PrintTree(Node \*Tree)

{

if (Tree != nullptr)

{

PrintTree(Tree->Left);

PrintTree(Tree->Right);

cout << Tree->x << " ";

}

else return;

}

void DeleteTree(Node \*Tree)

{

if (Tree != nullptr)

{

DeleteTree(Tree->Left);

DeleteTree(Tree->Right);

delete Tree;

}

else return;

}

void AddNode(int x, Node \*&MyTree) //Функция добавления звена в дерево

{

if (MyTree == nullptr) //Если дерева нет

{

MyTree = new Node;

MyTree->x = x;

MyTree->Left = MyTree->Right = nullptr;

return;

}

if(x < MyTree->x) //Если нововведенный элемент меньше,

чем элемент корня дерева, то уходим влево

{

if(MyTree->Left != nullptr) AddNode(x, MyTree->Left);//Запись

элемента на свободную ячейку

else

{

MyTree->Left = new Node;

MyTree->Left->Left = MyTree->Left->Right = nullptr;

//Инициализация пустых подзвеньев

MyTree->Left->x = x;

}

}

//Для правой ветви

if(x > MyTree->x)

{

if(MyTree->Right != nullptr) AddNode(x,MyTree->Right);

else

{

MyTree->Right = new Node;

MyTree->Right->Left = MyTree->Right->Right = nullptr;

MyTree->Right->x = x;

}

}

}

Node\* FindElem(Node \*Tree, int DesiredValue)

{

Node \*CurrentElem = Tree;

while(CurrentElem->x != DesiredValue)

{

if(CurrentElem->x > DesiredValue)

{

if(CurrentElem->Left == nullptr)

{

cout << "Элемент в бинарном дереве не найден" << endl;

return CurrentElem;

}

CurrentElem = CurrentElem->Left;

}

else

{

if(CurrentElem->Right == nullptr)

{

cout << "Элемент в бинарном дереве не найден" << endl;

return CurrentElem;

}

CurrentElem = CurrentElem->Right;

}

}

cout << "Значение " << CurrentElem->x << " найдено в бинарном дереве

поиска" << endl;

return CurrentElem;

}

Node\* FindMax(Node \*Tree)

{

Node\* CurrentElem = Tree;

while (CurrentElem->Right != nullptr)

{

CurrentElem = CurrentElem->Right;

}

cout << "Максимальный ключ в дереве = " << CurrentElem->x << endl;

return CurrentElem;

}

Node\* FindMin(Node \*Tree)

{

Node\* CurrentElem = Tree;

while (CurrentElem->Left != nullptr)

{

CurrentElem = CurrentElem->Left;

}

return CurrentElem;

}

Node\* GetParent(Node \*Tree, int DesiredValue)

{

Node \*Parent = nullptr;

Node \*CurrentElem = Tree;

while(CurrentElem->x != DesiredValue)

{

Parent = CurrentElem;

if(CurrentElem->x > DesiredValue)

{

if(CurrentElem->Left == nullptr)

{

cout << "Элемент в бинарном дереве не найден" << endl;

return nullptr;

}

CurrentElem = CurrentElem->Left;

}

else

{

if(CurrentElem->Right == nullptr)

{

cout << "Элемент в бинарном дереве не найден" << endl;

return nullptr;

}

CurrentElem = CurrentElem->Right;

}

}

return Parent;

}

Node\* NextElem(Node \*CurrentElem)

{

return FindMin(CurrentElem);

}

void DeleteElem(Node \*&Tree, int ElemToBeDeleted)

{

Node \*ElemForDelete;

Node \*Parent;

ElemForDelete = FindElem(Tree, ElemToBeDeleted); //Поиск удаляемого

элемента в дереве

if((ElemForDelete->Left == nullptr) && (ElemForDelete->Right

== nullptr))//Если удаляемый элемент без дочерних (лист)

{

Parent = GetParent(Tree, ElemForDelete->x);

if(Parent != nullptr)

{

if(Parent->Left == ElemForDelete)

{

Parent->Left = nullptr;

delete ElemForDelete;

cout << "" << endl;

}

else

{

Parent->Right = nullptr;

delete ElemForDelete;

cout << "" << endl;

}

}

}

else if((ElemForDelete->Left == nullptr && ElemForDelete->Right !=

nullptr) ||

(ElemForDelete->Right == nullptr && ElemForDelete->Left !=

nullptr))//У родителя только 1 потомок

{

Parent = GetParent(Tree, ElemForDelete->x);

Node \*Child;

if(Parent != nullptr)//Если удаляемый элемент не корень дерева

{

if(ElemForDelete->Left != nullptr)

{

Child = ElemForDelete->Left;

}

else

{

Child = ElemForDelete->Right;

}

cout << "Child = " << Child->x << endl;

cout << "ElemFor Delete = " << &ElemForDelete << endl

<< "Parent.left = " << Parent->Left << endl;

if(Parent->Left == ElemForDelete)

{

Parent->Left = Child;

delete ElemForDelete;

cout << "" << endl;

}

else if(Parent->Right == ElemForDelete)

{

Parent->Right = Child;

delete ElemForDelete;

cout << "Parent = " << Parent->x << endl;

}

}

else if(Parent == nullptr)

{

if(ElemForDelete->Left != nullptr)

{

Tree = ElemForDelete->Left;

}

else

{

Tree = ElemForDelete->Right;

}

delete ElemForDelete;

}

}

else //Когда есть оба поддерева

{

Node\* Next = NextElem(ElemForDelete->Right);

Parent = GetParent(Tree, ElemForDelete->x);

Node\* Child = Next->Right;

Node\* ParentFromNext = GetParent(Tree, Next->x);

if(ParentFromNext == ElemForDelete)

{

Node\* Parent = GetParent(Tree, ElemForDelete->x);

Parent->Right = ElemForDelete->Right;

Parent->Right->Left = ElemForDelete->Left;

delete ElemForDelete;

}

else

{

ParentFromNext->Left = Child;

ElemForDelete->x = Next->x;

delete Next;

}

}

}

int main()

{

//Чтение ключей из файла

Node \*Tree = NULL;

int KeyArray[NumKeys];

int DesiredValue; //Искомая величина

int NewNode;

int choice = 0;

ifstream fin("Text.txt");

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

{

fin >> KeyArray[i];

}

fin.close();//Закрытие файла ключей

cout << "\t СОЗДАНИЕ БИНАРНОГО ДЕРЕВА ПОИСКА" << endl;

cout << "Массив ключей: " << endl;

cout << "Keys[" << NumKeys << "] = {";

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

{

cout << KeyArray[i] << " ";

}

cout << "}" << endl << endl;

for(int i = 0; i < NumKeys ;i++) AddNode(KeyArray[i], Tree);

cout << "Обратный обход дерева: " << endl;

PrintTree(Tree);

cout <<

"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

<< endl;

cout << endl;

while(choice < 5)

{

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "| 1. ПОИСК УЗЛА С ЗАДАННЫМ ЗНАЧЕНИЕМ КЛЮЧА|" << endl;

cout << "| 2. ВСТАВКА НОВОГО УЗЛА |" << endl;

cout << "| 3. ПОИСК МАКСИМАЛЬНОГО КЛЮЧА ДЕРЕВА |" << endl;

cout << "| 4. УДАЛЕНИЕ КОРНЯ ДЕРЕВА |" << endl;

cout << "| \t \*ДЛЯ ВЫХОДА НАЖМИТЕ КЛАВИШУ > 4\* |" << endl;

cout << "-------------------------------------------" << endl;

cout << "\t\t\t Выберите действие: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "\t 1. ПОИСК УЗЛА С ЗАДАННЫМ ЗНАЧЕНИЕМ КЛЮЧА"

<< endl;

cout << "Введите значение ключа для поиска соответствующего

узла: ";

cin >> DesiredValue;

FindElem(Tree, DesiredValue);

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

break;

case 2:

cout << "\n\t 2. ВСТАВКА НОВОГО УЗЛА" << endl;

cout << "Введите значение ключа: ";

cin >> NewNode;

AddNode(NewNode, Tree);

PrintTree(Tree);

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

<< endl;

cout << endl;

break;

case 3:

cout << "\n\t 3. ПОИСК МАКСИМАЛЬНОГО КЛЮЧА ДЕРЕВА" << endl;

FindMax(Tree);

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

break;

case 4:

cout << " 4. УДАЛЕНИЕ КОРНЯ ДЕРЕВА" << endl;

int ElemToBeDeleted;

cout << "Введите элемент, который хотите удалить: " << endl;

cin >> ElemToBeDeleted;

DeleteElem(Tree, ElemToBeDeleted);

cout << "Обратный обход дерева: " << endl;

PrintTree(Tree);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

<< endl;

break;

default:

break;

}

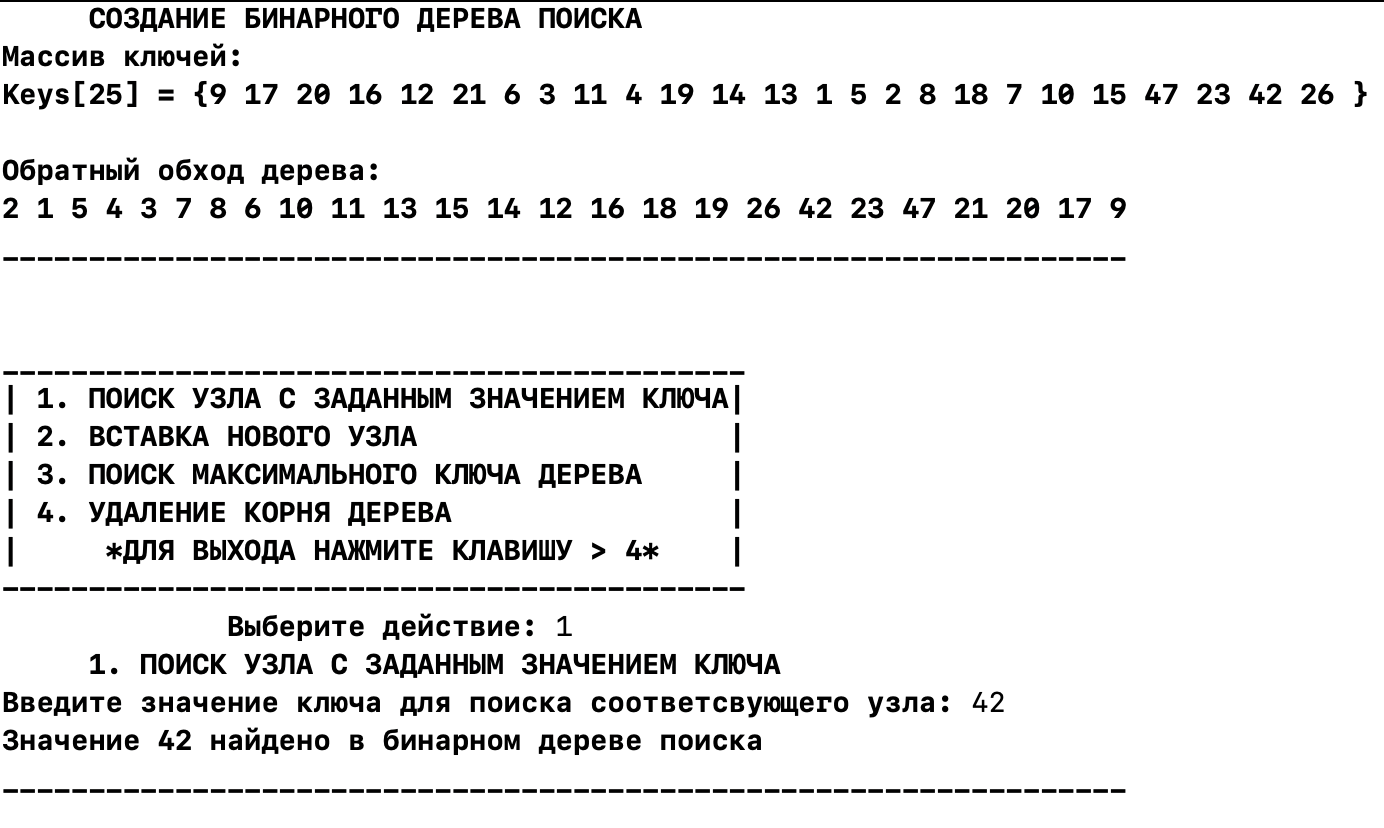
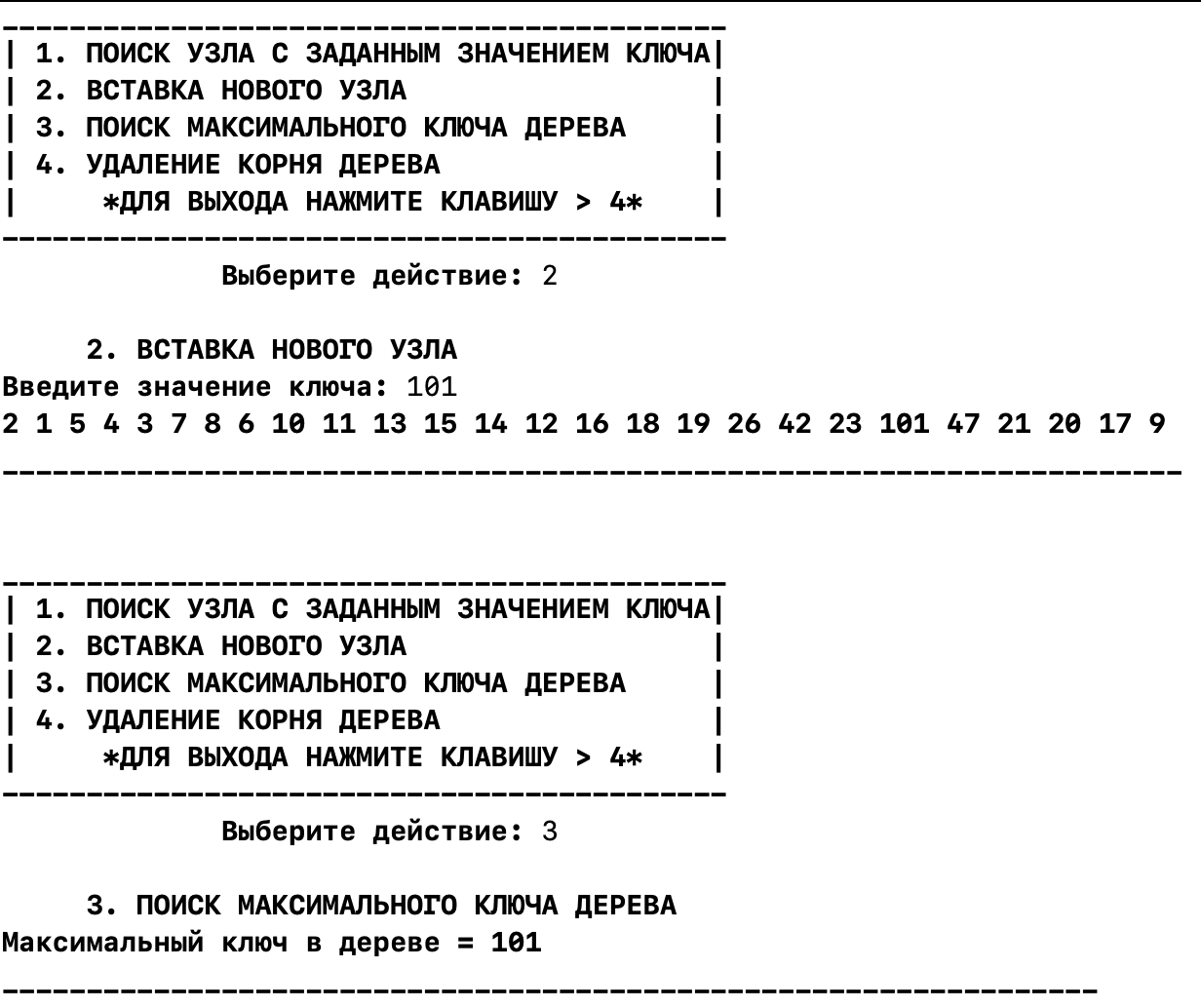
}

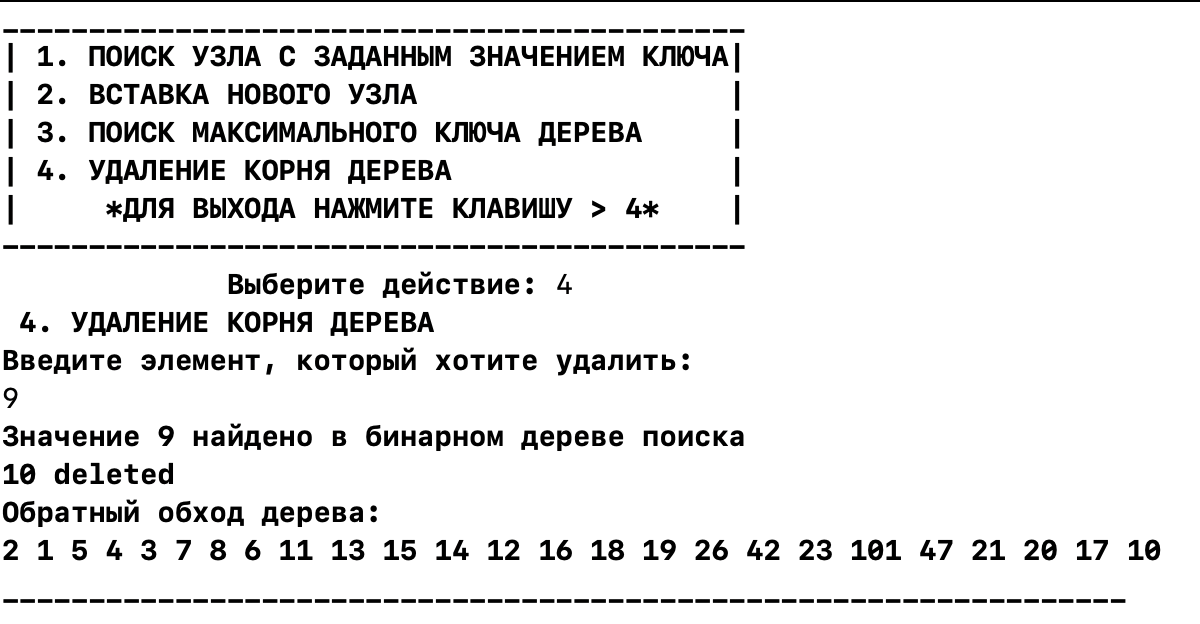
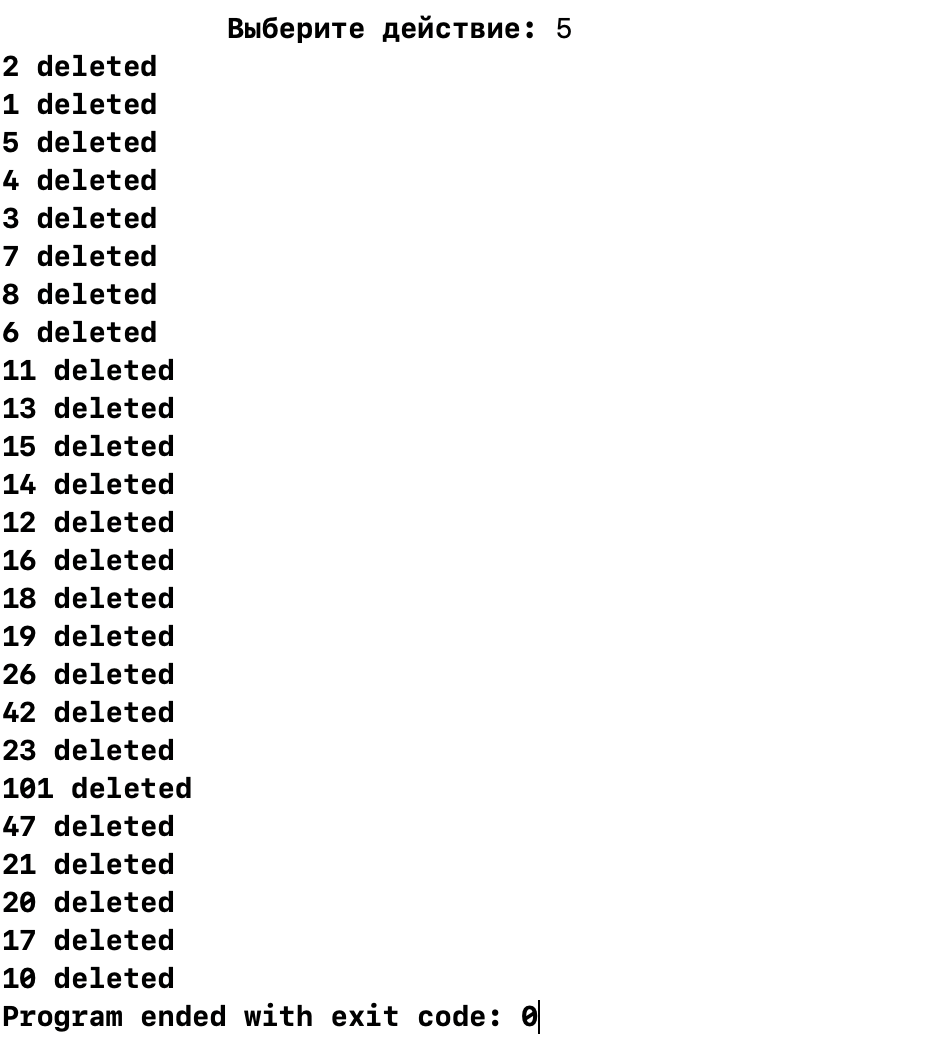
DeleteTree(Tree);

return 0;

}

**Примеры работы программы:**

**  
**

**  
**