Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**Деревья**

**Отчет по лабораторной работе №4**

**По дисциплине**

**«Структуры и алгоритмы обработки данных в ЭВМ»**

Выполнил: студент гр. 439-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зозуля Е.Д.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Проверил: ассистент каф. АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Яблонский Я.В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Томск 2020

# Задание на лабораторную работу

Вариант 7.

# Напишите программу, которая формирует бинарное дерево поиска, выводит построенное дерево на экран. Данные могут вводиться с клавиатуры, из файла или генерироваться с помощью генератора случайных чисел. Выбор способа ввода данных выполняется во время работы программы. В построенном дереве необходимо вычислить длину пути от корня до ближайшей вершины с элементом Е. Если элемент Е не входит в Т, то за ответ принять -1. Значение Е вводится с клавиатуры. Для реализации АТД «Дерево» используйте динамическое распределение памяти. Перед завершением работы программы освободить занимаемую динамическую память. Для этого используйте поэлементное удаление элементов динамической структуры данных.

# Алгоритм решения задачи

Выполняется обход с поиском элемента Е, где одновременно с переходом на левое или правое ответвление выполняется увеличение счётчика depth на 1, в случае если выполняется одно из двух условий depth присваивается значение -1 обход завершается.

1. Если Е меньше текущего элемента и при этом указатель на левый элемент равен NULL
2. Если Е больше текущего элемента и при этом указатель на правый элемент равен NULL

# Листинг программы

main.cpp

#include "BinaryTree.h"

#include <iostream>

#include <fstream>

#include <ctime>

using namespace std;

using namespace Me;

int main()

{

setlocale(LC\_CTYPE, "RUSSIAN");

srand(static\_cast<unsigned int>(time(0)));

BinaryTree tree;

cout << "Выберите режим: \n"

<<"1)Cчитать числа из файла \n"

<<"2)Ввести с клавиатуры\n"

<<"3)Сгенерировать с помощью генератора случайных чисел\n"

<<"Ваш выбор:";

int n;

cin >> n;

if (n == 1)

{

ifstream FileIn("Input.txt");

int rNum;

while (!FileIn.eof())

{

FileIn >> rNum;

tree.addNode(rNum);

}

//считали числа из файла в дерево

}

else if (n == 2)

{

cout << "Введите необходимое кол-во чисел: ";

int num;

cin >> num;

cout << "Введите числа:\n";

int info;

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

{

cin >> info;

tree.addNode(info);

}

}

else if (n == 3)

{

int num;

cout << "Введите необходимое кол-во чисел: ";

cin >> num;

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

{

tree.addNode(rand() % 100);

}

}

tree.Print();

tree.Count();

cin >> n;

cout << "\nГлубина:" << tree.getE(n) << endl;

}

BinaryTree.h

namespace Me

{

struct Node

{

int info;

Node\* left;

Node\* right;

};

class BinaryTree

{

public:

BinaryTree(); //конструктор

~BinaryTree(); //деструктор

void addNode(const int k);

void deleteNode(int k);

void Print();

int Count();

bool IsEmpty();

Node\* find(int k);

void prefixPrint(); //прямой обход (и печать)

void infixPrint(); //симметричный обход (и печать)

void postfixPrint(); //обратный обход (и печать)

double getE(int x); //среднее арефмитическое всех чисел

private:

Node\* root;

Node\* recurAddNode(Node\* &currentRoot, const int k);

void Delete(Node\* &T, int k);

void Del(Node\* &T, Node\* &P);

void recurPrint(Node\* currentRoot, int h);

void recurCount(Node\* currentRoot,int &count);

Node\* recurFind(Node\* currentRoot, int k);

void recurPrefixPrint(Node\* currentRoot);

void recurInfixPrint(Node\* currentRoot);

void recurPostfixPrint(Node\* currentRoot);

void recurDeleteTree(Node\* &currentRoot);

void recurFindE(Node\* currentRoot, int &k, int &depth, int &cont);

};

}

BinaryTree.cpp

#include "BinaryTree.h"

#include <iostream>

using namespace std;

namespace Me

{

BinaryTree::BinaryTree()

{

root = NULL;

}

BinaryTree::~BinaryTree()

{

this->recurDeleteTree(this->root);

this->root = NULL;

}

void BinaryTree::addNode(const int k)

{

if (root == NULL)

{

root = (Node\*)malloc(sizeof(Node));

root->info = k;

root->left = NULL;

root->right = NULL;

}

else

{

this->recurAddNode(root, k);

}

}

void BinaryTree::deleteNode(int k)

{

this->Delete(this->root, k);

}

void BinaryTree::Print()

{

if (root == NULL)

{

cout << "Список пуст\n";

}

else

{

this->recurPrint(this->root, 0);

}

}

bool BinaryTree::IsEmpty()

{

return (root == NULL);

}

Node\* BinaryTree::find(int k)

{

return (this->recurFind(this->root, k));

}

void BinaryTree::prefixPrint() //прямой

{

this->recurPrefixPrint(this->root);

cout << endl;

}

void BinaryTree::infixPrint() //симметричный

{

this->recurInfixPrint(this->root);

cout << endl;

}

void BinaryTree::postfixPrint() //обратный

{

this->recurPostfixPrint(this->root);

cout << endl;

}

int BinaryTree::Count()

{

int count = 0;

if (root == NULL)

{

cout << "Список пуст\n";

return count;

}

else

{

this->recurCount(this->root, count);

return count;

}

}

double BinaryTree::getE(int x)

{

int depth = 0;

int cont = 0;

recurFindE(this->root, x, depth, cont);

return depth;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Node\* BinaryTree::recurAddNode(Node\* &currentRoot, const int k)

{

if (currentRoot == NULL)

{

currentRoot = (Node\*)malloc(sizeof(Node));

currentRoot->info = k;

currentRoot->left = NULL;

currentRoot->right = NULL;

}

else

{

if (k < currentRoot->info)

{

currentRoot->left = recurAddNode(currentRoot->left, k);

}

else

{

currentRoot->right = recurAddNode(currentRoot->right, k);

}

}

return currentRoot;

}

void BinaryTree::Delete(Node\* &T, int k)

{

if (T != NULL)

{

if (k < T->info)

{

Delete(T->left, k);

}

else if (k > T->info)

{

Delete(T->right, k);

}

else

{

if ((T->left == NULL) && (T->right == NULL))

{

Node\* P = T;

T = NULL;

free(P);

}

else if (T->left == NULL)

{

Node\* P = T;

T = T->right;

free(P);

}

else if (T->right == NULL)

{

Node\* P = T;

T = T->left;

free(P);

}

else

{

Del(T->right, T);

}

}

}

}

void BinaryTree::Del(Node\* &T, Node\* &P)

{

if (T->left == NULL)

{

P->info = T->info;

Node\* Pl = T;

T = T->right;

free(Pl);

}

else

{

Del(T->left, P);

}

}

void BinaryTree::recurPrint(Node\* currentRoot, int h)

{

if (currentRoot != NULL)

{

this->recurPrint(currentRoot->right, h + 1);

for (int i = 0; i < h; i++) printf("\t");

printf("%d\n", currentRoot->info);

this->recurPrint(currentRoot->left, h + 1);

}

}

Node\* BinaryTree::recurFind(Node\* currentRoot, int k)

{

if ((k == currentRoot->info) || (currentRoot == NULL))

return currentRoot;

if (k < currentRoot->info)

{

return recurFind(currentRoot->left, k);

}

else

{

return recurFind(currentRoot->right, k);

}

}

void BinaryTree::recurPrefixPrint(Node\* currentRoot)

{

if (currentRoot != NULL)

{

cout << currentRoot->info << "; ";

recurPrefixPrint(currentRoot->left);

recurPrefixPrint(currentRoot->right);

}

}

void BinaryTree::recurInfixPrint(Node\* currentRoot)

{

if (currentRoot != NULL)

{

recurInfixPrint(currentRoot->left);

cout << currentRoot->info << "; ";

recurInfixPrint(currentRoot->right);

}

}

void BinaryTree::recurPostfixPrint(Node\* currentRoot)

{

if (currentRoot != NULL)

{

recurPostfixPrint(currentRoot->left);

recurPostfixPrint(currentRoot->right);

cout << currentRoot->info << "; ";

}

}

void BinaryTree::recurDeleteTree(Node\* &currentRoot)

{

if (currentRoot != NULL)

{

recurDeleteTree(currentRoot->left);

recurDeleteTree(currentRoot->right);

free(currentRoot);

currentRoot = NULL;

}

}

void BinaryTree::recurCount(Node\* currentRoot,int &count)

{

if (currentRoot != NULL)

{

this->recurCount(currentRoot->right, count += 1);

this->recurCount(currentRoot->left, count += 1);

}

}

void BinaryTree::recurFindE(Node\* currentRoot, int& k, int& depth, int& cont)

{

if ((k == currentRoot->info) || (currentRoot == NULL))

{

return;

}

if (k < currentRoot->info && currentRoot->left == NULL)

{

depth = -1;

return;

}

else if (k > currentRoot->info && currentRoot->right == NULL)

{

depth = -1;

return;

}

if (k < currentRoot->info)

{

return recurFindE(currentRoot->left, k, depth += 1, cont+=1);

}

else

{

return recurFindE(currentRoot->right, k, depth += 1, cont += 1);

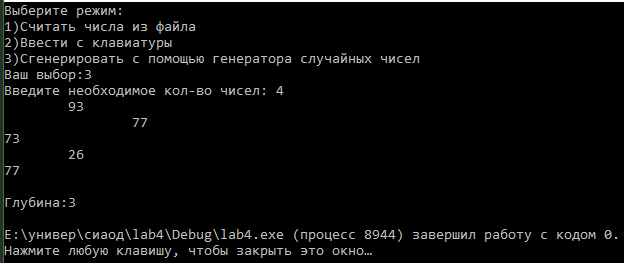
}

}

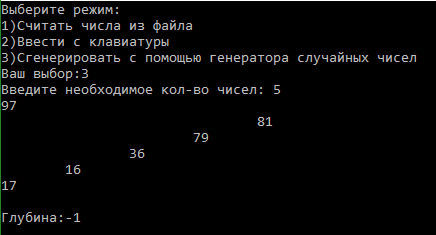
}

# Пример решения

Скриншоты успешной работы программы, входные данные тестового примера, выходные данные на рисунке 4.1 и 4.2.



*Рисунок 4.1.- Поиск существующего элемента*



*Рисунок 4.2.- Поиск несуществующего элемента*

# Вывод

Было изучено АТД дерево и использовано для решения задачи.