**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных».**

Тема: «Деревья»

Вариант 4 д.

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. 5383 |  | Допира В. Е. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2016

**Лабораторная работа 4**

**Деревья**

**Вариант 4**

**Задача:** Задано бинарное дерево b типа ВТ с произвольным типом элементов. Используя очередь и операции над ней, напечатать все элементы дерева b по уровням: сначала − из корня дерева, затем (слева направо) − из узлов, сыновних по отношению к корню, затем (также слева направо) − из узлов, сыновних по отношению к этим узлам, и т. д.

**2.Спецификация программы**

**2.1 Входные данные**

**Структуры данных:** в программе данные хранятся в структуре (динамическая реализация):

struct TreeNode // данные

{

char value;

TreeNode \*left;

TreeNode \*right;

TreeNode() { left = NULL; right = NULL; } // constructor

};

Работа с бинарным деревом осуществляется с помощью класса

class BinaryTree

{

private:

TreeNode \*head;

TreeNode \*link;

public:

TreeNode\* Create();

bool isNull();

void headBT();//определение вершины дерева

char RootBT(binTree b); // для непустого бин.дерева

TreeNode\* headRootBT( ) ;

binTree Left(binTree b);

binTree Right(binTree b);

void displayBT(binTree, ID );

void printTree();

void displayQueue(binTree , ID ); // формирование вектора

void displayQueueTree();

int getMaxDepth(binTree b, ID n); // глубина дерева

int getCount(binTree b, ID n); //общее количество не нулевых вершин

TreeNode\* ConsBT(const char &x, binTree &lst, binTree &rst);

TreeNode\* enterBT();

void destroyEL(binTree b);

void destroyTree();

};

**2.2Выходные данные**

Программа выводит на экран и записывает в файл элементы дерева по уровням.

**3.Алгоритм и структуры данных**

Программа выполняет следующую последовательность действий:

|  |
| --- |
| Функция создания дерева Create()  {  Присвоить текущему узлу NULL  Вернуть голову  } |
| Функция проверки пустое ли дерево isNull()  {  Вернуть ссылку = NULL  } |
| Функция определения вершины дерева headBT( )  {  Присвоить голове адрес текущего узла  } |
| Функция возвращения символа в текущей вершине для непустого бинарного дерева RootBT ( )  {  Если (дерево пустое) вывод сообщения об ошибки: «Error: RootBT(null)»  Иначе вернуть символ в текущей вершине  } |
| Функция возвращения головы для непустого бинарного дерева headRootBT ( )  {  Если (голова пуста) вывод сообщения об ошибки: «Error: headRootBT(null)»  Иначе вернуть голову  } |
| Функция переход к левому поддереву для непустого бинарного дерева Left ( )  {  Если (дерево пустое) вывод сообщения об ошибки: «Error: Left(null)»  Иначе вернуть и перейти к левой вершине  } |
| Функция переход к правому поддереву для непустого бинарного дереваRight ( )  {  Если (дерево пустое) вывод сообщения об ошибки: «Error: Right(null)»  Иначе вернуть и перейти к правой вершине  } |
| Функция формирования узла левой и правой ветви ConsBT ( )  {  Текущей ссылке присваиваем адрес нового узла  Если (элемент непустой)  {  Переход к корню дерева  Переход к левому элементу  Переход к правому элементу  Вернуть текущую ссылку  }  Иначе вывод сообщения «Memory not enough»  } |
| Функция ввода дерева из файла enterBT ( )  {  Формирование узлов  Если (файл не открыт) вывод сообщения: «Файл не может быть открыт!»  Иначе{  Пока не конец файла  {  Считываем с конца, пока не встретим «/»  Как только встретим «/» возвращаем NULL  Иначе {формирование узлов: p- нижняя ветвь, q - нижняя правая ветвь}  Рекурсивно вызываем функцию ConsBT ( )  Вернуть текущую ссылку  }  Закрыть файл  }  } |
| Функция вывод бинарного дерева на экран displayBT ( )  {  Если (дерево непустое)  {  Вывод корней с помощью функции RootBT()  Если (Правый узел непустой) { рекурсивный вызов функции displayBT ( )}  Иначе переход на следующую строку  Если (Левый узел непустой) {вывод пробелов  рекурсивный вызов функции displayBT ( )}  }  Иначе {}  } |
| Функция печати бинарного дерева на экран printTree ( )  {  Элементу дерева присвоить голову с помощью функции headRootBT()  Вызов функции displayBT ( )  } |
| Функция удаления элемента дерева destroyEL ( )  {  Если (дерево непустое)  {  Удаление левого элемента рекурсивно с помощью функции destroyEL ( )  Удаление правого элемента рекурсивно с помощью функции destroyEL ( )  Удаление вершины  Присвоить корню NULL  }  } |
| Функция удаления дерева destroyTree ( )  {  Рекурсивное с помощью функции destroyEL ( ) узлов headRootBT()  } |
| Функция формирования вектора displayQueue( )  {  Инициализация статического счетчика  Если (дерево непустое)  {  Создание вектора и записываем туда вершины с помощью функции RootBT()  Создание вектора для записывания уровней  Увеличение счетчика  Если (Правый узел непустой) рекурсивный вызов функции displayQueue()  Если (Левый узел непустой) рекурсивный вызов функции displayQueue()  }  } |
| Функция подсчета общего количества ненулевых вершин getCount()  {  Инициализация статического счетчика  Если (дерево непустое)  {  Увеличение счетчика  Рекурсивный вызов функции getCount() для левого узла  Рекурсивный вызов функции getCount() для правого узла  }  Вернуть количество ненулевых вершин  } |
| Функция обхода дерева getMaxDepth()  {  Если (дерево пустое) {Вернуть уровень, на котором находится узел}  Иначе вернуть максимальную глубину из вершин левого и правого узлов  } |
| Функция печати элементов дерева по уровням displayQueueTree()  {  Инициализировать корень с помощью функции headRootBT()  Общее количество не нулевых вершин с помощью функции getCount()  Инициализировать уровень, на котором находится узел  Формирование вектора с помощью функции displayQueue() узла headRootBT()  Печать общего количества ненулевых вершин  Для (счетчика от 0 до количества ненулевых вершин)  {  Вывод уровней вершин  Создание вектора и обход с помощью функции getMaxDepth узла headRootBT()  Для(счетчика от 0 до максимальной глубины)  {  Вывод и записывание в файл глубины  Для(счетчика от 0 до количества ненулевых вершин)  {  Занесение элемента в очередь  Вывод и записывание в файл элементов  }  }  Пока (очередь непустая)  {  Вывод первого элемента и его извлечение из очереди  }  }  } |
| ()  {  Корректное отображение кириллицы  Создание бинарного дерева с помощью функции Create()  Ввод из файла с помощью функции enterBT()  Определение вершины дерева с помощью функции headBT()  Проверка, пустое ли дерево с помощью функции isNull( ), и если так, то вывод сообщения: «Пустое БД»  Иначе  {  Вывод сообщения «Бинарное дерево: "  Печать дерева с помощью функции printTree()  Реализация задания с помощью функции displayQueueTree()  Удаление дерева с помощью функции destroyTree() и вывод сообщения об этом  }  Вернуть 0  } |

**Функции, используемые в программе**

TreeNode\* BinaryTree::Create()

Назначение: создание дерева

Возвращаемое значение: head - голова.

bool BinaryTree::isNull()

Назначение: проверка пустое ли дерево

Возвращаемое значение: пустой текущий узел.

void BinaryTree::headBT( )

Назначение: определение вершины дерева

char BinaryTree::RootBT(binTree b )

Назначение: возвращение символа в текущей вершине для непустого бинарного дерева

Параметры: binTree b – бинарное дерево

Возвращаемое значение символ в текущей вершине.

TreeNode\* BinaryTree::headRootBT( )

Назначение: возвращение головы для непустого бинарного дерева

Возвращаемое значение голова на дерево.

binTree BinaryTree::Left(binTree b)

Назначение: переход к левому поддереву для непустого бинарного дерева

Параметры: binTree b – бинарное дерево

Возвращаемое значение переход к левой вершине.

binTree BinaryTree::Right(binTree b)

Назначение: переход к правому поддереву для непустого бинарного дерева

Параметры: binTree b – бинарное дерево

Возвращаемое значение переход к правой вершине.

TreeNode\* BinaryTree::ConsBT(const char &x, binTree &lst, binTree &rst)

Назначение: формирование узла левой и правой ветви

Параметры: const char &x – корень дерева

binTree &lst – указатель на левый элемент

binTree &rst – указатель на правый элемент

Возвращаемое значение текущая ссылка с адресом узла.

TreeNode\* BinaryTree::enterBT()

Назначение: ввод дерева из файла

Возвращаемое значение текущая ссылка с адресом узла.

void BinaryTree::displayBT(binTree b, ID n)

Назначение: вывод бинарного дерева на экран

Параметры: binTree b – бинарное дерево

ID n – уровень, на котором находится узел

Возвращаемое значение нет.

void BinaryTree::printTree()

Назначение: печать бинарного дерева

Возвращаемое значение нет.

void BinaryTree::destroyEL(binTree b)

Назначение: удаление элемента в бинарном дереве

Параметры: binTree b – бинарное дерево

Возвращаемое значение нет.

void BinaryTree::destroyTree()

Назначение: удаление дерева

Возвращаемое значение нет.

void BinaryTree::displayQueue(binTree b, ID n)

Назначение: формирование вектора

Параметры: binTree b – бинарное дерево

ID n – уровень, на котором находится узел

Возвращаемое значение нет.

int BinaryTree::getCount(binTree b, ID n)

Назначение: подсчет общего количества ненулевых вершин

Параметры: binTree b – бинарное дерево

ID n – уровень, на котором находится узел

Возвращаемое значение i – количество ненулевых вершин.

int BinaryTree::getMaxDepth(binTree b, ID n)

Назначение: обход дерева

Параметры: binTree b – бинарное дерево

ID n – уровень, на котором находится узел

Возвращаемое значение максимальную глубину из вершин левого и правого узлов.

void BinaryTree::displayQueueTree()

Назначение: печать элементов дерева по уровням

Возвращаемое значение нет.

**4. Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Входные данные** | | **Результат** | |
| **Тесты** | **Бинарное дерево** | **Ожидаемый** | **Полученный** |
| 1 |  | Файл не может быть открыт!  Пустое БД | Файл не может быть открыт!  Пустое БД |
| 2 | ab/// | Глубина0 a  Глубина1 b | Глубина0 a  Глубина1 b |
| 3 | a/b// | Глубина0 a  Глубина1 b | Глубина0 a  Глубина1 b |
| 4 | abd//c//ef/gi//kj//// | Глубина0 a  Глубина1 e b  Глубина2 f c d  Глубина3 g  Глубина4 k i  Глубина5 j | Глубина0 a  Глубина1 e b  Глубина2 f c d  Глубина3 g  Глубина4 k i  Глубина5 j |
| 5 | abd/h//e//cfi//j//g/kl/// | Глубина0 a  Глубина1 c b  Глубина2 g f e d  Глубина3 k j i h  Глубина4 l | Глубина0 a  Глубина1 c b  Глубина2 g f e d  Глубина3 k j i h  Глубина4 l |
| 6 | ad/ej/k//fl///bg/h//cim/n//// | Глубина0 a  Глубина1 b d  Глубина2 c g e  Глубина3 i h f j  Глубина4 m l k  Глубина5 n | Глубина0 a  Глубина1 b d  Глубина2 c g e  Глубина3 i h f j  Глубина4 m l k  Глубина5 n |

1. **Текст программы:**

#include<iostream>

#include <fstream>

#include <string.h>

#include <vector>

#include <queue>

using namespace std;

ifstream in("in.txt");

ofstream fout("out.txt");

typedef int ID;

vector<int> used (20);

vector<char> d (20);

struct TreeNode // данные

{

char value;

TreeNode \*left;

TreeNode \*right;

TreeNode() { left = NULL; right = NULL; } // constructor

};

typedef TreeNode \*binTree; // "представитель" бинарного дерева

class BinaryTree

{

private:

TreeNode \*head;

TreeNode \*link;

public:

TreeNode\* Create();

bool isNull();

void headBT();//определение вершины дерева

char RootBT(binTree b); // для непустого бин.дерева

TreeNode\* headRootBT( ) ;

binTree Left(binTree b);

binTree Right(binTree b);

void displayBT(binTree, ID );

void printTree();

void displayQueue(binTree , ID ); // формирование вектора

void displayQueueTree();

int getMaxDepth(binTree b, ID n); // глубина дерева

int getCount(binTree b, ID n); //общее количество не нулевых вершин

TreeNode\* ConsBT(const char &x, binTree &lst, binTree &rst);

TreeNode\* enterBT();

void destroyEL(binTree b);

void destroyTree();

};

TreeNode\* BinaryTree::Create() //создание дерева

{

link=head=NULL;

return head;

}

bool BinaryTree::isNull()

{

return (link == NULL);

}

void BinaryTree::headBT( ) //определение вершины дерева

{

head=link;

}

char BinaryTree::RootBT(binTree b ) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }

else { return b->value; } //возвращает символ в текущей вершине

}

TreeNode\* BinaryTree::headRootBT( ) // для непустого бин.дерева

{

if (head == NULL) { cerr << "Error: headRootBT(null) \n"; exit(1); }

else {return head; } //возвращаем голову на дерево

}

//переходы по дереву

binTree BinaryTree::Left(binTree b) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }

else return b->left;

}

binTree BinaryTree::Right(binTree b) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }

else return b->right;

}

//формирование узла левой и правой ветви

TreeNode\* BinaryTree::ConsBT(const char &x, binTree &lst, binTree &rst)

{//текущей ссылке присваиваем адрес нового узла

binTree p;

p = new TreeNode;

if (p != NULL) {

p->value = x;

p->left = lst;

p->right = rst;

link=p;

return link;//текущая ссылка

}

else { cerr << "Memory not enough\n"; exit(1); }

}

TreeNode\* BinaryTree::enterBT() //ввод дерева из файла

{

char ch;

TreeNode\* p;

TreeNode\* q;

if (!in.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else

{ while(!in.eof())

{

in >> ch;

cout<<ch;//считывание с конца, пока не "/"

if (ch == '/'){ return NULL; }

else { p = enterBT(); //формирование узла

q = enterBT();//нижняя правая ветвь - q, p- нижняя ветвь

link=ConsBT(ch, p, q);

return link; }

}

in.close(); //закрытие файла

}

}

void BinaryTree::displayBT(binTree b, ID n)

{ // n - уровень узла

if (b != NULL) {

cout << ' ' << RootBT(b);

if (Right(b)!= NULL) { displayBT(Right(b), n + 1); }

else cout << endl; // вниз

if (Left(b)!= NULL) {

for (int i = 1; i <= n; i++) cout << " "; // вправо

displayBT(Left(b), n + 1);

}

}

else {};

}

void BinaryTree::printTree()

{

binTree b=headRootBT();

ID n=1;

displayBT( b, n);

}

void BinaryTree::destroyEL(binTree b)

{

if (b != NULL) {

destroyEL(b->left);

destroyEL(b->right);

delete b;

b = NULL;

}

}

void BinaryTree::destroyTree()

{

destroyEL(headRootBT());

}

void BinaryTree::displayQueue(binTree b, ID n) // формирование вектора

{

static int i=0; //при последующих вызовах наращивается

if (b != NULL) {

d[i]=RootBT(b);

used[i]=n;

i++;

if (Right(b)!= NULL) displayQueue(Right(b), n + 1);

if (Left(b)!= NULL) displayQueue(Left(b), n + 1);

}

}

int BinaryTree::getCount(binTree b, ID n) //общее количество ненулевых вершин

{

static int i=0;

if (b != NULL){i++;getCount(b->left, n + 1);getCount(b->right, n + 1);}

return i;

}

int BinaryTree::getMaxDepth(binTree b, ID n) // глубина дерева

{

if (b == NULL){ return n; }

else return max(getMaxDepth(b->left, n + 1), getMaxDepth(b->right, n + 1));

}

void BinaryTree::displayQueueTree()

{

binTree b=headRootBT();

int vecSize= getCount(b, 0); //общее количество не нулевых вершин(кол-во символов в строке ввода)

ID n=0;

displayQueue( headRootBT(), 0); // формирование вектора

cout << "Вектор "<<endl<<"Общее количество ненулевых вершин "<<vecSize<<endl;

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

{ cout << " "<<used[i]; } //уровни у вершин

cout<<endl;

queue<char> q;

int Depth=getMaxDepth(headRootBT(), 0);

for (int i = 0; i < Depth; i++) //по глубине

{ cout<<"Глубина "<<i<<endl;

fout<<"Глубина"<<i<<endl;

for (int j = 0; j < vecSize; j++) // по элементам вектора

if (used[j]==i) {q.push(d[j]); // занесение элемента в очередь

cout<<d[j]<<" ";

fout<<d[j]<<endl;

}

cout<<endl;

}

cout <<endl<< "Очередь "<< endl;

while (!q.empty())

{cout<< q.front()<<" "; //вывод первого элемента

q.pop(); //и его извлечение

}

cout << endl;

}

int main()

{

setlocale(LC\_ALL, "rus"); // корректное отображение Кириллицы

BinaryTree Bin;

Bin.Create();

Bin.enterBT();

Bin.headBT();

ID depth = 1;

if (Bin.isNull( )) cout << "Пустое БД" << endl;

else

{

cout << endl<<"Бинарное дерево: " << endl;

Bin.printTree();

Bin.displayQueueTree();

cout<<"Удаление"<<endl;

Bin.destroyTree();

}

return 0;

}

**6. Вывод**

В ходе данной работы были получены практические навыки работы с бинарными деревьями и реализации алгоритма, используя очередь в динамическую памяти, на языке программирования С++.