**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

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

Тема: «Бинарное дерево поиска»

Вариант 9.

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

Санкт-Петербург

2016

**Лабораторная работа 5**

**Вариант 9**

**Задача:** БДП: случайное\* БДП; действие: 1+2в.

1. По заданному файлу F (типа file of Elem), все элементы которого различны, построить БДП определённого типа;

2. в) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.

**2.Спецификация программы**

**2.1 Входные данные**

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

struct TreeNode // данные

{

char value; //текущий элемент

TreeNode \*left; //ссылка на левый

TreeNode \*right; //ссылка на правый

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

};

Работа с бинарным деревом поиска осуществляется с помощью класса:

class BinaryTree

{

private:

TreeNode \*head;

TreeNode \*link;

public:

TreeNode\* Create(); //создание дерева

bool isNull(); //проверка на пустоту

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 obhodBT(binTree b); // Обход и вывод по возрастанию

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

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

TreeNode\* enterBT();

TreeNode\* newElBT( char x);

TreeNode\* enterElBT(char x , binTree b);

void destroyEL(binTree b);

void destroyTree();

};

**2.2Выходные данные**

Программа выводит построенное бинарное дерево поиска на экран и записывает в файл элементы дерева в порядке возрастания.

**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 перейти к левой вершине  } |
| Функция переход к правому поддереву для непустого бинарного дереваRight ( )  {  Если (дерево пустое) вывод сообщения об ошибки: «Error: Right(null)»  Иначе перейти к правой вершине  } |
| Функция ввода листа newElBT()  {  Выделить память под новый элемент  Если (вершина непустая)  {  Добавить элемент в дерево и перейти к следующему узлу  }  } |
| Функция ввода вершины enterElBT()  {  Если (дерево непустое)  {  Последующий элемент ≤ предыдущего, переход рекурсивно к левому поддереву с помощью функции enterElBT()  В противном случае переходим вправо с помощью функции enterElBT()  }  Добавляем элемент с помощью функции newElBT()  } |
| Функция ввода дерева из файла enterBT ( )  {  Формирование узла  Если (файл не открыт) вывод сообщения: «Файл не может быть открыт!»  Иначе{  Считываем элементы, пока не конец файла  Если (узел непустой) Переходим по узлам функцией headRootBT() и выводим элементы с помощью функции enterElBT ( )  Иначе добавление элементов с помощью функции newElBT()  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 obhodBT( )  {  Если (дерево непустое)  {  Левый узел непустой, тогда рекурсивный вызов функции obhodBT()  Вывод элемента на экран и записывание в файл  Аналогичный обход правого поддерева с помощью функции obhodBT()  }  } |
| Функция подсчета общего количества ненулевых вершин getCount()  {  Если (дерево непустое)  {  Подсчет количества ненулевых вершин  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  } |
| ()  {  Корректное отображение кириллицы  Создание бинарного дерева с помощью функции Create()  Ввод из файла с помощью функции enterBT()  Проверка, пустое ли дерево с помощью функции isNull( ), и если так, то вывод сообщения: «Пустое БД»  Иначе  {  Вывод бинарного дерева поиска  Печать элементов БДП в порядке возрастания функцией obhodBT ()  Удаление дерева с помощью функции destroyTree() и вывод сообщения об этом  }  } |

**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::newElBT( char x)

Назначение: ввод листа

Параметры: int x – корень дерева

Возвращаемое значение: лист.

TreeNode\* BinaryTree::enterElBT(char x , binTree b)

Назначение: ввод вершины

Параметры: char x – корень дерева

binTree b– бинарное дерево

Возвращаемое значение дерево.

TreeNode\* BinaryTree::enterBT()

Назначение: ввод дерева из файла

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

void BinaryTree::printTree()

Назначение: печать бинарного дерева

void BinaryTree::destroyEL(binTree b)

Назначение: удаление элемента в бинарном дереве

Параметры: binTree b – бинарное дерево

void BinaryTree::destroyTree()

Назначение: удаление дерева

void BinaryTree::obhodBT(binTree b)

Назначение: вывод элементов в порядке возрастания

Параметры: binTree b– бинарное дерево

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.

**4. Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Входные данные** | | **Результат** | |
| **Тесты** | **Бинарное дерево** | **Ожидаемый** | **Полученный** |
| 1 |  | Файл не может быть открыт! | Файл не может быть открыт! |
| 2 | 5 3 7 6 9 2 1 3 4 0 8 m n | 0 1 2 3 3 4 5 6 7 8 9 m n | 0 1 2 3 3 4 5 6 7 8 9 m n |
| 3 | 1 7 9 0 2 | 0 1 2 7 9 | 0 1 2 7 9 |
| 4 | 129876poi305 | 0 1 2 3 5 6 7 8 9 i o p | 0 1 2 3 5 6 7 8 9 i o p |
| 5 | 01k734m82o59 | 0 1 2 3 4 5 7 8 9 k m o | 0 1 2 3 4 5 7 8 9 k m o |
| 6 | 38rnfkv76g2w,90! | ! , 0 2 3 6 7 8 9 f g k n r v w | ! , 0 2 3 6 7 8 9 f g k n r v w |
| 7 | 16d35a80 | 0 1 3 5 6 8 a d | 0 1 3 5 6 8 a d |

1. **Текст программы:**

#include<iostream>

#include <fstream>

#include <string.h>

using namespace std;

ifstream in("in.txt");

ofstream fin("out.txt");

typedef int ID;

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 obhodBT(binTree b); // Обход и вывод по возрастанию

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

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

TreeNode\* enterBT();

TreeNode\* newElBT(char x);

TreeNode\* enterElBT(char x , binTree b);

void destroyEL(binTree b);

void destroyTree();

};

//функции

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; }

}

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

{

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

else {

return head;

}

}

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;

}

TreeNode\* BinaryTree::newElBT(char x) //ввод листа

{

binTree p = new TreeNode;

if (p != NULL) {

p->value = x;

p->left = NULL;

p->right = NULL;

return p;

}

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

}

TreeNode\* BinaryTree::enterElBT(char x , binTree b) //ввод вершины

{

if (b!=NULL) { cout<<"Элемент "<<b->value<<" -> ";

if (x <= b->value) { b->left=enterElBT(x,b->left); }

else { b->right=enterElBT(x,b->right); }

}

else {

b = newElBT(x);

cout<<" добавлен элемент "<<b->value << endl;

}

return b;

}

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

{

char ch;

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

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

else

{

while(in >> ch )

{

if (!isNull()){ link=headRootBT();

link = enterElBT(ch,headRootBT());}

else {

link = newElBT(ch);

headBT(); }

}

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::obhodBT(binTree b)

{

if (b != NULL) {

if (Left(b)!= NULL) { obhodBT(Left(b)); }

cout <<"Элемент"<< ' ' << RootBT(b)<<endl;

fin<<RootBT(b)<<" ";

if (Right(b)!= NULL) { obhodBT(Right(b)); }

}

}

void BinaryTree::destroyEL(binTree b)

{

if (b != NULL) {

destroyEL(b->left);

destroyEL(b->right);

delete b;

b = NULL;

}

}

void BinaryTree::destroyTree()

{

destroyEL(headRootBT());

}

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

}

int main()

{

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

BinaryTree Bin;

Bin.Create();

Bin.enterBT();

ID depth = 1;

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

else

{

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

Bin.printTree();

cout << endl<<"Элементы БДП в порядке их возрастания: " << endl;

Bin.obhodBT(Bin.headRootBT());

cout<< endl<<"Удаление";

Bin.destroyTree();

}

return 0;

}

**6. Вывод**

В ходе данной работы были получены практические навыки работы с бинарными деревьями поиска и реализации алгоритма вывода элементов в порядке возрастания на языке программирования С++.