**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: Бинарные деревья**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6383 |  | Лавренкова Е.Л. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

Практическое применение и освоение представления и реализации бинарных деревьев для представления информации на языке программирования С++.

**Задание (вариант №2).**

Для заданного бинарного дерева b типа BT с произвольным типом элементов:

а) определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

б) вычислить длину внутреннего пути дерева b, т. е. сумму по всем узлам длин путей от корня до узла;

в) напечатать элементы из всех листьев дерева b;

г) подсчитать число узлов на заданном уровне n дерева b (корень считать узлом 1-го уровня);

д) определить, есть ли в дереве b хотя бы два одинаковых элемента.

**Основные теоретические положения.**

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

*a*

*c*

*b*

*f*

*d*

*h*

*e*

*j*

*g*

*i*

*k*

*l*

**Спецификация программы.**

*Назначение программы*.

Программа предназначена для нахождения максимальной глубины искомого дерева, его длины внутреннего пути, числа узлов на заданном уровне.

*Описание программы*.

На входе программа получает последовательность символов из файла “KLP.txt”. Для решения поставленных задач используются рекурсивные процедуры.

Исходные данные представляют собой строку, содержащую элементы бинарного дерева. Результат работы программы представляет собой числа – значения требуемых выражений.

*Реализация*.

|  |  |
| --- | --- |
| **Имя функции** | **Назначение** |
| binTree enterBT(); | Считывает бинарное дерево из файла. |
| int getMaxDepth(binTree q, int depth) | Считает глубину дерева. Выводит текущую глубину на экран.  На вход подается узел бинарного дерева |
| void displayBT(binTree b, int n) | Функция вывода бинарного дерева в консоль. |
| unInt lenBT(binTree b); | Считает внутреннюю длину бинарного дерева.  На вход подается указатель на текущий узел дерева |
| unInt NumOfLvl(binTree b, int a, int c = 1) | Функция, проверяющая, существует ли узел на заданном уровне |
| void levelBT(binTree b); | Считает количество уровней бинарного дерева. |

**Тестирование.**

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Исходные данные |
| 1 | (a(e(m(o//)(p//))(/))(d()/(z//))) | Результат в Приложении 2. |
| 2 | (/) | **Пустое БД** |
| 3 | (a(e(m//())(/))(d()/(z//))) | Высота БД 3  Внутренний путь БД 6  Уровень 3  Количество узлов на данном уровне 2 |
| 4 | (a//) | Высота БД 1  Внутренний путь 0  Уровень 1  Количество узлов на данном уровне 1 |
| 5 | (a(o//)(i//)) | Высота БД 2  Внутренний путь 2  Уровень 2  Количество узлов на данном уровне 2 |

**Выводы.**

В процессе выполнения работы были получены практические навыки по применению бинарных деревьев, реализации и применения деревьев на языке программирования С++.

**Приложение 1. Исходный код.**

***Файл main.cpp***

#include <iostream>

#include <fstream>

#include <fstream>

#include <cstdlib>

#include "Btree.h"

#include <windows.h>

#include <conio.h>

#include <algorithm>

using namespace std;

using namespace binTree\_modul;

typedef unsigned int unInt;

binTree enterBT();

void displayBT(binTree b, int n);

unInt sizeBT(binTree b);

unInt lenBT(binTree b);

unInt NumOfLvl(binTree b, int a, int c = 1);

int getMaxDepth(binTree b, int depth = 1);

void levelBT(binTree b);

ifstream infile("KLP.txt");

int main()

{

binTree b;

SetConsoleCP(1251); // для вывода кирилицы

SetConsoleOutputCP(1251); // для вывода кирилицы

b = enterBT();

int depth = 1;

if (isNull(b)) cout << "Пустое БД" << endl;

else

{

cout << "Бинарное дерево (повернутое): " << endl;

displayBT(b, 1);

cout << "\nВычисление высоты дерева: " << endl;

int a = getMaxDepth(b) - 1;

cout << "Высота дерева = " << a << "\n\n";

cout << "Длина внутреннего пути дерева = " << lenBT(b) << endl;

cout << "\nПоиск листьев дерева:" << endl; levelBT(b);

cout << "\nНахождение количества узлов на N-ом уровне. Введите N:" << endl;

int num;

cin >> num;

cout << "Количество узлов на заданном уровне = " << NumOfLvl(b, num);

destroy(b);

}

\_getch();

}

//---------------------------------------

binTree enterBT()

{

char ch;

binTree p, q;

infile >> ch;//считывание из файла

while (ch == '(' || ch == ')') infile >> ch;//проверка на скобки

if (ch == '/') { return NULL; }//проверка на пустой элемент

else { //иначе рекурсивная запись элементов

p = enterBT();

q = enterBT();

return ConsBT(ch, p, q);

}

}

int getMaxDepth(binTree q, int depth)//находит глубину дерева

{

if (q != NULL)//если узел не пуст

cout << "Узел " << q->info << " Текущая глубина=" << depth << endl; //Печать узла и текущей глубины

if (q == NULL) { return depth; }//Если элемент пуст, возвращается значение глубины

else return max(getMaxDepth(q->rt, depth + 1), getMaxDepth(q->lt, depth + 1));//В противном случае, рекурсивно продолжается процесс

}

//---------------------------------------

void displayBT(binTree b, int n) //Для печати повернутого дерева

{ // n - уровень узла

if (b != NULL)

{

cout << ' ' << RootBT(b); //Проверка элемента

if (!isNull(Right(b)))

displayBT(Right(b), n + 1); //Если правая ветка не пустая, то печатаем элементы

else cout << endl; // спускаемся вниз

if (!isNull(Left(b))) //Если левая ветка не пустая, то печатаем элементы

{

for (int i = 1; i <= n; i++)

cout << " "; // вправо

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

}

}

else {};

}

void levelBT(binTree b) //Поиск листьев дерева

{

if (!isNull(b)) //Если дерево не пусто

{

if (!(isNull(b->lt) && isNull(b->rt))) //Если не правый и не левый

cout << "Узел:" << b->info << endl;//то это узел, петатаем его

levelBT(b->lt); //Рекурчивно проверяем следующий элемент

if (isNull(b->lt) && isNull(b->rt)) //если правый и левый

cout << "Лист:" << b->info << endl; //то это лист , печать

levelBT(b->rt);

}

}

unInt NumOfLvl(binTree b, int lvl, int lvl\_now)

{

if (b != NULL && lvl >= lvl\_now) //Если элемент не равен нулю и уровень больше текущего

cout << lvl << " vs " << lvl\_now << " Узел " << b->info; //то этот элемент - узел

{

if (lvl == lvl\_now) cout << " Узел на заданном уровне существует!\n"; //есл уровень равен текущему уровню

else cout << endl;

}

if (lvl == lvl\_now) return 1;

else if (lvl >= lvl\_now)

return(((b->lt) ? NumOfLvl(b->lt, lvl, lvl\_now + 1) : 0) + ((b->rt) ? NumOfLvl(b->rt, lvl, lvl\_now + 1) : 0)); //Подсчет кол-ва узлов

}

//int len = 0;

unInt lenBT(binTree b) //Для внутреннего пути

{

if (!isNull(b)) //Если дерево не пусто

cout << "Узел " << b->info << " " << /\*Печать узла\*/

" Левый " << lenBT(Left(b)) << /\*Печать левого\*/

" Правый " << lenBT(Right(b)) << /\*Печать правого\*/

" Количество узлов " << sizeBT(b) - 1 << endl; /\*Печать количества узлов\*/

if (isNull(b)) { return 0; } //Если пустое дерево, возврат 0

else return lenBT(Left(b)) + lenBT(Right(b)) + sizeBT(b) - 1; //В противном случае, возрат внутреннего пути дерева

}

unInt sizeBT(binTree b) //Для подсчета количества узлов

{

if (isNull(b)) return 0; //Проверка на пустое дерево

else return sizeBT(Left(b)) + sizeBT(Right(b)) + 1; //Если нет , то рекурсивный подсчет кол-ва узлов

}

***Файл bt\_func.cpp***

#include <iostream>

#include <cstdlib>

#include "Btree.h"

using namespace std;

namespace binTree\_modul

{

binTree Create() { return NULL; }

bool isNull(binTree b) { return (b == NULL); } //Проверяет не лустое ли дерево

base RootBT(binTree b) // для непустого бин.дерева

{

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

else return b->info; //Если нет, возврат данных элемента

}

bool RooBT(binTree b)//Проверка элемента

{

if (b == NULL) return false; //если пустой

else return true; //Если не пуст

}

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

{

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

else return b->lt;

}

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

{

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

else return b->rt;

}

binTree ConsBT(const base &x, binTree &lst, binTree &rst)

{

binTree p;

p = new node;

if (p != NULL)

{

p->info = x;

p->lt = lst;

p->rt = rst;

return p;

}

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

}

void destroy(binTree &b)

{

if (b != NULL)

{

destroy(b->lt);

destroy(b->rt);

delete b;

b = NULL;

}

}

}

***Файл Btree.h***

#pragma once

namespace binTree\_modul

{

typedef char base;

struct node {

base info;

node \*lt;

node \*rt;

// constructor

node() { lt = NULL; rt = NULL; }

};

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

binTree Create(void);

bool isNull(binTree);

bool RooBT(binTree);

base RootBT(binTree); // для непустого бин.дерева

binTree Left(binTree);// для непустого бин.дерева

binTree Right(binTree);// для непустого бин.дерева

binTree ConsBT(const base &x, binTree &lst, binTree &rst);

void destroy(binTree&);

**Приложение 2.**



