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

Студентка гр. 7382 _____ Давкаева В.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2018

Цель работы.

Ознакомиться с основными понятиями и приёмами использования БДП , получить навыки решения задач с их помощью.

Формулировка задания (Вариант 9)

Необходимо написать код для реализации : БДП: случайное* БДП; действие: 1+2в.

1) По заданному файлу F (типа file of Elem), все элементы которого различны, построить БДП определённого типа;
2) Выполнить одно из следующих действий:

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

Теоретические положения.

Бинарное дерево поиска — это двоичное дерево, для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных самого узла X.
- У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равно*, нежели значение ключа данных самого узла X.

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения *меньше*.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных. Однако это касается реализации, а не природы двоичного дерева поиска.

Работа алгоритма.

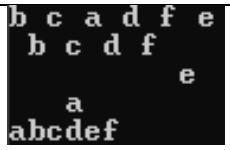
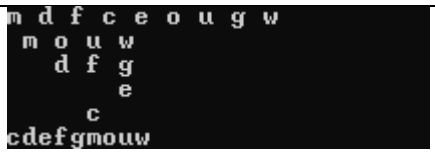
Алгоритм посимвольно считывает символы из файла и записывает их в БДП. То есть первый символ будет всегда корнем, следующие символы будут сверяться с корнем, меньшие значения будут записываться налево, большие

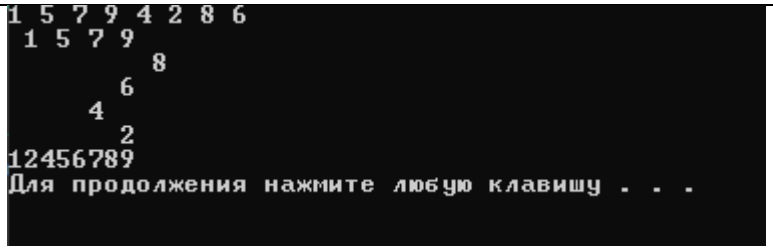
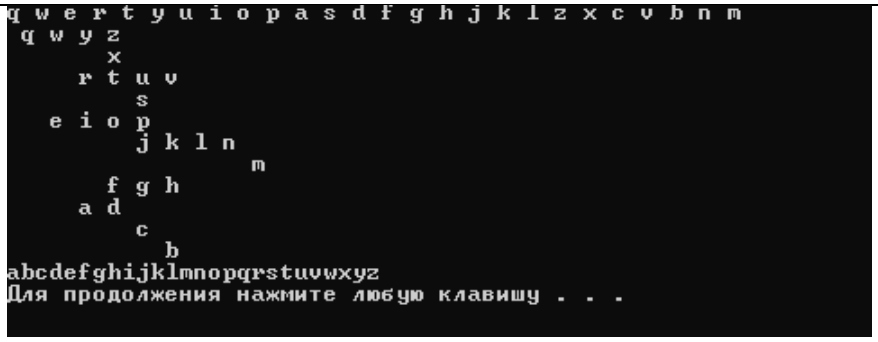
направо. Дерево будет выводиться на экран, а затем хранимые значения будут выводиться в порядке возрастания, используя обход дерева ЛКП.

Описание функций

Название	Описание
binSTree Create(void)	Создание нулевого дерева (конструктор)
bool isNull(binSTree)	Проверка на пустоту дерева
binSTree Left (binSTree)	Получение левого поддерева
binSTree Right (binSTree)	Получения правого поддерева
binSTree ConsBT(const base &x, binSTree &lst, binSTree &rst)	Построение дерева по корню и двум деревьям
void displayBT (binSTree b, int n)	Вывод дерева в консоль
void printLKP (binSTree b)	Печать ЛКП обхода в консоль
void printTEXTLKP (binSTree b)	Печать ЛКП обхода в файл

Тестирование программы:

Входные данные в файле	Вывод на экран	Содержимое выходного файла
b c a d f e		**abcdef**
m d f c e o u g w		**cdefgmouw**
		

1 5 7 9 4 2 8 6		**12456789**
q w e r t y u i o p a s d f g h j k l z x c v b n m		**abcdefghijklmnopqrstuvwxyz**

Тестирование.

Работа алгоритма на примере 1.

b c a d f e

1. Считыванте b. Так как b первый элемент, он будет корнем нашего дерева.
2. Считывание c. Так как c больше b, c – правое поддерево b.
3. Считывание a. Так как a меньше b, a – левое поддерево b.
4. Считывание d. Так как d больше b и больше c, d – правое поддерево c.
5. Считывание f. Так как f больше b,c и d, f-правое поддерево d.
6. Считывание e. Так как e больше b, c, d и меньше f, c-левое поддерево f.
7. Через функцию displayBT дерево отображается на экране. А именно все правые поддеревья записываются в одну строку вместе с корнем, а левые записываются на строку ниже с соответствующим количеством отступов.
8. Затем элементы записываются в порядке возрастания. Для этого используется обход дерева ЛКП. На примере уже построенного дерева:

8.1 Сначала выводится левое поддерево b. Выводится само a. Затем сам корень b.

8.2 Затем проверяется правое поддерево b. Выводятся c и d, так как у них только правые поддеревья.

8.3 Рассмотрим f. У f есть левое поддерево, где хранится e, значит выводим e, а затем и f.

Вывод.

В результате выполнения данной работы, ознакомились с основными понятиями и приёмами использования бинарного дерева поиска, получили навыки разработки программ на языке программирования C++.

Код программы:

- **ALG5.cpp :**

```
#include "stdafx.h"
#include "Btree.h"
#include <iostream>
#include <Windows.h>
#include <fstream>
#define _CRT_SECURE_NO_WARNINGS
using namespace std;
using namespace binSTree_modul;


ifstream infile("inputf.txt"); //    файл для считывания
ofstream outfile("outputf.txt");  // файл вывода
int main()
{
	char ch;
	//SetConsoleCP(1251);
	//SetConsoleOutputCP(1251);// для вывода кирилицы
	binSTree  tree;
	tree = Create();
	while(infile >> ch) {             // чтение дерева
		cout << ch << " ";
		read(ch, tree);
	}
	cout << endl;
	displayBT(tree, 1);               // вывод дерева в консоль
	printLKP(tree);             // запись ЛКП в консоль
	cout << endl;
	printTEXTLKP(tree, outfile);// запись ЛКП в файл
	system("pause");
    return 0;
}
```

- **BTree.cpp:**

```
#include "stdafx.h"
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <algorithm>
#include "Btree.h"
using namespace std;

namespace binSTree_modul
{

	//-------------------------------------
	binSTree Create()
	{
		return NULL;
	}
	//-------------------------------------
	bool isNull(binSTree b)
	{
		return (b == NULL);
	}
```

```
	//-------------------------------------
	base RootBT(binSTree b)                  // для непустого бин.дерева
	{
		if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
		else return b->info;
	}
	//-------------------------------------
	unsigned int RootCountBT(binSTree b) // для непустого бин.дерева
	{
		if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
		else return b->count;
	}
	//-------------------------------------
	binSTree Left(binSTree b)         // для непустого бин.дерева
	{
		if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }
		else return b->lt;
	}
	//-------------------------------------
	binSTree Right(binSTree b)        // для непустого бин.дерева
	{
		if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }
		else return b->rt;
	}
	//-------------------------------------
	binSTree ConsBT(const base &x, binSTree &lst, binSTree &rst)
	{
		binSTree p;
		p = new node;
		if (p != NULL) {
			p->info = x;
			p->count = 1;
			p->lt = lst;
			p->rt = rst;
			return p;
		}
		else { cerr << "Memory not enough\n"; exit(1); }
	}
	//-------------------------------------

	void read(base x, binSTree &b)
	{
		binSTree p;
		if (b == NULL) {
			p = new node;
			if (p != NULL) {
				p->info = x;
				p->count = 1;
				p->lt = NULL;
				p->rt = NULL;
				//cout << x;
				b = p;
			}
			else { cerr << "Memory not enough\n"; exit(1); }
		}
		else if (x < b->info) read(x, b->lt);//{cout << x << "<-" << endl;
SearchAndInsert (x, b->lt);}
		else if (x > b->info) read(x, b->rt);//{cout << x << "->" << endl;
SearchAndInsert (x, b->rt);}
		else b->count++;
	}
////////////////////////////////////////////////
	void displayBT(binSTree b, int n)
	{	// n - уровень узла
		if (b != NULL) {
```

```
                cout << ' ' << RootBT(b);
                if (!isNull(Right(b))) { displayBT(Right(b), n + 1); }
                else cout << endl; // вниз
                if (!isNull(Left(b))) {
                        for (int i = 1; i <= n; i++) cout << "  "; // вправо
                        displayBT(Left(b), n + 1);
                }
        }
        else {};
}
//---------------------------------------
void printLKP(binSTree b)
{
        if (!isNull(b)) {
                printLKP(Left(b));
                cout << RootBT(b);
                printLKP(Right(b));
        }
}
//---------------------------------------

void printTEXTLKP(binSTree b, std::ofstream &out)
{
        if (!isNull(b)) {
                printTEXTLKP(Left(b), out);
                out << RootBT(b);
                printTEXTLKP(Right(b), out);
        }
}
}
```

- **BTree.h:**

```
#include <fstream>
namespace binSTree_modul
{
        typedef char base;

        struct node {
                base info;
                unsigned int count;
                node *lt;
                node *rt;
                // constructor
                node() { count = 0; lt = NULL; rt = NULL; }
        };

        typedef node *binSTree; // "представитель" бинарного дерева

        binSTree Create(void);
        bool isNull(binSTree);
        base RootBT(binSTree);
        unsigned int RootCountBT(binSTree); // для непустого бин.дерева
```

```
        binSTree Left(binSTree);// для непустого бин.дерева
        binSTree Right(binSTree);// для непустого бин.дерева
        binSTree ConsBT(const base &x, binSTree &lst, binSTree &rst);


        void read(base x, binSTree &b);

        void displayBT(binSTree b, int n);

        void printLKP(binSTree b);

        void printTEXTLKP(binSTree b, std::ofstream &out);

};
```