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

Студент гр. 9382 _____ Иерусалимов Н.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Реализовать идеально сбалансированное бинарное дерево

поиска. Реализовать поиск и запись элементов в нем.

Задание.

Вариант 21.

БДП: Идеально сбалансированное;
действие: 1+2а

В вариантах заданий 2-ой группы (БДП и хеш-таблицы) требуется:

1) По заданной последовательности элементов Elem построить структуру данных

определённого типа – БДП или хеш-таблицу;

2) Выполнить одно из следующих действий:

а) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если входит, то в скольких экземплярах. Добавить элемент e в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

б) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem,

и если входит, то удалить элемент e из структуры данных (первое обнаруженное

вхождение). Предусмотреть возможность повторного выполнения с другим элементом.

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

г) Другое действие.

Основные теоретические сведения

Бинарное дерево поиска называется идеально сбалансированным, если для каждой его вершины количество вершин в левом и правом поддереве различаются не более чем на 1.

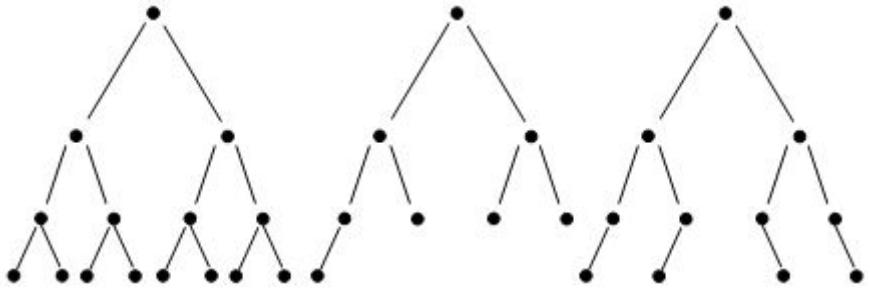

Рис.1. Примеры идеально сбалансированных бинарных деревьев

Алгоритм построения идеально сбалансированного дерева при известном числе вершин n лучше всего реализуется с помощью рекурсии (1). При этом необходимо лишь учесть, что для достижения минимальной высоты при заданном числе вершин, нужно располагать максимально возможное число вершин на всех уровнях, кроме самого нижнего. Это можно сделать очень просто, если распределять все поступающие в дерево вершины поровну слева и справа от каждой вершины.

1. Суть алгоритма:
   - Взять одну вершину в качестве корня
   - Левое поддерево строится с помощью формулы nl = n/2
   - Правое поддерево строится с помощью формулы nr = n - nl – 1
   - Пока n != 0 поддеревья которые были получены выше становятся новыми корнями для следующих поддеревьев.

Алгоритм представлен в Приложение А, в методе класса TREE:: Void Tree(…).

Описание алгоритма

Построение идеального бинарного дерева реализуется с помощью метода TREE** Tree(…). Поиск же реализуется с помощью метода Void Search().

1. Tree – рекурсивный метод который принимает в виде входных данных количество вершин и оригинальный указатель на корень дерева. С помощью количества вершин определяется количество узлов в левом поддереве и в правом. Потом записывается веденый элемент во временную переменную после чего вызывается этот же метод для левого

поддерева с вычисленным количество вершин для левого поддерева. Алгоритм выполняется пока количество вершин не будет равняться нулю. После чего вызывается метод уже для правого поддерева, с таким же алгоритмом.

2. Search – Рекурсивный метод который обходит дерево ЛПК обходом. Доходя до листьев, смотрит, равняется ли элемент искомому, если да то инкрементирует счетчик.

Описание основных функций.

Для создания БДП был создан класс TREE – в котором описывается структура дерева и методы для взаимодействия с ней:

1) Private: - переменные класса
   a) Int key – переменная куда будут записываться данные узла.
   b) TREE* duk – указатель на корень дерева.
   c) TREE* Left – указатель на левого ребенка.
   d) TREE* Right – указатель на правого ребенка.
2) Public: - методы класса
   a) TREE() – конструктор класса обнульяет корень.
   b) ~TREE() – Деструктор проходит по всем узлам ЛПК обходом и очищает память листьев - дерева.
   c) TREE**GetDuk() – Возвращает оригинал указателя на корень узла.
   d) Void Search(int num, int* count) – Производит поиск числа проходя по дереву с помощью ЛПК обхода.
      i) Int num – искомое число.
      ii) Int* count – указатель на счетчик.
   e) Void Tree(int, TREE**,char side, int depth, int parrent) – записывает в дерево элементы из консоли.
      i) Int – количество вершин.
      ii) TREE** - оригинальный указатель на корень дерева.

iii) Char side – для вывода промежуточных данных, дает понять куда идет элемент.

iv) Int depth - для вывода промежуточных данных, глубина рекурсии.

v) int parent - для вывода промежуточных данных, родитель нынешнего элемента.

Void TreeFromFile(int , TREE**,ifstream in,char side, int depth, int parrent) – записывает в дерево элементы из файла.

i) ifStream in – переменная для потокового считывания из файла.

ii) Все остальные аргументы те же что и у метода “e”

f) Void print1(TREE** root, short x. Short y, short a, char c)- выводит дерево в красивом виде.

i) TREE** root - оригинальный указатель на корень дерева.

ii) short x – первый отступ по x для позиционирования курсора в консоли

iii) Short y – первый отступ по y, для позиционирования курсора в консоли

iv) short a – количество уровней в дереве.

v) char c – определяет в какой стороне печатать символ лево право корень.

g) Void Vyvod(TREE** ,int) –более простая реализация вывода дерева для написания черновика дерева.

i) TREE** - оригинальный указатель на корень дерева.

ii) Int – количество вершин.

Выводы.

В данной задаче было создано Идеально сбалансированное бинарное дерево поиска. Реализован ввод данных в дерево, и поиск элементов в нем с последующим подсчетом их.

Тестирование:

Табл. №1

№	Входные данные	Выходные данные	Коменнтарии/ Промежуточные данные
1	5 1 2 3 4 5 Search - 5	Elemet : 5 amount 1	Enter the number of vertices - ... 5 Enter keys... 1 2 3 4 5 /\Add Root = 1 ; _____ /\\Add Left children = 2; /\\His parrent is = 1; /\\Became a tree node... _____ /\\\Add Left children = 3; /\\\His parrent is = 2; /\\\reached the leaf = 3 _____ /\\Add Right children = 4 ; /\\His parrent is = 1; /\\Became a tree node... _____ /\\\Add Left children = 5; /\\\His parrent is = 4; /\\\reached the leaf = 5 _____ A draft of the received data: 4 5 1 2 3

2	6 5 5 5 5 5 5 Search – 5 Search – 3 Search - 55	Elemet : 5 amount 6 Elemet : 3 amount 0 Elemet : 55 amount 0	Enter the number of vertices - ... 6 Enter keys... 5 /\Add Root = 5 ; _____ 5 /\\Add Left children = 5; /\\His parrent is = 5; /\\Became a tree node... _____ 5 /\\\Add Left children = 5; /\\\His parrent is = 5; /\\\reached the leaf = 5 _____ 5 /\\\Add Right children = 5 ; /\\\His parrent is = 5; /\\\reached the leaf = 5 _____ 5 /\\Add Right children = 5 ; /\\His parrent is = 5; /\\Became a tree node... _____ 5 /\\\Add Left children = 5;

			/\/\\His parrent is = 5; /\/\\reached the leaf = 5 _____ A draft of the received data: 5 5 5 5 5 5
3	8 15 65 9 8 65 45 32 1 Search – 65 Search – 1	Elemet : 65 amount 2 Elemet : 1 amount 1	
4	5 A	Eror, input should be digit!	
5	6 1 5 6 9 87 4 Search – f	Eror input should be digit! Elemet : 0 amount 0	
6	printTree	None tree!	
7	0	Empty tree!	
8	1 56	In Tree 1 elem !	Enter the number of vertices - 1 Enter keys... 56 /\Add Root = 56 ; _____ A draft of the received data: 56

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <conio.h>
#include <Windows.h>
using namespace std;

class TREE
{
private:


    int Key;
    int Count;
    TREE *duk; //Корень дерева.
    TREE *Left;
    TREE *Right;
public:
    TREE() { duk = nullptr; }
    ~TREE();
    TREE **GetDuk() { return &duk; }
    void Search(int num, int* count);
    void Tree (int, TREE **,char side,int depth, int parrent);
    void TreeFromFile (int n,TREE **p,ifstream& in,char side,int
depth,int parrent);
    void print1(TREE ** root, short x, short y, short a, char c);
    void printDepth(int depth);
    void Vyvod (TREE **, int);
};
void TREE::printDepth(int depth){
    for(int i =0; i < depth; ++i){
        cout << "/\\";
    }
}

TREE::~TREE(){
    if(duk->Left) duk->Left->~TREE();
    if(duk->Right) duk->Right->~TREE();
```

```
    delete duk;
}

void TREE::Search(int num,int* count) {
    if(this->duk != nullptr){
        duk->Search(num,count);
    }else{
        TREE *l =this->Left, *r = this->Right;
        if(l != nullptr){
            l->Search(num, count);
        }
        if(r != nullptr){
            r->Search(num, count);
        }
        if(Key == num) {
            *count += 1;
        }

    }
}


int main ()
{
    //system ("cls"); fflush (stdin);
    TREE A;
    int n=0, count=0, depth = 0;
    int choise, search, fileOrConsole;

    while(TRUE){
        cout << "0 - Exit. 1 - Create Tree. 2 - Print tree. 3 -
Search.\n";
        fflush (stdin);
        cin >> choise;
        switch(choise){
            case 0:
                exit (0);
                break;
            case 1:
                cout<<"1 - Console. 2 - File";
                if(!(cin>>fileOrConsole)) {
                    cout<<"Eror you should input only digit\n";
                    cin.clear();
                }else {
```

```
                    if (fileOrConsole == 2) {
                        string input_filename;
                        const string output_filename = "output.txt";
                        ifstream in;
                        ofstream out;
                        out.open(output_filename);
                        out << "";
                        out.close();
                        cout << "Enter the input file name: \n\n";
                        cin >> input_filename;
                        in.open(input_filename);
                        if (in.is_open()) {
                            in >> n;
                            A.TreeFromFile(n, A.GetDuk(), in, 'k', depth,
0);
                            in.close();
                            cout << "A draft of the received data:\n";
                            A.Vyvod(A.GetDuk(), n);
                        }

                    } else {
                        cout << "Enter the number of vertices -...\n";
                        if((cin >> n)){
                            if (n == 0) {
                                cout << "Empty array!\n";
                            } else {
                                cout << "Enter keys...\n";
                                A.Tree(n, A.GetDuk(), 'k', depth, 0);
                                cout << "A draft of the received
data:\n";
                                A.Vyvod(A.GetDuk(), n);

                            }
                        }else{
                            cout<<"Eror you should input only digit\n";
                            cin.clear();
                        }
                    }
                    system("pause");
                }
                break;
            case 2:
                system ("cls");
                if(n==0){
```

```
                    cout<<"None tree\n";
                }else {
                    cout << "Tree: \n";
                    A.print1(A.GetDuk(), 74, 2, 3, 'k');
                }
                cout<<"\n\n\n\n\n";

                system ("pause");
                break;
            case 3:
                cout << "Enter the item you are looking for - ...\n";
                if(!(cin >> search)){
                    cout<<"Eror input should be digit!";
                }
                A.Search(search, &count);
                cout<<"Elemet : "<<search <<" amount "<< count<<'\n';
                count = 0;
                system ("pause");
                break;
            default:
                cout<<"Wrong enter!\n";
                break;
        }
        //system ("cls");
    }

}
void TREE::TreeFromFile (int n,TREE **p,ifstream& in,char side,int depth,
int parrent){
    TREE *now;
    int x,nl,nr;
    ++depth;
    now = *p;
    if  (n==0) *p = NULL;
    else
    {
        nl = n/2;
        nr = n - nl - 1;
        in>>x;
        now = new TREE;
        (*now).Key = x;
        printDepth(depth);
        if(side == 'l'){
            cout<<"Add Left children = "<< x<<";\n";
```

```
            printDepth(depth);
            cout<<"His parrent is = "<< parrent<<";\n";
            if(nl+nr==0){
                printDepth(depth);
                cout<<"reached the leaf = "<<x<<'\n';
            }else {
                printDepth(depth);
                cout<<"Became a tree node...\n";
            }
        }else if(side == 'r'){
            cout<<"Add Right children = "<< x<<" ;\n";
            printDepth(depth);
            cout<<"His parrent is = "<< parrent<<";\n";
            if(nl+nr==0){
                printDepth(depth);
                cout<<"reached the leaf = "<<x<<'\n';
            }else {
                printDepth(depth);
                cout << "Became a tree node...\n";
            }
        }else if(side == 'k'){
            cout<<"Add Root = "<< x<<" ;\n";
        }
        cout<<"\n____________________\n";
        //else {
        //     cout<<"in a node "<<x<<" - "<<nl+nr<<" childrens\n";
        // }
        TreeFromFile (nl,&((*now).Left),in,'l',depth, x);
        --depth;
        TreeFromFile (nr,&((*now).Right), in, 'r',depth,x);
        *p = now;

    }
    --depth;
}

void TREE::Tree (int n,TREE **p,char side,int depth,int parrent){
// Построение идеально сбалансированного
//           дерева с n вершинами.
// *p - указатель на корень дерева.

    TREE *now;
    int x,nl,nr;
    ++depth;
```

```
now = *p;
if  (n==0) *p = NULL;
else
{
    nl = n/2;
    nr = n - nl - 1;
    if(!(cin>>x)){

        cout<<"Eror input should be digit!\n";
        cin.clear();
        exit(0);
    }
    else {
        now = new TREE;
        (*now).Key = x;
        printDepth(depth);
        if (side == 'l') {
            cout << "Add Left children = " << x << ";\n";
            printDepth(depth);
            cout << "His parrent is = " << parrent << ";\n";
            if (nl + nr == 0) {
                printDepth(depth);
                cout << "reached the leaf = " << x << '\n';
            } else {
                printDepth(depth);
                cout << "Became a tree node...\n";
            }
        } else if (side == 'r') {
            cout << "Add Right children = " << x << " ;\n";
            printDepth(depth);
            cout << "His parrent is = " << parrent << ";\n";
            if (nl + nr == 0) {
                printDepth(depth);
                cout << "reached the leaf = " << x << '\n';
            } else {
                printDepth(depth);
                cout << "Became a tree node...\n";
            }
        } else if (side == 'k') {
            cout << "Add Root = " << x << " ;\n";
        }
        cout << "\n____________________\n";
        Tree(nl, &((*now).Left), 'l', depth, x);
        Tree(nr, &((*now).Right), 'r', depth, x);
```

```
            *p = now;
        }
    }
    --depth;
}

void TREE::Vyvod (TREE **w,int l)
// Изображение бинарного дерева, заданного
// указателем *w на экране дисплея.
{
    if  (*w!=NULL)
    {
        Vyvod (&((**w).Right),l+1);
        for  (int i=1; i<=l; i++) {
            cout<<"\t";
        }
        cout<<(**w).Key<<endl;
        Vyvod (&((**w).Left),l+1);
    }
}
void GoToXY (short x, short y)
{
    HANDLE StdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD coord = {x, y};
    SetConsoleCursorPosition(StdOut, coord);
}

void TREE::print1(TREE ** root, short x, short y, short a, char c)
{
    if ((*root) != nullptr)
    {
        if (a>0 && c!='k')
        {
            if (c=='l')
                x-=10*a;
            else
                x+=10*a;
        }
        else
        if (c!='k')
            if (c=='l')
                x-=4;
            else
                x+=4;
```

```
        GoToXY (x,y+=2);

        a--;

        cout<<((**root).Key);
        print1(&((*root)->Left), x, y, a, 'l');
        print1(&((*root)->Right), x, y, a, 'r');
    }
}
```