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: «Бинарные деревья поиска и алгоритмы сжатия»

Студент гр. 7381 Дорох С.В.

Преподаватель Фирсов М.А.

Санкт-Петербург

2018

Задание.

7. БДП: случайное* БДП. Действие: 1+2а.

В варианте задания 1+2а требуется:

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

2) Выполнить одно из следующих действий:

а) Для построенного БДП проверить, входит ли в него элемент *e* типа *Elem*, и если не входит, то добавить элемент *e* в дерево поиска.

Пояснение задания:

На вход программе подаётся файл, содержащий произвольное количество значений заданного типа, с помощью которых строится дерево. Следом идёт элемент заданного типа данных – элемент, с которым необходимо выполнить второе задание.

Описание алгоритма.

Считывается первая строка файла, содержащая элементы дерева, параллельно элементы проверяются на принадлежность одному типу, после этого элементы добавляются в дерево. Элемент для проверки считывается и записывается в отдельную переменную. Алгоритм добавления при это игнорирует элементы, уже добавленные в дерево, т.е. если в дерево добавить n раз один и тот же элемент, то в дереве он будет записан всего 1 раз.

После считывания первой строки файла и построения дерева происходит вывод его состояния до поиска. Далее происходит считывание элемента для поиска и начинается работа с ним. Функция поиска элемента работает рекурсивно, если наш элемент имеет значение меньшее, чем значение узла, то вызывается эта же функция для поиска в левом поддереве, иначе в правом, также по ходу поиска выводится состояние работы функции. Если элемент найден – выводится соответствующее сообщение, если нет – то данный

элемент добавляется в дерево в соответствии с правилами построения бинарного дерева поиска.

Описание функций и структур данных.

Для выполнения лабораторной работы был написан класс BST.

`struct Node` – структура, представляющая собой узел дерева. Содержит в себе:

`Type data` – значение узла типа `Type`;

`Node* left` – указатель на левого сына типа `Node`;

`Node* right` – указатель на правого сына типа `Node`.

`Node* makeEmpty(Node* t)` – метод, удаляющий дерево.

Принимаемые аргументы:

`Node* t` – указатель на узел дерева.

Возвращаемое значение: 0 – в случае если узел пуст, или элемент дерева удалён.

`Node* insert (Type x, Node* t)` – метод, вставляющий элемент в дерево.

Принимаемые аргументы:

`Type x` – элемент для вставки типа `Type`;

`Node* t` – узел дерева типа Node.

Возвращаемое значение: ссылка на новый узел дерева.

`Node* findMin(Node* t)` – метод нахождения узла с минимальным значением.

Принимаемые аргументы:

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: ссылку на минимальный элемент дерева.

`Node* findMax(Node* t)` – метод нахождения узла с максимальным значением.

Принимаемые аргументы:

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: ссылку на максимальный элемент дерева.

`Node* remove(Type x, Node*t)` – метод удаления узла дерева.

Принимаемые аргументы:

`Type x` – элемент для удаления типа `Type`;

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: возвращает узел, вставший на место удалённого.

`void printSpaces(size_t deep)` – метод, печатающий пробелы в соответствии с глубиной дерева.

Принимаемые аргументы:

`size_t deep` – глубина дерева.

Возвращаемое значение: метод ничего не возвращает.

`void inorder(Node* t, size_t lvl)` – метод, выводящий значение текущего узла в дереве.

Принимаемые аргументы:

`Node* t` – узел дерева типа `Node`.

`size_t lvl` – глубина дерева.

Возвращаемое значение: метод ничего не возвращает.

`Node* find*(Type x, Node* t)` – метод, ищущий определённый узел дерева.

Принимаемые аргументы:

`Type x` – элемент для поиска типа `Type`;

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: метод возвращает указатель на найденный элемент или 0 в случае не нахождения элемента.

`BST()` – конструктор, присваивающий корню дерева NULL.

Принимаемые аргументы:

`~BST()` – деструктор, удаляющий дерево.

`Void insert(Type x)` – метод, осуществляющий вставку заданного элемента в дерево.

Принимаемые аргументы:

`Type x` – искомое значение.

Возвращаемое значение: метод ничего не возвращает.

`void remove(Type x)` – метод, удаляющий узел с заданным значением.
Принимаемые аргументы:

`Type x` – удаляемое значение.

Возвращаемое значение: метод ничего не возвращает.

`void display ()` – метод, выводящий дерево в древовидной форме в стандартный поток вывода.

Принимаемые аргументы: ничего не принимает.

Возвращаемое значение: метод ничего не возвращает.

`int search (Type x)` – метод, ищущий заданный элемент в дереве.

Принимаемые аргументы:

`Type x` – заданный элемент.

Возвращаемое значение: 1 – если элемент найден, 0 – не найден.

Тестирование.

Для тестирования был использован bash-скрипт, использованный в первых 4 лабораторных работах с небольшими доработками. Данные тестирования представлены в таблице ниже. Ввиду большого объёма выводимой информации большая часть тестов искусственно урезана.

№	Входные данные	Выходные данные
1	1 123 684 456 19	Test 1: 1 123 684 456 19 Enter leafs for tree: 1 123 684 456 Tree before search: # / 684 \ # / 456 \ # / 123 \

		# / 1 \ # Enter an item to search: 19 Element 19 is more than 1 , go to the right subtree. Element 1 was not 19! Element 19 is less than 123 , go to the left subtree. Element 123 was not 19! Element wasn't found! Add an element in BST: 19 # / 684 \ # / 456 \ # / 123 \ # / 19 \ # / 1 \ #
2	3 7 2 1 10 9 0	Test 2: 3 7 2 1 10 9 0 Enter leafs for tree: 3 7 2 1 10 9

		Enter an item to search: 0 Element 0 is less than 3 , go to the left subtree. Element 3 was not 0! Element 0 is less than 2 , go to the left subtree. Element 2 was not 0! Element 0 is less than 1 , go to the left subtree. Element 1 was not 0! Element wasn't found! Add an element in BST: 0

3	
	777
		Test 3:
		          777
		Enter leafs for tree:
		Tree before search:
		BST is empty!
		Enter an item to search: 777
		Element wasn't found! Add an element in
		BST: 777
		    #
		  /
		777
		  \
		    #

4	0 2 4 6 8 10 12 14 16 18 20 -2 -4 -6 -8 -10 -12 -14 -16 -18 -20 -7	Test 4: 0 2 4 6 8 10 12 14 16 18 20 -2 -4 -6 -8 -10 -12 -14 -16 -18 -20 -7 Enter leafs for tree: 0 2 4 6 8 10 12 14 16 18 20 -2 -4 -6 -8 -10 -12 -14 -16 -18 -20 Enter an item to search: -7 Element -7 is less than 0 , go to the left subtree. Element 0 was not -7! Element -7 is less than -2 , go to the left subtree. Element -2 was not -7! Element -7 is less than -4 , go to the left subtree. Element -4 was not -7! Element -7 is less than -6 , go to the left subtree.

		Element -6 was not -7! Element -7 is more than -8 , go to the right subtree. Element -8 was not -7! Element wasn't found! Add an element in BST: -7
5	7 1 8 1 3 12 47 qwe	Test 5: 7 1 8 1 3 12 47 qwe Enter leafs for tree: 7 1 8 1 3 12 47 Enter an item to search:qwe Error! Items for search can only be of one type(digits)!
6	8 8 1 4 oops 12 47 10	Test 6: 8 8 1 4 oops 12 47 10 Enter leafs for tree: 8 8 1 4 oops 12 47 Error! o - was not digit! Stop building the tree... Error! o - was not digit! Stop building the tree... Error! p - was not digit! Stop building the tree... Error! s - was not digit! Stop building the tree... Enter an item to search: 10 Element 10 is more than 8 , go to the right subtree. Element 8 was not 10! Element wasn't found! Add an element in BST: 10

Выводы.

В ходе выполнения лабораторной работы была изучена такая структура данных как бинарное дерево поиска. Был написан шаблонный класс БДП для работы с произвольным типом данных. В качестве языка разработки был использован C++, bash-скрипты.

ПРИЛОЖЕНИЯ

Приложение А. Код основной программы.

```
#include "BST.hpp"
#include <iostream>
#include <cstring>
#include <sstream>

int main() {
    std::string list;
    std::cout << "Enter leafs for tree:" << std::endl;
    getline(std::cin, list);
    std::stringstream ss;
    for(size_t i = 0; i < list.size(); i++)     //Проверка элемента на
принадлежность к целому типу
        if(isalpha(list[i]))
            std::cout << "Error! " << list[i] << " - was not digit!
Stop building the tree..." << std::endl;
    ss.str(list);
    BST<int> tree;
    int value;
    while(ss >> value)
        tree.insert(value);
    std::cout << "Tree before search:" << std::endl;
    tree.display();
    std::cout << "Enter an item to search:" << std::endl;
    std::cin >> value;
    if(std::cin.fail()) {                       //Проверка символа для
поиска на принадлежность целому типу
        std::cout << "Error! Items for search can only be of one
type(digits)!" << std::endl;
        std::cout << "----------------------------------------------
------------------------------------------------------------------
--" << std::endl;
        return 0;
    }
    if(!tree.search(value)) {
        std::cout << "Element wasn't found! Add an element in BST: "
<< value << std::endl;
        tree.insert(value);
    }
    tree.display();
    std::cout << "----------------------------------------------------
------------------------------------------------------------------"
<< std::endl;
 }
```

Приложение Б. Код заголовочного файла БДП.

```
#ifndef _BST_HPP_
#define _BST_HPP_

#include <iostream>
#include <cstring>

template <class Type>
class BST {
    struct Node {
        Type data;
        Node* left;
        Node* right;
    };

    Node* root;

    Node* makeEmpty(Node* t) {              //Функция удаления дерева
        if(!t)
            return 0;
        makeEmpty(t->left);
        makeEmpty(t->right);
        delete t;
        return 0;
    }

    Node* insert(Type x, Node* t) {         //Функция вставки элемента
в дерево
        if(!t) {
            t = new Node;
            t->data = x;
            t->left = t->right = 0;
        }
        else if(x < t->data)
            t->left = insert(x, t->left);
        else if(x > t->data)
            t->right = insert(x, t->right);
        return t;
    }

    Node* findMin(Node *t) {            //Функция нахождения
минимального элемента в дереве
        if(!t)
            return 0;
        else if(!t->left)
            return t;
        else return findMin(t->left);
    }
```

```
    Node* findMax(Node* t) {            //Функция нахождения
максимального элемента в дереве
        if(!t)
            return 0;
        else if(!t->right)
            return t;
        else return findMax(t->right);
    }

    Node* remove(Type x, Node* t) {     //Функция удаления элемента в
дереве
        Node* temp;
        if(!t)
            return 0;
        else if(x < t->data)
            t->left = remove(x, t->left);
        else if(x > t->data)
            t->right = remove(x, t->right);
        else if (t->left && t->right) {
            temp = findMin(t->right);
            t->data = temp->data;
            t->right = remove(t->data, t->right);
        }
        else {
            temp = t;
            if(!t->left)
                t = t->right;
            else if(!t->right)
                t = t->left;
            delete temp;
        }
        return t;
    }

    void printSpaces(size_t deep) {         //Функция печати пробелов
в соответствии с глубиной дерева
        for(size_t i = 0; i < deep; i++)
            std::cout << ' ';
    }

    void inorder(Node* t, size_t lvl) {     //Функция выводящая
значение текущего элемента в дереве
        if(!t) {
            printSpaces(lvl);
            std::cout << "#" << std::endl;
            return;
        }
        inorder(t->right, lvl+4);
        if(lvl)
            printSpaces(lvl);
        std::cout << "  /" << std::endl;
```

```
        printSpaces(lvl);
        std::cout << t->data << std::endl << ' ';
        printSpaces(lvl);
        std::cout << " \\" << std::endl;
        inorder(t->left, lvl+4);
    }

    Node* find(Type x, Node* t) {        //Функция поиска элемента в
дереве
        if(!t)
            return 0;
        else if(x < t->data) {
            std::cout << "Element  " << x << " is less than " << t-
>data << " , go to the left subtree." << std::endl;
            std::cout << "Element " << t->data << " was not " << x <<
'!' << std::endl;
            return find(x, t->left);
        }
        else if(x > t->data) {
            std::cout << "Element  " << x << " is more than " << t-
>data << " , go to the right subtree." << std::endl;
            std::cout << "Element " << t->data << " was not " << x <<
'!' << std::endl;
            return find(x, t->right);
        }
        else return t;
    }

    public:
        BST() {            //Конструктор
            root = NULL;
        }

        ~BST() {
            root = makeEmpty(root);   //Деструктор
        }

        void insert(Type x) {         //Функция вставки заданного
элемента
            root = insert(x, root);
        }

        void remove(Type x) {         //Функция удаления заданного
элемента
            root = remove(x, root);
        }

        void display() {        //Функция отображения дерева
            if(!root) {
                std::cout << "BST is empty!" << std::endl;
                return;
```

```
            }
            inorder(root, 0);
            std::cout << std::endl;
        }

        int search (Type x) {   //Функция поиска элемента в дереве
            Node* temp = find(x, root);
            if(temp) {
                std::cout << "Element was found!" << std::endl;
                return 1;
            }
            else return 0;
        }
};

#endif
```