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: Деревья поиска

Студент гр. 9303 _____ Лойконен М.Р.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Реализовать одну из представленных структур данных для быстрого поиска.

Задание.

Вариант 8.

БДП: случайное* БДП; действие: 1+2б

1) По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;

2б) 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.

Основные теоретические положения.

Двоичные деревья представляют собой иерархическую структуру, в которой каждый узел имеет не более двух потомков.

Двоичное дерево упорядоченно, если для любой его вершины x справедливы такие свойства: все элементы в левом поддереве меньше элемента, хранимого в x; все элементы в правом поддереве больше элемента, хранимого в x; все элементы дерева различны.

Случайные деревья поиска представляют собой упорядоченные бинарные деревья поиска, при создании которых элементы (их ключи) вставляются в случайном порядке.

Выполнение работы.

В ходе выполнения работы был реализован класс `BinSearchTree`, который содержит структуру `Node` — структура узла дерева, имеет поля `int data`, `shared_ptr<Node> left`, `shared_ptr<Node> right`, и

указатель на первый элемент дерева `shared_ptr<Node> head`. Для работы с деревом был создан набор методов, часть из которых являются приватными. `void add(shared_ptr<Node> &node, int el)` — добавляет элемент `el` в дерево, `int findMin(shared_ptr<Node> node)` — возвращает минимальный элемент дерева, `int findMax(shared_ptr<Node> node)` — возвращает максимальный элемент дерева, `void deleteEl(shared_ptr<Node>& node, int el)` — удаляет элемент `el` из дерева, `void print(shared_ptr<Node> node, int color, int level)` — печать дерева на экран. Также были написаны публичные методы: `void addElem(int el)` — вызывает метод добавления элемента, `void deleteElem(int el)` — вызывает метод удаления элемента, `void printTree()` — вызывает метод печати дерева на экран, `bool isEmpty()` — проверяет дерево на отсутствие элементов.

Исходную последовательность элементов, по которой строится дерево, программа считывает с файла.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В ходе выполнения работы была изучена структура данных типа случайное бинарное дерево поиска. Написана программа, в которой был реализован класс данной структуры данных, и написаны методы для работы с ней — добавление элемента, удаление, проверка на пустоту и печать дерева на экран.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <memory>

using namespace std;

class BinSearchTree{
private:
     struct Node{
          int data;
          shared_ptr<Node> left;
          shared_ptr<Node> right;
          Node(int     el):     data(el),     left(nullptr),
right(nullptr) {}
     };
     void add(shared_ptr<Node> &node, int el){
          if (!node){
               node = make_shared<Node>(el);
               return;
          }
          if (el < node->data){
               add(node->left, el);
          }
          else if (el > node->data){
               add(node->right, el);
          }
     }

     int findMin(shared_ptr<Node> node){
          auto cur = node;
          while (cur->left){
               cur = cur->left;
          }
          return cur->data;
     }

     int findMax(shared_ptr<Node> node){
          auto cur = node;
          while (cur->right){
```

```
            cur = cur->right;
        }
        return cur->data;
    }

    void deleteEl(shared_ptr<Node>& node, int el){
        if (!node){
            cout << "Данного элемента нет в дереве\n";
            return;
        }
        if (el > node->data){
            deleteEl(node->right, el);
        }
        else if (el < node->data){
            deleteEl(node->left, el);
        }
        else{
            if (node->right){
                int tmp = findMin(node->right);
                deleteEl(node, tmp);
                node->data = tmp;
            }
            else if (node->left){
                int tmp = findMax(node->left);
                deleteEl(node, tmp);
                node->data = tmp;
            }
            else
                node = nullptr;
        }
    }

    void print(shared_ptr<Node> node, int color, int level)
{
        if (node){
            print(node->right, 2, level+5);
            for (int i = 0; i<level; i++){
                cout << "\033[0;0m ";
            }
            if (color == 1)
                cout << "\033[1;34m" << node->data << "\
033[0;0m" << '\n';
            else if (color == 2)
```

```
                    cout << "\033[1;31m" << node->data << "\
033[0;0m" << '\n';
               else
                    cout << "\033[1;32m" << node->data << "\
033[0;0m" << '\n';
          print(node->left, 3, level+5);
     }
}

     shared_ptr<Node> head;
public:
     BinSearchTree(): head(nullptr) {};
     ~BinSearchTree() = default;

     void addElem(int el){
          add(head, el);
     }

     void deleteElem(int el){
          deleteEl(head, el);
     }

     bool isEmpty(){
          if (head)
               return false;
          else
               return true;
     }

     void printTree(){
          if (head)
               print(head, 1, 0);
          else
               cout << "В дереве нет элементов";
     }

};

int main(){
     string name;
     cout << "Введите название файла: ";
     cin >> name;
     ifstream fin(name);
     BinSearchTree* bst = new BinSearchTree();
```

```
          int elem;
          char opt = 'y';
          cout << "Исходная последовательность элементов: ";
          while(!fin.eof()){
               fin >> elem;
               cout << elem << " ";
               bst->addElem(elem);
          }
          cout << '\n';
          bst->printTree();
          while (opt == 'y'){
               cout << "Введите элемент, который нужно удалить:
";
               cin >> elem;
               bst->deleteElem(elem);
               if (bst->isEmpty()){
                    cout << "В дереве нет элементов\n";
                    break;
               }
               cout << '\n';
               bst->printTree();
               cout << '\n';
               cout << "Хотите удалить ещё один элемент? y - да |
n - нет : ";
               cin >> opt;
          }
          delete bst;
          return 0;
     }
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	test1.txt 15 n	Исходная последовательность элементов: 15 4 1 54 3 0 2 1 57 53 30 8 6 5 60 35 39 7 60 57 54 53 39 35 30 15 8 7 6 5 4 3 2 1 0	Тест пройден.
2.	test2.txt 78 y 240 n	Исходная последовательность элементов: 128 78 24 240 178 400 784 25 0 21 37 784 400 178 128 37 25	Тест пройден.

		24 21 0	
3.	test3.txt -42 n	Исходная последовательность элементов: -4 5 10 327 -24 -42 54 -27 -39 327 54 10 5 -4 -24 -27 -39	Тест пройден.