

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**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: БДП**

Студент гр. 9384

\_\_\_\_\_

Николаев А.А.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Реализовать рандомизированное БДП. И реализовать метод нахождения количества вхождений указанного ключа в БДП.

### **Задание.**

#### **ВАРИАНТ 12.**

БДП. Рандомизированная дерамида поиска (treap). Действие 1+2a

### **Выполнение работы.**

Был описан шаблонный класс Treap, создающий БДП, а так же методы добавления нового элемента и поиска кол-ва вхождений ключа.

### **Тестирование.**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
2 2 2 4 4 1 5 1 10 15 6	1 1 2 2 2 4 4 5 6 10 15
Find(2)	3
Ahsdsfugals	a a a d f g h l s s s u
Find(a)	3

### **Выводы.**

В ходе выполнения лабораторной работы было реализовано БДП.

## ПРИЛОЖЕНИЕ

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "treap.h"

int main()
{
    /*srand(time(0));
    char nums[] = { "ahsdsfugals" };
    int a = sizeof(nums)/sizeof(char);

    Treap<char> root;
    for (char n: nums)
        root.insert(n);
    std::cout << "Constructed Treap:";
    root.print();
    std::cout << "\n";
    std::cout << root.find('a');
    std::cout << "\n";
    root.insert('a');
    std::cout << "Constructed Treap:";
    root.print();
    std::cout << "\n";
    std::cout << root.find('a');*/

    srand(time(0));
    Treap<int> treap;
    std::cout << "File (0) or Console(1)?\n";
    int tmp;
    std::cin >> tmp;

    std::vector<int> vec;
    if (tmp == 0)
    {
        std::string path;
        std::cout << "Enter file path: ";
        std::cin >> path;
        std::ifstream file(path);
        if (file.is_open())
```

```

    {
        int temp;
        while (file >> temp)
        {
            vec.push_back(temp);
        }
    }
    file.close();
}
else if (tmp == 1)
{
    size_t size;
    std::cout << "Enter size: ";
    std::cin >> size;

    int temp;
    for (size_t index = 0; index < size; index++)
    {
        std::cout << "Enter [" << index << "] index: ";
        std::cin >> temp;
        vec.push_back(temp);
    }
}
else
{
    std::cout << "Not supported type!\n";
    return 1;
}

for (int num : vec)
    treap.insert(num);
std::cout << "\n===== \n";
treap.print();
std::cout << "\n===== \n";

for (;;)
{
    std::cout << "\ninsert (0), find(1), exit(2)?\n";
    std::cin >> tmp;

    if (tmp == 0)
    {
        std::cout << "Enter num: ";
        int num;
    }
}

```

```

        std::cin >> num;
        treap.insert(num);
    }
    else if (tmp == 1)
    {
        std::cout << "Enter num: ";
        int num;
        std::cin >> num;
        std::cout << treap.find(num);
    }
    else if (tmp == 2)
    {
        break;
    }
    else
    {
        std::cout << "Not supported type!\n";
    }

    std::cout << "\n===== \n";
    treap.print();
    std::cout << "\n===== \n";

}

return 0;
}

```

Название файла: Treap.h

```

#ifndef TREAP_H
#define TREAP_H

#include <iostream>
#include <cstdlib>
#include <ctime>

template <typename T>
class Treap
{
public:
    Treap() {}

```

```
Treap(T key, int priority) : key(key), priority(priority), left(nullptr), right(nullptr)
{};
```

```
void insert(T key)
{
    _insert(root, new Treap<T>(key, rand()));
}
```

```
void print()
{
    _print(root);
}
```

```
int find(T key)
{
    count = 0;
    return _find(root, key);
}
```

private:

```
void split(Treap* t, T& key, Treap*& left, Treap*& right)
{
    if (t == nullptr)
    {
        left = right = nullptr;
    }
    else if (key < t->key)
    {
        split(t->left, key, left, t->left);
        right = t;
    }
    else
    {
        split(t->right, key, t->right, right);
        left = t;
    }
}
```

```
void merge(Treap*& t, Treap* left, Treap* right)
{
    if (!left || !right)
        t = left ? left : right;
    else if (left->priority >= right->priority)
    {

```

```

        merge(left->right, left->right, right);
        t = left;
    }
    else
    {
        merge(right->left, left, right->left);
        t = right;
    }
}

void _insert(Treap*& t, Treap* it)
{
    if (t == nullptr)
    {
        t = it;
        return;
    }

    if (it->priority > t->priority)
    {
        split(t, it->key, it->left, it->right);
        t = it;
    }
    else
    {
        _insert(it->key < t->key ? t->left : t->right, it);
    }
}

void _print(Treap*& t)
{
    if (t == nullptr) return;
    this->_print(t->left);
    std::cout << t->key << ' ';
    this->_print(t->right);
}

int _find(Treap*& t, T key)
{
    if (t != nullptr)
    {
        if (t->key == key)
        {
            count += 1;
        }
    }
}

```



```

        _find(t->left, key);
        _find(t->right, key);
    }
    else if (t->key > key)
    {
        _find(t->left, key);
    }
    else
    {
        _find(t->right, key);
    }
}
return count;
}
private:
    T key;
    int priority;
    Treap* left, * right;
    Treap* root = nullptr;
    size_t count;
};
#endif // TREAP_H

```