ММИНОБРНАУКИ 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.

Студент гр. 9384 _____ Прашутинский К.И.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

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

Задание.

В вариантах заданий 1-ой группы (кодирование и декодирование) на вход подаётся файл с закодированным или незакодированным содержимым. Требуется раскодировать или закодировать содержимое файла определённым алгоритмом.

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.

Вариант 14: БДП: Рандомизированная дерамида поиска (treap); действие: 1+2в

Выполнение работы.

Было создано рандомизированная дерамида поиска по входящим данным. Затем, выводит количество найденных элементов. Затем, выводит дерево.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
abcdasdfgfhdsfrgrghjnki	Treap is: a a< b\ c d/ d< d f< f f< g g< g\ h h/ i< j k< n r/ r< s s/

Выводы.

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

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

int main()
{
	srand(time(0));
	std::ifstream input("test.txt");
	std::string str;
	treap<char> root;
	int i = 0;
	if (!input.is_open()) {
		std::cerr << "File not open!\n";
		return 1;
	}

	input >> str;



	while (str[i]!='\0') {
		root.insert(str[i]);
		i++;
	}
	std::cout << root.find('f') << '\n';

	std::cout << "Treap is:" << '\n';
	root.print();

	input.close();
	system("pause>>void");
	return 0;
}
```

Название файла: treap.h

```
#ifndef TREAP_H
#define TREAP_H

#include <iostream>


template <typename base>
class treap
{
public:
	treap():key(0), priority(0), left(nullptr), right(nullptr) {};
	treap(base key, int priority) : key(key), priority(priority), left(nullptr), right(nullptr) {};
```

```
	void insert(base key)
	{
		InsertTree(root, new treap<base>(key, rand()));
	}

	int find(base key)
	{
		count = 0;
		return FindTree(root, key);
	}

	void print()
	{
		PrintTree(root, 0);
	}

	int HeightTree() {
		if (this->left && this->right)
			return this->left->HeightTree() > this->right->HeightTree() ? 1 + this->left->HeightTree() : 1 + this->right->HeightTree();
		else if (this->left) {
			return 1 + this->left->HeightTree();
		}
		else if (this->right) {
			return 1 + this->right->HeightTree();
		}
		else return 1;
	}

private:
	void split(treap* t, base& 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 InsertTree(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
	{
		InsertTree(it->key < t->key ? t->left : t->right, it);
	}
}

int FindTree(treap*& t, base key)
{
	if (t != nullptr)
	{
		if (t->key == key)
		{
			count += 1;
			FindTree(t->left, key);
			FindTree(t->right, key);
		}
		else if (t->key > key)
		{
			FindTree(t->left, key);
		}
		else
		{
			FindTree(t->right, key);
		}
	}
	return count;
}

void PrintTree(treap*& t, int level)
{
	if (t)
	{
		if (!t) return;
		level += 5;
		PrintTree(t->left, level);
		for (int i = 0; i < level; i++) std::cout << " ";
```

```
                        if(t->left && t->right)
                                std::cout << t->key << '<' <<std::endl;
                        else if (t->left)
                                std::cout << t->key << '/' << std::endl;
                        else if (t->right)
                                std::cout << t->key << '\\' << std::endl;
                        else
                                std::cout << t->key << std::endl;
                        PrintTree(t->right, level);
                        level -= 5;
                        return;
                }
        }

        base key;
        int priority;
        treap* left, *right;
        treap* root = nullptr;
        size_t count;
};
#endif // treap_H
```