МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: кодирование и декодирование, БДП, хеш-таблицы, сортировки

Студент гр. 8304 _____ Кирьянов Д.И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Вариант 13

Цель работы

Изучить основные способы кодирования и декодирования, сортировки, представления данных в виде БДП и хеш-таблиц.

Задание

По заданному файлу F (типа file of Elem), все элементы которого различны, построить Рандомизированное БДП.

Для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если не входит, то добавить элемент *e* в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

Описание алгоритма

1. Открывается файл, либо считывается с консоли строка.
2. Создается вектор указателей.
3. В каждом указателе инициализируются поля. Поле ключа в зависимости от введенных пользователем значений. Поле приоритета – рандомизацией.
4. Исходя из предположения о корректности, на основе полученных данных в памяти формируется Treap-структура.
5. Помним последний добавленный y(k), добавляем y(k+1)
6. y(k+1) > y(k) => y(k+1) = right(y(k))
7. Иначе идем вверх пока y(l) <y(k+1)
8. y(k+1) = right(y(l))
9. y(l).pre_right = y(k+1).left
10. После выполнения алгоритма, полученное дерево выводится в порядке клп.
11. Реализована функция для общения с пользователем, где ему предлагается еще ввести элемены для добавления в структуру. При каждом добавлении дерево выводится для отслеживания изменений.
12. После всех преобразований выводится конечный вид дерева.

Описание основных функций

```
void create_int_vec(std::vector<nodePtr<int, int>>& mypairs, std::string& array)
```

-Функция для создания вектора указателей.

```
static void insert(nodePtr<elem, priority>& t, nodePtr<elem, priority> it)
static void split(nodePtr<elem, priority> head, elem key, nodePtr<elem, priority>& left,
nodePtr<elem, priority>& right)
```

-Метод класса для вставки нового элемента в дерево, а также вспомогательный метод.

```
void printtree(nodePtr<elem, priority>& head)
```

-Функция для вывода хранящегося дерева.

```
void dialog(nodePtr<elem, priority>& head)
```

-Функция для общения с пользователем.

```
static bool check_rep(nodePtr<elem, priority>& t, elem key)
```

-Метод класса, проверяющий наличие введенного элемента в дереве.

Вывод.

Был получен опыт работы с Treap-структурами данных. Изучены основные способы кодирования и декодирования, сортировки, представления данных в виде БДП и хеш-таблиц.

ПРИЛОЖЕНИЕ А

Тестирование программы

```
3 8 11
3 20998
8 17648
11 27355
((11;27355)((3;20998)#((8;17648)##))#)
Do you want to insert new element?
Please, enter.
'1' - Yes
'2' - No
1
Please, enter a key
5
((5;30868)((3;20998)##)((11;27355)((8;17648)##)#))
Do you want to insert another one?
Please, enter.
'1' - Yes
'2' - No
2
((5;30868)((3;20998)##)((11;27355)((8;17648)##)#))
```

ПРИЛОЖЕНИЕ Б

Файл lab5.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <ctime>
#include <memory>
#include <climits>

template<typename elem, typename priority>
class Node;
template<typename elem, typename priority>
using nodePtr = std::shared_ptr<Node<elem, priority>>;

template<typename elem, typename priority>
class Node {
public:
	elem key, prior;
	nodePtr<elem, priority> left, right = nullptr;
	Node() = default;
	~Node() = default;
	static void split(nodePtr<elem, priority> head, elem key, nodePtr<elem, priority>&
left, nodePtr<elem, priority>& right) {
		if (!head) {
			left = nullptr;
			right = nullptr;
		}
		else if (key < head->key) {
			split(head->left, key, left, head->left);
			right = head;
		}
		else {
			split(head->right, key, head->right, right);
			left = head;
		}
	}
	static void insert(nodePtr<elem, priority>& t, nodePtr<elem, priority> it) {
		if (!t)
			t = it;
		else if (it->prior > t->prior) {
			split(t, it->key, it->left, it->right);
			t = it;
		}
		else
			insert(it->key < t->key ? t->left : t->right, it);
	}
	static bool check_rep(nodePtr<elem, priority>& t, elem key) {
		bool result = true;
		if (t->key == key)
			return false;
		else
			result=check_rep(key < t->key ? t->left : t->right, key);
		return result;
	}
};

void create_int_vec(std::vector<nodePtr<int, int>>& mypairs, std::string& array){
	srand((unsigned int)time(0));
	unsigned long int index = 0;
	while (array[index] == ' ')
		++index;
```

```
	while (index != array.length()) {
		nodePtr<int, int> el = std::make_shared<Node<int, int>>();
		el->key = std::stoi(array.substr(index));
		el->prior = rand() % INT_MAX;
		mypairs.push_back(el);
		while (isdigit(array[index]))
			++index;
		while (array[index] == ' ')
			++index;
	}
}

template<typename elem, typename priority>
void dialog(nodePtr<elem, priority>& head) {
	int flag=0;
	std::cin >> flag;
	nodePtr<int, int> new_el = std::make_shared<Node<int, int>>();
	switch (flag){
	case(1):
		std::cout << "Please, enter a key" << std::endl;
		std::cin >> new_el->key;
		new_el->prior = rand() % INT_MAX;
		Node<int, int>::insert(head, new_el);
		printtree(head);
		std::cout << std::endl;
		std::cout << "Do you want to insert another one?" << std::endl << "Please, en-
ter." << std::endl;
		std::cout << "'1' - Yes" << std::endl << "'2' - No" << std::endl;
		dialog(head);
		break;
	case(2):
		break;
	default:
		std::cout << "Wrong symbol, please enter again" << std::endl;
		dialog(head);
		break;
	}
}
template<typename elem, typename  priority>
void printtree(nodePtr<elem, priority>& head) {
	if (!head) {
		std::cout << '#';
		return;
	}
	std::cout << '(';
	std::cout<< "(" << head->key << ";" << head->prior<<")";
	printtree(head->left);
	printtree(head->right);
	std::cout << ')';
}

int main(int argc, char* argv[]) {
	std::string array;
	if (argc == 1) {
		std::getline(std::cin, array);
		std::vector<nodePtr<int, int>> mypairs;
		create_int_vec(mypairs, array);
		for (unsigned long int i = 0; i < mypairs.size(); i++) {
			std::cout << mypairs[i]->key << ' ' << mypairs[i]->prior << std::endl;
		}
		nodePtr<int, int> head = nullptr;
		for (size_t i = 0; i < mypairs.size(); ++i){
			Node<int, int>::insert(head, mypairs[i]);
		}
		printtree(head);
```

```
		std::cout << std::endl;
		std::cout << "Do you want to insert new element?" << std::endl << "Please, en-
ter." << std::endl;
		std::cout << "'1' - Yes" << std::endl << "'2' - No" << std::endl;
		dialog(head);
		printtree(head);
		std::cout << std::endl;
	}
	else {
		std::ifstream in(argv[1]);
		if (!in.is_open()) {
			std::cout << "Can't open file" << std::endl;
			return 0;
		}
		while (std::getline(in, array)) {
			std::cout << array << "\n";
			std::vector<nodePtr<int, int>> mypairs;
			create_int_vec(mypairs, array);
			for (unsigned long int i = 0; i < mypairs.size(); i++) {
				std::cout << mypairs[i]->key << ' ' << mypairs[i]->prior <<
std::endl;
			}
			nodePtr<int, int> head = nullptr;
			for (size_t i = 0; i < mypairs.size(); ++i) {
				Node<int, int>::insert(head, mypairs[i]);
			}
			printtree(head);
			std::cout << std::endl;
			std::cout << "Do you want to insert new element?" << std::endl <<
"Please, enter." << std::endl;
			std::cout << "'1' - Yes" << std::endl << "'2' - No" << std::endl;
			dialog(head);
			printtree(head);
			std::cout << std::endl;
		}
		in.close();
	}
	return 0;
}
```