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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Рандомизированное БДП

Студент гр. 8304 _____ Кирьянов Д.И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Кирьянов Д.И.

Группа 8304

Тема работы: Рандомизированное БДП

Исходные данные: необходимо выполнить демонстрацию алгоритма вставки и удаления в рандоминизированное бинарное дерево поиска, включающее возможность самостоятельно редактировать дерево.

Содержание пояснительной записки:
«Содержание», «Введение», «Задание», «Описание программы», «Тестирование», «Исследование», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент		Кирьянов Д.И.
Преподаватель		Фирсов М.А.

АННОТАЦИЯ

В ходе выполнения курсовой работы была разработана программа с GUI, позволяющая исследовать алгоритм вставки в рандоминизированное бинарное дерево поиска, а также удаление заданного элемента. Программа обладает следующей функциональностью: построение РБДП по входному файлу, работа с пользовательской консолью, вывод РБДП.

SUMMARY

In the course work a program with a GUI was developed that allows you to examine the algorithm for inserting into a randomized binary search tree, as well as deleting a given element. The program has the following functionality: building a Treap from an input file, UI, Treap output.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Реализация и демонстрация алгоритмов работы с рандомизированными бинарными деревьями поиска.

Основные задачи

Демонстрация изменений рандомизированного бинарного дерева поиска при использовании доступных для пользователя команд.

Методы решения

Разработка программы велась на базе операционной системы Windows 10 в среде разработки MSVS 2019. Язык C++ стандарта 2017 года.

1. ЗАДАНИЕ

Необходимо провести визуализацию структур данных, алгоритмов, действий. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание основного класса для рандоминизированных бдп

Для реализации бдп был создан шаблонный класс Node<typename elem, typename Priority>. Основные методы класса представлены в табл. 5.

Таблица 5 – Основные функции работы с декартовым деревом

Функция	Назначение
Node() = default;	Конструктор по умолчанию
~Node() = default;	Деструктор по умолчанию
static void split(nodePtr<elem, priority>, elem, nodePtr<elem, priority> &, nodePtr<elem, priority> &);	Разбиение по ключу
static void insert(nodePtr<elem, priority> &, nodePtr<elem, priority>);	Вставка нового элемента
static void merge(nodePtr<elem, priority> &, nodePtr<elem, priority>, nodePtr<elem, priority>);	Слияние двух РБДП
static bool erase(nodePtr<elem, priority> &, elem);	Удаление элемента

Программа имеет возможность отображения дерева в графической записи.

2.2. Описание алгоритма вставки и удаления в рандоминизированном бдп

Известно, что если заранее перемешать как следует все ключи и потом построить из них дерево (ключи вставляются по стандартной схеме в полученном после перемешивания порядке), то построенное дерево окажется неплохо сбалансированным (его высота будет порядка $2log_2 n$ против $log_2 n$ для

идеально сбалансированного дерева). Любой вводимый ключ может оказаться корнем с вероятностью $\frac{1}{n+1}$ (n — размер дерева до вставки), следовательно выполняется с указанной вероятностью вставка в корень, а с вероятностью 1-$\frac{1}{n+1}$ — рекурсивную вставка в правое или левое поддерево в зависимости от значения ключа в корне.

Удаление происходит по ключу — ищется узел с заданным ключом и этот узел удаляется из дерева. Основное свойство дерева поиска — любой ключ в левом поддереве меньше корневого ключа, а в правом поддереве — больше корневого ключа. Это свойство позволяет очень просто организовать поиск заданного ключа, перемещаясь от корня вправо или влево в зависимости от значения корневого ключа. Далее происходит объединение левого и правого поддеревьев найденного узла, удаляется узел и возвращается корень объединенного дерева.

3. ТЕСТИРОВАНИЕ

3.1. Вид программы

Вид программы после запуска представлен на рис. 2.

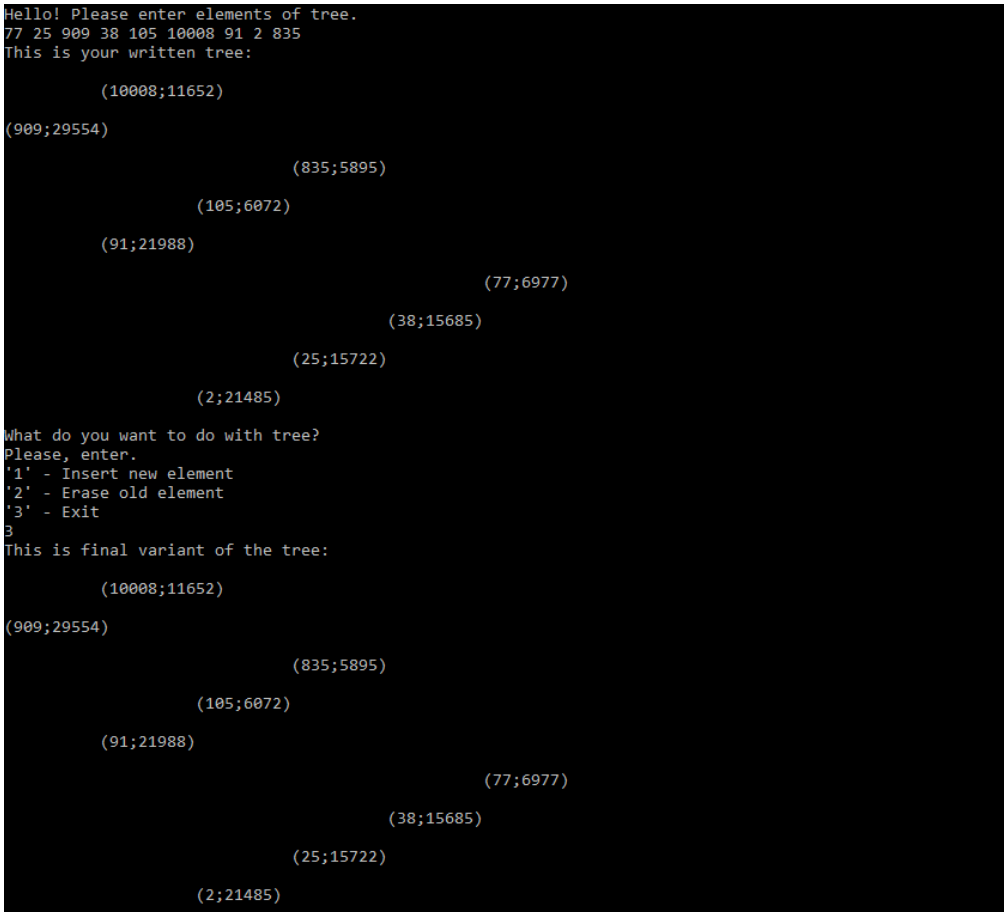
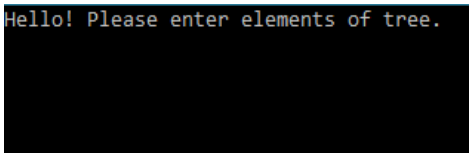


Рисунок 2 – Вид программы после запуска

4. ДЕМОНСТРАЦИЯ

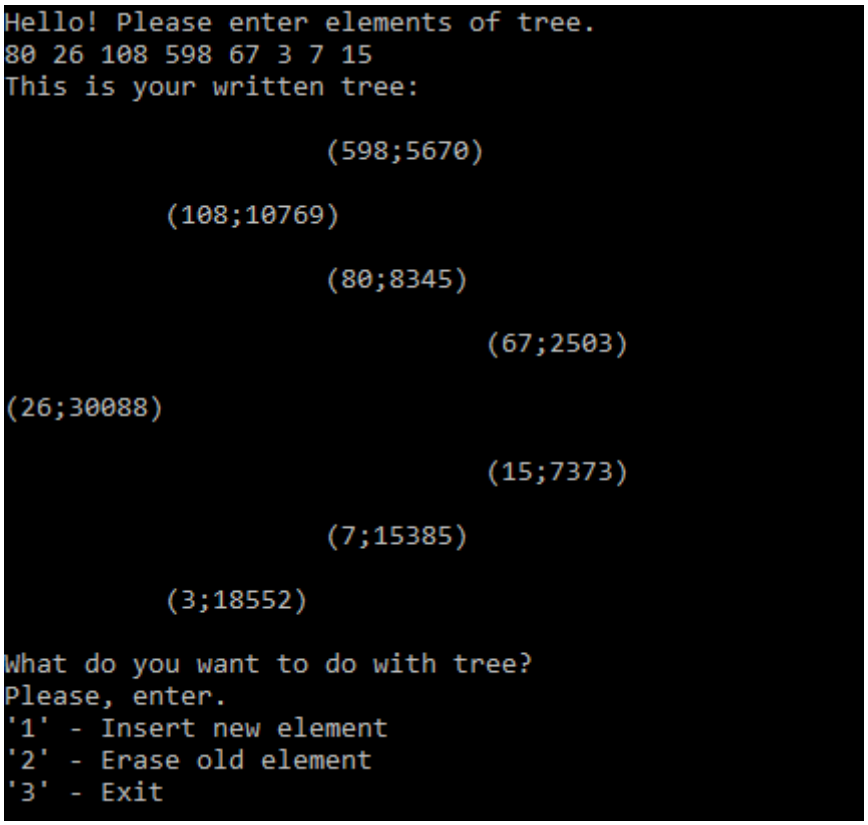
4.1. Начало работы.

При запуске программы выводится приветствие пользователя с просьбой ввести элементы бдп.



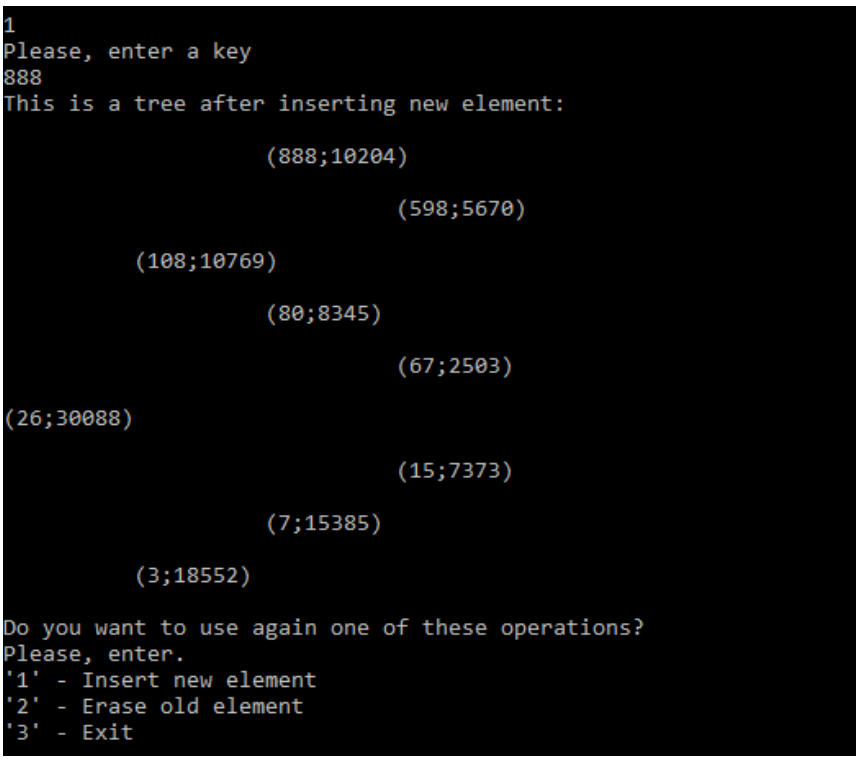
4.2. Демонстрация введенного дерева.

После введения пользователем всех элементов дерева выводится его графическое изображение. После чего программой предлагается функционал для работы с данным деревом.



4.3. Демонстрации вставки элемента.

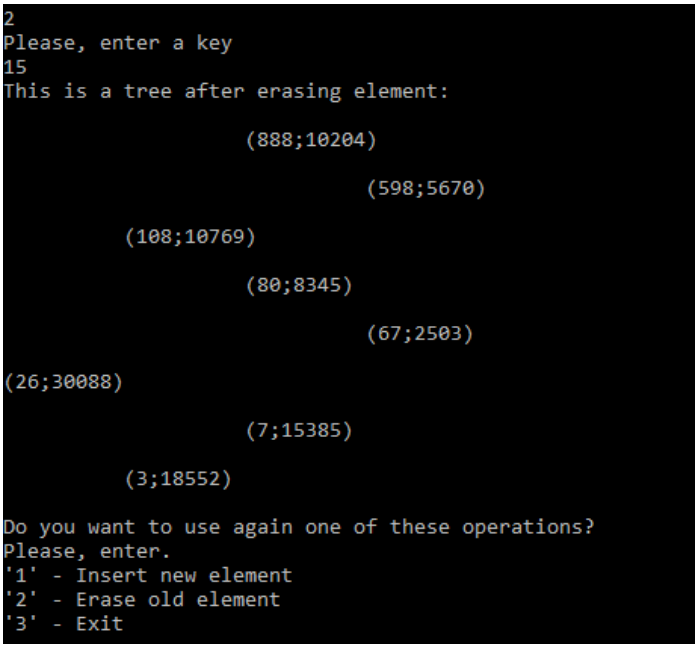
При выборе пользователем вставки элемента программа просит ввести требуемый элемент, после чего добавляет его в дерево и выводит его обновленный вид.



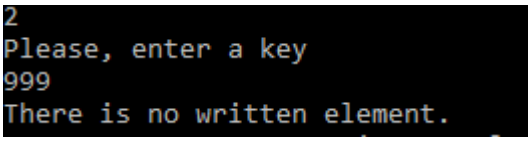
После вывода обновленного дерево пользователю снова предлагается весь функционал.

4.4. Демонстрация удаления элемента.

При выборе пользователем удаления элемента программа просит ввести требуемый элемент, после чего убирает его из дерева и выводит его обновленный вид.



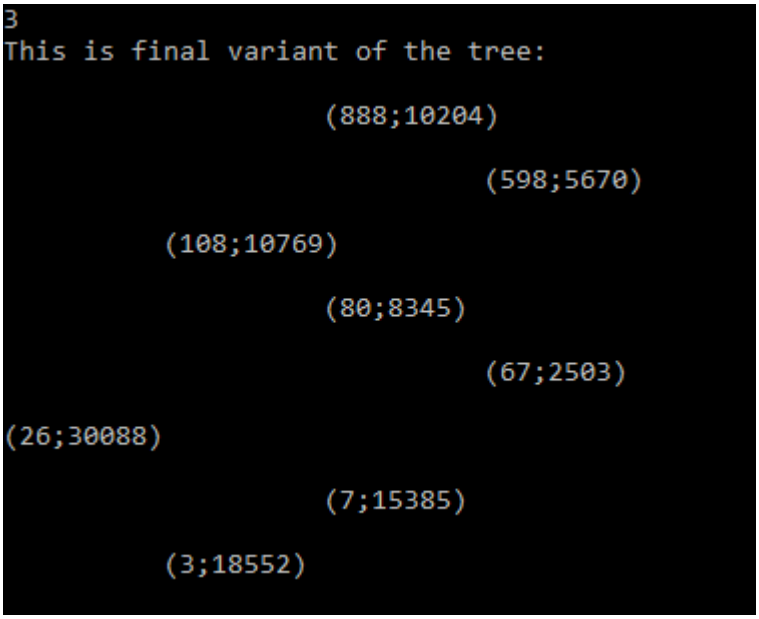
Если введенный элемент не был найден, то программа сообщает нам об этом и удаление соответственно не выполняется.



После выполнения всех действий пользователю снова предлагается весь функционал.

4.5. Выход из программы.

При выборе пользователем выхода из программы выводится итоговый вид преобразованного дерева, после чего программа завершает свою работу.



ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа, которая обладает следующей функциональностью: построение РБПД по входному файлу, графический вывод РБПД, работа с пользовательской консолью. С помощью программы было проведена демонстрация всего функционала алгоритма вставки в рандомизированное бинарное дерево и алгоритма удаления заданного значения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bjarne Stroustrup. A Tour of C++. M.: Addison-Wesley, 2018. 217 с.
2. Treap // GeeksforGeeks https://www.geeksforgeeks.org/treap-a-randomized-binary-search-tree/ (дата обращения: 18.12.2000)
3. Qt Documentation // Qt. URL: https://doc.qt.io/qt-5/index.html (дата обращения: 18.12.2000)
4. Рандомизированные деревья поиска URL: https://habr.com/ru/post/145388/#reed (дата обращения: 17.12.2000)
5. The Height of a Random Binary Search Tree // BRUCE REED URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.1289&rep=rep1&type=pdf (дата обращения: 18.12.2000)

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. CW.CPP

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <ctime>
#include <memory>
#include <climits>
#define COUNT 10

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 void merge(nodePtr<elem, priority>& t, nodePtr<elem, priority> l, nodePtr<elem,
priority> r) {
		if (!l || !r)
			t = l ? l : r;
		else if (l->prior > r->prior) {
			merge(l->right, l->right, r);
			t = l;
		}
		else {
			merge(r->left, l, r->left);
			t = r;
		}
	}
```

```
	static bool erase(nodePtr<elem, priority>& t, elem key) {
		if (!t)
			return false;
		if (t->key == key) {
			merge(t, t->left, t->right);

			return true;
		}
		return erase(key < t->key ? t->left : t->right, key);
	}
};

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) {
	if (flag==0)
		std::cin >> flag;
	nodePtr<int, int> new_el = std::make_shared<Node<int, int>>();
	elem element;
	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);
		std::cout << "This is a tree after inserting new element:" << std::endl;
		printtree(head, 0);
		std::cout << std::endl;
		std::cout << "Do you want to use again one of these operations?" << std::endl
<< "Please, enter." << std::endl;
		std::cout << "'1' - Insert new element" << std::endl << "'2' - Erase old
element" << std::endl << "'3' - Exit" << std::endl;
		dialog(head, 0);
		break;
	case(2):
		std::cout << "Please, enter a key" << std::endl;
		std::cin >> element;
		if (Node<int, int>::erase(head, element)) {
			std::cout << "This is a tree after erasing element:" << std::endl;
			printtree(head, 0);
			std::cout << std::endl;
		}
		else
			std::cout << "There is no written element." << std::endl;
		std::cout << "Do you want to use again one of these operations?" << std::endl
<< "Please, enter." << std::endl;
		std::cout << "'1' - Insert new element" << std::endl << "'2' - Erase old
element" << std::endl << "'3' - Exit" << std::endl;
```

```
                dialog(head, 0);
                break;
        case(3):
                break;
        default:
                std::cout << "Wrong symbol, please enter again" << std::endl;
                dialog(head, 0);
                break;
        }
}
template<typename elem, typename priority>
void printtree(nodePtr<elem, priority>& root, int space) {
        if (root == nullptr)
                return;
        space += COUNT;
        printtree(root->right, space);
        std::cout << std::endl;
        for (int i = COUNT; i < space; i++)
                std::cout << " ";
        std::cout << "(" << root->key << ";" << root->prior << ")\n";
        printtree(root->left, space);
}

int main(int argc, char* argv[]) {
        std::cout << "Hello! Please enter elements of tree." << std::endl;
        std::string array;
        if (argc == 1) {
                std::getline(std::cin, array);
                std::vector<nodePtr<int, int>> mypairs;
                create_int_vec(mypairs, array);
                nodePtr<int, int> head = nullptr;
                for (size_t i = 0; i < mypairs.size(); ++i) {
                        Node<int, int>::insert(head, mypairs[i]);
                }
                std::cout << "This is your written tree:" << std::endl;
                printtree(head, 0);
                std::cout << std::endl;
                std::cout << "What do you want to do with tree?" << std::endl << "Please,
enter." << std::endl;
                std::cout << "'1' - Insert new element" << std::endl << "'2' - Erase old
element" << std::endl << "'3' - Exit" << std::endl;
                dialog(head, 0);
                std::cout << "This is final variant of the tree:" << std::endl;
                printtree(head, 0);
                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);
                        nodePtr<int, int> head = nullptr;
                        for (size_t i = 0; i < mypairs.size(); ++i) {
                                Node<int, int>::insert(head, mypairs[i]);
                        }
                        std::cout << "This is your written tree:" << std::endl;
                        printtree(head, 0);
                        std::cout << std::endl;
```

```
                    std::cout << "What do you want to do with tree?" << std::endl <<
"Please, enter." << std::endl;
                    std::cout << "'1' - Insert new element" << std::endl << "'2' - Erase
old element" << std::endl << "'3' - Exit" << std::endl;
                    dialog(head, 0);
                    std::cout << "This is final variant of the tree:" << std::endl;
                    printtree(head, 0);
                    std::cout << std::endl;
                }
                in.close();
        }
        return 0;
}
```