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: Рандомизированные дерамиды поиска – вставка и исключение.

Текущий контроль

Студент гр. 9382 _____ Павлов Р.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Павлов Р.В.

Группа 9382

Тема работы: Рандомизированные дерамиды поиска – вставка и исключение.

Текущий контроль (14 вариант)
Исходные данные:

"Текущий контроль" - создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам, проходящим ТК; все задания должны касаться конкретных экземпляров структуры данных (т.е. не должны быть вопросами по теории); ответы должны позволять удобную проверку правильности выполнения заданий.
Содержание пояснительной записки:

«Содержание», «Введение», «Структура программы», «Тестирование» «Выводы», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 13.12.2020

Дата защиты реферата: 13.12.2020

Студент _____ Павлов Р.В.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

Курсовая работа представляет собой программу, предназначенную для генерации заданий для проведения текущего контроля. По заданному количеству вариантов создаются наборы , состоящие из трёх заданий, а также в процессе их генерации в отдельные файлы записываются ответы к ним, полученные с помощью соответствующих алгоритмов решения. После завершения работы программы доступны файлы с заданиями и файлы с ответами, лежащие в двух отдельных папках.

Код программы приведён в приложении А.

SUMMARY

This course work represents the program, intended for generation of tasks that are used for monitoring. For a given quantity, sets consisting of three tasks are created, and algorithm-derived answers to them are written to separate files during generation process. After terminating the program the files with tasks and the files with answers are available in two separate folders.

Progrgam code is given in attachment A.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: создание программы, генерирующей набор заданий и ответов к ним для проведения текущего контроля по заданной теме.

Задачи:

- реализация соответствующей заданию структуры данных (дерамида) и связанных с ней операций
- написание функций, генерирующих случайные структуры, задания различных типов и ответы к ним
- добавление записи полученных данных в файлы

Дерамида — это структура данных, которая хранит пары (x, y) в виде бинарного дерева таким образом, что она является бинарным деревом поиска по x и бинарной пирамидой по y. Значения x называются ключами, а значения y — приоритетами. Если некоторый элемент дерева содержит (x_0,y_0), то у всех элементов в левом поддереве $x < x_0$, у всех элементов в правом поддереве $x > x_0$, а также и в левом, и в правом поддереве $y < y_0$.

Разработка велась в IDE Microsoft Visual Studio версии 4.8.03752 на ОС Windows 10.

В результате была создана программа, которая в указанном пользователем количестве генерирует наборы заданий для проведения текущего контроля.

1. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1. Типы заданий

1) Задание на скобочное представление дерамиды. Ответом служит сокращённое скобочное представление дерева.

2) Задание на перечисление узлов дерамиды слева направо по уровням, начиная с первого, после добавления нового элемента в дерамиду / удаления из неё существующего элемента. Ответ — последовательность узлов результирующей дерамиды в указанном порядке.

3) Задание на перечисление узлов дерамиды в прямом/обратном/концевом порядках (КЛП-, ЛКП- и ЛПК-обходы соответственно). Ответ — последовательность узлов исходной дерамиды в указанном порядке.

Примечание: элементы записываются в формате *(ключ:приоритет)*

1.2. Описание алгоритма

1) Указывается количество вариантов, которое требуется сгенерировать, затем оно передаётся в функцию-генератор, которая соответствующее количество раз повторяет создание вариантов теста (по три задания на вариант).

2) Для каждого типа задания вызывается функция, записывающая взятую из шаблона оформления формулировку задания, а затем и случайно сгенерированную структуру, в файл с заданиями.

3) Из функции генерации задания вызывается функция записи ответа в соответствующей этому заданию форме в файл. Таких функций может существовать несколько для одного типа заданий в зависимости от того, сколько вариаций имеет задание данного типа.

Алгоритмы нахождения ответа:

- Первый тип заданий:

Алгоритм начинает работу с корня дерева. Печатается открывающая скобка, затем текущий элемент в заданном формате, после чего рекурсивно вызывается тот же алгоритм сначала для левого, а затем и для правого поддерева. В конце печатается закрывающая скобка.

- Второй тип заданий:

Выполняется заранее выбранная операция (добавление/удаление элемента), после чего h раз (где h – высота дерева) вызывается функция для печати элементов, находящихся на одном уровне, для каждого уровня. Алгоритм рекурсивно вызывает сам себя, если текущий уровень меньше, чем требуемый. Если же указанный уровень достигнут, в файл записывается элемент. Поскольку используется обход дерева слева направо, то и элементы одного уровня выводятся в том же направлении.

- Третий тип заданий:

В зависимости от того, какой был задан порядок обхода, вызываются соответствующие функции:

  - КЛП-обход — запись элемента в файл, спуск по левой ветви, спуск по правой ветви
  - ЛКП-обход — спуск по левой ветви, запись элемента в файл, спуск по правой ветви

- ЛПК-обход — спуск по левой ветви, спуск по правой ветви, запись элемента в файл

1.3. Описание функций и стрктур данных

- **template <typename T> struct Pair {**

  T x;

  T y;

  Pair(T x, T y) : x(x), y(y) {}

  }; –

  структура из пары значений (здесь используется либо пара значений **int**, либо пара указателей на **Node**)
- **struct Node {**

  Pair<int> p;

  Node* l, * r;

  Node(int x, int y) : p(Pair<int>(x, y)), l(nullptr), r(nullptr) {}

  Node(int x, int y, Node* l, Node* r) : p(Pair<int>(x, y)), l(l), r(r) {}

  }; –

  Структура узла дерамиды. Содержит пару целочисленных значений и указатели на левое и правое поддеревья, а также два конструктора.
- **void indent(int n, ostream& write = cout)** – вывод отступа, соответствующего глубине рекурсии; n – глубина рекурсии
- **void dispose(Node* t) —** освобождение памяти от данных дерамиды; t – указатель на корень дерева
- **bool isIn(Node* t, int k)** — проверка на вхождение элемента с указанным ключом в дерамиду; t - указатель на корень дерева, k – ключ искомого элемента

- **int height(Node* t) —** нахождение высоты дерамиды; t – указатель на корень дерева
- **Node* merge(Node* l, Node* r, int& depth)** — слияние двух поддеревьев; l – указатель на «левое» поддерево, r - указатель на «правое» поддерево, depth – сслыка на счётчик, отслеживающий глубину
- **Pair<Node*> split(Node* t, int k, int& depth) —** разбиение дерева по ключу; t – указатель на корень дерева, k – ключ, по которому разбивается дерамида, depth – сслыка на счётчик, отслеживающий глубину
- **Node* add(Node* t, int k, int v)** — добавление элемента в дерамиду; t – указатель на корень дерева, k – ключ добавляемого элемента, v – значение добавляемого элемента
- **Node* remove(Node* t, int k, int& depth)** – удаление элемента из дерамиды; t – указатель на корень дерева, k – ключ удаляемого элемента, depth – сслыка на счётчик, отслеживающий глубину
- **void printTreap(Node* t, int& depth, ostream& write)** – вывод дерамиды в указанный поток в наглядной форме; t – указатель на корень дерева, depth – ссылка на счётчик, отслеживающий глубину, write — ссылка на поток вывода
- **ostream& operator<<(ostream& out, const Node& t)** – вывод дерамиды на экран; out – ссылка на поток вывода, t – ссылка на корень дерева
- **Node* getRandom(Node* t, int& depth, int h)** – получение случайного элемента дерева; t – указатель на корень дерева, depth – ссылка на счётчик, отслеживающий глубину, h – случайно выбранный в пределах высоты уровень, при достижении которого, даже если элемент не концевой, он возвращается
- **Node* generateTreap()** - генерация случайной дерамиды
- **void writeBracketsAnswer(Node* t, ostream& answer)** – запись дерамиды в скобочном представлении; t - указатель на корень дерева, answer – ссылка на поток вывода

- **void writeLevelsAnswer(Node* t, ostream& answer, int& depth, int level) –** перечисление узлов на указанном уровне слева направо; t - указатель на корень дерева, answer – ссылка на поток вывода, depth – ссылка на счётчик, отслеживающий глубину, level – требуемый уровень
- **void writeKLPAnswer(Node* t, ostream& answer)** – перечисление узлов дерамиды в прямом порядке; t - указатель на корень дерева, answer – ссылка на поток вывода
- **void writeLKPAnswer(Node* t, ostream& answer)** – перечисление узлов дерамиды в обратном порядке; t - указатель на корень дерева, answer – ссылка на поток вывода
- **void writeLPKAnswer(Node* t, ostream& answer)** – перечисление узлов дерамиды в концевом порядке; t - указатель на корень дерева, answer – ссылка на поток вывода
- **void generateBrackets(ostream& task, ostream& answer)** – генерация задания на скобочное представление; task – ссылка на поток вывода для текста задания, answer – ссылка на поток вывода для ответа на задание
- **void generateLevels(ostream& task, ostream& answer)** – генерация задания на перечисление узлов по уровням слева направо; task – ссылка на поток вывода для текста задания, answer – ссылка на поток вывода для ответа на задание
- **void generateTraversal(ostream& task, ostream& answer, int cond)** – генерация задания на перечисление узлов в указанном порядке; task – ссылка на поток вывода для текста задания, answer – ссылка на поток вывода для ответа на задание, cond – условие генерации (случайно выбранный порядок обхода)
- **void generateTasks(int n)** – генерация указанного количества вариантов теста; n – количество вариантов, которое необходимо создать
- **int main()** - ввод количества вариантов для генерации из консоли

1.4. Описание интерфейса пользователя

В начале работы на экран выводится приветствие, информация о программе и краткая инструкция по использованию генератора заданий. У пользователя есть возможность указать то количество вариантов теста (от 1 до 30), которое необходимо сгенерировать. При вводе недопустимого числа предлагается ввести его повторно, таким образом исключается возможность создания слишком большого количества заданий или передача программе неположительного числа. Также во время выполнения программы на экран выводятся сообщения о текущих действиях, по которым можно отследить ход её работы.

2. ТЕСТИРОВАНИЕ

Данные, полученные в ходе тестирования, представлены в таблице 1.
Для самостоятельного ознакомления с результатами работы программы запустите её и сгенерируйте несколько вариантов теста, затем проверьте задания в папке “Tasks” и ответы в папке “Answers”.

Таблица 1.

№ варианта	Задания	Ответы
1	1. Запишите скобочное представление дерамиды, представленной ниже. 44:-48 38:-22 33:-23 31:13 30:40 14:5 10:-50 6:-10 2:-20 -35:-3 -38:5 -44:18 -48:5 2. Перечислите узлы следующей дерамиды по уровням после добавления элемента (9:5). 42:-13 38:41 26:-2 -4:28 -5:31 -15:33 -17:-4 -20:2 -48:8	1. ((30:40)((-44:18)((-48:5))((14:5)((-38:5)#((-35:-3)#((6:-10)((2:-20))((10:-50))))))) ((31:13)#((38:-22)((33:-23))((44:-48))))). 2. (38:41)(-15:33)(42:-13)(-48:8)(-5:31)(-20:2)(-4:28)(-17:-4)(9:5). 3. (-9:-43)(-8:17)(-7:-49)(9:-35)(22:1)(27:1)(31:-26)(40:30)(42:-2).

	3. Перечислите узлы данной дерамиды в обратном порядке. 42:-2 40:30 31:-26 27:1 22:1 9:-35 -7:-49 -8:17 -9:-43	
2	1. Запишите скобочное представление дерамиды, представленной ниже. 34:9 27:-15 11:-29 7:-9 1:36 -7:-11 -15:12 -26:-36 -37:-4 -42:-35 2. Перечислите узлы следующей дерамиды по уровням после удаления элемента (32:-4). 32:-4 30:-45 26:-49 22:-39 17:-7 13:26	1. ((1:36)((-15:12)((-37:-4)((-42:-35))((-26:-36)))((-7:-11)))((34:9)((7:-9)#((27:-15)((11:-29)))))). 2. (-29:31)(13:26)(-1:3)(17:-7)(7:-9)(22:-39)(8:-12)(30:-45)(11:-20)(26:-49). 3. (-50:40)(36:27)(-43:22)(-44:-41)(32:18)(-37:-4)(-31:-7)(-17:-24)(-18:-44)(8:-48).

	11:-20 8:-12 7:-9 -1:3 -29:31 3. Перечислите узлы данной дерамиды в прямом порядке. 36:27 32:18 8:-48 -17:-24 -18:-44 -31:-7 -37:-4 -43:22 -44:-41 -50:40	
3	1. Запишите скобочное представление дерамиды, представленной ниже. 36:-24 32:-1 29:10 9:27 6:-39 -16:5 -17:34 -20:-32 -24:-34 -45:-41 2. Перечислите узлы следующей дерамиды по уровням после добавления элемента (-39:14). 45:-29	1. ((-17:34)((-20:-32)((-24:-34)((-45:-41))))((9:27)((-16:5)#((6:-39)))((29:10)#((32:-1)#((36:-24)))))). 2. (11:25)(-40:21)(15:9)(-50:-3)(-39:14)(37:-21)(4:-20)(45:-29)(-1:-28)(9:-27). 3. (49:45)(7:30)(-12:23)(-13:21)(-49:10)(-10:-9)(-9:-32)(-7:-34)(27:-18)(11:-28).

	37:-21 15:9 11:25 9:-27 4:-20 -1:-28 -21:-41 -40:21 -50:-3 3. Перечислите узлы данной дерамиды в прямом порядке. 49:45 27:-18 11:-28 7:30 -7:-34 -9:-32 -10:-9 -12:23 -13:21 -49:10	
4	1. Запишите скобочное представление дерамиды, представленной ниже. 48:8 42:-20 18:-22 1:34 -3:-26 -12:-48 -13:34 -15:-30 -27:-37 -39:36 -44:-9 2. Перечислите узлы	1. ((-39:36)((-44:-9))((1:34)((-13:34)((-15:-30)((-27:-37)))((-3:-26)((-12:-48))))((48:8)((42:-20)((18:-22)))))). 2. (5:40)(-11:32)(15:27)(-33:28)(41:-3)(-32:15)(32:-6)(-27:4)(24:-23). 3. (-33:-22)(-25:-15)(-11:-15)(-3:13)(-49:15)(-1:27)(11:-42)(29:-41)(42:-8)(43:34)(5:44)(44:45).

	следующей дерамиды по уровням после удаления элемента (-42:-23). 41:-3 32:-6 24:-23 15:27 5:40 -11:32 -27:4 -32:15 -33:28 -42:-23 3. Перечислите узлы данной дерамиды в концевом порядке. 44:45 43:34 42:-8 29:-41 11:-42 5:44 -1:27 -3:13 -11:-15 -25:-15 -33:-22 -49:15	
5	1. Запишите скобочное представление дерамиды, представленной ниже. 47:-37 42:-23 34:-10 32:-7 17:-28	1. ((9:35)((-35:33)#((-12:32)((-23:-16))((-3:-12)((-5:-19)((-9:-24))))))((32:-7)((17:-28))((34:-10)#((42:-23)#((47:-37)))))). 2. (28:37)(-38:22)(40:32)(-50:17)(10:19)(45:1)(0:3)(26:-3)(-22:-16)(1:-3)(5:-10).

	9:35 -3:-12 -5:-19 -9:-24 -12:32 -23:-16 -35:33 2. Перечислите узлы следующей дерамиды по уровням после удаления элемента (42:46). 45:1 42:46 40:32 28:37 26:-3 10:19 5:-10 1:-3 0:3 -22:-16 -38:22 -50:17 3. Перечислите узлы данной дерамиды в прямом порядке. 48:-33 39:3 35:2 32:7 29:-44 9:13 0:-4 -11:-33 -19:35 -39:-13 -48:12	3. (-19:35)(-48:12)(-39:-13)(9:13)(0:-4)(-11:-33)(32:7)(29:-44)(39:3)(35:2)(48:-33).

6	1. Запишите скобочное представление дерамиды, представленной ниже. 47:34 45:-16 32:45 26:-14 8:-42 4:-32 3:16 1:-6 -23:-42 -26:-40 2. Перечислите узлы следующей дерамиды по уровням после добавления элемента (33:24). 45:18 20:-32 16:40 9:18 -3:-44 -4:-38 -5:18 -17:26 -32:-35 3. Перечислите узлы данной дерамиды в обратном порядке. 48:-1 44:41 39:27 33:-3 29:-5 21:18	1. ((32:45)((3:16)((1:-6)((-26:-40)#((-23:-42))))((26:-14)((4:-32)#((8:-42)))))((47:34)((45:-16)))). 2. (16:40)(-17:26)(33:24)(-32:-35)(9:18)(20:-32)(45:18)(-5:18)(-4:-38)(-3:-44). 3. (-48:27)(-40:-16)(-33:-20)(-11:-39)(7:-2)(14:-7)(21:18)(29:-5)(33:-3)(39:27)(44:41)(48:-1).

	14:-7 7:-2 -11:-39 -33:-20 -40:-16 -48:27	

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была создана программа, позволяющая генерировать несколько вариантов наборов заданий для проведения текущего контроля среди студентов. Некоторые из заданий имеют несколько вариаций, реализован ключевой тип заданий в данной работе — задания на вставку/удаление элементов. Тексты заданий и ответы к ним выводятся в отдельные файлы в удобном для предоставления решающим / проверки преподавателем виде.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. MAXimal. URL: https://e-maxx.ru/algo/treap
2. Вики-конспекты ИТМО. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
3. Алгоритмы и структуры данных. URL: https://acm.khpnets.info/w/index.php?title=%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
4. Алгоритмика. URL: https://algorithmica.org/ru/treap
5. Хабр. URL: https://habr.com/ru/post/101818/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: CourseWork2020.cpp

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

using namespace std;

//Пара значений (необходимо для разных типов данных)
template <typename T> struct Pair {
	T x;
	T y;

	Pair(T x, T y) : x(x), y(y) {}
};

//Узел - элемент дерева
struct Node {
	Pair<int> p;
	Node* l, * r;

	Node(int x, int y) : p(Pair<int>(x, y)), l(nullptr), r(nullptr) {}
	Node(int x, int y, Node* l, Node* r) : p(Pair<int>(x, y)), l(l), r(r) {}
};

//Отступ (нужен для рекурсивного вывода)
void indent(int n, ostream& write = cout) {
	for (int i = 0; i < n; i++) {
		write << "\t";
	}
}

//Освобождение памяти (удаление дерева)
void dispose(Node* t) {
	if (t == nullptr) {
		return;
	}
	dispose(t->l);
	dispose(t->r);
	delete t;
}

//Проверка на вхождение элемента с заданным ключом в дерево
bool isIn(Node* t, int k) {
	if (t != nullptr) {
		if (k > t->p.x) {
			return isIn(t->r, k);
		}
		else if (k < t->p.x) {
			return isIn(t->l, k);
		}
		return true;
	}
	return false;
}

//Расчёт высоты дерамиды
```

```
int height(Node* t) {
	int rHeight = 0, lHeight = 0;
	if (t->l != nullptr) {
		lHeight = height(t->l);
	}
	if (t->r != nullptr) {
		rHeight = height(t->r);
	}
	if (lHeight > rHeight) return lHeight + 1;
	return rHeight + 1;
}

//Слияние двух деревьев
Node* merge(Node* l, Node* r, int& depth) {
	if (!depth) {
		cout << "\nAttempting to merge trees\n";
	}
	if (l == nullptr) {			//Если одно из деревьев - nullptr, возвращаем другое (тоже может быть nullptr)
		return r;
	}
	if (r == nullptr) {
		return l;
	}

	if (l->p.y > r->p.y) {
		//Если у левого дерева приоритет больше, возвращаем его корень,
		//предварительно рекурсивно вызвав функцию слияния для его правого поддерева,
		//сохраняя неизменным исходное правое дерево.
		indent(depth);
		cout << "Merging with the right branch of  " << l->p.x << "  " << l->p.y << "\n";
		depth++;
		l->r = merge(l->r, r, depth);
		depth--;
		return l;
	}
	//Иначе возвращем корень правого дерева, перед этим
	//выполнив аналогичные действия с его левым поддеревом и
	//сохраняя исходное левое дерево
	indent(depth);
	cout << "Merging with the left branch of  " << r->p.x << "  " << r->p.y << "\n";
	depth++;
	r->l = merge(l, r->l, depth);
	depth--;
	return r;
}

//Разбиение дерева на два поддерева
Pair<Node*> split(Node* t, int k, int& depth) {
	if (!depth) {
		cout << "\nAttempting to split the tree.\n";
	}
	if (t == nullptr) {
		return Pair<Node*>(nullptr, nullptr);
	}
	else if (k > t->p.x) {
		//Если заданный ключ больше ключа текущего элемента, спускаемся по правой ветви
		//и возвращаем пару значений - исходный элемент и правое поддерево результата разбиения
```

```
            indent(depth);
            cout << "Descending to  the right branch of  " << t->p.x << "  " <<
t->p.y << "\n";
            depth++;
            Pair<Node*> nPair = split(t->r, k, depth);
            depth++;
            t->r = nPair.x;
            return Pair<Node*>(t, nPair.y);
      }
      else {
            //Иначе спускаемся по левой ветви и возвращаем левое поддерево
результата разбиения
            //и исходный элемент
            indent(depth);
            cout << "Descending to  the left branch of  " << t->p.x << "  " <<
t->p.y << "\n";
            depth++;
            Pair<Node*> nPair = split(t->l, k, depth);
            depth--;
            t->l = nPair.y;
            return Pair<Node*>(nPair.x, t);
      }
      return Pair<Node*>(t, nullptr);
}

//Удаление элемента из дерева
Node* remove(Node* t, int k, int& depth) {
      if (!depth) {
            cout << "\nAttempting to remove existing element.\n";
      }
      if (t != nullptr) {
            indent(depth);
            cout << "Current element: " << t->p.x << "  " << t->p.y << "\n";
            if (t->p.x < k) {
                  //Если заданный ключ больше, спускаемся к правому поддереву
                  depth++;
                  t->r = remove(t->r, k, depth);
                  depth--;
                  return t;
            }
            else if (t->p.x > k) {
                  //Если меньше - к левому поддереву
                  depth++;
                  t->l = remove(t->l, k, depth);
                  depth--;
                  return t;
            }
            else {
                  //Если нашёлся данный ключ, удаляем его, сливая затем его
поддеревья
                  indent(depth);
                  cout << "\nRequired element found: " << t->p.x << "  " << t-
>p.y << " , removing...\n";
                  Node* l = t->l, * r = t->r;
                  t->l = nullptr;
                  t->r = nullptr;
                  dispose(t);
                  return merge(l, r, depth);
            }
      }
      //Если t = nullptr
      return t;
}
```

```
//Добавление заданного элемента
Node* add(Node* t, int k, int v) {
	int depth = 0;
	if (isIn(t, k)) {
		cout << "\n\nElement with key = " << k << " already exists.
Replacing...";
		t = remove(t, k, depth);
	}
	cout << "\n\nSplitting the treap by key " << k << "\n";
	Pair<Node*> p = split(t, k, depth);	//Разбиение по заданному ключу
	Node* n = new Node(k, v);
	return merge(merge(p.x, n, depth), p.y, depth); //Слияние левого поддерева
результата разбиения
                                                                                //
с новым элементом, а затем с правым поддеревом
}

//Вывод дерева в файл
void printTreap(const Node* t, int& depth, ostream& write) {
	if (t != nullptr) {
		if (t->r != nullptr) {
			//П
			depth++;
			printTreap(t->r, depth, write);
			depth--;
		}
		//К
		indent(depth, write);
		write << t->p.x << ":" << t->p.y << "\n";
		if (t->l != nullptr) {
			//Л
			depth++;
			printTreap(t->l, depth, write);
			depth--;
		}
	}
}

//Переопределение оператора вывода в поток для дерамиды
ostream& operator<<(ostream& out, const Node& t) {
	int depth = 0;
	printTreap(&t, depth, out);
	return out;
}


//Получение случайного элемента из кучи
Node* getRandom(Node* t, int& depth, int h) {
	bool dir = rand() % 2;		//Выбор направления

	//Если спуск по левой ветви и не достигнут указанный уровень
	if (depth != h && dir && t->l != nullptr) return getRandom(t->l, depth,
h);

	//Если спуск по правой ветви и не достигнут указанный уровень
	else if (depth != h && !dir && t->r != nullptr) return getRandom(t->r,
depth, h);

	//Если был достигнут указанный уровень или элемент является концевым
	else {
		cout << "RANDOMLY GOT ELEMENT : (" << t->p.x << ":" << t->p.y << ")\
n\n";
```

```
            return new Node(t->p.x, t->p.y);
      }
}

//Генерация случайной дерамиды
Node* generateTreap() {
      Node* treap = nullptr;
      cout << "GENERATING RANDOM TREAP\n\n";
      for (int i = 0; i < 10 + rand()%5; i++) {
            treap = add(treap, rand() % 100 - 50, rand() % 100 - 50);
      }
      return treap;
}


//Запись ответа в скобочном представлении
void writeBracketsAnswer(Node* t, ostream& answer) {
      if (t != nullptr) {
            cout << "WRITING ELEMENT (" << t->p.x << ":" << t->p.y << ") TO A
FILE\n\n";
            answer << "((" << t->p.x << ":" << t->p.y << ")";

            //Проверка на то, какие есть дочерние объекты - нужно для
сокращённого представления
            if (t->l != nullptr) {
                  writeBracketsAnswer(t->l, answer);
                  writeBracketsAnswer(t->r, answer);
            }
            else if (t->r != nullptr) {
                  answer << "#";          //Пустое поддерево
                  writeBracketsAnswer(t->r, answer);
            }
            answer << ")";
      }
}

//Запись ответа по уровням слева направо
void writeLevelsAnswer(Node* t, ostream& answer, int& depth, int level) {
      if (t != nullptr) {
            //Только на заданном уровне
            if (depth == level) {
                  cout << "WRITING ELEMENT (" << t->p.x << ":" << t->p.y << ")
TO A FILE\n\n";
                  answer << "(" << t->p.x << ":" << t->p.y << ")";
            }
            //Иначе спуск, пока возможно
            else {
                  depth++;
                  writeLevelsAnswer(t->l, answer, depth, level);
                  writeLevelsAnswer(t->r, answer, depth, level);
                  depth--;
            }
      }
}

//Запись узлов в прямом (КЛП) порядке
void writeKLPAnswer(Node* t, ostream& answer) {
      if (t != nullptr) {
            cout << "WRITING ELEMENT (" << t->p.x << ":" << t->p.y << ") TO A
FILE\n\n";
            answer << "(" << t->p.x << ":" << t->p.y << ")";
            writeKLPAnswer(t->l, answer);
            writeKLPAnswer(t->r, answer);
```

```
	}
}

//Запись узлов в обратном (ЛКП) порядке
void writeLKPAnswer(Node* t, ostream& answer) {
	if (t != nullptr) {
		writeLKPAnswer(t->l, answer);
		cout << "WRITING ELEMENT (" << t->p.x << ":" << t->p.y << ") TO A
FILE\n\n";
		answer << "(" << t->p.x << ":" << t->p.y << ")";
		writeLKPAnswer(t->r, answer);
	}
}

//Запись узлов в концевом (ЛПК) порядке
void writeLPKAnswer(Node* t, ostream& answer) {
	if (t != nullptr) {
		writeLPKAnswer(t->l, answer);
		writeLPKAnswer(t->r, answer);
		cout << "WRITING ELEMENT (" << t->p.x << ":" << t->p.y << ") TO A
FILE\n\n";
		answer << "(" << t->p.x << ":" << t->p.y << ")";
	}
}

//Генерация задания на скобочное представление
void generateBrackets(ostream& task, ostream& answer) {
	ifstream read;
	string curString;
	Node* treap = generateTreap();

	//Получение шаблона оформления
	read.open("Templates/brackets.txt");
	getline(read, curString);
	//Запись формулировки задания и дерамиды
	cout << "WRITING TASK TO A FILE\n\n";
	task << "1. " << curString << "\n\n" << *treap << "\n";

	cout << "WRITING ANSWER TO A FILE\n\n";
	answer << "1. ";
	//Запись ответа
	writeBracketsAnswer(treap, answer);
	answer << ".\n\n";

	read.close();

	dispose(treap);
}

//Генерация задания на перечисление по уровням слева направо
void generateLevels(ostream& task, ostream& answer) {
	ifstream read;
	string curString;
	Node* treap = generateTreap();
	Node* unit;
	int depth = 0;
	int h = height(treap);


	//Получение шаблона оформления
	read.open("Templates/levels.txt");
	getline(read, curString);
	//Запись формулировки задания
```

```
	cout << "WRITING TASK TO A FILE\n\n";
	task << "2. " << curString;

	//Случайный выбор - добавление либо удаление элемента
	bool addrem = rand() % 2;
	if (addrem) {
		//Создание нового элемента
		unit = new Node(rand() % 100 - 50, rand() % 100 - 50);
		getline(read, curString);
		//Запись добавления
		task << curString;
		getline(read, curString);
	}
	else {
		//Получение элемента для удаления
		unit = getRandom(treap, depth, rand()%h);
		getline(read, curString);
		getline(read, curString);
		//Запись удаления
		task << curString;
	}

	//Запись выбранного элемента и дерамиды
	getline(read, curString);
	task << curString << "(" << unit->p.x << ":" << unit->p.y << ")" << ".\n\
n" << *treap << "\n";

	//Добавление/удаление элемента непосредственно
	if (addrem) {
		treap = add(treap, unit->p.x, unit->p.y);
	}
	else {
		treap = remove(treap, unit->p.x, depth);
	}

	//Запись ответа
	cout << "WRITING ANSWER TO A FILE\n\n";
	answer << "2. ";
	for (int i = 0; i < h; i++) {
		writeLevelsAnswer(treap, answer, depth, i);
	}
	answer << ".\n\n";

	read.close();
	dispose(unit);
	dispose(treap);
}

//Генерация задания на перечисление узлов в указанном порядке
void generateTraversal(ostream& task, ostream& answer, int cond) {
	ifstream read;
	string curString;
	Node* treap = generateTreap();

	//Получение шаблона оформления
	read.open("Templates/traversal.txt");
	getline(read, curString);
	//Запись формулировки задания
	cout << "WRITING TASK TO A FILE\n\n";
	task << "3. " << curString;

	//Запись подходящего порядка
	switch (cond) {
```

```
	case 0:				//Прямой
		getline(read, curString);
		task << curString;
		getline(read, curString);
		getline(read, curString);
		break;
	case 1:				//Обратный
		getline(read, curString);
		getline(read, curString);
		task << curString;
		getline(read, curString);
		break;
	case 2:				//Концевой
		getline(read, curString);
		getline(read, curString);
		getline(read, curString);
		task << curString;
		break;
	}
	getline(read, curString);
	task << curString;

	//Запись дерамиды
	task << "\n\n" << *treap << "\n";

	//Запись соответствующего ответа
	cout << "WRITING ANSWER TO A FILE\n\n";
	answer << "3. ";
	switch (cond) {
	case 0:
		writeKLPAnswer(treap, answer);
		break;
	case 1:
		writeLKPAnswer(treap, answer);
		break;
	case 2:
		writeLPKAnswer(treap, answer);
		break;
	}
	answer << ".\n\n";

	read.close();
	dispose(treap);
}

//Создание наборов заданий
void generateTasks(int n) {
	ofstream task, answer;
	string curVar, curAns;

	//Общий формат названий файлов
	const string var = "Tasks/variant", ans = "Answers/answer";

	cout << "\n\nGENERATION STARTED\n\n";
	char buff[3];
	for (int i = 0; i < n; i++) {
		cout << "\n\nGENERATING VARIANT #" << i + 1 << "\n\n";
		//Добавление к исходным строкам номера варианта
		_itoa_s(i + 1, buff, 10);
		curVar = var + buff + ".txt";
		curAns = ans + buff + ".txt";

		task.open(curVar);
```

```
            answer.open(curAns);

            //Запись заданий и ответов
            cout << "\nGENERATING \"BRACKETS\" TYPE TASK\n\n";
            generateBrackets(task, answer);
            cout << "\nGENERATING \"LEVELS\" TYPE TASK\n\n";
            generateLevels(task, answer);
            cout << "\nGENERATING \"TRAVERSAL\" TYPE TASK\n\n";
            generateTraversal(task, answer, rand()%3);

            task.close();
            answer.close();
      }
      cout << "\n\n\nGENERATION COMPLETED\n";
}

int main() {
      srand(time(nullptr));
      setlocale(LC_ALL, "rus");

      //Вывод инструкции на экран
      string s;
      ifstream instruction;
      instruction.open("Templates/Instruction.txt");
      while (!instruction.eof()) {
            getline(instruction, s);
            cout << s << "\n";
      }

      int nStudents = 0;

      cout << "Введите количество вариантов теста, которое необходимо
сгенерировать (не более 30!): \n\n";
      cin >> nStudents;
      while (nStudents <=0 || nStudents > 30) {
            cout << "Недопустимое количество, повторите ввод: ";
            cin >> nStudents;
      }

      generateTasks(nStudents);
      system("pause");
      return 0;
}
```