

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Рандомизированные деревья поиска**

Студент гр. 9303

\_\_\_\_\_

Максимов Е.А.

Преподаватель

\_\_\_\_\_

Филатов Ар.Ю.

Санкт-Петербург

2020

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент: Максимов Е.А.

Группа: 9303

Тема работы: Рандомизированные дерамиды поиска – вставка и исключение (демонстрация)

Исходные данные:

произвольный набор данных, заданный пользователем

Содержание пояснительной записки:

«Содержание», «Введение», «Основные теоретические положения», «Описание кода программы», «Заключение».

Предполагаемый объем пояснительной записки:

Не менее 10 страниц

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 25.12.2020

Дата защиты реферата: 25.12.2020

Студент

\_\_\_\_\_

Максимов Е.А.

Преподаватель

\_\_\_\_\_

Филатов Ар.Ю.

## АННОТАЦИЯ

Курсовая работа представляет собой консольное приложение, предназначенное для визуализации (демонстрации) устройства дерамиды. Исходный код программы был написан на языке программирования C++. Запуск программы осуществляется на операционной системе Windows.

В процессе разработки были использованы стандартные библиотеки языка программирования C++. Для получения данных в виде изображений формата PNG были использованы библиотеки Graphviz.

Полученная программа создаёт изображения, содержащие пошаговое построение дерамиды. При запуске программа информирует пользователя о необходимых объектах, в которых нуждается программа, информацию о рассматриваемом типе данных и замечания про некоторые неочевидные свойства логики написанной программы.

Для проверки работоспособности программы было проведено тестирование. Исходный код, результаты тестирования и скриншоты, подтверждающие корректную работу приложения, представлены в приложении.

## СОДЕРЖАНИЕ

Введение	5
1. Основные теоретические положения	6
1.1. Общие сведения	6
1.2. Свойства реализации	6
2. Описание кода программы	8
2.1. Класс узла дерамиды TreapNode	8
2.2. Класс дерамиды Treap	8
2.3. Функции программы	10
2.4. Функция main	11
3. Описание интерфейса пользователя	12
Заключение	13
Приложение А. Исходный код программы	14
Приложение Б. Результаты тестирования программы	21

## **ВВЕДЕНИЕ**

Цель работы – разработка приложения, представляющее собой наглядное пособие при ознакомлении со структурой данных «декартово дерево», которая ранее не известна пользователю.

Для достижения поставленной цели требуется реализовать задачи, перечисленные ниже.

1. Изучение теоретического материала по необходимой теме.
2. Реализация программного кода для обработки данных декартова дерева.
3. Реализация способов взаимодействия пользователя и приложения.
4. Тестирование программного кода.

# 1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

## 1.1. Общие сведения.

Декартово дерево (или дерамида) – это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу. Декартово дерево состоит из узлов, которые содержат 2 значения, обычно называемые «ключ» ( $X$ ) и «приоритет» ( $Y$ ).

Если рассматривать ключи узлов дерамиды ( $X$ ) и исключить приоритеты, то полученная структура данных будет представлять собой бинарное дерево поиска. Если рассматривать приоритеты узлов дерамиды ( $Y$ ), то полученная структура данных будет представлять собой структуру данных типа кучи.

Дерамида обладает всеми свойствами бинарного дерева поиска и кучи. Также следует отметить, что множество узлов определяют дерамиду однозначно.

## 1.2. Свойства реализации.

В силу однозначности построения, удаление или добавление лишь одного элемента может сильно изменить структуру дерамиды. Для работы с дерамидами необходимо реализовать две вспомогательные функции, которые будут использоваться в подавляющем большинстве прочих реализуемых функций.

1. Функция *Merge* принимает на вход два декартовых дерева, ключи которых образуют изолированные множества. Рекурсивная функция на каждом шаге рассматривает верхние элементы дерамиды, сравнивает их и устанавливает их в нужное место дерамиды.

2. Функция *Split* обратна функции *Merge*. Функция принимает на вход одно декартово дерево и ключ  $x$  – значение, по которому требуется разделить декартово дерево на две части. В результате работы рекурсивной функции получается два декартовых дерева, в одном из которых находятся все элементы исходного дерева с ключами, меньшими, чем ключ  $x$ , а в другом – с большими.

Благодаря двум функциям возможно реализовать остальные базовые функции работы со структурами данных.

3. Функция *Add* добавляет узел к дерамиде посредством разделения дерамиды, а затем последовательному слиянию.

4. Функция *Remove* удаляет узел дерамиды посредством разделения дерамиды на 3 части, одна из которых содержит единственный узел и удаляется, а затем слиянию двух других.

На основе уже последних двух функций представляется возможность реализации других, более сложных функций для работы с дерамидой.

## 2. ОПИСАНИЕ КОДА ПРОГРАММЫ

### 2.1. Класс узла дерамиды *TreapNode*.

Класс *TreapNode* представляет собой узел дерамиды.

Свойства класса:

- *int key* – данные узла;
- *int priority* – приоритет узла;
- *int count* – количество экземпляров с одинаковыми данными;
- *TreapNode\* left, right* – указатели на соседние узлы.

Методы класса:

1. Конструктор класса принимает на вход целочисленную переменную и создаёт узел со значением количество *count* = 1, указателями на соседние узлы *nullptr* и случайным приоритетом с помощью функции *rand()*.

### 2.2. Класс дерамиды *Treap*.

Класс *Treap* содержит всю структуру дерамиды и методы для работы с ней.

Свойства класса:

- *TreapNode\* root* – указатель на корневой узел дерамиды.

Методы класса:

1. Класс имеет два конструктора.
  - 1.1. Конструктор принимает на вход указатель на корень дерамиды *TreapNode\* root*. Конструктор устанавливает соответствующее поле на это значение.
  2. Конструктор принимает на вход вектор целочисленных переменных *vector<int> keys*. Для каждого элемента вызывается метод *add* (см. ниже).
  3. Деструктор класса вызывает рекурсивную функцию *destruct(root)* для освобождения динамической памяти (см. ниже).
  4. *void print()* вызывает рекурсивную функцию *printNode()* для печати данных дерамиды (см. ниже).



5. *TreapNode\* find(int key, TreapNode\* root)* принимает на вход данные, которые необходимо найти в дерамиде и указатель на корень структуры дерамиды. Метод возвращает *nullptr*, если узла с такими данными нет, или указатель на узел с требуемыми данными.

6. *void add(int key)* добавляет данные в структуру дерамиды. Приоритет узла выбирается случайным образом. Если узел с такими данными уже есть, то его поле *count* увеличивается на 1. Метод не имеет возвращаемого значения.

7. *void remove (int key)* удаляет узел с данными, которые указаны в методе. Если узла с такими данными нет, то структура не изменяется. Метод не имеет возвращаемого значения.

8. Приватный метод *void destruct (TreapNode\* node)* рекурсивно удаляет выделенную память на узлы дерамиды. Метод не имеет возвращаемого значения.

9. Приватный метод *void merge(Treap\* another)* объединяет вместе две дерамиды при условии, что границы множества узлов дерамид не пересекаются. Метод вызывает рекурсивный метод *mergeTreapNodes* (см. ниже). После объединения другая структура дерамиды удаляется.

10. Приватный метод *Treap\* split (int delimiter)* принимает на вход значение, по которому нужно разделить дерамиду на две независимые дерамиды. Метод вызывает рекурсивный метод *splitTreapNode* (см. ниже). Метод возвращает указатель на новую дерамиду.

11. Приватный метод *Pair splitTreapNode (TreapNode\* node, int delimiter)* принимает на вход указатель на узел дерамиды и целочисленную переменную, относительно которой необходимо разделить дерамиду. Метод возвращает пару узлов, которые являются корнями дерамиды, представляющую собой тип данных *Pair*.

12. Приватный метод *TreapNode\* mergeTreapNodes (TreapNode\* L, TreapNode\* R)* принимает на вход указатели на узлы, которые необходимо объединить. Функция рекурсивно объединяет узлы в одну дерамиду согласно

их данным и приоритетам. Функция возвращает указатель на корень новой дерамиды.

13. *void printNode(TreapNode\* node, int flag = 1)* принимает на вход указатель на узел. Функция рекурсивно печатает данные узлов в виде таблицы, содержащей значения данных в корневом узле, направление (слева или справа от текущего узла), значение данных в текущем узле, приоритет и количество экземпляров с таким типом данных.

14. Приватный метод *void toDotTreapNode(TreapNode\* node, ofstream& output, int& i)* принимает на вход указатель на узел дерамиды, поток вывода и счётчик. Метод записывает данные во временный файл *dotCode.txt* в виде кода на языке DOT, который может распознать библиотека Graphviz для дальнейшей визуализации декартова дерева.

15. Метод *void getImage(string imageName, int key, int flagAddRem)* принимает на вход строку с названием файла, ключ и флаг о способе взаимодействия с дерамидой. Метод взаимодействует с библиотекой Graphviz для создания файла формата PNG. Метод не имеет возвращаемого значения.

### 2.3. Функции программы.

В программе реализованы функции, которые не относятся ни к одному из классов.

1. Функция *vector<int> readVector()* считывает данные из файла «input.txt» и создает на основе этих данных вектор целых чисел, который будет обработан конструктором класса *Treap*.

2. Функция *bool isNumber(char c)* возвращает *true*, если символ *c* является цифрой, и *false* в противном случае.

3. Функция *clearImagesFolder* удаляет все изображения из соответствующей папки. Функция имеет параметр по умолчанию  $i = 0$ , который отвечает за номер изображения, удаляемый из папки. Функция не имеет возвращаемого значения.

## 2.4. Функция **main**.

Функция *int main()* вызывает необходимые функции для обработки данных из файла и обрабатывает данные пользователя как из консоли, так из определённого файла. Функция управляет процессом создания и редактирования пирамиды, созданием и удалением изображений и временных файлов.

Исходный код программы см. в приложении А.

Результаты тестирования программы см. в приложении Б.

### 3. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

При запуске приложения пользователь видит консоль, в которой указана следующая важная информация.

1. Необходимые условия для корректной работы программы.
2. Краткая информация о структуре данных «Декартово дерево».
3. Некоторые особенности реализации программы.

Пользователю предлагается создать текстовый файл определённого названия, который будет содержать элементы декартова дерева (однако пользователь может и не делать этого).

Далее пользователю предлагается поочерёдно вводить дополнительные данные для расположения их в декартовом дереве. Если пользователь не желает вводить данные, он может ввести пустую строку, чтобы закончить.

Далее пользователю предлагается поочерёдно вводить данные ключей для удаления их из декартова дерева. Если пользователь не желает вводить данные, он может ввести пустую строку, чтобы закончить.

После этого программа указывает пользователю директорию сохранения данных.

Программа на каждом шаге работы создаёт изображение формата PNG, содержащее наглядное представление декартова дерева с необходимой информацией об изменениях на каждом шаге.

Примеры работы программы см. в приложении В.

## ЗАКЛЮЧЕНИЕ

Для успешного достижения поставленной цели были реализованы следующие задачи.

1. Был изучен теоретический материал по необходимой теме декартова дерева.
  2. Был реализован программный код для обработки входных данных для построения декартова дерева.
  3. Был реализован интерфейс (консоль) для взаимодействия с пользователем.
  4. Был протестирован программный код на корректность выполнения задачи.
1. Результатом работы стало консольное приложение, представляющее собой наглядное пособие для ознакомления со структурой данных «декартово дерево», которая ранее не известна пользователю, которую можно использовать для обучения.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp.

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <vector>
#include <string>
#include <cstring>
#include <cstdio>

using namespace std;

class Treap;
class TreapNode;
typedef pair<TreapNode*, TreapNode*> Pair;      // Pair(this, another)
bool isNumber(char c);
vector<int> readVector();

class TreapNode
{
public:
    int key, priority, count;
    TreapNode* left;
    TreapNode* right;

    TreapNode(int key)
    {
        this->key = key;
        this->priority = rand();
        left = nullptr;
        right = nullptr;
        count = 1;
    }

    ~TreapNode() {}
};

class Treap
{
public:
    TreapNode* root = nullptr;

    Treap(TreapNode* root = nullptr) { this->root = root; }
    Treap(vector<int> keys)
    {
        for (int i=0; i < keys.size(); i++)
```

```

    {
        add(keys[i]);
        getImage(string("treap_") + to_string(i) + string(".png"), keys[
i], 0);
    }
}

~Treap() { destruct(root); }
void print() { printNode(this->root); }

TreapNode* find(int key, TreapNode* root)
{
    if (root == nullptr)
        return nullptr;
    else if (key < root->key)
        return find(key, root->left);
    else if (key > root->key)
        return find(key, root->right);
    else
        return root;
}

void add(int key)
{
    TreapNode* findNode = this->find(key, this->root);
    if (findNode != nullptr)
    {
        findNode->count++;
        return;
    }
    Treap* another = this->split(key);
    Treap* aloneNode = new Treap((new TreapNode(key)));
    this->merge(aloneNode);
    this->merge(another);
    return;
}

void remove(int key)
{
    TreapNode* findNode = this->find(key, this->root);
    if (findNode == nullptr)
        return;
    Treap* right = this->split(key);
    Treap* middle = this->split(key-1);
    delete middle;
    merge(right);
    return;
}

```

```

void getImage(string imageName, int key, int flagAddRem)
{
    ofstream output("dotCode.txt");
    if (!output.is_open())
    {
        cout << "Can't open \"dotCode.txt\" data file. Stopping...\n";
        exit(0);
    }
    int i = 0;
    output << "digraph Tree{\n";
    toDotTreapNode(this->root, output, i);

    if(flagAddRem == 0)
        output << "label=\"Added key: " << key << "\";\n";
    else
        output << "label=\"Removed key: " << key << "\";\n";
    output << "labelloc=top;\n";

    output << "}";
    output.close();
    string sysCommand = ".\\graphviz\\dot.exe -Tpng dotCode.txt -
o.\\images\\" + imageName;
    system(sysCommand.c_str());
}

private:
void destruct(TreapNode* node)
{
    if (node == nullptr)
        return;
    if (node->left != nullptr)
        destruct(node->left);
    if (node->right != nullptr)
        destruct(node->right);
    delete node;
    return;
}

void merge(Treap* another)
{
    this->root = mergeTreapNodes(this->root, another->root);
    another->root = nullptr;
    delete another;
    return;
}

Treap* split(int delimiter)
{
    Pair splitResult = splitTreapNode((this->root), delimiter);
    this->root = splitResult.first;
}

```



```

    Treap* out = new Treap(splitResult.second);
    return out;
}

Pair splitTreapNode(TreapNode* node, int delimiter)
{
    if (node == nullptr)
        return {nullptr, nullptr};

    if (node->key <= delimiter)
    {
        Pair temp = splitTreapNode(node->right, delimiter);
        node->right = temp.first;          // Первый, слева
        return {node, temp.second};
    }
    else // Тогда node->key > delimiter
    {
        Pair temp = splitTreapNode(node->left, delimiter);
        node->left = temp.second;          // Второй, справа
        return {temp.first, node};
    }
}

TreapNode* mergeTreapNodes(TreapNode* L, TreapNode* R)
{
    if (L == nullptr)
        return R;
    if (R == nullptr)
        return L;
    if (L->priority > R->priority)
    {
        L->right = mergeTreapNodes(L->right, R);
        return L;
    }
    else // Тогда L->priority <= R->priority
    {
        R->left = mergeTreapNodes(L, R->left);
        return R;
    }
}

void printNode(TreapNode* node, int flag = 1)
{
    if (node == nullptr)
        return;
    if (flag)
        cout << "\t\t";
}

```

```

    cout << node->key << "\t" << node->priority << "\t\t" << node-
>count << "\n";
    if (node->left != nullptr)
    {
        cout << node->key << ":\tleft\t";
        printNode(node->left, 0);
    }
    if (node->right != nullptr)
    {
        cout << node->key << ":\tright\t";
        printNode(node->right, 0);
    }
}

void toDotTreapNode(TreapNode* node, ofstream& output, int& i)
{
    if (node->left != nullptr)
    {
        output << "\"\" << node->key << " (" << node->priority << ")\" -
> \"\" << node->left->key << " (" << node->left->priority << ")\";\n";
        toDotTreapNode(node->left, output, i);
    }
    else
    {
        output << i << " [shape=point];\n";
        output << "\"\" << node->key << " (" << node->priority << ")\" -
> " << i << ";\n";
        i++;
    }
    if (node->right != nullptr)
    {
        output << "\"\" << node->key << " (" << node->priority << ")\" -
> \"\" << node->right->key << " (" << node->right->priority << ")\";\n";
        toDotTreapNode(node->right, output, i);
    }
    else
    {
        output << i << " [shape=point];\n";
        output << "\"\" << node->key << " (" << node->priority << ")\" -
> " << i << ";\n";
        i++;
    }
}
};

vector<int> readVector()
{
    ifstream infile("data/input.txt");

```

```

vector<int> inputVector;
if (!infile.is_open())
{
    cout << "\nCan't open \"input.txt\" file. Continue...\n";
    return inputVector;
}
else
    cout << "\n\"input.txt\" file opened successfully.\n";

int input;
while(infile >> input)
    inputVector.push_back(input);
infile.close();
return inputVector;
}

bool isNumber(char c) { return ((c>=48) && (c<=57)); }

void clearImagesFolder(int i = 0)
{
    string removeImage = string(".\\images\\treap_") + to_string(i) + string(".png");
    if(remove(removeImage.c_str()) == 0)
        clearImagesFolder(++i);
}

int main()
{
    srand(time(NULL));
    system("color 0B");
    cout << "\tWelcome to the Treap Visualizer\n";
    cout << "\tMade by Maximov E. (AandSD coursework)\n\n";
    cout << "Before continue, make sure that application's folder contains graphviz folder, otherwise there will be no output images.\n";
    cout << "You can create data/input.txt file and write numbers for treap into it before start.\n";
    cout << "All images will be saved in \"images\" folder. You can open images before program ends.\n\n";
    cout << "Info: Treap - special data structure. Each node of treap has 2 values: key and priority.\n";
    cout << "Keys organized like binary search tree. Priority organized like heap data structure.\n\n";
    cout << "Repeating numbers will be drawn on image only once.\n";
    cout << "Program chooses priority at random for optimized treap creating. Priority value will be written in brackets.\n\n";
    clearImagesFolder();
    system("pause");
}

```

```

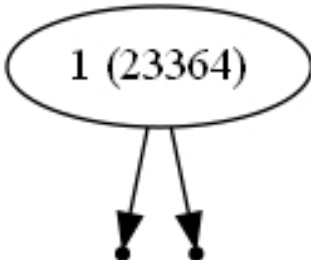
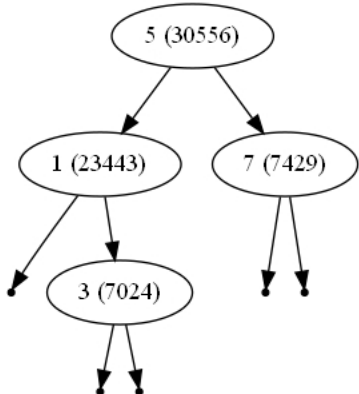
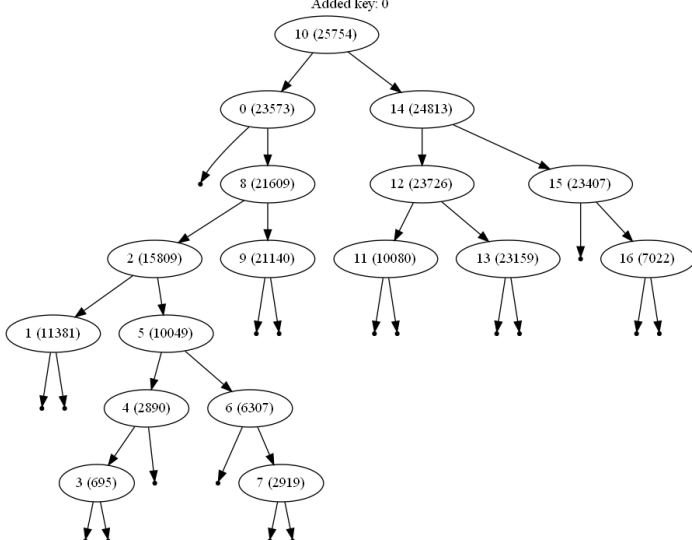
vector<int> inputVector = readVector();
Treap treap(inputVector);
string tempValueString, inputString;
int imageIndex = inputVector.size();
while(true)
{
    cout << "\nPlease, write treap element for add or empty string to continue:\t";
    getline(cin, inputString);
    if(inputString.length() == 0)
        break;
    if(!isNumber(inputString[0]))
        break;
    treap.add(stoi(inputString));

    treap.getImage(string("treap_") + to_string(imageIndex) + string(".png"), stoi(inputString), 0);
    imageIndex++;
}
while(true)
{
    cout << "\nPlease, write treap element for delete or empty string to continue:\t";
    getline(cin, inputString);
    if(inputString.length() == 0)
        break;
    if(!isNumber(inputString[0]))
        break;
    treap.remove(stoi(inputString));

    treap.getImage(string("treap_") + to_string(imageIndex) + string(".png"), stoi(inputString), 1);
    imageIndex++;
}
cout << "\nStep-by-step treap images were saved in \"images\" folder.\n\n";
remove("dotCode.txt");
system("pause");
return 0;
}

```

# **ПРИЛОЖЕНИЕ Б** **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ**

№	Входные данные	Выходные данные	Комментарий
1.	1	<p align="center">Added key: 1</p> 	Тест обработки одного элемента.
2.	1 7 1 3 7 3 5 7 5	<p align="center">Added key: 5</p> 	Тест обработки одинаковых элементов.  Представлен конечный результат работы программы.
3.	0 1 3 5 7 9 11 13 15 0 2 4 6 8 10 12 14 16 0	<p align="center">Added key: 0</p> 	Тест композиции двух тестов выше.  Представлен конечный результат работы программы.