**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 пирамиды поиска**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Головина Е.С. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Головина Е.С. | | |
| Группа 7382 | | |
| Тема работы : Рандомизированные пирамиды поиска (Treaps) - вставка и удаление. Демонстрация. | | |
| Содержание пояснительной записки:   1. Введение 2. Формальная постановка задачи 3. Описание алгоритма 4. Описание структур данных и функций 5. Тестирование 6. Демонстрация 7. Заключение | | |
| Предполагаемый объем пояснительной записки:  Не менее 25 страниц. | | |
| Дата выдачи задания: 19.10.2018 | | |
| Дата сдачи курсовой работы: | | |
| Дата защиты курсовой работы: | | |
| Студентка |  | Головина Е.С. |
| Преподаватель |  | Фирсов М.А. |

**Аннотация**

В ходе курсовой работы требуется разработать программу на языке программирования C++. Программа должна производить работу с рандомизированной пирамидой поиска все элементы которого различны, осуществлять вставку и исключение заданного элемента пирамиды. Также, должна быть реализована визуализация структур данных, алгоритмов и действий.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc532754631)

[1 ЦЕЛЬ РАБОТЫ 7](#_Toc532754632)

[2 ПОСТАНОВКА ЗАДАЧИ И СПЕЦИФИКАЦИЯ ПРОГРАММЫ 7](#_Toc532754633)

[2.1 Постановка задачи. 7](#_Toc532754634)

[2.2 Спецификация программы. 7](#_Toc532754635)

[3 ОПИСАНИЕ АЛГОРИТМА 8](#_Toc532754636)

[4 ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ И СТРУКТУР ПРОГРАММЫ 10](#_Toc532754637)

[5 ТЕСТИРОВАНИЕ 12](#_Toc532754638)

[6 ДЕМОНСТРАЦИЯ 16](#_Toc532754639)

[ЗАКЛЮЧЕНИЕ 16](#_Toc532754640)

# ВВЕДЕНИЕ

Пирамида поиска (декартово дерево) — это двоичное дерево, в узлах которого хранятся:

* ссылки на правое и левое поддерево;
* значения *k* и *p*, по которым пирамида поиска является двоичным деревом поиска по ключу *k* и двоичной кучей (пирамидой) по приоритету *p*; а именно, для любого узла дерева *n*:
  + ключи *k* узлов правого (левого) поддерева больше (меньше) либо равны ключа *k* узла *n*;
  + приоритеты *p* узлов правого и левого сына меньше либо равны приоритету *p* узла *n*.

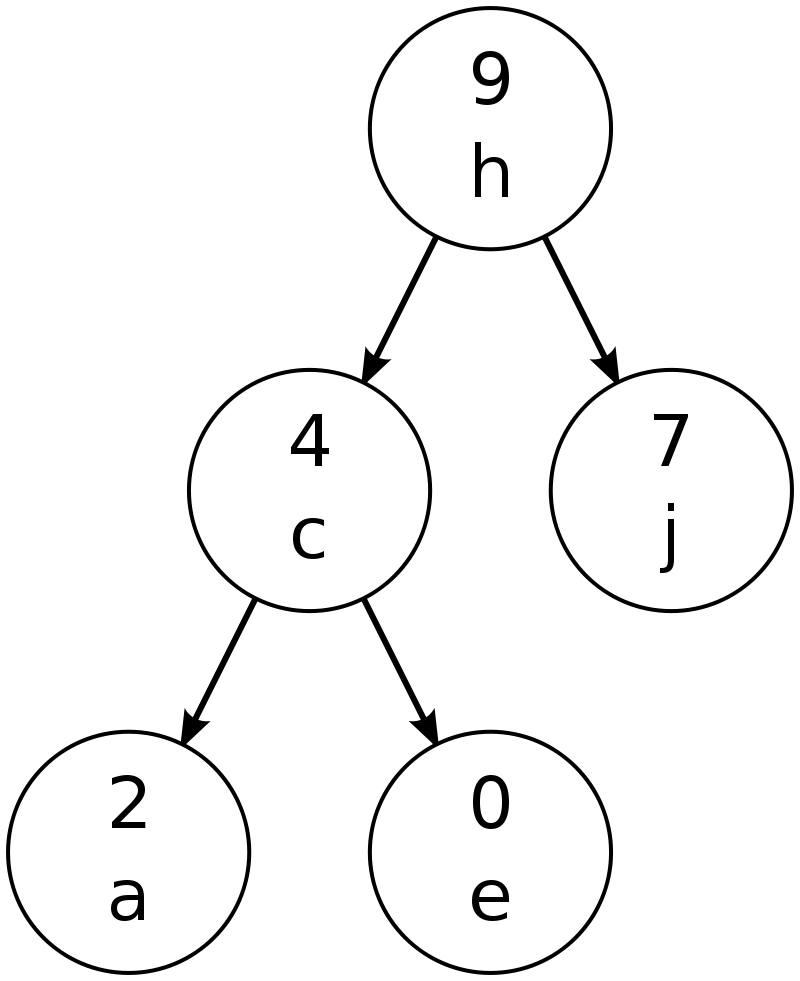


Рис.1 - Пример пирамиды поиска (ключ=символ, приоритет=целое число)

Рандомизированная пирамида не является самобалансирующейся в обычном смысле, и применяют его по следующим причинам:

1. Проще реализуется, по сравнению, например, с настоящими самобалансирующимися деревьями вроде красно-чёрного.
2. Хорошо ведёт себя «в среднем», если приоритеты *p* раздать случайно.
3. Типичная для сортирующего дерева операция «расчленить по ключу *k* на „меньше *k*0“ и „не меньше *k*0“» работает за O(*h*), где *h* – высота дерева. На красно-чёрных деревьях придётся восстанавливать балансировку и окраску узлов.

Недостатки рандомизированной пирамиды:

1. Большие накладные расходы на хранение: вместе с каждым элементом хранятся два-три указателя и случайный приоритет *p*.
2. Скорость доступа O(*n*) в худшем, хотя и маловероятном, случае. Поэтому декартово дерево недопустимо, например, в ядрах ОС.

Рандомизированной пирамидой поиска называют пирамиду поиска, которая получена последовательной вставкой входной последовательности ключей (подобно случайному БДП) таким образом, что приоритеты при каждой вставке разыгрываются случайно и являются случайной величиной, распределенной равномерно, например, на интервале вещественных чисел [0,1].

Добавление узла k в рандомизированную пирамиду поиска:

1. Узел вставляется в дерево поиска по ключу k (как лист дерева);
2. Случайно разыгрывается значение приоритета p;
3. Если необходимо, то с помощью левого или правого поворота восстанавливается свойство пирамиды.

# ЦЕЛЬ РАБОТЫ

Разработать программу работы с рандомизированной пирамидой поиска (Treap), все элементы которой различны. Произвести демонстрацию, т.е. сопроводить программу подробными и понятными пояснениями, чтобы ее можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

# ПОСТАНОВКА ЗАДАЧИ И СПЕЦИФИКАЦИЯ ПРОГРАММЫ

## Постановка задачи.

Для 13 варианта задания необходимо:

По заданному файлу F, все элементы которого различны, построить БДП типа рандомизированная пирамида поиска. Продемонстрировать работу программы с подробным пояснением каждого шага.

## Спецификация программы.

Входные данные: последовательность целых чисел, считанная с файла. Необходимо будет написать путь к файлу, если он находится не в домашней директории.

Выходные данные: демонстрация создания рандомизированной пирамиды поиска, поиск элемента по ключу и удаление элементов из нее.

# ОПИСАНИЕ АЛГОРИТМА

Для решения задачи построения БДП типа рандомизированная пирамида поиска используется динамическая реализация. Изначально создается объект класса randTreap, содержащий только нулевой указатель на узел БДП. Далее последовательно создается пирамида поиска с помощью метода randTreap::insert(elem key,node \*\* temp).

Заходим в метод randTreap::insert(elem key, node \*\* temp).

1. Проверяется пустой указатель на узел дерева или нет. Если да п.2., если нет п.3.

2. Тогда выделяется память для нового элемента, со случайным заданием приоритета. Далее п.4.

3. Проверяется, если новый элемент, который необходимо добавить, меньше текущего, происходит переход к левому поддереву, если больше к правому. Т.е. происходит рекурсивный запуск этого же метода, но для другого узла. И возвращаемся в п.1.

4. Полученное дерево проверяем на соответствие требуемому типу (treap) с помощью метода randTreap::iscorrect(node \*), если все хорошо работа метода завершается. Если нет — п.5.

5. Заходим в метод randTreap::check\_heap. Для начала проверяется не пустой ли указатель на узел, если пустой, работа метода завершается, если нет — п.6.

6. Далее проверяем есть ли у узла правый сын, если есть, то проверяем, что приоритет отца выше приоритета сына, иначе поворачиваем пирамиду влево с помощью метода randTreap::left\_turn. И после поворота возвращаемся обратно. п.7.

7. Далее проверяем есть ли у узла левый сын, если есть, то проверяем, что приоритет отца выше приоритета сына, иначе поворачиваем пирамиду вправо с помощью метода randTreap::right\_turn. И после поворота возвращаемся обратно. п.8.

8. Снова проверяем есть ли у узла правый сын, если есть, то метод randTreap::check\_heap(node\*\*) применяется для правого сына. И переходим в п.5.

9. Снова проверяем есть ли у узла левый сын, если есть, то метод randTreap::check\_heap(node\*\*) применяется для левого сына. И переходим в п.5.

10. В итоге получается корректная рандомизированная пирамида поиска.

# ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ И СТРУКТУР ПРОГРАММЫ

* + - 1. **Структура для узла БДП**

struct node{

elem key;

double priority;

node \* left;

node \* right;

};

key— ключ, хранящийся в узле рандомизированной пирамиды;

priority — приоритет, хранящийся в узле рандомизированной пирамиды;

left — указатель на левого сына узла;

right — указатель на правого сына узла.

* + - 1. **Класс randTreap**

struct randTreap{

struct node{

elem key;

double priority;

node \* left;

node \* right;

};

node\* top;

};

struct node — структура с содержимым узла пирамиды;

top — указатель на вершину пирамиды.

**3. void randTreap::clean(node \* temp)**

temp — указатель на узел пирамиды.

Метод очищает выделенную под пирамиду память.

**4. void randTreap::insert(elem key,node \*\* temp)**

key— значение ключа, которое нужно добавить в пирамиду;

temp — указатель на указатель на узел пирамиды.

Метод выполняет вставку нового элемента со случайным выбором приоритета.

**5. void randTreap::check\_heap(node \*\* temp)**

temp — указатель на указатель на узел пирамиды.

метод выполняет изменение пирамиды, чтоб свойство пирамиды по приоритетам было соблюдено.

**6. bool iscorrect(node \* temp)**

temp — указатель на узел пирамиды.

Метод проверяет корректность построенной пирамиды. Если все верно, возвращает true, иначе false.

**7. void print(node \* temp, int height)**

temp — указатель на узел пирамиды;

height — высота пирамиды

Метод печатает вид пирамиды в виде уступчатого списка.

**8. int height(node \* temp)**

temp — указатель на узел пирамиды.

Высчитывает высоту пирамиды.

**9. void left\_turn(node \*\* temp)**

temp — указатель на указатель на узел пирамиды.

Метод выполняет поворот пирамиды влево.

**10. void right\_turn(node \*\* temp)**

temp — указатель на указатель на узел пирамиды.

Метод выполняет поворот пирамиды вправо.

**11. node \* search(node \* temp,elem key)**

temp — указатель на узел пирамиды;

key — искомый ключ.

Метод выполняет поиск элемента в пирамиде и если находит его — возвращает указатель на него или NULL.

**12. void deleteEl(elem key, node \*\* ptr\_to\_top)**

key — удаляемый ключ;

ptr\_to\_top - указатель на указатель на узел пирамиды.

Метод находит элемент и удаляет его из БДП.

**13. node \* find\_father(node \* temp, elem key, node \* father)**

temp — указатель на узел пирамиды;

key — искомый ключ;

father — отец рассматриваемого поддерева (у дерева = NULL).

Метод нужен, чтобы найти отца определенного элемента. Используется при удалении элементов для обнуления указателя на удаляемого сына.

# ТЕСТИРОВАНИЕ

Было сделано 9 тестов для демонстрации и проверки работы программы.

Файл с данными для 1 и 2, 3 и 4, 5 и 6, 7 и 8 один, чтобы продемонстрировать, что пирамида действительно рандомизированная.

Таблица 1 – Тестирование программы

|  |  |  |
| --- | --- | --- |
| Тест № | Данные | Результат |
| 1 | 40 50 20 80 10  77 | Элемент 77 не найден. Сейчас произойдет его добавление.  Пирамида без нового элемента:  ----50, 0.6  ---10, 0.6  --40, 0.4  -20, 0.4  ---80, 0.6  Пирамида с новым элементом:  ----50, 0.6  ---10, 0.6  --40, 0.4  -20, 0.4  ---80, 0.6  --77, 0 |
| 2 | 40 50 20 80 10  40 | Элемент 40 найден!  Построенное БДП:  -----10, 0.7  ----20, 0.6  ---40, 0.5  --80, 0.5  -50, 0.3 |
| 3 | 10 7 33 32 88 63 55 332 90  90 | Элемент 90 найден!  Построенное БДП:  ----55, 0.9  ---7, 0.6  --32, 0.6  -10, 0.4  -33, 0.4  ---88, 0.6  --63, 0  --90, 0.5  -332, 0.3 |
| 4 | 10 7 33 32 88 63 55 332 90  123 | Элемент 123 не найден. Сейчас произойдет его добавление.  Пирамида без нового элемента:  -----332, 0.9  ----88, 0.7  ---33, 0.6  --10, 0.5  -7, 0.3  -32, 0.5  --55, 0.6  -63, 0.3  ---90, 0  Пирамида с новым элементом:  -----332, 0.9  ----88, 0.7  ---33, 0.6  --10, 0.5  -7, 0.3  -32, 0.5  --55, 0.6  -63, 0.3  ---123, 0.1  --90, 0 |
| 5 | 10 7  15 | Элемент 15 не найден. Сейчас произойдет его добавление.  Пирамида без нового элемента:  --7, 0.6  -10, 0.4  Пирамида с новым элементом:  ---7, 0.6  --10, 0.4  -15, 0.4 |
| 6 | 10 7  10 | Элемент 10 найден!  Построенное БДП:  --10, 0.5  -7, 0.3 |
| 7 | 10 20 30 40 50 60 70 80 90 100  50 | Элемент 50 найден!  Построенное БДП:  -----70, 0.9  ----20, 0.6  ---10, 0.4  ---40, 0.6  --30, 0.4  --50, 0.6  -60, 0  ----90, 0.5  ---80, 0.3  ---100, 0.5 |
| 8 | 10 20 30 40 50 60 70 80 90 100  55 | Элемент 55 не найден. Сейчас произойдет его добавление.  Пирамида без нового элемента:  -----80, 0.9  ----50, 0.7  ---30, 0.6  --10, 0.5  -20, 0.3  --40, 0.5  ---70, 0.6  --60, 0.3  ----100, 0.1  ---90, 0  Пирамида с новым элементом:  -----80, 0.9  ----50, 0.7  ---30, 0.6  --10, 0.5  -20, 0.3  --40, 0.5  ---70, 0.6  --60, 0.3  -55, 0  ----100, 0.1  ---90, 0 |
| 9 | 10 | БДП пусто! Программа завершена! |

Для более наглядной демонстрации работы программы был создан bash-скрипт, последовательно выводящий содержимое очередного теста и результат работы программы для этого теста.

Выберем 5 тест для детального обзора. Программа считывает название файла, хранящего последовательность элементов для добавление в пирамиду. Далее считывает элемент, который надо найти в пирамиде. Поэлементно считываются из файла ключи, которые надо добавить в пирамиду.

1. Принимаем элемент 10

2. Заходим в метод randTreap::insert. Полученный методом указатель на узел — пустой, т. к. изначально создается новая пирамида, где ничего не содержится, поэтому сразу выделяется память для нового узла с ключом 10. Случайным образом выбирается приоритет. В этот раз он равен 0.4. Он также записывается в узел, указатели на сыновей обнуляются. Полученный узел стал вершиной дерева. Выходим из метода randTreap::insert.

3. Считываем следующий элемент = 7. Заходим в метод randTreap::insert. Так как теперь вершина дерева не пустая сравниваем новый ключ с ключом в вершине, он меньше 10, поэтому переходим на левого сына и снова заходим в метод randTreap::insert. Указатель на левого сына пустой, поэтому выделяется память для нового узла с ключом 7. Случайным образом выбирается приоритет. В этот раз он равен 0.6. Проверяем пирамиду на корректность. Она некорректна, т. к. приоритет сына больше приоритета отца. Поэтому поворачиваем пирамиду вправо. Теперь вершина дерева (7, 0.6), а (10, 0.4) его правый сын.

4. Заходим в метод randTreap::search, чтобы найти искомый элемент 15. Ни в одном из узлов не содержится 15, поэтому нам необходимо добавить этот ключ.

5. Снова заходим в метод randTreap::insert. Пирамида непустая, поэтому происходит сравнение ключа с вершиной. 15 > 7, поэтому переходим на правого сына узла 7. Правый сын также существует и его ключ равен 10. 15>10, поэтому переходим на правого сына узла (10, 0.4). Выделяем память под новый узел и случайным образом выбираем приоритет. В этот раз он равен 0.4. Приоритет отца такой же, значит свойства пирамиды не нарушается и дальнейших преобразований не требуется. Пирамида построена.

# ДЕМОНСТРАЦИЯ

Программа демонстрирует каждый шаг своей работы, показывая, как выглядит БДП после каждого изменения. Вид реализован в виде уступчатого списка. Далее приведен вид для теста с данными из 1 теста, последовательность целых чисел: 40 50 20 80 10. И элемент 77 для добавления.

Test 1

Здравствуйте!

Данная программа создана для демонстрации работы с БДП типа

"Рандомизированная пирамида поиска"(Treap).

В ходе программы будет показан:

1) каждый шаг формирования БДП,

2) как происходит поиск, добавление и удаление элментов из пирамиды.

Вывод БДП реализован в виде уступчатых списков.

Для начала работы, пожалуйста, введите имя файла в формате <имя\_файла>.txt

(текстовый файл, где у вас хранится последовательность элементов для формирования БДП)

./Tests/test1.txt

Введите ключ (целое число), которое вы хотите найти:

77

Входная последовательность:

40 50 20 80 10

---------------------

Шаг № 1

Программа приняла 40

Сейчас произойдет поиск места для вставки.

Текущая БДП:

Пустое БДП!

Место для нового элемента (по ключу) найдено!

Сгенерирован приоритет = 0.3

Вид пирамиды после вставки элемета (по ключу):

- (40; 0.3)

По приоритету БДП соотвествует пирамиде! Дальнейших преобразований не требуется!

---------------------

Шаг № 2

Программа приняла 50

Сейчас произойдет поиск места для вставки.

Текущая БДП:

- (40; 0.3)

Новый ключ (50) больше 40. Переходим на правую ветвь.

Место для нового элемента (по ключу) найдено!

Сгенерирован приоритет = 0.4

Вид пирамиды после вставки элемета (по ключу):

-- (40; 0.3)

- (50; 0.4)

Приоритет отца (0.3) меньше приоритета правого сына (0.4)

=> по приоритету БДП не соотвествует пирамиде!

Необходим поворот БДП налево, т.к. 0.3 < 0.4

Поворот налево:

(40; 0.3) становится левым сыном своего правого сына (50; 0.4)

Место его правого сына займет левый сын (50; 0.4) (если он есть, иначе он останется без правого сына).

Вид пирамиды после поворота:

-- (50; 0.4)

- (40; 0.3)

---------------------

Шаг № 3

Программа приняла 20

Сейчас произойдет поиск места для вставки.

Текущая БДП:

-- (50; 0.4)

- (40; 0.3)

Новый ключ (20) меньше 50. Переходим на левую ветвь.

Новый ключ (20) меньше 40. Переходим на левую ветвь.

Место для нового элемента (по ключу) найдено!

Сгенерирован приоритет = 0.8

Вид пирамиды после вставки элемета (по ключу):

--- (50; 0.4)

-- (40; 0.3)

- (20; 0.8)

Приоритет отца (0.3) меньше приоритета левого сына (0.8)

=> по приоритету БДП не соотвествует пирамиде!

Необходим поворот БДП направо, т.к. 0.3 < 0.8

Поворот направо:

(40; 0.3) становится правым сыном своего левого сына (20; 0.8)

Место его леввого сына займет правый сын (20; 0.8) (если он есть, иначе он останется без левого сына).

Вид пирамиды после поворота:

--- (50; 0.4)

-- (20; 0.8)

- (40; 0.3)

Приоритет отца (0.4) меньше приоритета левого сына (0.8)

Необходим поворот БДП направо, т.к. 0.4 < 0.8

Поворот направо:

(50; 0.4) становится правым сыном своего левого сына (20; 0.8)

Место его леввого сына займет правый сын (20; 0.8) (если он есть, иначе он останется без левого сына).

Вид пирамиды после поворота:

--- (20; 0.8)

-- (50; 0.4)

- (40; 0.3)

---------------------

Шаг № 4

Программа приняла 80

Сейчас произойдет поиск места для вставки.

Текущая БДП:

--- (20; 0.8)

-- (50; 0.4)

- (40; 0.3)

Новый ключ (80) больше 20. Переходим на правую ветвь.

Новый ключ (80) больше 50. Переходим на правую ветвь.

Место для нового элемента (по ключу) найдено!

Сгенерирован приоритет = 0.5

Вид пирамиды после вставки элемета (по ключу):

--- (20; 0.8)

-- (50; 0.4)

- (40; 0.3)

- (80; 0.5)

Приоритет отца (0.4) меньше приоритета правого сына (0.5)

=> по приоритету БДП не соотвествует пирамиде!

Необходим поворот БДП налево, т.к. 0.4 < 0.5

Поворот налево:

(50; 0.4) становится левым сыном своего правого сына (80; 0.5)

Место его правого сына займет левый сын (80; 0.5) (если он есть, иначе он останется без правого сына).

Вид пирамиды после поворота:

---- (20; 0.8)

--- (80; 0.5)

-- (50; 0.4)

- (40; 0.3)

---------------------

Шаг № 5

Программа приняла 10

Сейчас произойдет поиск места для вставки.

Текущая БДП:

---- (20; 0.8)

--- (80; 0.5)

-- (50; 0.4)

- (40; 0.3)

Новый ключ (10) меньше 20. Переходим на левую ветвь.

Место для нового элемента (по ключу) найдено!

Сгенерирован приоритет = 0.3

Вид пирамиды после вставки элемета (по ключу):

---- (20; 0.8)

--- (10; 0.3)

--- (80; 0.5)

-- (50; 0.4)

- (40; 0.3)

По приоритету БДП соотвествует пирамиде! Дальнейших преобразований не требуется!

---------------------

Поиск элемента 77 в БДП (по ключу)...

Элемент 77 не найден. Сейчас произойдет его добавление.

---------------------

Новый ключ (77) больше 20. Переходим на правую ветвь.

Новый ключ (77) меньше 80. Переходим на левую ветвь.

Новый ключ (77) больше 50. Переходим на правую ветвь.

Место для нового элемента (по ключу) найдено!

Сгенерирован приоритет = 0.9

Вид пирамиды после вставки элемета (по ключу):

---- (20; 0.8)

--- (10; 0.3)

--- (80; 0.5)

-- (50; 0.4)

- (40; 0.3)

- (77; 0.9)

Приоритет отца (0.4) меньше приоритета правого сына (0.9)

=> по приоритету БДП не соотвествует пирамиде!

Необходим поворот БДП налево, т.к. 0.4 < 0.9

Поворот налево:

(50; 0.4) становится левым сыном своего правого сына (77; 0.9)

Место его правого сына займет левый сын (77; 0.9) (если он есть, иначе он останется без правого сына).

Вид пирамиды после поворота:

----- (20; 0.8)

---- (10; 0.3)

---- (80; 0.5)

--- (77; 0.9)

-- (50; 0.4)

- (40; 0.3)

Приоритет отца (0.5) меньше приоритета левого сына (0.9)

Необходим поворот БДП направо, т.к. 0.5 < 0.9

Поворот направо:

(80; 0.5) становится правым сыном своего левого сына (77; 0.9)

Место его леввого сына займет правый сын (77; 0.9) (если он есть, иначе он останется без левого сына).

Вид пирамиды после поворота:

---- (20; 0.8)

--- (10; 0.3)

--- (77; 0.9)

-- (50; 0.4)

- (40; 0.3)

-- (80; 0.5)

Приоритет отца (0.8) меньше приоритета правого сына (0.9)

Необходим поворот БДП налево, т.к. 0.8 < 0.9

Поворот налево:

(20; 0.8) становится левым сыном своего правого сына (77; 0.9)

Место его правого сына займет левый сын (77; 0.9) (если он есть, иначе он останется без правого сына).

Вид пирамиды после поворота:

---- (77; 0.9)

--- (20; 0.8)

-- (10; 0.3)

-- (50; 0.4)

- (40; 0.3)

--- (80; 0.5)

---------------------

Пирамида с новым элементом:

---- (77; 0.9)

--- (20; 0.8)

-- (10; 0.3)

-- (50; 0.4)

- (40; 0.3)

--- (80; 0.5)

Введите ключ (целое число), которое вы хотите удалить:

Удаление элемента 77 из БДП

Удаляемый элемент - не лист, сделаем его листом

Для этого в его приоритет запишем 0.0

Поворотами восставновим свойство пирамиды в БДП и сделаем элемент листом

Вид пирамиды до поворота:

---- (77; 0.0)

--- (20; 0.8)

-- (10; 0.3)

-- (50; 0.4)

- (40; 0.3)

--- (80; 0.5)

Приоритет отца (0) меньше приоритета левого сына (0.8)

Необходим поворот БДП налево, т.к. 0 < 0.5

Поворот налево:

(77; 0.0) становится левым сыном своего правого сына (80; 0.5)

Место его правого сына займет левый сын (80; 0.5) (если он есть, иначе он останется без правого сына).

Необходим поворот БДП направо, т.к. 0 < 0.8

Поворот направо:

(77; 0.0) становится правым сыном своего левого сына (20; 0.8)

Место его леввого сына займет правый сын (20; 0.8) (если он есть, иначе он останется без левого сына).

Необходим поворот БДП направо, т.к. 0 < 0.4

Поворот направо:

(77; 0.0) становится правым сыном своего левого сына (50; 0.4)

Место его леввого сына займет правый сын (50; 0.4) (если он есть, иначе он останется без левого сына).

Вид пирамиды после поворота:

---- (80; 0.5)

--- (20; 0.8)

-- (10; 0.3)

-- (50; 0.4)

- (40; 0.3)

- (77; 0.0)

Приоритет отца (0.5) меньше приоритета левого сына (0.8)

Необходим поворот БДП направо, т.к. 0.5 < 0.8

Поворот направо:

(80; 0.5) становится правым сыном своего левого сына (20; 0.8)

Место его леввого сына займет правый сын (20; 0.8) (если он есть, иначе он останется без левого сына).

Вид пирамиды после поворота:

---- (20; 0.8)

--- (10; 0.3)

--- (80; 0.5)

-- (50; 0.4)

- (40; 0.3)

- (77; 0.0)

Теперь элемент 77 - лист, его можно удалять!

Вид пирамиды после удаления:

---- (20; 0.8)

--- (10; 0.3)

--- (80; 0.5)

-- (50; 0.4)

- (40; 0.3)

Таким образом, все этапы работы с БДП типа рандомизированная пирамида поиска были отражены в ходе программы. Демонстрация для всех тестов расположена в прилагающемся файле out.txt, т. к. из-за очень подробного описания каждого шага она занимает очень много места.

# ЗАКЛЮЧЕНИЕ

В ходе работы была разработана программа, реализующая работу рандомизированной пирамиды поиска. Была проведена демонстрация добавления элемента, поиск элемента по ключу и удаление элементов БДП. Каждый шаг программы был описан простым и понятным языком, так что она может быть использована в учебных целях, либо для пояснения работы данного типа БДП.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД, СОДЕРЖАЩИЙ ГОЛОВНУЮ ФУНКЦИЮ**

#include <iostream>

#include <iomanip>

#include <fstream>

#include <cstdlib>

#include <string>

#include <ctime>

#include "structure2.h"

#include <clocale>

int main(){

char name[20];

std::ifstream file;

int key=0, new\_key=0;

int forrand=time(0);

std::string str\_treap;

int counter=1;

setlocale(LC\_ALL, "Russian");

//receiving file .txt

std::cout << "Здравствуйте!\nДанная программа создана для демонстрации "

<< "работы с БДП типа\n\"Рандомизированная пирамида поиска\"(Treap).\n"

<< "В ходе программы будет показан:\n1) каждый шаг формирования БДП,\n"

<< "2) как происходит поиск, добавление и удаление элментов из пирамиды.\n"

<< "Вывод БДП реализован в виде уступчатых списков.\n"

<< "Для начала работы, пожалуйста, введите имя файла в формате <имя\_файла>.txt\n"

<< "(текстовый файл, где у вас хранится последовательность элементов для формирования БДП)"

<< std::endl;

std::cin.getline(name,20);

file.open(name);

if (!file.is\_open()){

std::cout << "\nНевозможно открыть файл.\nПрограмма завершена!" << std::endl;

return 0;

}

//receiving key for the search

std::cout << "Введите ключ (целое число), которое вы хотите найти: ";

std::cin >> new\_key;

std::cout << std::endl;

std::getline(file,str\_treap);

std::cout << "Входная последовательность: \n" << str\_treap << std::endl;

file.clear();

file.seekg(0, file.beg);

//making class object for future treap

randTreap<int> my\_treap;

//receiving elements from file and adding them to the treap

do{

srand(forrand);

file >> key;

if (file.eof()) break;

std::cout << "---------------------\n";

std::cout << "Шаг № " << counter++ << "\n";

std::cout << "\nПрограмма приняла " << key << "\n";

std::cout << "Сейчас произойдет поиск места для вставки.\n";

std::cout << "Текущая БДП:\n";

my\_treap.print(my\_treap.top,my\_treap.height(my\_treap.top));

my\_treap.insert(key,&(my\_treap.top));

forrand++;

}while (1);

std::cout << "---------------------\n";

if (!my\_treap.height(my\_treap.top)){

std::cout << "БДП пусто! Программа завершена!\n";

exit(1);

}

file.close();

//searching for new\_key if it is found print the treap,

//if not add new\_key to the treap and print the result.

std::cout << "Поиск элемента "<< new\_key << " в БДП (по ключу)...\n";

if (my\_treap.search(my\_treap.top,new\_key)){

std::cout << "Элемент " << new\_key << " найден!\n";

std::cout << "Построенное БДП:\n";

my\_treap.print(my\_treap.top,my\_treap.height(my\_treap.top));

}

else{

std::cout << "Элемент "<< new\_key <<" не найден. Сейчас произойдет его добавление.\n";

// std::cout << "Пирамида без нового элемента:\n";

// my\_treap.print(my\_treap.top,my\_treap.height(my\_treap.top));

std::cout << "---------------------\n";

my\_treap.insert(new\_key,&(my\_treap.top));

std::cout << "---------------------\n";

std::cout << "Пирамида с новым элементом:\n";

my\_treap.print(my\_treap.top,my\_treap.height(my\_treap.top));

}

std::cout << "Введите ключ (целое число), которое вы хотите удалить: ";

std::cin >> new\_key;

std::cout << std::endl;

my\_treap.deleteEl(new\_key,&(my\_treap.top));

return 0;

}

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД ЗАГОЛОВОЧНИКА**

//template for randomized treap, type of the key selectable

template <class elem>

struct randTreap{

//struct for the node of the treap, which contains pair(key,priority)

//and pointers to the sons

struct node{

elem key;

double priority;

node \* left;

node \* right;

};

//pointer to the top node of the randomized treap

node \* top;

//constructor

randTreap(){

top = NULL;

}

~randTreap(){

if (top)

randTreap::clean(top);

}

//release of the allocated memory, method for destructor

void clean(node \* temp){

if (temp->left)

randTreap::clean(temp->left);

if (temp->right)

randTreap::clean(temp->right);

delete temp;

}

//adding new node, with correction

void insert(elem key,node \*\* temp){

if (!(\*temp)){

std::cout << "Место для нового элемента (по ключу) найдено!\n";

\*temp = new node;

(\*temp)->key=key;

(\*temp)->priority=(double)((rand() % 10)+1)/10;

std::cout << "Сгенерирован приоритет = "<< (\*temp)->priority<<"\n";

(\*temp)->left=NULL;

(\*temp)->right=NULL;

std::cout << "Вид пирамиды после вставки элемета (по ключу):\n";

randTreap::print(top,randTreap::height(top));

//checking if treap is correct, if not

//turning it to the right direction

if(randTreap::iscorrect(top)){

std::cout << "По приоритету БДП соотвествует пирамиде! Дальнейших преобразований не требуется!\n";

}

else{

std::cout << "=> по приоритету БДП не соотвествует пирамиде!\n";

do{

randTreap::check\_heap(&top);

std::cout << "Вид пирамиды после поворота:\n";

randTreap::print(top,randTreap::height(top));

}while(!(randTreap::iscorrect(top)));

// std::cout << "After correction:\n";

// randTreap::print(top,randTreap::height(top));

}

}

else if (key < (\*temp)->key){

std::cout << "Новый ключ ("<<key<<") меньше " << (\*temp)->key <<". Переходим на левую ветвь.\n";

randTreap::insert(key,&((\*temp)->left));

}

else if (key > (\*temp)->key){

std::cout << "Новый ключ ("<<key<<") больше " << (\*temp)->key <<". Переходим на правую ветвь.\n";

randTreap::insert(key,&((\*temp)->right));

}

}

void deleteEl(elem key, node \*\* ptr\_to\_top){

node \* temp=randTreap::search(\*(ptr\_to\_top),key);

if (temp){

std::cout << "Удаление элемента " << key <<" из БДП\n";

if (temp->left == NULL && temp->right==NULL){

std::cout << "Удаляемый элемент - лист, значит можно его удалить\n";

}

else{

std::cout << "Удаляемый элемент - не лист, сделаем его листом\n"

<< "Для этого в его приоритет запишем 0.0\n"

<< "Поворотами восставновим свойство пирамиды в БДП и сделаем элемент листом\n";

temp->priority=0.0;

std::cout << "Вид пирамиды до поворота:\n";

randTreap::print(top,randTreap::height(top));

while(!(randTreap::iscorrect(top))){

randTreap::check\_heap(&top);

std::cout << "Вид пирамиды после поворота:\n";

randTreap::print(top,randTreap::height(top));

}

std::cout << "Теперь элемент " << key << " - лист, его можно удалять!\n";

temp=randTreap::search(\*(ptr\_to\_top),key);

}

node \* father=randTreap::find\_father(\*(ptr\_to\_top),key,NULL);

if (father->left == temp) father->left=NULL;

if (father->right == temp) father->right=NULL;

delete temp;

std::cout << "Вид пирамиды после удаления:\n";

randTreap::print(top,randTreap::height(top));

}

else{

std::cout << "Данного элемента нет - удалять нечего!\n";

return;

}

}

//changing treap to correct one

void check\_heap(node \*\* temp, double max=1.0){

if (\*temp){

if ((\*temp)->right){

if ((\*temp)->priority < (\*temp)->right->priority){

std::cout << "Необходим поворот БДП налево, т.к. "

<< (\*temp)->priority <<" < " << (\*temp)->right->priority

<< std::endl;

randTreap::left\_turn(temp);

}

}

if((\*temp)->left){

if ((\*temp)->priority < (\*temp)->left->priority){

std::cout << "Необходим поворот БДП направо, т.к. "

<< (\*temp)->priority <<" < " << (\*temp)->left->priority

<< std::endl;

randTreap::right\_turn(temp);

}

}

if ((\*temp)->right)

randTreap::check\_heap(&((\*temp)->right));

if((\*temp)->left)

randTreap::check\_heap(&((\*temp)->left));

}

}

//check if treap is correct

bool iscorrect(node \* temp){

if (!temp) return true;

if (temp->left){

if (temp->priority < temp->left->priority){

std::cout << "Приоритет отца ("<< temp->priority <<") меньше приоритета левого сына ("<< temp->left->priority<<")\n";

return false;

}

}

if (temp->right){

if (temp->priority < temp->right->priority){

std::cout << "Приоритет отца ("<< temp->priority <<") меньше приоритета правого сына ("<< temp->right->priority<<")\n";

return false;

}

}

return randTreap::iscorrect(temp->left) && randTreap::iscorrect(temp->right);

}

//printing treap

void print(node \* temp, int height){

char \* fstr = new char[10];

if (!height){

std::cout << "Пустое БДП!\n";

return;

}

std::string dashes(height,'-');

if (temp){

sprintf(fstr,"(%d; %.1lf)",temp->key,temp->priority);

// std::cout.width(height+20);

std::cout << std::setw(height+1)<<dashes << std::setw(12) <<fstr << std::endl;

delete [] fstr;

}

if (temp->left){

randTreap::print(temp->left,height-1);

}

if (temp->right){

randTreap::print(temp->right,height-1);

}

}

//counting the height of the tree

int height(node \* temp){

if (!temp) return 0;

else return 1+std::max(randTreap::height(temp->left),randTreap::height(temp->right));

}

//making left turn of the treap

void left\_turn(node \*\* temp){

std::cout << "Поворот налево:\n";

node \* forswap=NULL;

char \* node\_str = new char[10];

if (\*temp){

sprintf(node\_str,"(%d; %.1lf)",(\*temp)->key,(\*temp)->priority);

std::cout << node\_str << " становится левым сыном своего правого сына ";

forswap = (\*temp)->right;

sprintf(node\_str,"(%d; %.1lf)",forswap->key,forswap->priority);

std::cout << node\_str << "\nМесто его правого сына займет левый сын " << node\_str

<< " (если он есть, иначе он останется без правого сына).\n";

(\*temp)->right = forswap->left;

forswap->left = (\*temp);

(\*temp) = forswap;

}

delete [] node\_str;

}

//making right turn of the treap

void right\_turn(node \*\* temp){

std::cout << "Поворот направо:\n";

node \* forswap=NULL;

char \* node\_str = new char[10];

if (\*temp){

sprintf(node\_str,"(%d; %.1lf)",(\*temp)->key,(\*temp)->priority);

std::cout << node\_str << " становится правым сыном своего левого сына ";

forswap = (\*temp)->left;

sprintf(node\_str,"(%d; %.1lf)",forswap->key,forswap->priority);

std::cout << node\_str << "\nМесто его леввого сына займет правый сын " << node\_str

<< " (если он есть, иначе он останется без левого сына).\n";

(\*temp)->left = forswap->right;

forswap->right = (\*temp);

(\*temp) = forswap;

}

delete [] node\_str;

}

//searching for the key

node \* search(node \* temp,elem key){

if (!temp) return NULL;

if (temp->key == key) return temp;

else if (key< temp->key) randTreap::search(temp->left,key);

else if (key > temp->key) randTreap::search(temp->right,key);

// else return randTreap::search(temp->left,key) || randTreap::search(temp->right,key);

}

node \* find\_father(node \* temp, elem key, node \* father){

if (!temp) return NULL;

if (temp->key == key) return father;

else if (key< temp->key){

randTreap::find\_father(temp->left,key,temp);

}

else if (key > temp->key){

randTreap::find\_father(temp->right,key,temp);

}

}

};