# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МОЭВМ**

# КУРСОВАЯ РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: RB-дерево vs Хеш-таблица (открытая адресация). Исследование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Камынин А. А. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург 2022

# ЗАДАНИЕ

**НА КУРСОВУЮ РАБОТУ**

Студент: Камынин А. А.

Группа 1384

Тема работы : RB-дерево vs Хеш-таблица (открытая адресация). Исследование

Исходные данные:

Содержание пояснительной записки:

Аннотация, Содержание, Введение, Отчет, Примеры работы программы, Заключение

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 22.03.2022 Дата сдачи реферата: 01.06.2022 Дата защиты реферата: 01.06.2022

Студент Камынин А. А.

Преподаватель Жангиров Т. Р.

# АННОТАЦИЯ

П

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

В рамках работы необходимо реализовать Красно-черное дерево и Хеш-таблицу с открытой адресацией, а именно основные операции для работы с ними: вставку, поиск, удаление. Необходимо сравнить данные структуры данных между собой на основе собственных сгенерированных входных данных, сравнить результат с теоретической оценкой.

# РЕАЛИЗАЦИЯ СТРУКТУР ДАННЫХ

# Красно-черное дерево

Для реализации красно-черного дерева были написаны два класса на языке Python: Node и RBTree. Класс Node представляет собой узел, элемент дерева, который хранит в себе поля key (значение узла), left, right и parent (указатели на левого и правого ребенка, а также на родителя данного узла в дереве) и цвет (красный или черный). Для обозначения цветов были использованы строки, записанные в глобальные переменные BLACK и RED.

При создании красно-черного дерева указателю на корень присваивается None, количеству неудаленных узлов 0 (данное поле необходимо для проверки корректности удаления в дереве), а фиктивному листу nil, не несущему никакой значительной информации, но тоже являющемся частью красно-черного дерева (необходим для корректного удаления в дереве), присваивается объект класса Node со значением -1 и черным цветом.

Реализация вставки. Для вставки в красно-черное дерево узла с ключом key были реализованы методы insert(key) и fix\_insert(node). В первом методе проверяется, не пусто ли текущее дерево (указатель на корень не равен None), и если это так, то узел с переданным ключом записывается в корень. В ином случае, пользуясь свойством красно-черного дерева, что слева от текущего узла располагаются элементы с меньшим ключом, а справа с большим, ищется подходящий лист для заданного ключа. Таким образом, мы проходимся от корня к нужному листу, тем самым осуществив h сравнений, где h - высота красно-черного дерева. Так как после вставки элемента свойства могли нарушиться, то вызывается метод fix\_insert(node), восстанавливающий их. При восстановлении в цикле узел node поднимается каждый раз снизу вверх (если при этом не оказалось, что свойства уже восстановлены), в худшем случае доходя до корня дерева. При восстановлении свойств используется левое и правое малое вращение. Так как после восставления корень может стать красным, то в таком случае он перекрашивается в черный цвет.

Реализация удаления. Для удаления из красно-черного дерева узла с переданным ключом key были реализованы методы delete(key) и fix\_delete(node). В методе delete(key) проходом от корня до листа ищется узел к удалению: если данный узел не найден, об этом выводится сообщение, значение поля count\_not\_delete\_nodes увеличивается на 1; в ином случае записывается указатель на удаляемый элемент в node\_to\_delete. При удалении вершины проверяется три случая: у вершины нет детей, тогда указатель на родителя у фиктивного листа nil изменяется на данный элемент (это делается еще до проверок остальных случаев); у вершины один ребенок, тогда меняем местами существующего ребенка и удаляемый узел; у вершины есть оба ребенка, то находится узел с большим значением ключа. Так как нарушение свойств красно-черного дерева может нарушиться только тогда, когда удаляемая вершина красная, то метод fix\_delete(node) вызывается только в том случае, если она черная. При восстановлении свойств меняются, в зависимости от случая, цвета узлов, применяются левые и правые вращения.

Реализация поиска. Поиск узла со значением key осуществлен в методе search(key). В данном методе осуществляется проход от корня до листов с проверками, больше или меньше ключ key текущего рассматриваемого узла (соответственно, это интерпретируется, как левее или правее, согласно свойствам красно-черного дерева). Как только ключ совпал, осуществляется выход из цикла и выводится сообщение, что ключ найден. Так как проход осуществляется от корня к последнему уровню, то в общем случае будет h итераций, где h - высота дерева.

# Хеш-таблица с открытой адресацией

Для реализации хеш-таблицы с открытой адресацией были созданы классы Probe и HashTable. Класс Probe описывает собой элемент хеш-таблицы, который хранит в себе ключ и значение idx (idx не является точным индексом элемента в хеш-таблице; скорее, он представляет собой дополнительный параметр, по которому можно определить точное расположение в массиве текущей пары (ключ, значение).

При создании хеш-таблицы инициализируется ее размер (поле size), текущее количество записанных элементов current\_size иницализируется 0. Сама таблица представляет собой список размера size, в котором будут храниться объекты класса Probe (изначально записывается None). Также определены поля hash\_type для выбора конкретного метода пробирования открытой адресации, параметры k, c1, c2, используемые в линейном и квадратичном пробировании. Вызывается метод choose\_hash\_type(), который позволяет выбрать желаемое хеширование.

Реализация вставки. Для вставки в хеш-таблицу реализован метод insert(key). В данном методе в текущее смещение idx записывается 0, а значение хеш-функции (результат метода hash\_function(key)) в hash\_value. В цикле while вычисляется на основе хеш-значения и перебираемых от 0 целых idx номер ячейки: вызывается метод hashing(hash\_value, i), который на основании выбранного пользователем метода пробирования вызывает соответсвующий; линейное пробирование реализовано в методе linear\_hashing(hash\_value, i), квадратичное - в quadratic\_hashing(hash\_value, i). После проверяется, что в списке ячейка пустая или удаленная, и в таком случае элемент записывается в хеш-таблицу. Если текущий размер таблицы стал равен 2/3 от максимального размера, то вызывается метод resize(), расширяющий таблицу вдвое.

Реализация удаления. Для удаления из хеш-таблицы элемента с ключом key реализован метод delete(key). В данном методе проверяется, что таблица не пустая - тогда удалять нечего. В ином случае, аналогично вставке, вычисляется хеш-значение, записываемое в hash\_value, осуществляется перебор idx от 0 до size-1 в цикле for. Постоянно пересчитывая номер текущей рассматриваемой ячейки cell\_number сверяется значения ключа к удалению и ключа в ячейке с текущим номером - если они совпали, то данная ячейка помечается как deleted, текущий размер current\_size уменьшается на 1. Яче йка помечается, как удаленная, так как если в нее записать None, то в дальнейшем мы не сможем найти те элементы, в момент которых в таблице данное место было занято (и из-за чего был выбран дальний элемент в последовательности испробованных мест).

Реализация поиска. Для поиска элемента с заданным ключом был создан метод search(key). В данном методе проверяется, что таблица не пустая - тогда искать нечего. В ином случае, аналогично удалению и вставке, вычисляется хеш-значение, записываемое в hash\_value, осуществляется перебор idx от 0 до size-1 в цикле for. Постоянно пересчитывая номер текущей рассматриваемой ячейки cell\_number сверяется значения ключа для поиска и ключа в ячейке с текущим номером - если они совпали, то возвращается номер найденной ячейки.

# СРАВНЕНИЕ СТРУКТУР ДАННЫХ

# Заголовок 1.

# Заголовок 2.

# ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

# ЗАКЛЮЧЕНИЕ

По итогам курсовой работы была создана программа...

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: