|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

**«ОБРАБОТКА ДЕРЕВЬЕВ, ХЕШ-ФУНКЦИЙ»**

Студент ???

Группа ИУ7 – 3?Б

Преподаватель Силантьева А.В.

*2021 г.*

**Цель работы**

Цель работы – построить дерево, вывести его на экран в виде дерева, реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов, сбалансировать дерево, сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления; построить хеш-таблицу и вывести ее на экран, устранить коллизии, если они достигли указанного предела, выбрав другую хеш-функцию и реструктуризировав таблицу; сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска(ДДП),в хеш-таблицахи в файлах. Сравнить эффективность реструктуризации таблицы для устранения коллизий и поиска в ней с эффективностью поиска в исходной таблице.

**Описание условия задачи**

Построить ДДП, сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность поиска в ДДП в АВЛ деревеи в хеш-таблице (используя открытую или закрытую адресацию)и в файле. Вывести на экран деревья и хеш-таблицу. Подсчитать среднее количество сравнений для поиска данных в указанных структурах. Произвести реструктуризацию хеш-таблицы, если среднее количество сравнений больше указанного. Оценить эффективность использования этих структур (по времени и памяти) для поставленной задачи. Оценить эффективность поиска в хеш-таблице при различном количестве коллизий.

**Описание технического задания**

В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного целого числа в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию. Если количество сравнений в хеш-таблице больше указанного(вводить),то произвести реструктуризацию таблицы, выбрав другую функцию.

Входные данные:

1. Номер команды: целое число в диапазоне от 0 до 4.
2. Элемент данных: целое число.
3. Максимальное количество сравнений: цело число.

Выходные данные:

1. Результат выполнения определенной команды.
2. Сообщение об ошибке.
3. Сравнение времени поиска элемента в различных структурах.

Функции программы:

1. Загрузить данные из файла.
2. Вывести деревья и хеш-таблицу.
3. Добавить элемент.
4. Сравнить время поиска элемента в разных структурах.

0. Выход из программы.

Обращение к программе:

Запускается через терминал.

Аварийные ситуации:

1. Некорректный ввод номера команды.

На входе: число, большее чем 4 или меньшее, чем 0.

На выходе: сообщение «Invalid menu code»

2. Некорректный ввод номера команды.

На входе: буква.

На выходе: сообщение «Invalid menu code»

3. Поиск в пустой таблице.

На входе: число.

На выходе: сообщение «There is not any data.»

Описание структуры данных

*Структура, описывающая узел дерева:*

**typedef struct tree\_node tree\_node;**

**struct tree\_node**

**{**

**int number;**

**int height;**

**tree\_node \*left;**

**tree\_node \*right;**

**};**

Поля структуры

**number –** число

**height –** высота дерева в этом узле

**\*left –** указатель на левую ветку дерева

**\*right –** указатель на правую ветку дерева

*Структуры, описывающие хэш-таблицу:*

* *Структура, описывающая данные таблицы:*

***struct cell\_t***

***{***

***int number;***

***int id;***

***cell\_t \*next;***

***};***

Поля структуры:

**number –** число

**id –** номер в односвязном списке

**\*next –** указатель на следующий элемент односвязного списка (для устранения коллизий)

* *Структура, описывающая саму таблицу:*

***typedef struct***

***{***

***cell\_t \*******data;***

***int coll;***

***int now\_count\_elem;***

***int count\_elem;***

***} table\_t;***

Поля структуры:

***data* –** указатель на данные

**coll –** количество коллизий

***now\_count\_elem* –** количество считанных элементов

***count\_elem* –** размер таблицы

**Описание алгоритма добавления в хэш-таблицу**

1. Считается количество элементов в файле.
2. Находится минимальное простое число большее количества элементов.
3. Считываемое число делится на это простое число, полученный остаток от деления есть индекс элемента в таблице.
4. Если количество сравнений превышает максимальное, то находится следующее простое число, большее текущего и повторяются пункты 1 – 3.

**Описание действий пользователя**

1. Пользователь вводит номер команды из предложенного меню.
2. Пока пользователь не введет 0, ему будет предложено вводить номера команд.

**Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название теста** | **Пользователь вводит** | **Вывод** |
| 1 | Некорректный ввод команды | 45 (разрешено от 0 до 4) | Invalid menu code |
| 2 | Пустой ввод | Буква. | Invalid menu code |
| 3 | Команда 1 | 1 | Данные считаны |
| 5 | Команда 2 (без вызова команды 1) | 2 | Созданы пустые структуры данных |
| 6 | Команда 2 (с вызовом команды 1) | 2 | Созданы необходимые структуры данных |
| 7 | Команда 3 (неверный ввод числа) | А | Invalid input |
| 8 | Команда 3 (добавление в пустые структуры данных) | 1 | Число добавлено |
| 11 | Команда 4 (без вызова команды 1) | 4 | There is not any data |
| 12 | Команда 4 (с вызова команды 1) | 4 | Успешный вывод |

# 

# **Оценка эффективности**

***Поиск элемента***

***Время (в тактах процессора)***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Бинарное дерево | Сбалансированное дерево | Хеш-таблица | Файл |
| 10 | 920 | 840 | 890 | 102270 |
| 50 | 1420 | 1380 | 760 | 121600 |
| 100 | 1480 | 1320 | 1250 | 149920 |
| 500 | 2880 | 2760 | 900 | 313580 |

***Память (в байтах)***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Бинарное дерево | Сбалансированное бинарное дерево | Хеш-таблица | Файл |
| 10 | 240 | 240 | 200 | 26 |
| 50 | 1200 | 1200 | 872 | 172 |
| 100 | 2400 | 2400 | 1640 | 332 |
| 500 | 12000 | 12000 | 8072 | 2154 |

***Количество сравнений***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Бинарное дерево | Сбалансированное бинарное дерево | Хеш-таблица | Файл |
| 10 | 4 | 3 | 2 | 6 |
| 50 | 4 | 4 | 1 | 29 |
| 100 | 9 | 5 | 1 | 36 |
| 500 | 12 | 9 | 1 | 170 |

# **Ответы на контрольные вопросы**

***1.Что такое дерево?***

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Дерево с базовым типом Т определяется рекурсивно либо как пустая структура (пустое дерево), либо как узел типа Т с конечным числом древовидных структур этого же типа, называемых поддеревьями.

***2.Как выделяется память под представление деревьев?***

Способ выделения памяти под деревья определяется способом их представления в программе. C помощью матрицы или списка может быть реализована таблица связей с предками или связный список сыновей. Целесообразно использовать списки для упрощенной работы с данными, когда элементы требуется добавлять и удалять, т. е. выделять память под каждый элемент отдельно. При реализации матрицей память выделяется статически.

***3.Какие стандартные операции возможны над деревьями?***

Основные операции с деревьями: обход дерева, поиск по дереву, включение в дерево, исключение из дерева. Обход вершин дерева можно осуществить следующим образом:

* сверху вниз (префиксный обход)
* слева направо (инфиксный обход)
* снизу вверх (постфиксный обход)

***4.Что такое дерево двоичного поиска?***

Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше.

Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень. С учетом этого свойства поиск узла в двоичном дереве поиска можно осуществить, двигаясь от корня в левое или правое поддерево в зависимости от значения ключа поддерева.

***5.Чем отличается идеально сбалансированное дерево от АВЛ дерева?***

Идеально сбалансированное дерево : при добавлении узлов в дерево мы будем их равномерно располагать слева и справа, то получится дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу.

АВЛ-дерево : двоичное дерево называется сбалансированным, если у каждого узла дерева высота двух поддеревьев отличается не более чем на единицу.

***6.Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?***

Временная сложность поиска элемента в АВЛ дереве – О(log2n)

Временная сложность поиска элемента в дереве двоичного поиска –от О(log2n) до O(n).

***7.Что такое хеш-таблица, каков принцип ее построения?***

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей. Функцию, по которой можно вычислить этот индекс. называется хеш-функцией. Принято считать, что хорошей является такая функция, которая удовлетворяет следующим условиям:

* функция должна быть простой с вычислительной точки зрения;
* функция должна распределять ключи в хеш-таблице наиболее равномерно.
* функция должна минимизировать число коллизий

***8.Что такое коллизии? Каковы методы их устранения.***

Коллизия - ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2.

Первый метод –внешнее(открытое) хеширование (метод цепочек)

В случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется связный список. Таким образом, если для нескольких различных значений ключа возвращается одинаковое значение хеш-функции, то по этому адресу находится указатель на связанный список, который содержит все значения.

Второй метод - внутреннее (закрытое) хеширование (открытая адресация). Оно, состоит в том, чтобы полностью отказаться от ссылок. В этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку (с шагом 1), до тех пор, пока не будет найден ключ K или пустая позиция в таблице.

***9.В каком случае поиск в хеш-таблицах становится неэффективен?***

Поиск в хеш-таблицах становится менее эффективен, если наблюдается большое число коллизий. Тогда вместо ожидаемой сложности О(1) получим сложность O(n).

***10.Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах***

Хеш-таблица - от О(1) до O(n)

АВЛ-дерево - О(log2n)

Дерево двоичного поиска – от О(log2n) до O(n).

# **Вывод**

Из переведенной выше оценки эффективности можно сделать вывод, что лучше всего и по памяти, и по времени работает хеш-таблица. Это связано с тем, что поиск в хэш-таблице сводится, по сути, к вычислению индекса, где находится элемент.

Однако, такой хороший результат невозможен без нахождения хорошей хэш-функции. В том случае, если такую функцию создать сложно, то для поиска элементов лучше использовать деревья. Поиск в файле, как правило, на порядок дольше.

Если говорить о деревьях, то для поиска элементов лучше использовать сбалансированные деревья.