|  |  |
| --- | --- |
| 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 «Программная инженерия»

**ОТЧЁТ**

**«**Работа №6, Деревья, Хеш-таблицы**»**

Выполнил студент: \_\_***Бугаенко Андрей Павлович***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа: \_\_\_\_***ИУ7-35Б***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверила.**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*подпись, дата*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2020 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-таблицах и в файлах. Сравнить эффективность реструктуризации таблицы для устранения коллизий и поиска в ней с эффективностью поиска в исходной таблице.

**Условие задания**

Построить ДДП, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Добавить указанное слово, если его нет в дереве (по желанию пользователя) в исходное и сбалансированное дерево. Сравнить время добавления и объем памяти. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана, используя метод цепочек для устранения коллизий. Вывести построенную таблицу слов на экран. Осуществить добавление введенного слова, вывести таблицу. Сравнить время добавления, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев, хеш-таблиц и файла.

**Техническое задание**

Построить ДДП, сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность заданной операции в ДДП в АВЛ дереве и в хеш-таблице. Вывести на экран деревья и хеш-таблицу. Произвести реструктуризацию хеш-таблицы, если среднее количество сравнений больше указанного. Оценить эффективность использования этих структур (по времени и памяти) для поставленной задачи.

**Исходные данные и результат**:

В качестве входных данных программа получает файл с элементами, в нашем случае это слова. После получения файла пользователь должен выбрать способ работы с данным файлом. При выполнении некоторых команд от программы поступит запрос на ввод дополнительных данных (слово для добавления в дерево/хеш-таблицу или данные для построения хеш-таблицы).  
  
В качестве результата пользователь получает:

1. В случае работы с бинарным деревом поиска:
   1. Построенное по данным из файла ДДП в виде графа.
   2. Построенное по данным из файла АВЛ в виде графа.
   3. ДДП, в которое было добавлено слово, введённое пользователем.
   4. АВЛ, в которое было добавлено слово, введённое пользователем.
   5. Сравнение времени добавления слов в АВЛ и ДДП в виде количества тиков.
2. В случае работы с хеш-таблицей мы можем:
   1. Таблица созданная из файла.
   2. Построенная ранее таблица.
   3. Успешность операции добавления слова в таблицу.

**Описание задачи, реализуемой программой:**

Программа должна на основе файла, название которого было получено от пользователя, создать по выбору пользователя либо хеш-таблицу, либо ДДП. Также программа должна предоставить возможность проводить с этими структурами данных такие операции как:

* Вывод деревьев на экран.
* Создание ДДП на основе файла.
* Создание АВЛ на основе файла.
* Добавление слова в АВЛ и ДДП.
* Сравнение добавления слова в АВЛ и ДДП.
* Создание хеш-таблицы на основе данных из файла
* Добавление слова в хеш-таблицу
* Расчёт эффективности структур данных при добавлении слова

**Способ обращения к программе:**

Для работы с программой пользователь должен использовать консольный клавиатурный ввод. Для запуска программы пользователь использует команду ./app.exe. После этого пользователь попадёт в меню, в котором каждое действие определено как цифра, при вводе которой это действие происходит. Также для выполнения некоторых операций программе могут потребоваться дополнительные данные, о необходимости ввода которых она уведомит пользователя. Данные вводятся в соответствии с запросом программы и если данные введены неправильно, то программа выведет сообщение об ошибке.

**Описание возможных аварийных ситуаций и ошибок пользователя:**

В случае ошибки во вводе программа прекратит выполнение команды и выведет описание ошибки, которая произошла.

Типы ошибок ввода:

* неправильно была введена команда - в этом случае программа уведомит пользователя о том, что была совершена ошибка при попытке вызова команды.
* неправильный ввод дополнительных данных - введённые данные не соответствуют формату ввода, либо их использование в программе не представляется возможным в силу их свойств.

Типы ошибок работы со стеком:

* попытка создать хеш-таблицу или ДДП на основе пустого файла.
* попытка совершать действия с ещё не созданными хеш-таблицей или структурой данных.

**Описание внутренних структур данных:**

В данной программе было реализовано несколько структур данных для представления двоичного дерева поиска и хеш-таблицы.

Для экономии памяти хеш-таблица была реализована в виде динамического массива структур вида:

typedef struct hash\_linked hash\_linked\_t;

struct hash\_linked

{

    char \*key;

    hash\_linked\_t \*next;

};

где:  
key - поле для записи данных  
next - указатель на следующий узел  
table[size] - вид хеш-таблицы  
В данной реализации таблицы хеш-код - индекс элемента массива.  
  
АВЛ было реализовано следующей структурой:

typedef struct tree\_node tree\_node\_t;

struct tree\_node

{

    char \*key;

    long height;

    tree\_node\_t \*left;

    tree\_node\_t \*right;

};

где:  
key - поле для записи данных  
height - высота узла  
left - левый потомок  
right - правый потомок

ДДП реализовано следующей структурой:

typedef struct tree\_uns\_node tree\_uns\_node\_t;

struct tree\_uns\_node

{

    char \*key;

    tree\_uns\_node\_t \*left;

    tree\_uns\_node\_t \*right;

};

где:  
key - поле для записи данных  
left - левый потомок  
right - правый потомок

**Описание алгоритма:**

Для хеш-таблицы было использовано два алгоритма, так как было использовано открытое хеширование, то элементы, у которых значение хеш-кода одинаковое, попадают в одну и ту же ячейку хеш-таблицы. Количество коллизий для ячейки хеш-таблицы считается как количество элементов списка значений в этой ячейке. В процессе хеширования использовалась хеш-функция, определяющая код на основе суммы кодов букв строки и для реструктуризации хеш-функция на основе бинарных сдвигов.  
  
При построении сбалансированного ДДП происходят повороты вокруг тех или иных узлов дерева, если обнаружена разбалансировка. Вставка осуществляется рекурсивно, при этом сравниваются ключи дерева и перемещение идёт по левой или правой ветви соответственно.

**Набор тестов:**

|  |  |  |
| --- | --- | --- |
| Наименование теста | Входные данные | Выходные данные |
| Некорректное имя файла | 1 > к | Файл не найден. Проверьте имя файла и попробуйте еще раз. |
| Пустой файл | 1 > \пустой файл\ | Файл пуст. Проверьте содержимое файла и попробуйте еще раз. |
| Неверно введенное число коллизий | 1 > f | Число введено неверно. |
| Работа с непостроенной хеш- таблицей | 2 > 2 | Сначала хеш - таблицу нужно построить. |
| Работа с несбалансированным деревом | 1 > 1 > 4 | Сначала дерево нужно сбалансировать. |
| Работа с непостроенным деревом | 1 > 4 | Сначала дерево нужно построить. |
| Построение дерева | 1 > 1 | Вывод дерева в виде графа |
| Балансировка дерева | 1 > 1 > 2 | Вывод сбалансированного дерева в виде графа |
| Вывод хеш-таблицы | 2 >2 | Вывод хеш-таблицы |
| Добавление слова в хеш- таблицу | 2>3 | Успешно. |

**Анализ скорости обработки данных:**

Частные результаты работы программы:

Время добавления в файл: 211755

Время добавления в АВЛ: 3234

Время добавления в ДДП: 1194

Время добавления в хеш-таблицу: 519

Количество сравнений АВЛ: 3

Количество сравнений ДДП: 3

Количество сравнений хеш-таблицы: 2

Память АВЛ: 544

Память ДДП: 408

Память хеш-таблицы: 272

Скорость добавления: быстрее всего работает хеш-таблица, затем идёт ДДП, затем АВЛ.

Расход памяти: меньше всего памяти расходует хеш-таблица, затем идёт ДДП и АВЛ.

**Вывод:**Хеш-таблица всегда выигрывает по времени у ДДП и АВЛ, так как доступ к элементам имеет не рекурсивный характер, однако хеш-таблицы не всегда эффективны по памяти, поскольку каждый хеш-код требует выделение памяти, и в случае неточно подобранной хеш-функции распределение будет неплотное. При сравнении работы деревьев, ДДП выигрывает по времени вставки нового элемента у АВЛ, так как при вставке элемента в АВЛ может возникнуть необходимость перебалансировки дерева.

**Контрольные вопросы:**

1. Что такое дерево?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим»

2. Как выделяется память под представление деревьев?

Память выделяется динамически аналогично связному списку - под каждый элемент.

3. Какие стандартные операции возможны над деревьями?

Добавление узла, вставка узла, удаление узла, поиск узла.

4. Что такое дерево двоичного поиска?

Дерево, левый потомок каждого узла которого больше или равен родителю, а правый – строго меньше.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

У АВЛ - дерева разница глубины узлов не более 1, а у идеально сбалансированного количество вершин в каждом поддереве различается не более чем на 1.

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

Поиск в АВЛ дереве быстрее, чем в дереве двоичного поиска.

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

Таблица вида «ключ–хеш код». Такая таблица возвращает один и тот же код для одного значения, но не всегда возвращает разные коды для разных значений. Коды рассчитываются на основе хеш-функции, которая позволяет определить хеш-код для элемента.

8. Что такое коллизии? Каковы методы их устранения.

Коллизия – это совпадение хеш–кодов. Из принципов устранения выделяют открытое и закрытое хеширование.

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

При большом количестве коллизий эффективность поиска сильно снижается.

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

Таблица : O(1), в худшем случае O(n).   
АВЛ – O(log2n).   
ДДП – O(n), где n – высота дерева.