



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»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
«ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ»
по курсу «Типы и структуры данных»

Студент: Чепиго Дарья Станиславовна

Группа: ИУ7-34Б

Студент

подпись, дата

Чепиго Д.С.

фамилия, и.о.

Преподаватель

подпись, дата

фамилия, и.о.

Оценка _____

Условие задачи

Вариант 5

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

Входные данные

- Целое число от 0 до 7 – пункт меню
- Числа и строки, требуемые в меню программы

Для сборки программы существует makefile. Программа «app.exe» запускается через консоль, после чего можно увидеть меню программы, состоящее из 8 пунктов

Пункты меню:

- 1 - считать дерево из файла
- 2 - создать сбалансированное дерево
- 3 - вывести исходное дерево
- 4 - вывести сбалансированное дерево

Работа с хэш-таблицой:

- 5 - считать хэш-таблицу из файла
- 6 - вывести хэш-таблицу на экран

7 - добавить слово в деревья и в таблицу

0 - выход из программы

Выходные данные:

В соответствии от действий пользователя может быть следующее:

- вывод деревьев в png-формате
- вывод хэш-таблицы на экран
- вывод измерений времени и количества сравнений при добавлении нового элемента в структуры данных
- информация об ошибке

Описание алгоритма

1. Пользователь выбирает пункт меню
2. Пользователь обязан ввести максимальный размер структур и текущий размер структур
3. Генерируется файл со словами, количество которых ввел пользователь пунктом ранее(текущий размер структур)
4. В зависимости от пункта меню пользователь может создать и вывести дерево, сбалансированное дерево и хэш-таблицу
5. Для того, чтобы добавить элемент в структуры они все должны существовать

6. Если пользователь захочет завершить программу, то для этого есть отдельный пункт меню.

Исходное состояние — пустое дерево, все используемые структуры пусты. Чтение элементов из файла продолжается до конца файла.

При добавлении элемента в ДДП и АВЛ происходит сравнение добавляемого элемента со встречающимися ему на пути при обходе. Если такое слово встретилось, то оно не добавляется ещё раз в структуры. АВЛ-дереву также необходима перебалансировка, которая осуществляется правым или левым поворотом (в зависимости от степени сбалансированности). При добавлении элемента в хеш-таблицу осуществляется вычисление значения хеш-функции. Так как мы работаем со строками, хеш-функцией был выбран метод исключающего или (или же XOR). В случае возникновения коллизий, они устраняются методом цепочек — добавлением в список текущего элемента.

Аварийные ситуации:

- Выбор несуществующего пункта меню
- Переполнение деревьев или таблицы
- Ошибка выделения памяти
- Добавление слова, которое уже есть в структурах данных

Описание структур данных

Структура для реализации деревьев:

```
struct tree_node
{
    const char *name;           // значение узла - слово
    unsigned char height;       // высота в дереве
    struct tree_node *left;     // указатель на меньшие узлы
    struct tree_node *right;    // указатель на большие узлы
};
```

Структура для реализации хэш-таблицы:

```
struct hash_table
{
    int max_size; //максимальное количество элементов хэш-таблицы

    node_table_t *array; //указатель на массив элементовтаблицы
};
```

Структура для реализации элементов хэш-таблицы:

```
typedef struct node_table node_table_t;

struct node_table
{
    char *name;           // значение элемента - слово
    node_table_t *next;   // указатель на следующий элемент
};
```

Сравнение эффективности

Время — в тактах, Память — в байтах

Берётся среднее из 5 измерений. После добавления элемента структура становится полностью заполненной.

Размер структуры после добавления	ДДП	АВЛ	Хэш-таблица	Файл
10	7	7	9	12
100	9	8	10	10
150	9	8	9	11
250	8	7	8	9

Исходя из этих замеров нельзя сделать однозначный вывод кроме того, что запись в файл самая медленная, а хэш-таблицы немного проигрывают деревьям.

Измерим память в байтах:

Размер структуры после добавления	ДДП	АВЛ	Хэш-таблица	Файл
10	320	320	176	76
100	3200	3200	1616	702
150	4800	4800	2480	1000
250	8000	8000	4064	1700

Несложно заметить, что хоть где-то выигрывает файл! На втором месте идет хэш-таблица, ибо там мы храним только размер и списки. А после идут деревья, а так как у меня для них одинаковая структура данных, то их память совпадает(но при этом, в ДДП не используется высота, поэтому если бы я сэкономила память, то ДДП выигрывала бы по памяти без этого поля).

Изучим количество сравнений.

Размер структуры после добавления	ДДП	АВЛ	Хэш-таблица	Файл
10	4	4	1	9
100	8	7	0	99
150	7	6	1	149
250	7	9	1	249

В процессе измерения времени и количества сравнений можно заметить, что эти параметры для деревьев зависят от слова. Ибо если мы добавляем слово на букву У или А, то это означает, что мы вставляем в самые дальние позиции дерева, следовательно получаем большее время и количество сравнений. Легко заметить, что при добавлении слова в файл мы пробегаемся по всему файлу, ибо вставляем слово в конец, а значит получаем наибольшее количество сравнений. При добавлении в хэш-таблицу мы получаем наименьшее количество сравнений, ибо хэш-функция зависит от размера таблицы и коллизии возникают не так часто. Также видно, что количество сравнений в АВЛ дереве меньше, чем в ДДП.

Тестирование

Позитивные тесты.

Входные данные	Действия программы	Выходные данные
Пункт 1 <i>Ввод максимального количества элементов и текущего</i>	Корректная работа программы Создание ДДП	Ожидание следующего ключа
Пункт 2	Корректная работа программы	Ожидание следующего ключа

	Создание на основе ДДП АВЛ дерева	
Пункт 3	Корректная работа программы Вывод на экран png изображения ДДП	Ожидание следующего ключа
Пункт 4	Корректная работа программы Вывод на экран png изображения АВЛ- дерева	Ожидание следующего ключа
Пункт 5	Корректная работа программы Вывод на экран png изображения АВЛ- дерева	Ожидание следующего ключа
Пункт 6	Корректная работа программы Вывод на экран хэш- таблицы	Ожидание следующего ключа
Пункт 7 Ввод валидного слова	Корректная работа программы Добавление слова во все структуры данных Вывод информации о добавлении	Ожидание следующего ключа

Негативные тесты

Входные данные	Действия программы	Выходные данные
Ввод числа 10	Информация о неверном	Завершение программы

	ключе меню	
Пункт 1 ввод чисел 80, 90 251 -14	Информация о неверном количестве элементов в структурах	Завершение программы
Пункт 7, добавление уже существующего слова	Информация, что данное слово уже есть в структурах данных	Ожидание нового ключа

Контрольные вопросы

1. Что такое дерево?

Дерево — нелинейная структура данных, используемая при представлении иерархических связей, имеющих отношение «один ко многим». Также деревом называется совокупность элементов, называемых узлами или вершинами, и отношений («родительских») между ними, образующих иерархическую структуру узлов.

2. Как выделяется память под представление деревьев?

Деревья могут представляться как списком, так и массивом. При реализации списком соответственно память выделяется под каждый элемент (для значения, правого и левого потомков), при реализации массивом выделяется с запасом фиксированная длина. При этом размер массива выбирается исходя из максимально возможного количества уровней двоичного дерева, и чем менее полным является дерево, тем менее рационально используется память.

3. Какие стандартные операции возможны над деревьями

Поиск, добавление, удаление, обход дерева, балансировка.

4. Что такое дерево двоичного поиска?

Двоичным деревом поиска называют дерево, все вершины которого упорядочены, каждая вершина имеет не более двух потомков (назовём их левым и правым), и все вершины, кроме корня, имеют родителя.

5. Чем отличается идеально сбалансированное дерево от AVL дерева?

Критерий для AVL-дерева: дерево называется сбалансированным тогда и только тогда, когда высоты двух поддеревьев каждой из его вершин отличаются не более чем на единицу.

В случае идеально сбалансированного дерева не более чем на единицу должно отличаться число вершин в левом и правом поддеревьях.

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

Так как высоты поддеревьев AVL-дерева отличаются не более чем на 1, количество сравнений в таком дереве заметно уменьшается. ДДП же в худшем случае (дерево представляет из себя линейный список) имеет количество сравнений равное количеству элементов, что значительно замедляет процесс поиска.

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

Хеш-таблица — массив, заполненный в порядке, определенным хеш-функцией, т.е. это структура данных вида «ассоциативный массив», которая ассоциирует ключи со значениями.

Для каждого исходного элемента вычисляется значение хеш-функции, в соответствии с которым элемент записывается в определенную ячейку хеш-таблицы.

8. Что такое коллизии? Каковы методы их устранения.

Коллизия — совпадение хеш-адресов для разных ключей.

Устранение коллизий можно производить методом цепочек (также открытое хеширование). Его суть заключается в том, что каждая ячейка хеш-таблицы представляет собой связный список, содержащий все элементы, значение хеш-функции которых совпадает с текущим.

Также можно бороться с коллизиями методом закрытого хеширования. Хеш-таблица в данном случае представляет из себя обычный массив, который заполняется по такому принципу: если ячейка со значением хеш-функции свободна, туда записывается элемент, иначе проверяется другая ячейка. При этом адресация может быть линейной, квадратичной или произвольной.

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

Хеш-таблицы перестают быть эффективными при большом количестве коллизий.

В таком случае количество сравнений будет значительно расти, независимо от способа их разрешения.

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

Нетрудно заметить из таблиц выше, что поиск в AVL-дереве намного эффективнее по времени, чем поиск в ДДП. Однако даже AVL проигрывает хеш-таблице в случае малого количества коллизий, так как в идеале количество сравнений в хеше будет равно 1. При большом количестве коллизий хеш-таблица может стать даже хуже ДДП.