Министерство образования и науки 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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
«РАБОТА С ДЕРЕВЬЯМИ»

Студент, группа **Кононенко С., ИУ7-33Б**

2019 г.

Описание условия задачи

Построить дерево двоичного поиска, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из слов текстового файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного слова 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.

Техническое задание

Входные данные:

1. **Имя файла с деревом:** строка, содержащая имя файла.
2. **Максимальное значение допустимых коллизий:** целое число.
3. **Слово для поиска:** строка, содержащая слово, которое планируется найти в дереве.

Выходные данные:

1. Псевдографическое изображение бинарного дерева.
2. Псевдографическое изображение сбалансированного бинарного дерева.
3. Хеш-таблица вершин дерева.
4. Количественная характеристика выполнения моделирования (время поиска вершины в различных представлениях дерева, количество занимаемой деревом памяти в различных представлениях, количество сравнений для достижения результата (поиска) в различных представлениях, среднее количество сравнений (при поиске) в различных представлениях).

Функция программы: программа выполняет ряд функций в порядке:

1. Вывод бинарного дерева.
2. Вывод сбалансированного бинарного дерева.
3. Вывод хеш-таблицы.
4. Вывод количественной характеристики моделирования, указанного в секции выходных данных.

Обращение к программе: запускается из терминала с аргументом в виде файла, содержащего значения вершин дерева.

Аварийные ситуации:

1. Некорректный ввод имени файла.
   На входе: имя файла, несуществующего в системе.
   На выходе: сообщение «Неверное имя файла! Повторите попытку.»
2. Пустой файл в качестве аргумента программы.
   На входе: пустой файл.
   На выходе: сообщение «Файл пуст! Проверьте файл.»
3. Некорректный ввод максимального числа коллизий.
   На входе: буква или, любой другой нечисловой символ или отрицательное число.
   На выходе: сообщение «Введено недопустимое значение! Повторите попытку.»

Структуры данных

Реализация линейного односвязного списка:

```
typedef struct list
{
   char *value;
   struct list *next;
} list_t;
```

Поля структуры:

- ***char* *value –** указатель на массив символов;
- ***struct list* *next –** указатели на следующий элемент списка.

Реализация листа дерева:

```
typedef struct tree_node
{
   char *value;
   struct tree_node *left;
   struct tree_node *right;
} tree_node;
```

Поля структуры:

- ***char* *value –** указатель на массив символов;
- ***struct tree_node* *left –** указатель на левого потомка;
- ***struct tree_node* *right –** указатель на правого потомка.

Реализация динамического массива (для реализации хеш-таблицы):

```
typedef struct
{
   tree_node **arr;
   int size;
   int mem_size;
} dynarr_t;
```

Поля структуры:

- ***tree_node* **arr –** указатель на массив указателей листов дерева;
- ***int* size –** фактическая ёмкость массива;
- ***int* mem_size –** аллоцированная ёмкость массива.

Алгоритм

Сначала происходит балансировка дерева. Исходное дерево вытягивается в отсортированный односвязный список, затем происходил рекурсивное построение АВЛ-дерева. После постройки дерева строится хеш-таблица с помощью примитивной функции (DJB2 хеширование) по дереву. В случае, если количество коллизий превосходит допустимое, таблица самоперестраивается на основе улучшенной хеш-функции (Jenkins-At-One-Time хеширование) и выводится количественная характеристика моделирования.

Тесты

	Тест	Пользовательский ввод	Результат
1	Некорректный ввод имени файла	data/tree07.txt (файл не существует)	Неверное имя файла! Повторите попытку.
2	Пустой файл	data/tree06.txt (пустой файл)	Файл пуст! Проверьте файл.
3	Некорректный ввод максимального количества коллизий	A	Введено недопустимое значение! Повторите поппытку
4	Некорректный ввод максимального количества коллизий	0	Введено недопустимое значение! Повторите поппытку
5	Некорректный ввод максимального количества коллизий	-1	Введено недопустимое значение! Повторите поппытку
6	Ввод несуществующего слова	Hi (слова нет в дереве)	Слово “Hi” не найдено.
7	Ввод количества коллизий, большего, чем текущее максимальное	3 (при максимальном 2)	Результат достижим за введенное количество коллизий. Пересоздание таблицы не требуется.
8	Ввод количества коллизий, меньшего, чем текущее максимальное	1 (при максимальном 2)	Пересоздание хеш-таблицы
9	Корректный ввод всех характеристик	Корректный файл, корректный ввод числа коллизий	Количественная характеристика моделирования

Оценка эффективности

Измерения эффективности реализаций решения задачи на дереве будут производиться в единицах измерения – тактах процессорв. Для измерения была специально написана ассемблерная функция, поэтому погрешность измерений минимальна.

Поиск слова (в тактах процессора)*:

Количество элементов дерева	**Бинарное дерево**	**Сбалансированное бинарное дерево**	**Хеш-таблица****	**Файл**
16	1140	976	636	7841
32	1563	923	720	11362
50	1767	1351	785	21022
500	1989	1610	842	43696
1000	2017	1724	966	86655

*в таблице указаны средние значения для поиска по всей структуре данных

****указанные результаты справедливы для хеш-таблиц без коллизий

Объём занимаемой памяти (в байтах):

Количество элементов дерева	**Бинарное дерево**	**Сбалансированное бинарное дерево**	**Хеш-таблица (без коллизий/80 % коллизий от исходного числа)**	**Файл**
16	384	384	1432 / 488	119
32	768	768	2104 / 824	222
50	1200	1200	7384 / 1896	379
500	12000	12000	241944 / 11160	3737
1000	24000	24000	858232 / 22120	7624

Контрольные вопросы

1. Что такое дерево?

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

2. Как выделяется память под представление деревьев?

В виде связного списка — динамически под каждый узел.

3. Какие стандартные операции возможны над деревьями?

Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

4. Что такое дерево двоичного поиска?

Двоичное дерево поиска - двоичное дерево, для каждого узла которого сохраняется условие:
левый потомок больше или равен родителю, правый потомок строго меньше родителя (либо наоборот).

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

У АВЛ дерева для каждой его вершины высота двух её поддеревьев различается не более чем на 1, а у идеально сбалансированного дерева различается количество вершин в каждом поддереве не более чем на 1.

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

Поиск в АВЛ дереве происходит быстрее, чем в ДДП.

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

Хеш-таблицей называется массив, заполненный элементами в порядке, определяемом хеш-функцией. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс. Функция должна быть простой для вычисления,распределять ключи в таблице равномерно и давать минимум коллизий.

8. Что такое коллизии? Каковы методы их устранения?

Коллизия – ситуация, когда разным ключам хеш-функция ставит в соответствие один и тот же индекс. Основные методы устранения коллизий:

открытое и закрытое хеширование. При открытом хешировании к ячейке по данному ключу прибавляется связанны список, при закрытом – новый элемент кладется в ближайшую свободную ячейку после данной.

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

Поиск в хеш-таблице становится неэффективен при большом числе коллизий –сложность поиска возрастает по сравнению с O(1). В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции.

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

В хеш-таблице минимальное время поиска O(1). В АВЛ: O(log2n). В дереве двоичного поиска O(h), где h - высота дерева (от log2n до n).

Вывод

Использование хеш-таблицы всегда эффективно по времени, но не всегда эффективно по памяти (в случае хорошей дистрибуции функции распределение будет не плотным), так как требует выделенной памяти под каждый хеш (если отсутствуют коллизии, хорошая дистрибуция). В случае деревьев, АВЛ дерево не всегда выигрывает по времени поиска у несбалансированного дерева, так как порядок вершин при балансировке меняется, но всегда выигрывает по среднему значению количества сравнений и среднему времени поиска по дереву.