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 Хамзина Регина Ренатовна
фамилия, имя, отчество

Студент, группа ИУ7-33Б Хамзина Р.Р.

2020 г.

Описание условия задачи

Построить ДДП, сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность поиска в ДДП в АВЛ дереве и 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.

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. 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. *Целое число* — номер пункта меню, который вызывает описанное в пункте действие. [0… 5]
2. *Строка* — слово для поиска в дереве, хеш-таблице и файле. Максимальная длина строки равна 20.

Выходные данные:

1. *Изображение дерева слов*
2. *Хеш-таблица вершин дерева слов*
3. *Временная характеристика* — время работы функций.
4. *Объемная характеристика* — объем структуры данных.

Функция программы:

Работа с деревом слов — создание и вывод дерева двоичного поиска, создание и вывод сбалансированного дерева, построение, вывод и реструктуризация (при

необходимости) хеш-таблицы. Поиск слова в двоичном дереве поиска слов, в сбалансированном дереве слов, в хеш-таблице и в файле.

Обращение к программе:

Программа запускается из терминала в директории с проектом при помощи команды «./app.exe name.txt», где name.txt — имя текстового файла слов для создания дерева слов и работы с ним.

Аварийные ситуации

1. Некорректный ввод номера пункта меню
Входные данные : не целое число или целое число, большее 5, или целое число, меньшее 0.
Выходные данные : сообщение «Команда введена неверно».

2. Неверное имя файла.
Входные данные : имя несуществующего файла.
Выходные данные : сообщение «Ошибка открытия файла».

3. Пустой файл
Входные данные : пустой файл.
Выходные данные : сообщение «Файл пуст».

4. Неверное число допустимых сравнений при поиске слова
Входные данные : не цифра или число, меньшее 1.
Выходные данные : сообщение «Неверное число допустимых сравнений».

Внутренняя структура данных

Структура для хранения дерева:

```
typedef struct tree_t
{
   char *word;
   int height;
   struct tree_t *left;
   struct tree_t *right;
} tree_t;
```

Её поля:

*char *word* — значение текущей вершины;

int height — высота вершины относительно других вершин;
*struct tree_t *left* — указатель на левого потомка;
*struct tree_t *right* — указатель на правого потомка;

Структуры для хранения хеш-таблицы:

```
typedef struct hash_table_t
{
   int count;
   hash_t **array;
} hash_table_t;
```

Её поля:

int count — размер хеш-таблицы;
*hash_t **array* — массив указателей на список — структуру данных, описанную следующим образом:

```
typedef struct hash_t
{
   char word[MAGIC_SIZE];
   struct hash_t *next;
} hash_t;
```

Её поля:

char word[MAGIC_SIZE] — значение текущего элемента списка *MAGIC_SIZE* = 20;
*struct hash_t *next* — указатель на следующий элемент списка;

Алгоритм

Дерево двоичного поиска:

Искомое слово сравнивается со словом, находящимся в текущей вершине. Если они совпадают — поиск завершен, если искомое слово меньше — поиск продолжается в левом поддереве вершины, иначе — в правом.

Сбалансированное дерево:

Алгоритм аналогичен ДДП.

Хеш-таблица:

Для каждого элемента таблицы: определяется хеш-значение, по хеш-значению добавляется в односвязный список (метод цепочек устранения коллизий).

Хеш-значение по слову S определяется так:
h(S) = S[0] + S[1] * P + S[2] * P^2 + S[3] * P^3 + ... + S[N] * P^N, где P — изначальное число вершин (полиномиальная хеш-функция), N — число символов в строке.

При поиске в хеш-таблице считается число сравнений для элемента. Если оно превышает введенное пользователем, происходит реструктуризация хеш-таблицы.

Реструктуризация хеш-таблицы: число P увеличивается в 2 раза и для нового размера таблицы для всех слов получают новые хеш-значения и заполняют таблицу.

Функции программы

Для работы с деревом двоичного поиска:

*int create_tree(tree_t **root, FILE *f_in, int *count)* — создание дерева;

*tree_t *create_node(char *word, int h)* — создание узла;

*void free_tree(tree_t *tree)* — освобождение памяти для дерева;

*void print_tree(tree_t *tree, int place)* — печать дерева;

*int find(tree_t *tree, char *word, int *count_cmprs)* — поиск слова в дереве;

Для работы со сбалансированным деревом:

*int create_balance(tree_t **root, FILE *f_in)* - создание дерева;

*int balance_find(tree_t *tree, char *word, int *count_cmprs)* — поиск слова в дереве;

Для работы с хеш-таблицей:

*int get_hash(char *word, const int size)* — получение хеша;

*int hash_table_init(hash_table_t *table, const int table_size)* — инициализация хеш-таблицы;

*int create_hash_table(FILE *file, hash_table_t *result)* — создание хеш-таблицы;

*void print_table(hash_table_t *table)* — печать таблицы;

*void free_table(hash_table_t *table)* — освобождение таблицы;

*int restructuring(FILE *file, hash_table_t *table)* — реструктуризация таблицы;

*hash_t *hash_find_in_table(hash_table_t *hash_table, char *word, int collision, int *code, int *cmprs)* — поиск слова в таблице;

Для работы с файлом:

*int search_file(char *to_find, FILE *f, int *cmprs)* — поиск слова в файле;

Тесты

	Тест	Ввод	Вывод
1	Неверный пункт меню: больше 5	6	Команда введена неверно
2	Неверный пункт меню: меньше 0	-1	Команда введена неверно
3	Неверный пункт меню: не целое число	a	Команда введена неверно
4	Неверное имя файла	Несуществующее имя файла	Ошибка открытия файла
5	Пустой файл	Пустой файл	Файл пустой
6	Неверное число допустимых сравнений: буква	5, a	Неверное число допустимых сравнений
7	Неверное число допустимых сравнений: меньше 1	5, 0	Неверное число допустимых сравнений
8	Любое действие меню 2-5, если не загружены данные	2-5	Дерево не загружено из файла
9	Вывод ДДП	2	ДДП

10	Вывод сбалансированного дерева	3	Сбалансированное дерево
11	Вывод хеш-таблицы	4	Хеш-таблица
12	Поиск	5	Временная и количественная характеристика

Примеры работы программы

```
Введите слово для поиска: product

Введите допустимое число сравнений в хеш-таблице: 3


Поиск в дереве двоичного поиска

Слово product найдено за 4784 тактов процессора
Дерево двоичного поиска занимает - 3200 байт
Число сравнений до искомого слова - 8


Поиск в сбалансированном дереве

Слово product найдено за 2990 тактов процессора
Сбалансированное дерево занимает - 3200 байт
Число сравнений до искомого слова - 2


Поиск в таблице

Слово product найдено за 2028 тактов процессора
Хеш-таблица занимает - 1916 байт
Число сравнений до искомого слова - 3


Поиск в файле

Слово product найдено за 16562 тактов процессора
Файл занимает - 619 байт
Число сравнений до искомого слова - 29
```

Оценка эффективности

Время поиска элемента (тики) — средние значения:

Число элементов	ДДП	Сбалансированное дерево	Хеш-таблица	Файл
10	780	608	161	2763
100	1106	905	164	12549
500	1232	912	173	49753
1000	1913	1139	197	77071

Объем памяти (байты):

Число элементов	ДДП	Сбалансированное дерево	Хеш-таблица	Файл
10	320	320	208	60
100	3200	3200	1916	619
500	16000	16000	9112	3240
1000	32000	32000	17820	6522

Ответы на 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?

Дерево реализуется при помощи односвязного списка, поэтому память выделяется для каждого узла отдельно.

3. Какие стандартные операции возможны над деревьями?

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

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка (меньше, либо равны), а все правые – старше (больше, либо равны). Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень.

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

У идеально сбалансированного дерева число вершин в левом и правом поддеревьях отличается не более, чем на единицу. У каждого узла АВЛ-дерева высота двух поддеревьев отличается не более, чем на единицу.

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

Поиск в АВЛ-дереве происходит быстрее, чем поиск в дереве двоичного поиска и с меньшим числом сравнений.

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

Хеш-таблица - массив, заполненный в порядке, определенным хеш-функцией.

Принцип построения: хеш-функция ставит в соответствие каждому ключу k_i индекс ячейки j, где расположен элемент с этим ключом. Таким образом:

$h(k_i) = j$, если $j=(1, m)$, где j принадлежит множеству от 1 до m, а m. – размерность массива.

8. Что такое коллизии? Каковы методы их устранения?

Коллизии - ситуации, когда разным ключам соответствует одно значение хеш-функции, то есть, когда $h(K1)=h(K2)$, в то время как $K1 \neq K2$.

Методы устранения:

1) 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.

2) Внутреннее (закрытое) хеширование (открытая адресация). 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. При этом, если индекс следующего просматриваемого элемента определяется добавлением какого-то постоянного шага (от 1 до n), то данный способ разрешения коллизий называется линейной адресацией. Для вычисления шага можно также применить формулу: $h = h + a^2$, где a – это номер попытки поиска ключа. Этот вид адресации называется квадратичной или произвольной адресацией.

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

Если для поиска элемента необходимо более 3–4 сравнений, то эффективность использования хеш-таблицы пропадает.

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

АВЛ-деревья : $O(\log_2(n))$

ДДП : $O(\log_2(n))$ - $O(n)$

Хеш-таблица: $O(1)$

Выводы

При сравнении поиска слова в четырех структурах данных (дерево двоичного поиска, сбалансированное, хеш-таблица и файл) получили следующие результаты.

Самой эффективной структурой данных по времени обработки является хеш-таблица (в 7 раз быстрее, чем ДДП и в 5 раз быстрее, чем сбалансированное). Выигрыш по времени объясняется числом сравнений при поиске (при отсутствии коллизий количество сравнений при поиске слова равно 1). Объем памяти хеш-таблицы меньше в 2 раза, чем объем памяти, выделенной под деревья.

Самой эффективной структурой данных по занимаемому объему памяти является файл (в 5 раз меньше, чем деревья и в 3 раза меньше, чем хеш-таблица). При этом файл является самой неэффективной структурой данных по времени, так как все слова в файле идут последовательно и необходимо пройти все слова, лежащие до искомого.

Сравнение двух реализаций деревьев (ДДП и сбалансированного) показало, что поиск слова в сбалансированном дереве происходит быстрее в 1.5 раза, чем в ДДП, что объясняется меньшей высотой сбалансированного дерева. Объем занимаемой памяти двух реализаций одинаков.