Лабораторная работа №6.

Обработка деревьев и хэш-таблиц.

Работу выполнил: Назиров Илхомджон

группа ИУ7-35Б

МГТУ им. Баумана
2021

Условие задачи

Получить навыки применения двоичных деревьев. Построить и обработать хеш-таблицы, сравнить эффективность работы в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Произвести реструктуризацию хеш-таблицы, если среднее количество сравнений больше указанного. Оценить эффективность поиска в этих СД при различном количестве коллизий

Техническое задание

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 в исходное и сбалансированное дерево. 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.

Входные данные

Имя текстового файла, текстовый файл со словами, добавляемые элементы в дерево и хеш-таблицу, размер хеш-таблицы, максимальное среднее количество сравнений для поиска в хеш-таблице.

Выходные данные

Состояние ДДП, сбалансированного ДДП, хеш-таблицы, результаты обработки этих СД.

Возможные аварийные ситуации

Некорректный ввод размера хеш-таблицы или максимального количества сравнений.

Структуры данных

Структура дерева

```
typedef struct tree_el
{
	st_node *ptr;
	int height;
	int size;
}
```

root — указатель на корень дерева.

size — кол-во вершин в дереве.

height — высота дерева.

Структура вершины дерева

```
typedef struct node_el
{
        node_el *right;
        node_el *left;
        char *key;
        int height;
} st_node;
```

left — указатель на левую вершину.

right — указатель на правую вершиу.

key — содержимое вершины.

height — высота вершины.

Структура хеш-таблицы

```
typedef struct hash_element
{
        ash_element *ptr;
        int size;
        int total_compare;
} st_hash_table;
```

ptr — указатель на первый элемент массива, содержащего элементы хеш-таблицы.

size — размер этого массива.

total_compare = кол сравнений

Структура элемента хеш-таблицы

```
typedef struct node
{
   char *string;
   struct node *next;
} node_el;
```

string — содержимое ячейки хеш-таблицы

next — указатель на следующий элемент списка, в случае если в данной ячейке находится

несколько значений

Алгоритм

Алгоритм балансировки ДДП

Сначала исходное дерево «вытягивается» в отсортированный односвязный список, далее из этого списка рекурсивного строится АВЛ-дерево.

Алгоритм работы хеш-функции

Алгоритм №1. Хеш-функция подсчитывает сумму кодов символов исходной строки, и делит их на размер массива. Остаток от деления и будет ключом.

Алгоритм №2. Каждый код символа последовательно умножается на P^n, где P — размерность алфавита, а n — текущий номер символа в строке. Далее, эта сумма делится на размер массива.

Тестирование

Время

Добавление элементов

Входные данные

1. Количество элементов: 12

ДДП	Сбалансированое ДДП	Хеш-таблица	Файл
12000 ~ тиков	38000 ~ тиков	1200 ~ тиков	20000 ~ тиков

Формирование АВЛ-дерева

Количество элементов	Список
10	35000 тиков
50	81000 тиков
100	120000 тиков

Поиск

Среднее количество сравнений

Входные данные:

1. Размер хеш-таблицы: 7

Количество элементов	ДДП	АВЛ-дерево	Хеш-таблица (первый алгоритм хеширования)	Хеш-таблица (второй алгоритм хеширования)

10	3.1 сравнения	2.6 сравнения	2.6 сравнения	2.1 сравнения
50	6.4 сравнения	4.7 сравнения	33 сравнения	32 сравнения
100	19.1 сравнения	6.1 сравнения	220.1 сравнения	236.17 сравнения

Память

Занимаемая память

Количество элементов	Дерево	Хеш-таблица
10	320 байт	160 байт
100	3200 байт	1600 байт
1000	32000 байт	16000 байт

Контрольные вопросы

Что такое дерево?

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

Как выделяется память под представление деревьев?

В виде связного списка — динамически под каждый узел

Какие стандартные операции возможны над деревьями?

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

Что такое дерево двоичного поиска?

Двоичное дерево, для каждого узла которого сохраняется условие:
Левый потомок больше или равен родителю, правый потомок строго меньше родителя (либо наоборот)

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.

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

Поиск в АВЛ дереве происходит быстрее, чем в ДДП.

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

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.

В каком случае поиск в хеш-таблицах становится неэффективен?

Поиск в ХТ 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.

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).

Выводы по проделанной работе

Основным преимуществом рассмотренных структур данных является возможная высокая эффективность реализации алгоритмов добавления, поиска, удаления элементов. Среднее время выполнения этих операций для таблиц намного меньше (для добавления нужно в среднем *в 10 раз больше времени для ДДП, и в 30 раз больше для АВЛ-дерева*), чем для деревьев. Так же, для хранения хеш-таблицы нужно *в 2 раза меньше памяти*, чем для хранения деревьев. Но, у деревьев есть по крайней мере одно заметное преимущество по сравнению с хеш-таблицей: в них можно выполнить проход по возрастанию или убыванию ключей и сделать это быстро. Стоит отметить, что время добавление в файл будет всегда константным, потому что перемещение каретки в конец файла происходит за константное время. Поэтому, выбор структур данных напрямую зависит от области их применения.