Отчёт по лабораторной работе №6 "Обработка деревьев и хэш-таблиц", Вариант 1

Выполнил: Калашков Павел ИУ7-36Б, Вариант 7 (по журналу)

Описание условия задачи

Цель работы – получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов; построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах.

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

Исходные данные

Исходными данными для программы является:

- название файла с входными данными (содержащий слова);
- данные в файле (слова);
- значение слова для поиска;

Результаты

Выходными данными являются:

- графы двоичного дерева поиска, обычного и сбалансированного (АВЛ-дерева)
- хеш-таблица, полученная из файла с входными данными;
- информация, связанная с результатами поиска (найдено или нет, сколько сравнений);
- результаты измерений времени работы операций поиска, а также используемой памяти

Задача, реализуемая программой

Задачей программы является построение ДДП, в вершинах которого находятся слова из текстового файла, а также вывод его на экран в виде дерева с последующей балансировкой. Программа должна строить хеш-таблицу из слов текстового файла, а для устроения коллизий использовать метод цепочек. Программа должна осуществлять поиск введённого слова в ДДП, в сбалансированном дереве, в хеш-таблице и в файле, а также сравнивать время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше введённого, то программа должна производить реструктуризацию таблицы, выбрав другую функцию.

Способ обращения к программе

Обращение к программе происходит посредством запуска исполняемого файла app.exe

Возможные аварийные ситуации и ошибки пользователя

Аварийной ситуацией является:

- ввод некорректного имени файла;
- несоблюдение формата данных (слова разделены не пробелами);

Описание внутренних структур данных

Для моделирования обработки очередей были созданы 4 структуры:

```
typedef struct bst_node bst_node_t;

struct bst_node
{
    char *value;
    bst_node_t *left;
    bst_node_t *right;
};

typedef struct avl_node avl_node_t;

struct avl_node
{
    char *value;
    char height;
    avl_node_t *left;
    avl_node_t *right;
};

struct node
{
    void *data;
    node_t *next;
};

node_t *hash_table[ARRAY_SIZE] = { NULL };
```

Описанный алгоритм

Алгоритм делится на четыре главных части:

- 1 - вывод начальной информации;
- 2 - запрос имени файла;
- 3 - построоение ДДП, АВЛ-дерева, хеш-таблицы;
- 4 - запрос слова для поиска;
- 5 - поиск слова, вывод результатов измерений;

Сравнение структур данных

Сравним среднее время работы и объём затрачиваемой памяти, а также количество сравнений для операции поиска элемента для четырёх структур данных: двоичного дерева поиска, АВЛ-дерева, хеш-таблицы и файла. Для каждой структуры рассмотрим худший, средний, и лучший случаи.

Измерения будем проводить на одном файле размером 9 байт

Двоичное дерево поиска:

Случай	Время поиска, нс	Объём используемой памяти, байты	Количество сравнений
Худший	280	240	10
Средний	140	240	5
Лучший	110	240	4

АВЛ-дерево:

Случай	Время поиска, нс	Объём используемой памяти, байты	Количество сравнений
Худший	110	320	4
Средний	110	320	3
Лучший	110	320	4

Хеш-таблица:

Случай	Время поиска, нс	Объём используемой памяти, байты	Количество сравнений
Худший	275	80	10
Средний	75	80	2
Лучший	75	80	1

Файл:

Случай	Время поиска, нс	Объём используемой памяти, байты	Количество сравнений
Худший	445	10	10
Средний	200	10	6
Лучший	85	10	1

Теоретическая часть

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. У дерево есть ссылка не только на один элемент, а на элемент справа и слева (для двоичного дерева).

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.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

У АВЛ дерева для каждой его вершины высота двух её помков 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 дереве двоичного поиска, и АВЛ дерево не зависит от изначально вводимых значений, а ДДП зависит.

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

Хеш-таблицей - массив, заполненный элементами в порядке, определяемом ключом. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс (например подбирает остаток от деления ключа от индекса)

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 связанный 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 количестве 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).

Выводы по проделанной работе

В ходе лабораторной работы я научился создавать и использовать такие структуры данных, как двоичные деревья поиска, АВЛ-деревья, хеш-таблицы (на примере операции поиска). Хеш-таблица является наиболее выгодной структурой данных, т.к. выигрывает как по памяти, так и по времени, имея среднее время работы пропорционально O(1). Однако сложность хеш-таблиц заключается в правильном подборе хеш-функции, а также в разрешении коллизий. Также АВЛ-дерево является хорошим вариантом хранения данных, по которым часто будет производиться поиск(ведь среднее время работы пропорционально log(n)) За счёт этого АВЛ-деревья первосходят обычные двоичные деревья поиска (т.к. являются более сбалансированными). Минут АВЛ-дерева - временные затраты на балансировку дерева, что является особенно заметно при частом выполнении вставки и удаления. В то же время ДДП являются хорошей структурой данных по сравнению с файлом.