|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 **ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04** Программная инженерия

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 7 |

**Название:**

Сбалансированные деревья, хеш-таблицы

**Дисциплина:** Типы и структуры данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-36Б |  | | А.А. Жаворонкова |
|  | (Группа) | |  | (И.О. Фамилия) |
|  |  | |  |  |
| Преподаватель: | Никульшина Т. А. | |  |  |

Москва, 2022

Описание условия задачи

Используя предыдущую программу (задача №6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана. Осуществить поиск введенного слова в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных.

Описание ТЗ

*Описание исходных данных и результатов*

*Исходные данные:* пункт меню (см. описание задачи), файл строк.

*Результаты:* дерево, АВЛ-дерево, хеш-таблицы, эффективность поиска.

*Описание задачи, реализуемой программой*

Меню:

1. Загрузить данные из файла
2. Сбалансировать дерево
3. Добавить элемент
4. Удалить элемент
5. Найти элемент
6. Вывести хеш-таблицы
7. Реструктурировать функцию
8. Сравнить время поиска в деревьях и хеш-таблице
9. Вывести меню
10. Выйти из программы

*Способ обращения к программе*

Вызов программы происходит через терминал (main.exe). Дальнейшая работа с программой выполняется при помощи меню, выводимого на экран.

*Описание возможных ошибок пользователя*

* Ввод неверного пункта меню
* Ввод неверного имени файла
* Попытка удаления несуществующего элемента
* Вывод несозданных хеш-таблиц

Описание внутренних структур данных

АВЛ-дерево и двоичное дерево поиска представлены следующей структурой:

typedef struct node

{

    char \*word;

    int height;

    struct node \*left;

    struct node \*right;

} node\_t;

word – указатель на строку  
height – высота (в ДДП не используется)  
left – указатель на элемент, меньше текущего  
right – указатель на элемент, больше текущего

Хеш-таблица с открытым хешированием:

typedef struct hash

{

    int k;

    char \*word;

    struct hash \*next;

} hash\_node\_t;

typedef struct

{

    int len;

    hash\_node\_t \*\*arr;

} open\_hash\_t;

k - ключ  
word – указатель на строку  
next – указатель на следующий элемент  
len – длина массива  
arr – массив элементов хеш-таблицы

Хеш-таблица с закрытым хешированием:

typedef struct

{

    int k;

    char \*word;

} elem\_t;

typedef struct

{

    int len;

    elem\_t \*\*arr;

} close\_hash\_t;

k - ключ  
word – указатель на строку  
len – длина массива  
arr – массив элементов хеш-таблицы

Набор тестов с указанием проверяемого параметра

|  |  |  |
| --- | --- | --- |
| Входные данные | Выходные данные | Что проверяется |
| Имя файла | «Не удалось открыть файл» | Неверное имя файла |
| Пункт меню: 2 | «Дерево еще не считано» | Балансировка несозданного дерева |
| Пункт меню: 4 | «Элемент не найден» | Удаление несуществующего элемента |
| Пункт меню: 6 | «Хеш-таблица еще не создана» | Вывод несозданной хеш-таблицы |

Описание алгоритма

Поиск в двоичном девере и АВЛ-дереве выполняется следующим образом: от корня дерева в зависимости от результата сравнения с текущим элементом «уходим» либо в правое, либо в левое поддерево.

Поиск в хеш-таблице с открытым хешированием: по ключу вычисляется индекс в массиве и просматривается линейный список, пока не найдем нужный элемент.  
Поиск в хеш-таблице с закрытым хешированием: по ключу вычисляется индекс в массиве, с которого нужно начать поиск, и массив просматривается далее, пока не найдется нужный элемент.

Выбранная хеш-функция:

int hash\_func(int key, int add, int size)

{

    return (key + add) % size;

}

key – ключ  
add – целое число, добавляемое к ключу (изменяется при устранении коллизий)  
size – размер хеш-таблицы

Временная эффективность и затраты памяти

Время было измерено, как среднее арифметическое времен поиска каждого слова. При создании хеш-таблиц их размер был равен количеству элементов.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Общее количество элементов | Двоичное дерево | | АВЛ дерево | | Открытая адресация | | Закрытая адресация | |
| Время, нс | Память, б | Время, нс | Память, б | Время, нс | Память, б | Время, нс | Память, б |
| 10 | 24 | 320 | 17 | 320 | 13 | 256 | 14 | 176 |
| 50 | 46 | 1 600 | 44 | 1 600 | 15 | 1 216 | 15 | 816 |
| 100 | 43 | 3 200 | 41 | 3 200 | 11 | 2 368 | 11 | 1 616 |
| 500 | 70 | 16 000 | 52 | 16 000 | 11 | 11 512 | 10 | 8 016 |

Как видно из таблицы, поиск в АВЛ дереве происходит быстрее, чем в двоичном. Поиск в хеш-таблицах с открытой и закрытой адресацией происходит за примерно одинаковое время.  
Видно, что затраты памяти у хеш-таблиц меньше, чем у деревьев. При моей реализации хеш-таблицы с закрытой адресацией каждый ее элемент имел размер 16 б, а с открытой адресацией – 24 б. Отсюда такая существенная разница в размерах у хеш-таблиц.

Вывод

Поиск в АВЛ дереве происходит быстрее, чем в бинарном, так как уменьшается высота дерева. Хеш-таблицы с открытой и закрытой адресацией выполняют поиск, в среднем затрачивая одинаковое время. При этом поиск в хеш-таблицах происходит быстрее, чем в деревьях, потому что при отсутствии коллизий трудоемкость поиска становится O(1). Также видно, что поиск в деревьях зависит от количества элементов: с увеличением их числа, увеличивается время поиска; в хеш-таблицах вне зависимости от размера поиск происходит за одно и то же время.

Таким образом, хеш-таблицы имеют преимущество перед деревьями и по времени, и по затрачиваемой памяти.

Ответы на вопросы

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Идеально сбалансированное дерево – дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу.  
АВЛ-дерево – двоичное дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу.

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

АВЛ-дерево имеет наименьшую высоту дерева, а так как высота дерева определяет длину пути поиска в нем, то, следовательно, и поиск укорачивается.

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

Хеш-таблица – массив, заполненный в порядке, определенным хеш-функцией. Принцип построения – запись элемента в массив на такой индекс, который определила хеш-функция для конкретного ключа.

1. Что такое коллизии? Каковы методы их устранения.

Коллизия – ситуация, когда разным ключам соответствует одно значение хеш-функции. Методы устранения: использование открытой и закрытой адресации. При открытой адресации по одному индексу хранится связный список, в котором находятся все элементы, для которых хеш-функция определила этот индекс. При закрытой адресации по одному индексу хранится только один элемент, если хеш-функция определяет уже занятый индекс, то элемент записывается далее по массиву в первый свободный индекс.

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

Если для поиска элемента необходимо более 3–4 сравнений, то эффективность использования такой хеш-таблицы пропадает и ее следует реструктуризировать (т.е. найти другую хеш-функцию).

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

Поиск в двоичном дереве происходит медленнее, чем в АВЛ дереве, потому что АВЛ дерево имеет меньшую высоту, что укорачивает поиск. Поиск в хеш-таблице с открытой адресацией происходит медленнее, чем в закрытой. Поиск в файле происходит медленнее всего.