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 И ХЕШ-ТАБЛИЦ»

Студент Ковалец Кирилл

Группа ИУ7 – 33Б

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.

Описание технического задания

В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать закрытое хеширование для устранения коллизий. Осуществить добавление введенного целого числа, если его там нет, в ДДП, в сбалансированное дерево, в хеш-таблицу и в файл. Сравнить время добавления, 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.

Входные данные:
Номер команды, отвечающий за определённое действие.

Команды:

1. Работа с деревьями;

    1) Добавить элемент в дерево;
    2) Удалить элемент из дерева;
    3) Закончить работу с деревьями;

2. Работа с хеш-таблицей;

    1) Добавить элемент в хеш-таблицу;
    2) Удалить элемент из хеш-таблицы;
    3) Закончить работу с хеш-таблицей;

3. Добавить элемент в файл;
4. Замеры времени и памяти;
5. Выйти из программы.

Выходные данные:

1. Результат выполнения определённой команды;
2. Результат сравнения обработки 2-х типов деревьев и хеш-таблицы по времени и памяти.

Обращение к программе:
Запускается через терминал командой make run

Сообщения при аварийных ситуациях:

1. Не удалось прочитать номер команды;
2. Номер команды должен быть >= 0 и <= 4;
3. Буфер переполнен;
4. Не удалось прочитать число из файла;
5. Не удалось прочитать кол-во сравнений;
6. Кол-во сравнений должно быть больше нуля;
7. Не удалось выделить память.

Описание структуры данных

vertex_t - структура, содержащая информацию о вершине дерева.

```
typedef struct vertex
{
   struct vertex *left;
   struct vertex *right;
   int number;
   int height;
} vertex_t;
```

Поля структуры:

1) struct vertex *left – указатель на вершину левого поддерева;
2) struct vertex *right – указатель на вершину правого поддерева;
3) int number – значение элемента дерева;
4) int height – текущая высота дерева.

tree_t - структура, содержащая информацию о дереве.

```
typedef struct tree
{
   vertex_t *root;
   int size;
   int height;
```

```
    int total_compare;
} tree_t;
```

Поля структуры:

1) vertex_t *root – указатель на “голову” дерева;
2) int size – кол-во элементов в дереве;
3) int height – высота дерева.
4) int total_compare – общее кол-во сравнений, требуемое для нахождения каждого элемента дерева.

hash_elem_t - структура, содержащая информацию об элементе хеш-таблицы.

```
typedef struct hash_elem
{
    int number;
    char bool;
} hash_elem_t;
```

Поля структуры:

1) int number – значение элемента хеш-таблице;
2) char bool – признак занятости ячейки таблицы.

hash_table_t - структура, содержащая информацию о хеш-таблице.

```
typedef struct hash_table
{
    hash_elem_t *array;
    int size;
    int divider;
    int elements;
    int total_compare;
} hash_table_t;
```

Поля структуры:

1) hash_elem_t *array – массив элементов хеш-таблицы;
2) int size – кол-во элементов в таблице;
3) int divider – делитель, требуемый для хеш-функции;
4) int total_compare – общее кол-во сравнений, требуемое для нахождения каждого элемента хеш-таблицы.

Описание алгоритма

1. Выводится меню программы (каждой команде присвоен номер);
2. Пользователь вводит номер команды, который отвечает за определённое действие;
3. Ввод осуществляется до того момента, пока не будет введён 0, являющийся признаком выхода из программы.

Набор тестов

№	Название теста	Входные данные	Результат
2	Номер команды - число	k	Не удалось прочитать номер команды
3	Номер команды >= 0 и <=4	8	Номер команды должен быть >= 0 и <= 4
	Открытие несуществующего файла	Iu7.txt	Такого файла не существует
4	Загрузить файл с некоректными данными	Элементы в файле: 1 2 k	Не удалось прочитать число из файла
5	Кол-во сравнений - число	k	Не удалось прочитать кол-во сравнений
6	Кол-во сравнений – положительное число	0, -1	Кол-во сравнений должно быть больше 0
7	Добавление нового числа в дерево (таблицу)	2	Успешное добавление числа
8	Добавление в дерево (таблицу) числа, уже существующего там	24	Такой элемент уже существует
9	Удаление числа из дерева (таблицы)	10	Успешное удаление числа

10	Удаление числа, не существующего в дереве (таблице)	100	Такого элемента нет в дереве (хеш-таблице)
11	Добавления числа в файл (число уже есть в нём)	10	Такой элемент уже есть в файле
12	Добавления нового числа в файл	1	Элемент успешно добавлен в файл
13	Работа с деревьями	Команда 1	Вывод ДДП и АВЛ-дерева и меню для добавления и удаления элементов из них
14	Работа с хеш-таблицей	Команда 2	Вывод хеш-таблицы и меню для добавления и удаления элементов из неё
15	Сравнение работы двух деревьев и хеш-таблицы по памяти и времени	Запрос на сравнение (Номер команды – 4)	Печать требуемых по заданию значений
16	Выход из программы	Команда 0	Выход из программы

Оценка эффективности

Сортировка каждой таблицы будет измеряться в тактах процессора (процессор со средней частотой 2.3gHz).

Добавление элементов (в тиках)

Размер	ДДП	АВЛ-дерево	Хеш-таблица	Файл
10	196	292	61	4283
25	201	398	64	4247
50	241	550	69	4031
100	329	610	62	4093

Сравнение памяти

Размер	ДДП	АВЛ-дерево	Хеш-таблица	Файл
10	264	264	104	25
25	624	624	224	70
50	1224	1224	424	146
100	2424	2424	824	346

Среднее кол-во сравнений для поиска элементов

Размер	ДДП	АВЛ-дерево	Хеш-таблица	Файл
10	3.50	2.90	1.00	5.50
25	7.24	3.96	1.00	13.00
50	13.50	4.86	1.00	25.50
100	26.00	5.80	1.00	50.50

Ответы на контрольные вопросы

1. Что такое дерево?
Дерево – рекурсивная структура данных, используемая для представления иерархических связей, имеющих отношение “один ко многим”.

2. Каким выделяется память под представление деревьев?
В виде связанного списка. Каждый лепесток – узел.

3. Какие стандартные операции возможны над деревьями?
Обход дерева, поиск по дереву, включение элемента в дерево, исключение элемента из дерева.

4. Что такое дерево двоичного поиска?
Двоичное дерево поиска – двоичное дерево, для каждого узла которого выполняется условие: левый потомок больше или равен родителю, правый потомок строго меньеш родителя (также можно наоборот).

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?
В АВЛ дереве для каждой его вершины высота её 2-х поддеревьев отличается не более, чем на 1. В идеально сбалансированном дереве количество вершин в каждом поддереве различается не более, чем на 1.

6. Чем отличается поиск в АВЛ дереве от поиска в дереве двоичного поиска?
В АВЛ дереве поиск происходит быстрее, чем в дереве двоичного поиска.

7. Что такое хеш-таблица, какой принцип ее построения?
Хеш-таблица – массив, заполненный элементами в порядке, который опредляется хеш-функцией
Хеш-функция ставит каждому элементу таблицы в соответствие определенный индекс. Функция должна быть простой для вычисления, распределять ключи в таблице равномерно и давать в результате минимум коллизий.

8. Что такое коллизии? Каковы методы их устранения?
Коллизия – ситуация, при которой разным ключам хем-таблицы ставится в соответствие один и тот же индекс.
Основные методы устранения коллизий: открытое хеширование, закрытое хеширование.

9. В каком случае поиск в хеш-таблице становится неэффективен?
При большом количестве коллизий поиск становится менее эффективным, сложность возрастает относительно O(1). В таком случае требуется реструктуризация таблицы, то есть перезаполнение таблицы с использованием новой хеш-функции.

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

Минимальное время поиска для хеш-таблицы: O(1) Минимальное время поиска для АВЛ дерева: O(log2n)

Минимальное время поиска а Двоичном дереве поиска O(h), где h – высота дерева (от log2n до n)

Вывод

При удалении или добавлении элемента, в АВЛ - дереве необходимо корректировать балансировку, тратя на это дополнительные ресурсы, из-за этого возникает разница во времени добавления новых вершин(от 49% до 98%) по сравнению с обычным деревом. При этом АВЛ-дерево гораздо эффективнее при поиске элемента в нём (в 5 раз при 100 элементах). Оба вида деревьев занимают одинаковую память.

Хеш-таблицы используют меньше памяти, и для них требуется меньшее кол-во операций сравнения при добавлении элемента (если отсутствую коллизии, то всего 1). Стоит ответственно подойти к выбору хеш-функции, чтобы избежать большого кол-ва коллизий.

Способ хранения элементов в файле выигрывает только по памяти (от 2 до 7 раз). При добавлении новых элементов или поиске нужного он проигрывать как деревьям, так и хеш-таблице (при 100 элементах время добавления больше от 7 (при использовании хеш-таблицы) до 66 раз (при использовании АВЛ-дерева), а среднее кол-во сравнений при поиске нужного элемента от 2 до 50 раз).