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 «Работа с деревью»

Студент Романов Семен Константинович

Группа ИУ7 – 35Б

Проверено Никульшина Татьяна Александровна

Описание задачи

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 удаление введенного целого числа в ДДП, в сбалансированном дерево, в хеш-таблице и в файле.** Сравнить время удаления, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного, то произвести реструктуризацию таблицы, выбрав другую функцию.

Техническое задание:

Входные данные:

1. **Текстовый файл:** Текстовый файл с целыми числами
2. **Числовые обозначения:** максимальное количество сравнений в хэш-таблице; число, необходимое удалить

Выходные данные:

1. **Результаты работы с деревом:** Графическое изображение сбалансированного и несбалансированного деревьев, хэш-таблица**.**
2. **Эффективность работы структур данных:** Время работы функции, занимаемая память структуры, количество сравнений при работе функции, связанной со структурой

Способ обращения к программе: запускается из терминала командой ./app.exe <filename>, где <filename> - имя входного файла

Функция программы:

1. Вывод бинарного дерева.
2. Вывод сбалансированного бинарного дерева.
3. Вывод хеш-таблицы.
4. Удаление введенного числа.
5. Вывод всех предыдущих структур данных после удаления числа
6. Вывод характеристик структур данных, указанных в секции выходных данных.

Аварийные ситуации:

1. Неверное название файла или пустой файл: Сообщение “Incorrect file! Exiting the program”
2. Неверный ввод при вводе максимального количества сравнений: Сообщение “Incorrect input! Exiting the program”

Структуры данных:

Структура узла ДДП дерева:

```
typedef struct tree_node
{
    int data;
    struct tree_node *left;
    struct tree_node *right;
} node_t;
```

Поля структуры:

- **Int data:** значение узла
- **Struct tree_node *left:** указатель на левый узел дерева (в случае головного элемента – указатель NULL)
- **Struct tree_node *right:** указатель на правый узел дерева (в случае головного элемента – указатель NULL)

Структура дерева:

```
typedef struct
{
    node_t *arr;
    int size;
} tree_arr;
```

Поля структуры:

- **Node_t *arr:** указатель на корень дерева
- **Int size:** Количество узлов дерева

```
typedef struct node_avl
{
    int data;
    int height;
    struct node_avl *left;
    struct node_avl *right;
} node_avl_t;
```

Поля структуры:

- **Int data:** значение узла
- **Int height:** высота поддерева (наидленейший путь)
- **Struct tree_node *left:** указатель на левый узел дерева (в случае головного элемента – указатель NULL)
- **Struct tree_node *right:** указатель на правый узел дерева (в случае головного элемента – указатель NULL)

Структура Хэш-таблицы:

```
typedef struct
{
    int *hash_table;
    int *is_occupied;
    int size;
    int hash_num;
} hash_t;
```

Поля структуры:

- **Int *hash_table:** Массив, предназначенный под хранение значений хэш-таблицы
- **Int *is_occupied:** Массив, в котором хранятся данные о том, занята ли ячейка в массиве хэш-таблицы или нет.
- **Int size:** Размер массива
- **Int hash_num:** Хэш-число, необходимое для операций

Алгоритм

1. При начале работы программы задается название файла, откуда будут считываться данные
2. При постройке дерева значение нового узла считывается из файла, после чего начинается рекурсивный обход дерева, в случае если значение нового узла меньше текущего узла, то начинается процесс от левого потомка, иначе от правого. Процесс проходит до тех пор, пока не дойдем до узла без потомков

3. При постройке АВЛ дерева, значения всех узлов записываются в массив, начиная с самого левого. Таким образом у нас получается упорядоченный массив, и для построения сбалансированного дерева мы будем рекурсивно разбивать массив на 2 части, где значение текущего узла будет равно центральному элементу, а разбитые части уйдут в левому и правому потомку.
4. При постройке хэш-таблицы мы используем одну хэш-функцию, где значение ключа вычисляется в ходе одной какой-либо операции. В случае если ячейка массива хэш-таблицы уже занята другим элементом, то значение ключа уменьшается на 1, пока не будет найдена пустая ячейка.

   Если текущее количество переходов будет превышать максимально допущенное, то таблица будет перестроена второй хэш-функцией, с последующим увеличением хэш-числа при повторении ситуации
5. При удалении числа из дерева, возможны 2 ситуации:
   a. *У узла есть потомки:*

      В таком случае ищется максимально правый узел левого поддерева, либо при отсутствии левого потомка, максимально левый узел правого поддерева, дабы сохранить бинарность дерева, после чего просиходит замена значений

   b. *У узла нет потомков:*

      В таком случае просто удаляется ссылка на него (с освобождением памяти)

6. При нарушении сбалансированности АВЛ-дерева после удаления элемента, применяются левые и правые повороты вокруг нужных нам узлов деревьев

Тесты

N	Test	Input	Output
1	Некорректное имя файла	Abc.txt (Файл не существует)	“Incorrect file! Exiting the program”
2	Пустой файл	Abc.txt (Файл пуст)	“Incorrect file! Exiting the program”
3	Некорректное максимальное количество сравнений	-1	“Invalid Input! Exiting the program”
4	Некорректное максимальное количество сравнений	Ab	“Invalid Input! Exiting the program”
5	Ввод несуществующего числа	123 (числа нет в дереве)	“Number not found”
6	Корректный ввод всех данных	Корректный файл, максимальное количество сравнений, корректное число для удаления	Характеристика структур данных

Оценка эффективности:

Все время измеряется в тактах процессора

Время удаления элемента:

Size of tree	Бинарное дерево	АВЛ дерево	Хэш-таблица	Файл
10	908	1108	538	884200
50	1024	1808	736	987072
100	1506	2276	806	1096736
200	2748	3420	992	1302976

Занимаемая память в байтах:

Все размеры в байтах.

Размер АВЛ дерева всегда равен ДДП (ввиду упаковки структур)

При работе с хеш-таблицей количество максимальных сравнений равно 10% от количества чисел в файле

Size of tree	Дерево	Хэш-таблица	Файл
10	216	568	35
50	1176	856	195
100	2376	1528	390
200	4728	2824	780

Контрольные вопросы

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.

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 (либо наоборот).

5. 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.

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

Поиск в АВЛ дереве происходит быстрее, чем в ДДП.

7. 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.

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 числе коллизий с открытой адресацией и при большом количестве сравнений с закрытой – 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).

Вывод

Использование хеш-таблицы всегда эффективно как по времени, так и эффективно по памяти (в случае хорошей дистрибуции функции распределение будет не плотным, например если отсутствуют коллизии, то это считается хорошей дистрибуцией). В случае деревьев, АВЛ дерево редко выигрывает по времени удаления у несбалансированного дерева, так как порядок вершин при балансировке меняется, также у АВЛ дерева тратятся дополнительные ресурсы на ребалансировку, но при этом оба варианта в разы выигрывают по времени удаления из файла.