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»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7
«СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ»

Студент: Кашима Ахмед

Группа: ИУ7-33Б

2022 г

1. Цель работ

Цель работы – построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнить эффективность устранения коллизий при внешнем и внутреннем хешировании.

2. Описание условия задачи

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

3. Описание технического задания

3.1. Входные данные

Целое число от 0 до 10 – номер пункта меню, позволяющий пользователю осуществить работу с бинарным деревом и хеш-таблицей. Элемент дерева и хеш-таблицы – целое число.

Для корректной работы программы необходимо загрузить данные из заранее подготовленного текстового файла.

3.2. Выходные данные

Выходными данными в зависимости от пунктов меню являются либо визуализация дерева в виде «png» картинки, либо результат сравнения

времени добавления вершины в файл и в дерево, либо сравнение поиска элемента в ДДП, сбалансированном дереве поиска и хеш-таблице.

3.3. Описание задачи, реализуемой программой

Задача программы – обеспечение работы с такими структурами данных, как бинарное дерево поиска и хеш-таблицы.

3.4. Способ обращения к программе

Ввод и вывод всех данных осуществляется через консоль.

Для корректной работы программы необходимо загрузить данные из заранее подготовленного текстового файла.

Программа собирается с помощью специального make-файла, и может быть запущена в командной строке с помощью слова make.

3.5. Описание возможных аварийных ситуаций и ошибок пользователя

Аварийные ситуации:

- Пустое поле ввода – ничего не произойдет, программа будет ждать ввода пользователя;
- Удаление вершины из пустого дерева – ничего не произойдет, будет выведено сообщение о том, что дерево пустое;
- Ввод имени файла, которого не существует – программа сгенерирует файл с данными с введенным названием.

Ошибки пользователя:

- Некорректный ввод: ввод неверного пункта меню, ввод не валидных данных при добавлении в дерево и при удалении из дерева, ввод не валидных данных при добавлении элемента в хеш-таблицу.

4. Описание внутренних струтур данных

Реализация вершины бинарного дерева описывается структурой «vertex_t».

```
typedef struct vertex_t vertex_t;

struct vertex_t
{
    int data;
    int height;      // высота вершины
    vertex_t *left;  // меньшие
    vertex_t *right; // большие
};
```

Листинг 1. Структура «vertex_t», содержащая описание одной вершины

Поля структуры «vertex_t»:

1. data – целое число, данные вершины дерева;
2. height – целое число, высота вершины для сбалансированного дерева;
3. left – указатель на левого (меньшего по значению) потомка;
4. right – указатель на правого (большего по значению) потомка.

Бинарное дерево описывается структурой «tree _t».

```
typedef struct tree_t tree_t;

struct tree_t
{
    vertex_t
*root; };
```

Листинг 2. Реализация бинарного дерева структурой «tree_t»

Поля структуры «tree_t»:

1. root – корень дерева.

Хеш-таблица была реализована с помощью структуры «hash_table_t».

```
typedef struct hash_table_t hash_table_t;

struct hash_table_t
{
    hash_t *data;
    int size; // количество
элементов };
```

Листинг 3. Реализация хеш-таблицы структурой «hash_table_t»

Поля структуры «hash_table_t»:

1. data – массив структур типа «hash_t»;
2. size – размер массива.

Поскольку для разрешения коллизий в хеш-таблице мною был выбран способ цепочек, то для реализации одного элемента массива в хеш-таблице была использована структура «hash_t».

```
typedef struct hash_t hash_t;

struct hash_t
{
    data_t *head;
    int hash;
    int collision;
    bool is_full;
};
```

Листинг 4. Реализация одного элемента массива структурой «hash_t»

Поля структуры «hash_t»:

1. head – указатель на односвязный список (цепочку), которая будет содержать данные с одинаковым хешем;
2. hash – вычисленное значение хеша для элемента массива;
3. collision – целочисленная переменная, количество коллизий для данного значения хеша;
4. is_full – булевая переменная, принимающая значение истины, если под текущим значением хеша уже был добавлен элемент.

Узел связного списка, содержащий данные, хранящиеся под вычисленным значением хеша, описываются структурой «data_t».

```
typedef struct data_t data_t;

struct data_t
{
    int data; // данные
    data_t *next;
};
```

Листинг 5. Реализация узла списка структурой «data_t»

Поля структуры «data_t»:

1. data – целочисленная переменная, элемент хеш-таблицы;
2. next – указатель на следующий элемент.

4.1. Описание алгоритма

Взаимодействие пользователя с программой осуществляется через консоль с помощью специального меню, в котором пользователю предлагается выполнить то, или иной действие.

При выполнении поиска в используемых структурах данных по результатам работы будут выведены результаты эффективности, а именно:

- время поиска в микросекундах;
- количество сравнений, получившееся в результате поиска;
- требуемый для хранения структуры объем памяти в байтах.

Если при поиске элемента в хеш-таблице количество сравнений превысит введенное пользователем максимально допустимое количество сравнений, то по окончании поиска выведется сообщение о том, что необходимо реструктуризировать таблицу, программа предложит пользователю сменить хеш-функцию, а затем нужно будет снова прочитать данные из текстового файла для отображения реструктуризированной таблицы.

```
------------------------------------------------------------|
Программа для обработки бинарного дерева и хеш таблицы.
Элементом дерева и хеш-таблицы является целое число.
Правила:
- добавить можно только уникальное число,
добавление дубликата будет проигнорировано;
Операции для обработки дерева:
1 - прочитать данные из файла;
2 - добавить в дерево число;
3 - удалить число из дерева;
4 - вывести обычное и сбалансированное бинарное дерево;
5 - поиск введенного с клавиатуры числа;
Операции для обработки хеш-таблицы:
6 - прочитать данные из файла;
7 - вывести хеш-таблицу;
8 - поиск введенного с клавиатуры числа;
9 - сменить хеш-функцию;
10 - добавить элемент в хеш-таблицу;
0
- выйти из программы.
------------------------------------------------------------|
Выберите пункт меню:
```

Листинг 3. Меню взаимодействия

4.2. Алгоритмы работы хеш-функций

Алгоритм работы первой хеш-функции: вычисляется сумма цифр переданного в качестве значения целого числа и от полученной суммы берется остаток от деления на размер массива в качестве результата работы хеш-функции.

Алгоритм работы второй хеш-функции: в качестве результата работы данной хеш-функции берется остаток от деления переданного в качестве значения целого числа на размер массива.

4.3. Ограничения на входные данные

Попытка удаления из пустого дерева обрабатывается программой, выдавая соответствующие сообщения, но не завершая программу с ненулевым кодом возврата.

При вводе любых не валидных данных программа выдает

соответствующее сообщение и завершает работу с ненулевым кодом возврата.

При визуализации бинарного дерева в виде png картинки элементы правого поддерева соединены дугами красного цвета, а элементы левого поддерева соединены дугами синего цвета.

Чтобы использовать добавление данных в хеш-таблицу вручную, необходимо прежде прочитать данные из текстового файла, поскольку для вычисления хеш-значения необходим размер массива.

5. Оценка эффективности работы алгоритмов

Результаты измерения времени поиска данных в бинарном дереве поиска, в сбалансированном бинарном дереве и в хеш-таблице при разных количествах элементов. Для каждого количества вершин было проведено 1000 измерений и взято среднее из временных интервалов.

Сравнение времени поиска числа в ДДП, в СДП и хеш-таблице

Количество элементов	**Поиск элемента (мкс)**		
	ДДП	СДП	Хеш-таблица
10	1.403000	1.101000	0.901000
100	3.170000	2.269000	0.964000
500	2.384000	2.231000	0.880000
700	2.994000	2.121000	1.003000
1000	4.734000	2.974000	1.277000

Зависимость количества сравнений от числа в ДДП, в СДП и хеш-таблице

Количество элементов	**Количество сравнений**		
	ДДП	СДП	Хеш-таблица
10	6	4	1
100	8	7	2
500	14	9	1
700	15	10	1
1000	15	10	3

Сравнение времени поиска числа в ДДП, в СДП и хеш-таблице

Количество элементов	**Память (байты)**		
	ДДП	СДП	Хеш-таблица
10	216	216	364
100	2232	2232	3604
500	7512	7512	18004
700	9216	9216	25204
1000	10416	10416	36004

6. Выводы

Анализируя полученные данные, можно заметить, что время поиска в сбалансированном дереве выполняется быстрее, чем в ДДП, но проигрывает времени поиска в хеш-таблице. Это согласуется с теорией, поскольку время поиска в бинарном дереве зависит от высоты дерева, а балансировка дает наименьшую высоту дерева, а так как высота дерева определяет длину пути поиска в нем, то, следовательно, и укорачивает поиск (сложность в среднем O(log(n))).

Поиск в хеш-таблице выполняется быстрее, чем в ДДП и СДП потому, что сложность поиска в хеш-таблице в среднем O(1), так как для элемента, которые требуется найти, вычисляется хеш, и по хешу находится элемент. Если же в хеш-таблице имеются коллизии, то по значению хеша идет поиск элемента в односвязном списке (цепочке), что увеличивает поиск до O(k), где k – количество узлов в цепочке.

При всех плюсах использования хеш-таблиц для поиска элемента видим, что для хранения этой структуры данных необходимо больше всего памяти, и чем больше количество элементов, тем большая разница наблюдается в хранении ДДП, СДП и хеш-таблицы (при небольших количествах элементов ~100 в 1,5 раза, а при больших данных ~1000 в 3,5 раза.

№ Теста	Входные данные	Выходные данные	Результат
01	Номер пункта меню 1, 6 Файл с данными существует и не пустой	Сообщение об успешности считывания данных из файла в дерево	Ожидание следующего действия.
02	Номер пункта меню 1, 6 Файла с данными не существует Ввод валидного количества генерируемых данных	Сообщение об успешности считывания данных из файла в дерево	Ожидание следующего действия.
03	Номер пункта меню 2 Ввод валидного элемента	Сообщение об успешном добавлении данных в деревья	Ожидание следующего действия.
04	Номер пункта меню 3 Ввод валидного элемента Удаляемого элемента нет	Сообщение о том, что удаляемого элемента нет	Ожидание следующего действия.
05	Номер пункта меню 3 Ввод валидного элемента Удаляемый элемент есть	Сообщение об успешном удалении данных из деревьев	Ожидание следующего действия.
06	Номер пункта 4 Дерево пустое	Сообщение о том, что дерево пустое	Ожидание следующего действия.
07	Номер пункта 4 Дерево не пустое	Создание png изображения ДДП и СДП	Ожидание следующего действия.
08	Номер пункта 5 Дерево пустое	Сообщение о том, что дерево пустое	Ожидание следующего действия.
09	Номер пункта 5 Дерево не пустое	Вывод результатов поиска	Ожидание следующего действия.
10	Номер пункта 7 Хеш-таблица пустая	Сообщение о том, что хеш-таблица пустая	Ожидание следующего действия.
10	Номер пункта 7 Хеш-таблица не пустая	Вывод хеш-таблицы на экран	Ожидание следующего действия.
11	Номер пункта 8 Хеш-таблица пустая	Сообщение о том, что хеш-таблица пустая	Ожидание следующего действия.
12	Номер пункта 8 Хеш-таблица не пустая	Вывод результатов поиска	Ожидание следующего действия.
13	Номер пункта 9 Валидный номер хеш-функции	Сообщение о смене хеш-функции	Ожидание следующего действия.

14	Номер пункта 10 Хеш-таблица пустая	Сообщение о необходимости прочесть данные из файла	Ожидание следующего действия.
15	Номер пункта 10 Хеш-таблица не пустая Валидный элемент для добавления	Сообщение об успешном добавлении данных	Ожидание следующего действия.

Негативные тесты

№ Теста	Входные данные	Выходные данные	Результат
01	Номер пункта меню 100	Сообщение о том, что введен неверный номер пункта меню	Завершение программы с ненулевым кодом возврата
02	Номер пункта равен 2, 3, 10 Элемент: папввывы	Сообщение о том, что введен некорректный элемент	Завершение программы с ненулевым кодом возврата
03	Номер пункта 9 Невалидный выбор хеш-функции	Сообщение о том, что введен невалидный номер хеш-функции	Завершение программы с ненулевым кодом возврата

8. Ответы на контрольные вопросы

8.1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В идеально сбалансированном дереве число вершин в левом и правом поддеревьях отличается не более чем на единицу, а в АВЛ дереве у каждого узла дерева высота двух поддеревьев отличается не более, чем на единицу.

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

Так как высоты поддеревьев в АВЛ дереве отличаются не более, чем на единицу, то количество сравнений при поиске элемента уменьшается (сложность O(log(n))). В ДДП в худшем случае, когда дерево представляет

из себя линейный односвязный список, временная сложность O(n) (в среднем также O(log(n))).

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

Хеш-таблица – массив, заполненный в порядке, определенном хеш-функцией. Для каждого исходного элемента вычисляется значение хеш-функции, в соответствии с которым элемент записывается в определенную ячейку массива.

8.4. Что такое коллизии? Каковы методы их устранения.

Коллизия – ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда h(K1) = h (K2), в то время как K1! = K2.

Первый метод – внешнее (открытое) хеширование (метод цепочек)

В случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется связный список. Таким образом, если для нескольких различных значений ключа возвращается одинаковое значение хеш-функции, то по этому адресу находится указатель на связанный список, который содержит все значения. Поиск в этом списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

Другой путь решения проблемы, связанной с коллизиями – внутреннее (закрытое) хеширование (открытая адресация). Оно, состоит в том, чтобы полностью отказаться от ссылок. В этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку (с шагом 1), до тех пор, пока не будет найден ключ K или пустая позиция в таблице. При этом, если индекс следующего просматриваемого элемента определяется добавлением какого-то постоянного шага (от 1 до n), то данный способ разрешения коллизий называется линейной адресацией.

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

Поиск в хеш-таблицах становится менее эффективен, если наблюдается большое число коллизий, тогда вместо ожидаемой сложности O(1) получим сложность O(n).

В первом методе – поиск в списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

Во втором методе – необходимо просматривать все ячейки, если есть много коллизий.

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

Хеш-таблица – от O(1) до O(n)

АВЛ дерево – O(log(n))

Дерево двоичного поиска – от O(log(n)) до O(n).