



**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» (ИУ)
КАФЕДРА «Программное обеспечение ЭВМ и информационные
технологии» (ИУ7)

**Лабораторная работа №7
“Сбалансированные деревья, хеш-таблицы”
Вариант №8**

Студент:
Князев Дмитрий Юрьевич, группа ИУ7-33Б

(подпись, дата)

Преподаватель:
Барышникова Марина Юрьевна

(подпись, дата)

2022 г.

Оглавление

Описание задачи	3
Входные и выходные данные	4
Аварийные ситуации и особенности реализации	5
Описание структур данных	6
Функциональные тесты	11
Сравнение эффективности	18
Вывод	21
Контрольные вопросы	22

Цель работы:

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.

Задание:

Построить хеш-таблицу по указанным данным. Сравнить эффективность поиска в сбалансированном двоичном дереве, в двоичном дереве поиска и в хеш-таблице (используя открытую и закрытую адресацию). 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 и при различных методах их разрешения.

Задание по варианту:

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.

Входные данные:

- Пункт меню (число от 0 до 6 включительно)
- В зависимости от выбранного пункта меню:
 - Файл с последовательностью целых чисел
 - Целое число

Выходные данные (в зависимости от выбранного пункта меню):

- Графическое изображение AVL дерева или ДДП в формате .svg файла в отдельном окне путём вызова xdg-open
- Содержимое хеш-таблиц в символьном представлении
- Информация об успешном или неудачном поиске, добавлении или удалении элемента в ДДП, AVL дереве, хеш-таблицах и массиве, о загрузке элементов из файла в AVL дерево, ДДП и в хеш-таблицы
- Результаты сравнения эффективности алгоритмов сортировки и поиска с использованием ДДП, AVL дерева, хеш-таблиц и массива

Функции меню

0. Выход
1. Загрузить числа из файла в AVL дерево, в ДДП и в хэш-таблицы
2. Вывести ДДП
3. Вывести AVL дерево
4. Вывести хэш-таблицы
5. Добавить элемент в ДДП, AVL дерево, хэш-таблицу и в файл
6. Удалить элемент из ДДП, AVL дерева, хэш-таблицы и из файла
7. Найти элемент в AVL дереве, в ДДП, в хэш-таблицах и в файле
8. Сравнить эффективность алгоритмов сортировки и поиска с использованием ДДП, AVL дерева и хэш-таблиц

Оценка эффективности производится в последнем пункте меню, предыдущие используются лишь для проверки работоспособности структур данных.

Аварийные ситуации:

1. Считываемый файл кроме чисел и переносов строк содержит другие символы

Особенности реализации:

1. Если файл содержит несколько одинаковых значений, они будут перезаписаны

2. Имена используемых файлов находятся в заголовочных файлах:

```
#define DATABASE_FILENAME "database/numbers.txt"
```

В файле `presets.h`

```
#define ITERATIONS 10
#define SMALL_FILE_NAME "database/numbers_small.txt"
#define MEDIUM_FILE_NAME "database/numbers_medium.txt"
#define BIG_FILE_NAME "database/numbers_big.txt"
#define SORTED_FILE_NAME "database/numbers_sorted.txt"
#define SORTED_REVERSE_FILE_NAME "database/numbers_sorted_reverse.txt"
```

В файле `efficiency.h`

Описание структур данных:

```
typedef struct tree_node
{
    int value;
    unsigned char height;
    struct tree_node *left;
    struct tree_node *right;
} tree_node_t;
```

Структура узла дерева

value — значение узла, информация пользователя дерева

height — высота узла, игнорируется в функциях для работы с деревом двоичного поиска

left — указатель на левое поддерево

right — указатель на левое поддерево

```
typedef struct hash_node
{
    int value;
    struct hash_node *next;
} hash_node_t;

struct hashtable_opened
{
    hash_node_t **data;
    bool *exists;
    size_t records_amount;
    size_t max_size;
};

struct hashtable_closed
{
    int *data;
    bool *exists;
    size_t records_amount;
    size_t max_size;
};
```

Структуры хеш-таблиц

struct `hash_node` — структура записи хеш-таблицы с закрытой адресацией

`value` — значение записи

`next` — указатель на следующую запись с таким же хешем

struct `hashtable_opened` — структура хеш-таблицы с закрытой адресацией

`data` — указатель на массив записей таблицы (узлы односвязного списка)

struct `hashtable_closed` — структура хеш-таблицы с открытой адресацией

`data` — указатель на массив значений таблицы (целые числа)

`exists` — указатель на массив ключей, определяющих существование элемента в таблице

`records_amount` — количество записей в таблице

`max_size` — размер таблицы

```
typedef struct array
{
    int *data;
    size_t size;
    size_t max_size;
} array_t;
```

Структура массива чисел, используемая для обработки файла

`data` — указатель на динамический массив, хранящий числа из файла

`size` — текущий размер массива

`max_size` — максимальный размер массива

Константы, связанные с хешированием

```
#define HASH_OPENED_COEFFICIENT 0.72
#define HASH_CLOSED_COEFFICIENT 1.5
#define HASHTABLE_STEP_DIVISION 10
```

В файле hashtable.h

Процесс выделения памяти заключается в нахождении количества элементов N , которые будут храниться в таблице. Затем N умножается на `HASH_OPENED_COEFFICIENT` в случае хеш-таблицы с открытым хешированием и на `HASH_CLOSED_COEFFICIENT` в случае хеш-таблицы с закрытым хешированием. Затем ищется простое число, превышающее полученное, которое будет размером новой хеш-таблицы, измеряемой в записях, к которому можно обратиться за $O(1)$.

Хеш-функции

```
size_t
get_hash_opened(int key,
                size_t max_size)
{
    return key % max_size;
}
```

Хеш-функция, используемая хеш-таблицей с закрытым хешированием, где ключом является сам элемент (целое число).

```
size_t
get_hash_closed(int key,
                size_t max_size,
                size_t step,
                size_t iterations)
{
    return (key + iterations * step) % max_size;
}
```

Хеш-функция, используемая хеш-таблицей с открытым хешированием. Параметр `step` определяет шаг, с которым происходит смещение элементов с одинаковым хешем (от хеш-функции указанной выше) в таблице. В качестве этого аргумента передаётся размер таблицы (поле `max_size`

структуры `hashtable_closed`), делённое на коэффициент `HASHTABLE_STEP_DIVISION`, равный 10. Таким образом достигается равномерность распределения ключей в таблице.

Реализация подсчёта сравнений

Листинг кода для подсчёта количества сравнений на примере открытой хеш-таблицы (аналогичный код есть и в остальных структурах данных):

```
size_t hashtable_comparisons;

size_t
get_hashtable_comparisons(void)
{
    return hashtable_comparisons;
}

void
set_hashtable_comparisons(size_t new_comparisons)
{
    hashtable_comparisons = new_comparisons;
}

int
hashtable_opened_find(hashtable_opened_t *hashtable,
                      int value)
{
    size_t hash;
    hash_node_t *curr_node;

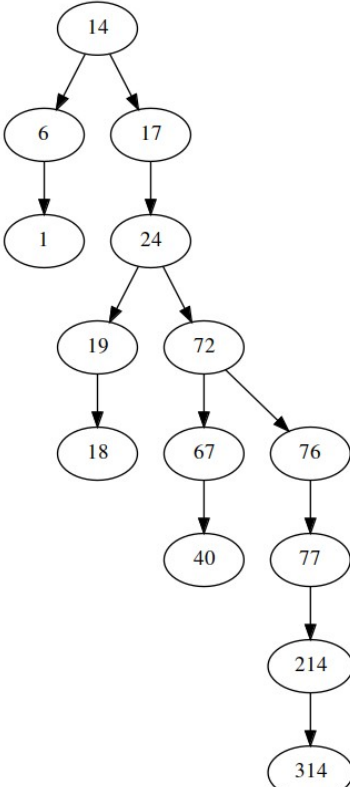
    hash = get_hash_opened(value, hashtable->max_size);

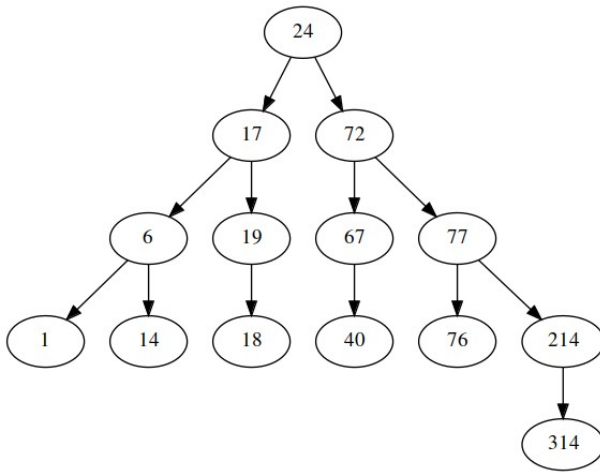
    curr_node = hashtable->data[hash];
    while (curr_node != NULL)
    {
        hashtable_comparisons++;
        if (curr_node->value == value)
        {
            return EXIT_SUCCESS;
        }
        curr_node = curr_node->next;
    }

    return ERR_NOT_FOUND;
}
```

Здесь `hashtable_comparisons` — статическая переменная, хранящая накопленное количество сравнений, она изменяется функцией поиска элемента в том же модуле, где и определена. Также её можно изменить и прочесть во внешних модулях с помощью функций `get_hashtable_comparisons` и `set_hashtable_comparisons`.

Функциональные тесты

Ввод	Вывод
<p>1. Загрузить числа из файла в AVL дерево, в ДДП и в хеш-таблицы</p>	<p>Очистка AVL дерева...</p> <p>Очистка ДДП...</p> <p>Очистка открытой хеш-таблицы...</p> <p>Очистка закрытой хеш-таблицы...</p> <p>Открытие файла на чтение...</p> <p>Запись в AVL дерево...</p> <p>Запись в ДДП...</p> <p>Запись в открытую хеш-таблицу...</p> <p>Запись в закрытую хеш-таблицу...</p> <p>Закрытие файла...</p>
<p>2. Вывести ДДП</p> <p>(После загрузки элементов из файла в структуры)</p>	 <pre> graph TD 14((14)) --> 6((6)) 14 --> 17((17)) 6 --> 1((1)) 17 --> 24((24)) 24 --> 19((19)) 24 --> 72((72)) 19 --> 18((18)) 72 --> 67((67)) 72 --> 76((76)) 67 --> 40((40)) 76 --> 77((77)) 77 --> 214((214)) 214 --> 314((314)) </pre>
<p>3. Вывести AVL дерево</p>	

<p>(После загрузки элементов из файла в структуры)</p>	 <pre> graph TD 24((24)) --> 17((17)) 24 --> 72((72)) 17 --> 6((6)) 17 --> 19((19)) 6 --> 1((1)) 6 --> 14((14)) 19 --> 18((18)) 72 --> 67((67)) 72 --> 77((77)) 67 --> 40((40)) 77 --> 76((76)) 77 --> 214((214)) 214 --> 314((314)) </pre>
<p>4. Вывести хеш-таблицы</p> <p>(После загрузки элементов из файла в структуры)</p>	<pre> Открытая хэш-таблица: Ключ Значения 1 1 6 6 11 314 12 214 14 14 17 17 18 18 19 19 24 24 40 40 67 67 72 72 76 76 77 77 Закрытая хэш-таблица: Ключ Значение 1 1 6 6 11 314 12 214 14 14 17 17 18 18 19 19 24 24 40 40 67 67 72 72 76 76 77 77 </pre>
<p>5. Найти элемент в AVL дереве, в ДДП, в хеш-таблицах и в файле</p>	<p>Поиск элемента в файле...</p>

<p>(После загрузки элементов из файла в структуры)</p> <p>Поиск числа 76, которое есть в файле</p> <p>(Максимальное среднее количество сравнений равно 2)</p>	<p>Элемент найден: 76</p> <p>Количество сравнений: 5</p> <p>Поиск элемента в AVL дереве...</p> <p>Элемент найден: 76</p> <p>Количество сравнений: 5</p> <p>Поиск элемента в двоичном дереве поиска...</p> <p>Элемент найден: 76</p> <p>Количество сравнений: 5</p> <p>Поиск элемента в открытой хэш-таблице...</p> <p>Элемент найден: 76</p> <p>Количество сравнений: 1</p> <p>Поиск элемента в закрытой хэш-таблице...</p> <p>Элемент найден: 76</p> <p>Количество сравнений: 2</p>
0. Выход	Осуществляется выход из программы
Вставка элемента, который находится в файле, но не в остальных структурах	<p>Вставка элемента в AVL дерево...</p> <p>Успешно</p> <p>Вставка элемента в ДДП...</p> <p>Успешно</p> <p>Вставка элемента в открытую хэш-таблицу...</p> <p>Успешно</p> <p>Вставка элемента в закрытую хэш-таблицу...</p>

	<p>Успешно</p> <p>Вставка элемента в файл...</p> <p>Элемент уже находится в файле</p>
Вставка элемента, который уже находится в файле и структурах	<p>Вставка элемента в AVL дерево...</p> <p>Элемент уже находится в AVL дереве</p> <p>Вставка элемента в ДДП...</p> <p>Элемент уже находится в ДДП</p> <p>Вставка элемента в открытую хэш-таблицу...</p> <p>Элемент уже находится в открытой хэш-таблице</p> <p>Вставка элемента в закрытую хэш-таблицу...</p> <p>Элемент уже находится в закрытой хэш-таблице</p> <p>Вставка элемента в файл...</p> <p>Элемент уже находится в файле</p>
Удаление элемента, который находится во всех структурах	<p>Удаление элемента из AVL дерева...</p> <p>Успешно</p> <p>Удаление элемента из ДДП...</p> <p>Успешно</p> <p>Удаление элемента из открытой хэш-таблицы...</p> <p>Успешно</p> <p>Удаление элемента из закрытой хэш-таблицы...</p> <p>Успешно</p> <p>Удаление элемента из файла...</p>

	Успешно
Удаление элемента, который не находится не в одной из структур	<p>Удаление элемента из АВЛ дерева...</p> <p>Элемент не найден в АВЛ дереве</p> <p>Удаление элемента из ДДП...</p> <p>Элемент не найден в ДДП</p> <p>Удаление элемента из открытой хэш-таблицы...</p> <p>Элемент не найден в открытой хэш-таблице</p> <p>Удаление элемента из закрытой хэш-таблицы...</p> <p>Элемент не найден в закрытой хэш-таблице</p> <p>Удаление элемента из файла...</p> <p>Элемент не найден в файле</p>

Вывод хеш-таблиц до и после
удаления элемента 72

```
Открытая хэш-таблица:
Размер: 11
Ключ    Значения
1        |23,      67
2        |123
3        |14
6        |17,      72,      6,      50
7        |18,      40
8        |19
9        |416
10       |76
```

```
Закрытая хэш-таблица:
Размер: 23
Ключ    Значение
0        |23
2        |40
3        |72
4        |50
6        |6
7        |76
8        |123
10       |416
14       |14
17       |17
18       |18
19       |19
21       |67
```

```
Открытая хэш-таблица:
Размер: 11
Ключ    Значения
1        |23,      67
2        |123
3        |14
6        |17,      6,      50
7        |18,      40
8        |19
9        |416
10       |76
```

```
Закрытая хэш-таблица:
Размер: 23
Ключ    Значение
0        |23
2        |40
4        |50
6        |6
7        |76
8        |123
10       |416
14       |14
17       |17
18       |18
19       |19
21       |67
Помеченные как удалённые:
3        |72
```


2. Вывести ДДП (До выполнения первого пункта меню)	Преобразование дерева в .dot файл... Дерево не имеет ни одного узла
3. Вывести AVL дерево (До выполнения первого пункта меню)	Преобразование дерева в .dot файл... Дерево не имеет ни одного узла
4. Вывести хеш-таблицы (До выполнения первого пункта меню)	Открытая хеш-таблица пуста Закрытая хеш-таблица пуста
5. Найти элемент в AVL дереве, в ДДП, в хеш-таблицах и в файле (До выполнения первого пункта меню) Поиск числа 123, которое есть в файле (из 12-ти элементов)	Поиск элемента в файле... Элемент найден: 123 Количество сравнений: 12 Поиск элемента в AVL дереве... Элемент не найден Количество сравнений: 1 Поиск элемента в двоичном дереве поиска... Элемент не найден Количество сравнений: 1 Поиск элемента в открытой хэш-таблице... Элемент не найден Количество сравнений: 1 Поиск элемента в закрытой хэш-таблице... Элемент не найден Количество сравнений: 1

Сравнение эффективности (на основе десяти замеров):

При каждом замере для поиска использовались случайные числа, находящиеся в соответствующих файлах.

Максимальное значение сравнений в данном сравнении, после которого происходит реструктуризация: **2,0**.

Поиск (указано среднее количество сравнений)

Структура данных	Маленький файл (100 элементов)	Средний файл (500 элементов)	Большой файл (1000 элементов)	Отсортированный файл (500 элементов)	Файл, отсортированный в обратном порядке (500 элементов)
АВЛ дерево	5.7	8.4	9.2	7.8	7.3
ДДП	6.3	14.9	8.9	284.7	217.3
Массив	48.3	203.4	346.5	284.7	217.3
Открытая хеш-таблица	1.3	1.0	1.4	1.1	1.0
Закрытая хеш-таблица	1.4	2.3	1.0	4.2	1.0
Открытая хеш-таблица (реструктуризированная)	-	-	-	-	-
Закрытая хеш-таблица (реструктуризированная)	-	1.0	-	1.0	-

Поиск

Так как время поиска прямо пропорционально количеству сравнений, проанализируем количество сравнений.

На файлах без с маленькой упорядоченностью данных ДДП и АВЛ деревья занимают примерно одинаковое количество сравнений. Однако в отсортированных файлах лучше себя показало АВЛ дерево, выигрывая до 37 раз по количеству сравнений у ДДП, которое по эффективности стало в точности как массив. В каждом случае деревья проигрывают хеш-таблицам в десятки (АВЛ дерево) и сотни (ДДП) раз.

Таблица с закрытой адресацией в среднем потребовала не больше 1,4 сравнений.

Таблица с открытой адресацией на некоторых файлах заняла большее количество сравнений, чем таблица с закрытой адресацией. Решить эту проблему помогла реструктуризация второй таблицы с увеличением её размера вдвое, поиск в таблице с увеличенным размером стал занимать в среднем 1 сравнение.

Занимаемая память (в байтах)

Структура данных	Маленький файл (100 элементов)	Средний файл (500 элементов)	Большой файл (1000 элементов)	Отсортированный файл (500 элементов)	Файл, отсортированный в обратном порядке (500 элементов)
АВЛ дерево	2400	12000	24000	12000	12000
ДДП	2400	12000	24000	12000	12000
Массив	424	2024	4024	2024	2024
Открытая хеш-таблица	2289	12455	26031	12455	12455

Закрытая хеш-таблица	787	3787	7587	3787	3787
Открытая хеш-таблица (реструктуризированная)	-	-	-	-	-
Закрытая хеш-таблица (реструктуризированная)	-	7587	-	7587	-

Память

Больше всего места заняла хеш-таблица с закрытой адресацией. Она занимает на 8% больше памяти, чем двоичное дерево поиска. Самой эффективной по памяти структурой данных является массив, так как он не требует никаких дополнительных полей, кроме, возможно, своего размера. Хеш-таблица с закрытой адресацией проигрывает массиву по памяти примерно в 6,5 раз.

Однако результаты намного лучше для хеш-таблицы с открытой адресацией. Она занимает примерно в 1,9 раз больше места, чем массив и примерно в 3,1 раза меньше места, чем двоичное дерево поиска.

Не потребовалась реструктуризация таблицы с закрытым хешированием, так как среднее количество сравнений при поиске не превысило 2,0.

Даже реструктуризированные таблицы с открытой адресацией занимают в среднем в 2 раза меньший объем памяти, чем таблицы с закрытой адресацией

Вывод

Хеш-таблица с закрытой адресацией в среднем показала себя лучше, чем таблица с открытой адресацией, но и потребовала значительно больше памяти.

Двоичные деревья поиска имеет смысл использовать на не слишком большом количестве элементов. Когда элементов много, выгоднее использовать хеш-таблицы. Однако сложность поиска в АВЛ дереве $O(\log_2 n)$, а в хеш-таблице — $O(n)$, но это верно только для таблиц с большим количеством коллизий, в среднем поиск в хеш-таблице с малым числом коллизий занимает $O(1)$.

Массив — самая маленькая структура данных, однако и требует больше всего сравнений при поиске.

Создание оптимальной по размеру хеш-таблицы с малым количеством коллизий представляет собой достаточно трудную задачу, так как при создании хеш-таблицы необходимо найти компромисс между её размером и количеством коллизий, а также правильный выбор хеш-функции.

Контрольные вопросы:

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В идеально сбалансированном дереве количество вершин в каждом поддереве различается не больше, чем на 1.

В АВЛ дереве для каждой его вершины высота двух её поддеревьев различается не более, чем на 1.

2. Чем отличается поиск в АВЛ дереве от поиска в дереве двоичного поиска?

В АВЛ дереве поиск в среднем работает быстрее за счёт меньшей высоты поддеревьев, а следовательно и меньшего количества сравнений.

Однако возможны случаи, когда поиск в ДДП будет быстрее за счёт малой высоты его поддеревьев, в которых содержится искомое число (или не содержится, если данного числа нет).

3. Что такое хеш-таблица, каков принцип её построения?

Хеш-таблица — массив, заполненный в порядке, определенным хеш-функцией,

4. Что такое коллизии? Каковы методы их устранения.

Коллизия — ситуация, когда у разных элементов совпадают их хеш-значения.

Коллизии устраняются с помощью открытой адресации, при которой элементы с одинаковым хеш-значением образуют список, дерево, или другую структуру данных, по которой продолжается поиск, либо закрытой адресации, при которой хеш-значение продолжает вычисляться до тех пор, пока не будет найден искомый элемент или не найдётся пустая ячейка.

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

Поиск в хеш-таблицах становится неэффективен при большом количестве коллизий

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

Для ДДП: $O(h)$, где h — высота дерева (в среднем), $O(n)$ — худший случай

Для АВЛ: $O(\log_2 n)$

хеш-таблица:

без коллизий: $O(1)$

с коллизиями: $O(n)$