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)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

«СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ»

по курсу:

«ТИПЫ И СТРУКТУРЫ ДАННЫХ»

Вариант: 1

Студент:
Авдейкина Валерия Павловна, группа ИУ7-33Б _____
(подпись, дата)

Руководители:
Преподаватель ИУ7 _____
Силантьева Александра Васильевна (подпись, дата)
Преподаватель ИУ7
Барышникова Марина Юрьевна _____
(подпись, дата)

Проверяющий:
Барышникова Марина Юрьевна _____
(подпись, дата)

Оценка: _____

2022 г.

Оглавление

Описание условий задачи

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.

Вариантное задание: Используя предыдущую программу (задача No6), 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.

Техническое задание

1 Входные данные

- Входные данные при вводе номера выбранной опции меню:
  - Целые беззнаковые числа [0; 14]
- Входные данные для различных опций меню:

Номер опции	Название опции	Входные данные
0	Выход	-
1	Задать файл для обрабтки	Имя файла, код адресации
2	Вывести на экран ДДП, АВЛ-дерево	-
3	Вывести на экран хеш-таблицу	-
4	Выполнить поиск с анализом	Максимальное количество сравнений
5	Создать новый файл с данными	Имя файла
6	Добавить новый элемент	Значение элемента
7	Выполнить поиск в структурах	Значение элемента
8	Очистить все	-
9	Изменить адресацию хеш-таблицы	-
10	Вывести меню	-
11	Вывести правила ввода	-

Таблица 1: Входные данные опций меню

- Имя файла: символы, не более 255
- Значение элемента: целое число из [-2147483648; 2147483647]
- Максимальное количество сравнений: вещественное число > 1
- Код адресации: 0 (закрытая), 1 (открытая)

2 Выходные данные

- Выходные данные при вводе номера выбранной опции меню:
  - Выходные данные команды в случае корректного ввода
- Выходные данные для различных опций меню:

Номер	Название опции	Выходные данные
0	Выход	Сообщение об успешном выполнении / сообщение об ошибке
1	Задать файл для обрабтки	Сообщение об успешном выполнении / сообщение об ошибке
2	Вывести на экран ДДП, АВЛ-дерево	Изображения деревьев
3	Вывести на экран хеш-таблицу	Хеш-таблица
4	Выполнить поиск с анализом	Результаты анализа
5	Создать новый файл с данными	Сообщение об успешном выполнении / сообщение об ошибке
6	Добавить новый элемент	Сообщение об успешном выполнении / сообщение об ошибке
7	Выполнить поиск в структурах	Сообщение об успешном выполнении / сообщение об ошибке
8	Очистить все	Сообщение об успешном выполнении / сообщение об ошибке
9	Изменить адресацию хеш-таблицы	Сообщение об успешном выполнении / сообщение об ошибке
10	Вывести меню	Меню
11	Вывести правила ввода	Правила ввода

Таблица 2: Выходные данные опций меню

3 Задача, реализуемая программой

Программа реализует обработку набора целых чисел с помощью составления двоичного дерева поиска, его балансировки и хеш-таблицы и оценивает эффективность разных способов поиска элемента.

4 Способ обращения к программе

Программа вызывается в командной строке без каких-либо аргументов. Ввод данных производится с клавиатуры после соответствующего приглашения к вводу («`Команда...`» / «`Введите..`»).

5 Возможные аварийные ситуации и ошибки со стороны пользователя

1. Некорректный ввод имени файла:
    1.1. Пустой ввод
    1.2. Недопустимые символы
    1.3. Превышение допустимого количества символов *(поведение программы не определено)*
    1.4. Имя несуществующего файла или директории
2. Некорректные данные в обрабатываемом файле / вводе с клавиатуры
3. Некорректный ввод номера опции:
    3.1. Пустой ввод
    3.2. Недопустимые символы (*поведение программы не определено*)
    3.3. Недопустимый номер
4. Вставка элемента в заполненную хеш-таблицу без реструктуризации

Описание внутренних структур данных

В ходе работы были составлены следующие структуры:

1. Дерево двоичного поиска — АТД, тип которого — tree_t, указатель на структуру tree_node:
   - struct tree_node:
     - key (метка узла) — поле типа int (key_t)
     - left (левый потомок) — поле типа tree_t (struct tree_node*)
     - right (правый потомок) — поле типа tree_t (struct tree_node*)

```
typedef struct tree_node* tree_t;
typedef int key_t;

struct tree_node
{
   key_t key;
   tree_t left;
   tree_t right;
};
```

Итоговый размер структуры: 20 байт.

2. Сбалансированное двоичное дерево (АВЛ-дерево) — АТД, тип которого — avltree_t, указатель на структуру avltree_node:
   - struct avltree_node:
     - key (метка узла) — поле типа int (key_t)
     - left (левый потомок) — поле типа avltree_t (struct avltree_node*)
     - right (правый потомок) — поле типа avltree_t (struct avltree_node*)
     - height (высота узла) — поле типа size_t

```
typedef struct tree_node* tree_t;
typedef int key_t;

struct tree_node
{
   key_t key;
   tree_t left;
   tree_t right;
   size_t height;
};
```

Итоговый размер структуры: 28 байт.

3. Хеш-таблица struct hash_table (данные — массив из элементов типа hash_elem):

```
struct hash_elem
{
   int not_empty;
   key_t key;
   hashtable_t next;
};

typedef struct hash_table *hash_t;

typedef struct hash_elem *hashtable_t;
typedef int (*hash_func_pt)(hash_t table, const key_t key);

struct hash_table
{
   hashtable_t data;
   size_t size;
   size_t comparisons;
   size_t collisions;
   size_t try_count;
   int addr_code;
   hash_func_pt hash_func;
};
```

Итоговый размер структуры: 48 байт.

Алгоритм

1. Вставка в АВЛ-дерево
   1.1. Поиск в АВЛ-дереве аналогичен поиску в ДДП
   1.2. Основные действия с вершиной (перераспределение указателей): малый поворот влево, малый поворот вправо, большой поворот вправо, большой поворот влево
   1.3. При вставке узла проверяется его наличие в дерево
   1.4. После рекурсивной вставки перерасчитываются высоты каждого узла и проводится рекурсивная ребалансировка:
      1.4.1. Если баланс-фактор (*разность высот правого и левого поддеревьев*) узла меньше -1 и при этом баланс-фактор левого поддерева:
         a) равен -1
            - происходит малый поворот узла вправо
         b) равен 1
            - происходит большой поворот узла вправо
      1.4.2. Если баланс-фактор узла больше 1 и при этом баланс-фактор правого поддерева:
         a) равен 1
            - происходит малый поворот узла влево
         b) равен -1
            - происходит большой поворот узла влево
2. Заполнение хеш-таблицы и поиск
   - Реструктуризация осуществляется с помощью увеличения размера хеш-таблицы в k = 1,5 раза
   - Размер хеш-таблицы является наименьшим простым числом, превосходящим количество элементов * k = 1,5
   - Хеш-таблица может быть заполнена двумя способами, которые отличаются методами разрешения коллизий (случаев совпадения значений хеш-функции каких-либо элементов):

2.1. Открытая адресация

- В хеш-таблице не используется возможность создавать списки
- **Хеш-функция**: hash = (hash + try_count ^ 2) % size
  - size — размер таблицы
  - key — ключ элемента
  - try_count — номер попытки вставки элемента в таблицу
- В случае *коллизии* try_count увеличивается на 1, вычисляется новый хеш и проверяется, свободен ли элемент с новым хешем; если нет — все повторяется (до тех пор, пока не найдется «пустое место»)
- *Поиск элемента* осуществляется с аналогичным использованием хеш-функции до того момента, пока не будет обнаружен пустой элемент хеш-таблицы

2.2. Закрытая адресация

- В хеш-таблице используются линейные односвязные списки
- **Хеш-функция**: hash = key % size
  - key — ключ элемента
  - size — размер таблицы
- В случае коллизии элемент добавляется в конец списка, связанного с полученным хешем
- *Поиск элемента* осуществляется с использованием хеш-функции для получения соответствующего хеша и дальнейшим простым перебором связанного с этим хешем списка

В реализации решения задачи были использованы следующие основные функции:

Функция	Описание
`int _balance_factor(avltree_t node)`	Получение баланс-фактора узла
`avltree_t rotate_right(avltree_t node)`	Малый поворот узла вправо
`avltree_t rotate_left(avltree_t node)`	Малый поворот узла влево
`avltree_t rotate_left_right(avltree_t node)`	Большой поворот узла вправо
`avltree_t`	Большой поворот узла влево

`rotate_right_left(avltree_t node)`	
`avltree_t _rebalance(avltree_t tree)`	Ребалансировка tree
`avltree_t avl_insert_node(avltree_t tree, avltree_t node)`	Вставка узла node в дерево tree
`avltree_t avl_search_node(avltree_t tree, const key_t key)`	Поиск ключа key в дереве tree
`int closed_hash_func(hash_t table, const key_t key)`	Хеш-функция открытой адресации
`int opened_hash_func(hash_t table, const key_t key)`	Хеш-функция закрытой адресации
`int closed_insert(hash_t table, const key_t key)`	Вставка в хеш-таблицу с открытой адресацией
`int opened_insert(hash_t table, const key_t key)`	Вставка в хеш-таблицу с закрытой адресацией
`int fill_hashtable(FILE *const f, hash_t table)`	Заполнение таблицы table из файла f
`int restructure_hashtable(FILE *const f, hash_t table)`	Реструктуризация таблицы table с данными из файла f
`int search_hashtable(hash_t table, const key_t key, const int code)`	Поиск key в table; code — VERBOSE / QUIET

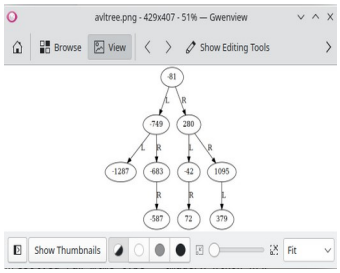
Таблица 3. Описание основных функций

Тестирование

- Позитивные тесты
    - **Команда 1**

```
Команда (10 - меню, 11 - правила): 1
Введите имя файла: ./data/10.txt
Файл с данными был успешно задан
Введите код адресации (0 - закр., 1 - откр.): 1
ДДП было успешно заполнено
АВЛ-дерево было успешно заполнено
Хеш-таблица была успешно заполнена
```

    - **Команда 2**



    - **Команда 3**

```
Команда (10 - меню, 11 - правила): 3
Текущая хеш-таблица:
Тип адресации: ОТКРЫТАЯ
|----------|----------|
| Хеш      | Значение |
|----------|----------|
| 0        | -749     |
|----------|----------|
| 4        | 72       |
|----------|----------|
| 5        | -81      |
|----------|----------|
| 6        | -1287    |
|----------|----------|
| 7        | 1095     |
|----------|----------|
| 8        | 280      |
|----------|----------|
| 9        | 379      |
|----------|----------|
| 10       | -42      |
|----------|----------|
| 11       | -587     |
|----------|----------|
| 15       | -683     |
|----------|----------|

Размер таблицы: 17
Количество КОЛЛИЗИЙ: 6
```

    - **Команда 6**

```
Команда (10 - меню, 11 - правила): 6
Введите элемент: 10
Элемент 10 был успешно вставлен в хеш-таблицу
Элемент 10 был успешно добавлен в файл ./data/20.txt
Элемент 10 был успешно вставлен в ДДП
Элемент 10 был успешно вставлен в АВЛ-дерево
```

    - **Команда 7**

```
Команда (10 - меню, 11 - правила): 7
Введите элемент: 10

Элемент 10 найден в ДДП: 10 [ L: -, R: -]
- Кол-во сравнений: 2
Элемент 10 найден в АВЛ-дереве: 10 [ L: -, R: -, H = 1]
- Кол-во сравнений: 4
Элемент 10 найден в хеш-таблице, хеш: 10
- Кол-во сравнений: 1
```

    - **Команда 8, 9**

```
Команда (10 - меню, 11 - правила): 9
Хеш-таблица была успешно заполнена

Команда (10 - меню, 11 - правила): 8
Выбор файла был успешно сброшен
ДДП было успешно очищено
АВЛ-дерево было успешно очищено
Хеш-таблица была успешно очищена
```

- Негативные тесты: вызов при пустом дереве / некорректный ввод

Оценка эффективности

Время поиска замерялось в двух плоскостях: в зависимости от количества входных чисел и в зависимости от типа адресации хеш-таблицы (а также при ее реструктуризации). Количество итераций для всех плоскостей: 20. В верхнем левом углу указано количество чисел в файле. Реструктуризация проводилась при количестве сравнений, превышающем 4.

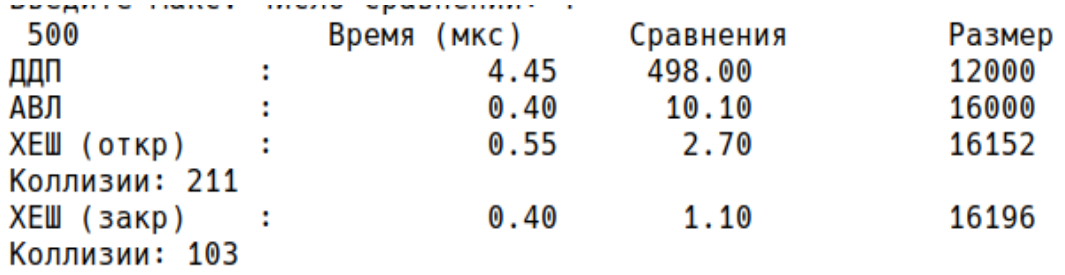


Рисунок 1: Результат для 500 чисел в случайном порядке

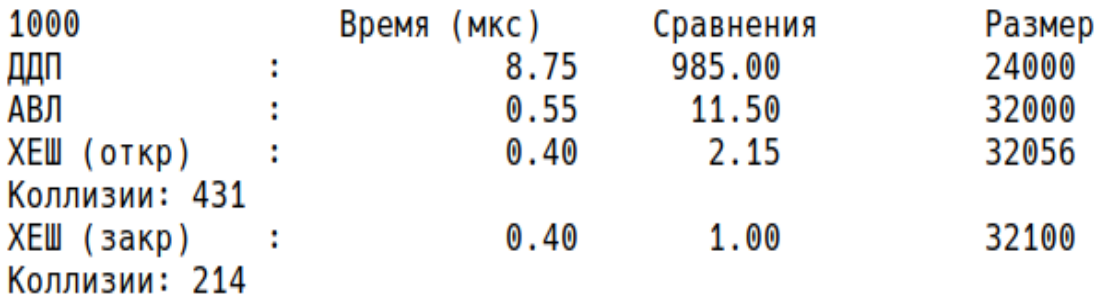


Рисунок 2: Результат для 1000 чисел в случайном порядке

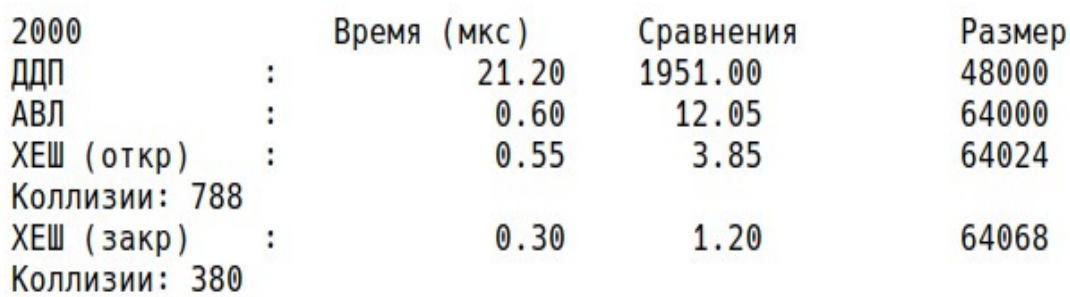


Рисунок 3: Результат для 2000 чисел в случайном порядке

```
 500                Время (мкс)       Сравнения          Размер
ДДП             :             5.35      500.00          12000
АВЛ             :             0.50       10.00          16000
ХЕШ (откр)      :             0.60        9.15          16152
Коллизии: 0
ХЕШ (ге)        :             0.45        5.50          32476
Коллизии: 0
ХЕШ (закр)      :             0.45        1.00          16196
Коллизии: 0
```

Рисунок 4: Результат для 500 чисел в порядке возрастания

```
 500                Время (мкс)       Сравнения          Размер
ДДП             :             5.00      500.00          12000
АВЛ             :             0.35       10.00          16000
ХЕШ (откр)      :             0.45        9.15          16152
Коллизии: 0
ХЕШ (ге)        :             0.50        5.50          32476
Коллизии: 0
ХЕШ (закр)      :             0.35        1.00          16196
Коллизии: 0
```

Рисунок 5: Результат для 500 чисел в обратном порядке

Исходя из результатов, можно сделать следующие выводы:

- Поиск в двоичном дереве без балансировки является наименее эффективным по времени для всех случаев, но наиболее экономным по памяти
- Поиск в АВЛ-дереве намного более эффективен по времени, чем поиск в двоичном дереве поиска, и более эффективен в случае отсортированных данных в сравнении с хеш-таблицей
- Поиск в хеш-таблице является наиболее эффективным как по времени, так и по количеству сравнений
- При как-либо отсортированных данных поиск в ДДП содержит в среднем максимальное число сравнений
- В силу сторонних процессов появляется большая погрешность при измерении времени в мск и эффективность поиска в хеш-таблице становится более заметной лишь при больших размерах входных данных

Контрольные вопросы

1 Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В АВЛ-дереве менее строгие требования: для каждой вершины разница максимальных высот левого и правого поддеревьев не больше 1, а для идеального — любых высотю

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

Поиск в АВЛ-дереве ничем не отличается от поиска в ДДП, однако в силу особенностей структур он быстрее.

3 Что такое хеш-таблица, каков принцип ее построения?

Хеш-таблица — массив, заполненный в порядке, определенном хеш-функцией. Принцип построения: по ключу элемента рассчитывается хеш, который является индексом в хеш-таблице.

4 Что такое коллизии? Каковы методы их устранения.

Коллизии — случаи, когда хеши двух разных элементов совпадают. Методы разрешения: открытая адресация (хеш увеличивается на количество попыток вставить) и закрытая (метод цепочек, для каждого хеша создается связный список).

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

Поиск в хеш-таблицах становится неэффективен, когда сравнений становится больше 2-4 (много данных с одинаковыми хешами)

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

Поиск в ДДП более эффективен по времени, чем поиск в файле.

Поиск в АВЛ-дереве более эффективен по времени, чем поиск в ДДП.

Поиск в хеш-таблице более эффективен по времени, чем поиск в АВЛ-дереве.

Выводы

В ходе работы было выяснено, что использование хеш-таблицы для поиска целых чисел является более эффективным по времени по сравнению с ДДП и АВЛ-деревьями, но в то же время используется больший объем памяти.