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 №7 «СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ»

Студент Родинков Алексей Глебович

Группа ИУ7 – 31БВ

Преподаватель Силантьева Александра Васильевна

2023 год.

Оглавление

Описание условия задачи 2

Описание технического задания 3

Описание структуры данных 4

Оценка эффективности (такты) 4

Описание алгоритма 5

Ответы на контрольные вопросы 5

Описание условия задачи

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.

Используя предыдущую программу (задача №6), построить дерево, например, для следующего выражения: 9+(8*(7+(6*(5+4)-(3-2))+1)). При постфиксном обходе дерева, вычислить значение каждого узла и результат записать в его вершину. Получить массив, используя инфиксный обход полученного дерева. Построить для этих данных дерево двоичного поиска (ДДП), сбалансировать его. Построить хеш-таблицу для значений этого массива. Осуществить поиск указанного значения. Сравнить время поиска, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев и хеш-таблиц.

Описание технического задания

Входные данные:

Непустой файл

Выходные данные:

Полученное ДДП, АВЛ-дерево, Хеш-таблица (деревья), Хеш-таблица(списки), Хеш-таблица без коллизий.

Обращение к программе:

Запускается через терминал командой: ./app.exe.

Аварийные ситуации:

1. Пустой файл
2. Ошибка выделения памяти.
3. Файл содержит слово с дефисом (из-за, из-под).

Описание структуры данных

Структура узла АВЛ дерева.

```
struct tree_t
{
    int counter;     // - количество повторений
    char *word;     // - слово
    struct tree_t *right; // - указатель на правое  дерево
    struct tree_t *left; // - указатель на левое дерево
};
```

Структура узла бинарного дерева.

```
struct tree_t
{
    int counter;     // - количество повторений
    char *word;     // - слово
    struct tree_t *right; // - указатель на правое  дерево
    struct tree_t *left; // - указатель на левое дерево
};
```

Структура Хеш-таблицы (деревья)
Дескриптор.

```c
// Структура открытого типа хеширования с коллизиями в виде ДДП
struct hash_table_col_t
{
    itemtree_t **itemtree;  // адрес начала массива элементов (деревья)
    size_t size;   // размер таблицы
    size_t count; // количество неповторяющихся элементов в хеш таблице
    size_t col_amount; // количество коллизий в хеш таблице
};

// Структура хеш-таблицы для обработки данных с коллизией (в виде дерева)
typedef struct hash_table_col_t tablec_t;
```

Элемент Хеш-таблицы.

```c
/* Блок работы с деревьями */
// единица данных для хеш функции с коллизией (дерево)
struct hashtree_item_t
{
    size_t key;         // ключ для поиска
    tree_t *tree_head; //  указатель на голову
};


typedef struct hashtree_item_t itemtree_t;
```

Элемент дерева.

```c
struct tree_t
{
    int counter;     // - количество повторений
    char *word;     // - слово
    struct tree_t *right; // - указатель на правое  дерево
    struct tree_t *left; // - указатель на левое дерево
};

typedef struct tree_t tree_t;
```

Структура Хеш-таблицы (списки)
Дескриптор.

```
// Структура открытого типа хеширования с коллизиями в виде односвязного линейного списка
struct hash_table_node_t
{
    itemnode_t **itemnode; // адреса начала массива элементов (списков)
    size_t size; // размер таблицы
    size_t count; // количество неповторяющихся элементов в хеш таблице
    size_t col_amount; // количество коллизий в хеш таблице
};


// Структура хеш-таблицы для обработки данных с коллизией (в виде списков)
typedef struct hash_table_node_t tablen_t;
```

Элемент Хеш-таблицы.

```
/* Блок работы со списками */
// единица данных для хеш функции с коллизией (список)
struct itemnode_t
{
    size_t key; // ключ для поиска ячейки хеш таблицы
    node_t *node_head; // указатель на голову списка
};


typedef struct itemnode_t itemnode_t;
```

Элемент списка.

```
typedef struct node_t node_t;

struct node_t
{
    char *word; // слово
    int count; // количество повторений слова
    node_t *next; // Указатель на следующий
};
```

Структура Хеш-таблицы закрытого типа массив.
Дескриптор.

```c
// Структура закрытого типа
struct hash_table_t
{
    item_t **item;  // адрес начала массива элементов
    size_t size;   // размер таблицы
    size_t count; // количество элементов в хеш таблице
};

//
typedef struct hash_table_t table_t;
```

Элемент хеш-таблицы.

```c
struct hash_item_t
{
    size_t key;  // ключ хеш таблицы
    int counter; // количество повторений
    char *word; // слово
};

typedef struct hash_item_t item_t;
```

Структура основного меню.

```
enum MENU
{
    EXIT,
    PRINTTREE,
    ADDWORDINTREE,
    ADDWORDINFILE,
    OPENNEWFILE,
    FIND,
    DELETEINTREE,
    DELETEINFILE,
    WORK_WITH_HASHTABLE,
    WORK_WITH_HASHTABLE_LIST,
    WORK_WITH,
};
```

Структура дополнительного меню для каждой из реализаций хеш-таблицы.

```
enum HASHMENU
{
    INITHASHTABLE,
    ADDTOHASH,
    DELETEFROMHASH,
    CHANGEHASHFUNCION,
    RESTRUCTHASH,
    FINDINHASH,
    ELEMS,
    EXITHASH,
};
```

Описание алгоритма

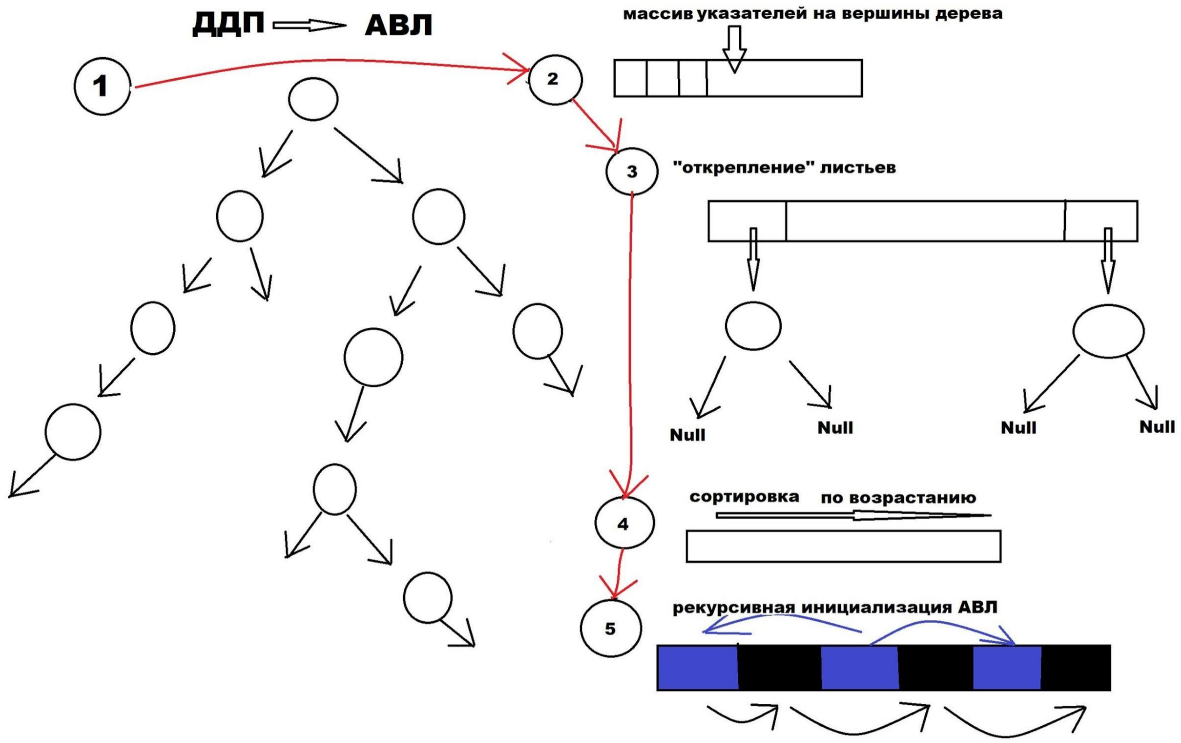
1. Выбор пункта меню

```
aleksei@aleksei-ubuntu:~/Tisd/lab_07/lab_07_app$ ./app.exe
Программа создает частотный словарь на основе файла
Открыт файл text.txt
Стандартный размер таблиц: 50

Основное меню программы
        0 - Из ДДП в АВЛ
        1 - Вывод дерева
        2 - Добавление слова в дерево
        3 - Добавление слова в файл
        4 - Открыть другой файл
        5 - Поиск
        6 - Удаление в дереве
        7 - Удаление в файле
        8 - Работа с хеш-таблицей( деревья )
        9 - Работа с хеш-таблицей( список )
        h - Закрытое хеширование
        e - Завершение работы программы
```

```
Хеш меню
        1 - Вывод
        2 - Добавление в хеш-таблицу
        3 - Удаление из хеш-таблицы
        4 - Изменение хеш-функция
        5 - Реструктуризация хеш-функции
        6 - Поиск в хеш-функции
        7 - Количество неповторяющихся элементов
        8 - Завершение работы программы
```

2) Алгоритм создания АВЛ дерева на основе ДДП:



- Создаем массив указателей на листья дерева.

- Откребляем листья (ставим указатели на Null).

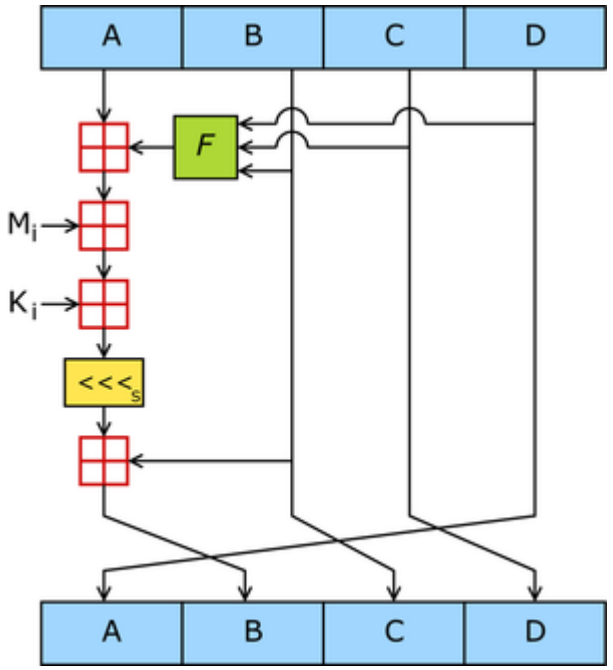
- Сортируем массив данных по возрастанию.

- Рекурсивно заполняем дерево (в отсортированном массиве каждый следующий элемент поддерева будет средним в указанном диапазоне). Т.е. массив (вектор 1 2 3 4 5 6 7 8 9 будет всегда преобразовываться в АВЛ вида (1) 2) (3) 4) (5) (6 (7) (8 (9).

3) Функции Хеширования (Стандартная, MD5, Керниган)

1.Стандартная функция подсчитывает количество байт и берет остаток от деления на размер таблицы.

2.MD5 представляет собой деление на константные значения, в 3 прохода, после каждого результат побитово сдвигается



3.Керниган представляет собой сумму всех остатков от деления следующего значения на предыдущее, результат которого делится на размер хеш таблицы.

4.Алгоритм закрытого Хеширования (для поиска места вставки в массив, линейный тип)

- Проверяет следующий элемент массива, если он занят, то переходит к следующему (кольцевая, т.е. при бОльшем значении позиции проверяет мЕньшие на наличие свободного места)

5.Алгоритм удаления из хеш-таблицы, реализованной на списках (Есть четыре ситуация удаления из списка: - Удаление головы:

- Следующий элемент существует → переназначим голову (head = head→next, а текущую голову очищаем)

- Следующий элемент пуст (очищаем голову, а указателю из массива указателей присваиваем значение Null)
- Удаление элемента:
- Следующий элемент существует →находим предыдущий элемент, переназначим следующий на следующего от удаляемого элемента, удаляем.
- Следующий элемент пуст -> очищаем удаляемый элемент.

6.Алгоритм добавления в дерево:

- Создается лист на основе слова, считанного из текста. Указатели на детей устанавливаются в «Null». Путем сравнения со словом в текущем корне дерева лист переходит либо в правое поддерево, либо в левое. Если указатель на левое или правое поддерево пуст, то «присоединяем» лист в нужное поддерево.

7.Алгоритм удаления слова из ДДП :

- Поиск заданного слова. Если найденный лист имеет только одного ребенка, то мы добавляем на место удаляемого листа лист с минимальным словом из соответствующего поддерева. Если найденный лист имеет два ребенка, то мы добавляем на место удаляемого листа лист, содержащий минимальное слово из правого поддерева.

8.Алгоритм реструктуризации хеш-таблицы:

- Открывается новый файл, туда записываются элементы дерева (слово→количество повторений) . Удаляется дерево / список каждого элемента. Удаляется массив указателей. Освобождается память из-под дескриптора. Создается новая хеш-таблица исходя из размеров.

Набор тестов

(негативные)

№	Название теста	Пользовательский ввод	Вывод
1	Некорректный ввод данных	a	Введите корректный пункт меню
2	Пустой файл	Empty.txt	Пустой файл!
3	Файл содержащий слово с дефис	Из-за	Не могу прочитать файл. Измените слова с дефисом.
4	Невозможно выделить память для реструктуризации хеш-таблицы	100000000	Ошибка выделения памяти!

(позитивные)

№	Название теста	Пользовательский ввод	Вывод
1	Удаление головы списка	малое	Слова малое удалено
2	Удаление головы списка со следующим элементом	отомстить	Слово отомстить удалено

3	Удаление элемента из списка	уродливый	Слово уродливый удалено
4	Добавление слова в список	Харам	Слово Харам добавлено
5	Добавление слова в хеш-таблицу закрытого типа.	екой	Слово екой добавлено в первый пустой элемент
6	Удаление слова из хеш-таблицы закрытого типа.	екой	Слово екой удалено.
7	ДДП → АВЛ.	(пункт меню) 0	Дерево составлено
8	Открытие нового файла	new.txt	Хеш-таблицы и деревья проинициализированы.
9	Высокий процент содержания коллизий	Реструктуризация (размер 10 новой хеш-таблицы)	Внимание высокий процент содержания коллизий,

			рекомендуется добавить изменить хеш функцию или реструктурировать таблицу.
10	Изменение хеш функции		Распределение слов другое, хеш функция успешно изменена

Оценка эффективности (мс)

	Кол-во неповторяющихся элементов	ДДП	АВЛ-дерево	Размер хеш-таблицы		Хеш-таблица (открытый тип деревья)		Хеш-таблица (открытый тип списки)		Хеш-таблица закрытый тип	
Усредненное время поиска (всех слов из файла)	20	1	1	23	-	1		1		1	
	428	12	6	50	500	3 / 2		2 / 2		3	
	920	15	8	50	620	4 / 1		5 / 1		4	
Усредненное кол-во сравнений	20	4.062	2.25	23	-	1.25		1.25		1.88	
	428	10	7	50	500	2.75	1	6.15	1	-	16
	920	12	8.33	50	940	3.65	1	12.562	1	-	32

АВЛ-дерево вследствие своей балансировки работает в два раза быстрее, чем ДДП. По этой же причине у АВЛ меньшее кол-во сравнений. Время поиска в хеш-таблице сравнимо с поиском в АВЛ дереве.

Память (байт)

ДДП	АВЛ-дерево	Хеш-таблица закрытого типа	Хеш-таблица открытого типа деревья	Хеш-таблица открытого типа списки
16 + данные	16 + данные	12 + данные	24 + данные	20 + данные

Двоичное дерево занимает столько же места, сколько и АВЛ-дерево, т.е. в 0.5 раза больше, чем хеш-таблица закрытого типа

Ответы на контрольные вопросы

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В идеально сбалансированном дереве кол-во элементов в правом и левом поддереве отличается не более чем на единицу. В АВЛ дереве высоты правого и левого поддерева отличается не более чем на единицу

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

Алгоритм одинаков.

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

Структура данных позволяющая получать по ключу элемент массива называется хеш-таблицей.

Для доступа по ключу используется хеш-функция. Она по ключу получает нужный индекс массива. Хеш-функция должна возвращать одинаковые значения для одного ключа и использовать все индексы с одинаковой вероятностью.

4. Что такое коллизии? Каковы методы их устранения.

Ситуация, когда из разных ключей хеш-функция выдаёт одни и тот же индекс, называется коллизией.

Метод цепочек – при коллизии элемент добавляется в список элементов этого индекса.

Линейная адресация – при коллизии ищется следующая незаполненная ячейка.

Произвольная адресация - используется заранее сгенерированный список случайных чисел для получения последовательности.

Двойное хеширование – использовать разность 2 разных хеш-функций.

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

При большом количестве коллизий.

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

Скорость поиска в хеш-таблице зависит от числа коллизий. При небольшом числе коллизий для поиска элемента совершается мало сравнений и поиск быстрее чем в деревьях.
АВЛ дерево быстрее при поиске за счёт более равномерного распределения элементов чем в ДДП.

Вывод

Скорость работы хеш-таблицы сравнима со скоростью работы АВЛ-дерева. Поиск в этих структурах данных выполнялся быстрее, чем в двоичном дереве поиска — эта характеристика зависит от выбранной хеш-функции, которая должна обеспечить как можно меньшее кол-во коллизий.

Деревья лишены этого недостатка.
Очевидно, что АВЛ-деревья более эффективны, чем Бинарные, так как высота АВЛ-дерева не превышает 1.44 * log(N), где N — кол-во узлов, но тратится время на балансировку после вставки узла.

По памяти самое эффективное решение – хеш-таблица.
По времени эффективнее других структур данных также оказалась хеш-таблица.