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»

по курсу «Типы и структуры данных»

Студент: Шимшир Эмирджан Османович

Группа: ИУ7-33Б

Студент _____ Шимшир Э.О.

подпись, дата *фамилия, и.о.*

Преподаватель _____ Барышникова М.Ю.

подпись, дата *фамилия, и.о.*

Оценка _____

Условие задачи

Используя предыдущую программу (задача No6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из слов текстового файла. Осуществить поиск введенного слова 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

Входные данные

Для корректной работы программы нужно заполнить из файла двоичное дерево поиска, сбалансированное дерево и хеш-таблицу.

Выходные данные

Вывод структур данных и результатов измерения времени при поиске элементов в структурах данных и файле.

Задачи, реализуемые программой

1. Ввод данных их файла
2. Вывод ДДП
3. Вывод АВЛ-дерева
4. Вывод хеш-таблицы
5. Поиск слова в ДДП, сбалансированном дереве, хеш-таблице и файле, вывод измерений времени, памяти и количества сравнений

Допущения

При вводе пункта меню необходимо вводить только числа.

Слова необходимо вводить на английском языке

Длина вводимых слов должна быть не более 20 символов

Пробелы и знаки препинания не допускаются

Описание внутренних структур данных

Структура для хранения дерева:

```
typedef struct tree_t
{
    char *word;
    int height;
    struct tree_t *left;
    struct tree_t *right;
} tree_t;
```

*char *word* - значение текущей вершины;

int height - высота вершины относительно других вершин;

*struct tree_t *left* - указатель на левого потомка;

*struct tree_t *right* - указатель на правого потомка;

Память под описанные выше структуры данных выделяется динамически

Структуры для хранения хеш-таблицы:

```
typedef struct hash_table_t
{
    int count;
    hash_t **array;
} hash_table_t;
```

int count — размер хеш-таблицы;

*hash_t **array* — массив указателей на список — структуру данных, описанную следующим образом:

```c
typedef struct hash_t
{
    char word[WORD_SIZE];
    struct hash_t *next;
} hash_t;
```

char word[WORD_SIZE] — значение текущего элемента списка;

*struct hash_t *next* — указатель на следующий элемент списка;

WORD _ SIZE = 20;

Описание меню и функций программы

```
emir_shimshir@mbp-emir ~/D/g/B/lab_07 (lab_07)> ./app.exe func_tests/test_100.txt
Программа для работы с бинарным, сбалансированным деревьями и хеш-таблицей.
Элементами структур данных являются слова.
Программа осуществляет поиск введенного слова.

Шимшир Эмирджан ИУ7-33Б

(При вводе пункта меню необходимо вводить только числа)
(Слова необходимо вводить на английском языке)
(Длина вводимых слов должна быть не более 20 символов)
(Пробелы и знаки препинания не допускаются)


Выберите номер пункта меню:
1) Ввести данные из файла
2) Вывести ДДП
3) Вывести АВЛ-дерево
4) Вывести хеш-таблицу
5) Найти слово в ДДП, сбалансированном дереве, хеш-таблице и файле
0) Выход

Введите команду:
```

1.

1. Пользователь вводит имя файла как аргумент командной строки при запуске программы

2. Пользователь загружает данные из файла

3. Пользователь выводит ДДП

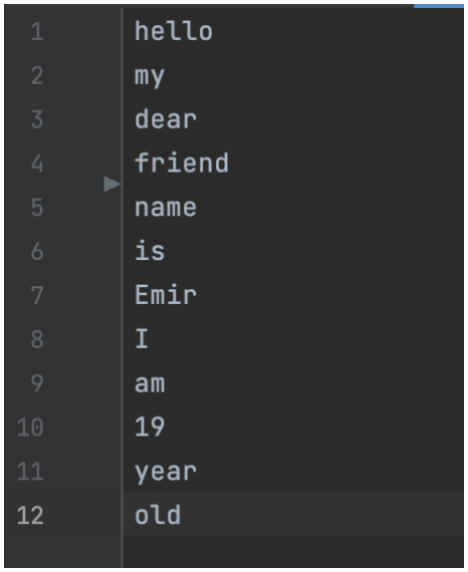
4. Пользователь выводит АВЛ-дерево

5. Пользователь выводит хеш-таблицу

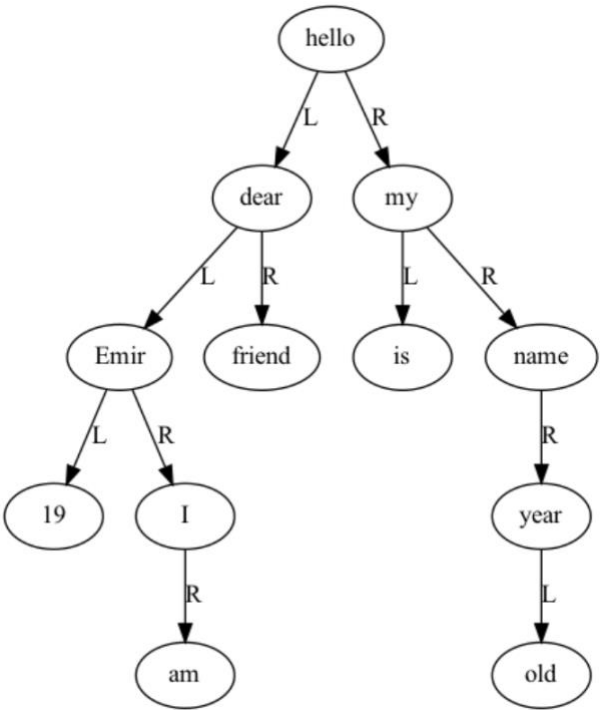
6. Пользователь ищет слова в деревьях, хеш-таблице и файле, также ему доступны результаты измерения времени, размера структур данных и количества сравнений при поиске

Пример вывода структур данных

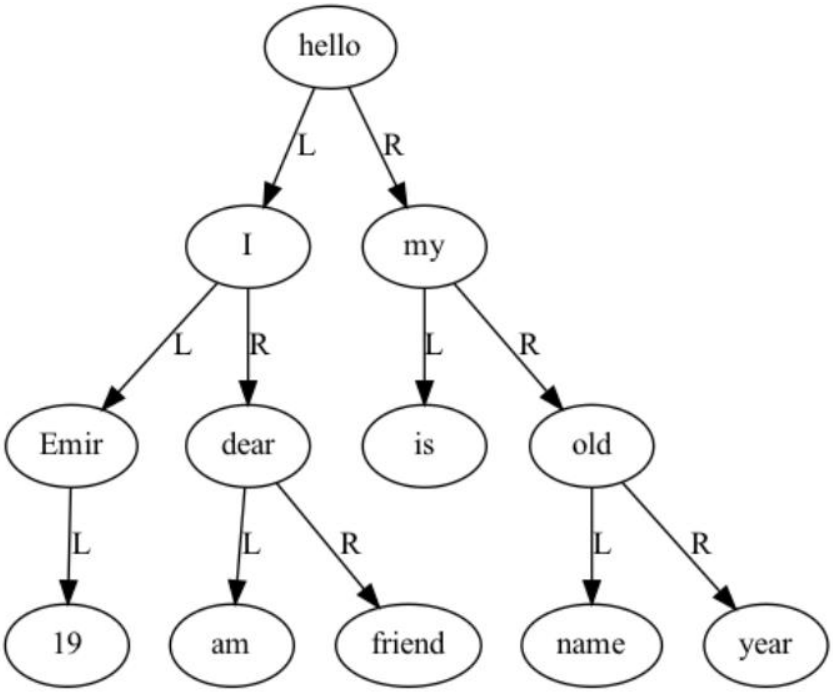
Исходный файл с данными:



Вывод ДДП в виде *.png* картинки (graphviz):



Вывод АВЛ - дерева в виде *.png* картинки (graphviz):



Вывод хеш-таблицы в консоли:

```
Хеш-таблица:

  0:  NULL
  1:  Emir ->  I ->  year ->  NULL
  2:  my ->  am ->  NULL
  3:  NULL
  4:  hello ->  dear ->  is ->  NULL
  5:  NULL
  6:  NULL
  7:  old ->  NULL
  8:  friend ->  NULL
  9:  name ->  NULL
 10:  19 ->  NULL
 11:  NULL
```

Описание функций

```c
/*
* Функция создает узел дерева
*
* Принимает слово и глубину, возвращает указатель на новый узел
*/
tree_t *create_node(char *word, int h);
```

```c
/*
* Функция добавляет новый узел в дерево
*
* Принимает указатель на корень, слово, указатель на глубину
*/
static tree_t *add_node(char *word, tree_t *tree, int *h);
```

```c
/*
* Функция очищает дерево
*
* Принимает указатель на корень дерева
*/
void free_tree(tree_t *tree);
```

```c
/*
* Функция ищет слово в дереве
*
* Принимает указатель на корень дерева, слово и указатель на количество
сравнений
*/
int find_tree(tree_t *tree, char *word, int *count_cmp);
```

```c
/*
* Функция балансирует дерево
*
* Принимает указатель на корень дерева
*/
static tree_t *balance(tree_t *tree);
```

```c
/*
* Функция создает хеш-таблицу
*
* Принимает указатель на хеш-таблицу и размер
*/
int hash_table_init(hash_table_t *table, const int table_size);
```

```c
/*
* Функция ищет слово в хеш-таблице
*
```

```
* Принимает указатель на хеш-таблицу, слово, максимальное число коллизий,
количество сравнений
*/
hash_t *hash_find_in_table(hash_table_t *hash_table, char *word, int
collision, int *code, int *cmp);
```

```
/*
* Функция печатает хеш-таблицу
*
* Принимает указатель на хеш-таблицу
*/
void print_table(hash_table_t *table);
```

Описание алгоритма и исследование полученных результатов

Дерево двоичного поиска

Искомое слово сравнивается со словом, находящимся в текущей вершине. Если они совпадают — поиск завершен, если искомое слово меньше — поиск продолжается в левом поддереве вершины, иначе — в правом.

Сбалансированное дерево

Алгоритм аналогичен ДДП.

Хеш-таблица

Для каждого элемента таблицы: определяется хеш-значение, по хеш-значению элемент добавляется в односвязный список (метод цепочек устранения коллизий).

ключом является остаток от деления суммы кодов символов строки на размер таблицы.

При поиске в хеш-таблице считается число сравнений для элемента. Если оно превышает введенное пользователем, происходит реструктуризация хеш-таблицы.

Реструктуризация хеш-таблицы: размер таблицы увеличивается в 2 раза и для нового размера таблицы все слова получают новые хеш-значения и запоняют таблицу.

Полученные результаты

Результаты измерения времени при поиске слов в деревьях, хеш-таблице и файле при различном количестве исходных элементов. Время измерений считается как среднее арифметическое при поиске всех слов из исходного файла. При проведении экспериментов программа была скомпилирована без оптимизаций (**-O0**), внешние задачи отсутствовали.

Время (в тиках):

Число элементов	**ДДП**	**Сбалансированное дерево**	**Хеш-таблица**	**Файл**
50	370	311	79	11902
100	415	361	93	17546
500	521	429	76	54198
1000	577	481	88	94374

Объем памяти (в байтах):

Число элементов	**ДДП**	**Сбалансированное дерево**	**Хеш-таблица**	**Файл**
50	1600	1600	1048	298
100	3200	3200	1608	618
500	16000	16000	7320	3239
1000	32000	32000	11548	6522

Тестирование

Позитивные тесты

№	Тест	Ввод	Вывод
1	Вывод ДДП	2	ДДП
2	Вывод сбалансированного дерева	3	Сбалансированное дерево
3	Вывод хеш-таблицы	4	Хеш-таблица
4	Поиск	5	Временная и количественная характеристика

Негативные тесты

№	Тест	Ввод	Вывод
1	Неверный пункт меню: больше 5	6	Команда введена неверно
2	Неверный пункт меню: меньше 0	-1	Команда введена неверно
3	Неверный пункт меню: не целое число	fsdfs	Команда введена неверно

4	Неверное имя файла	Несуществующее имя файла	Ошибка открытия файла
5	Пустой файл	Пустой файл	Файл пустой
6	Неверное число допустимых сравнений: буква	5, a	Неверное число допустимых сравнений
7	Неверное число допустимых сравнений: меньше 1	5, 0	Неверное число допустимых сравнений
3	Любое действие меню 2-5, если не загружены данные	2-5	Дерево не загружено из файла

Контрольные вопросы

1. Чем отличается идеально сбалансированное дерево от АВЛ-дерева?

У идеально сбалансированного дерева число вершин в левом и правом поддеревьях отличается не более, чем на единицу. У каждого узла АВЛ-дерева высота двух поддеревьев отличается не более, чем на единицу.

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

Поиск в АВЛ-дереве происходит быстрее, чем поиск в дереве двоичного поиска и с меньшим числом сравнений.

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

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

Принцип построения: хеш-функция ставит в соответствие каждому ключу ki индекс ячейки j, где расположен элемент с этим ключом. Таким образом:

h (ki) = j, если j=(1, m), где j принадлежит множеству от 1 до m, а m. – размерность массива.

4. Что такое коллизии? Каковы методы их устранения?

Коллизии - ситуации, когда разным ключам соответствует одно значение хеш- функции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2.

Методы устранения:

1) 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.

2) Внутреннее (закрытое) хеширование (открытая адресация). В этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку до тех пор, пока не будет найден ключ K или пустая позиция в таблице.

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

Если для поиска элемента необходимо более 3–4 сравнений, то эффективность использования хеш-таблицы пропадает.

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

АВЛ-деревья : $\mathbf{O(\log_2(n))}$ ДДП : $\mathbf{O(\log_2(n))}$ **- O(n)** Хеш-таблица: **O(1)**

Вывод

При сравнении поиска слова в четырех структурах данных (дерево двоичного поиска, сбалансированное, хеш-таблица и файл) я получил следующие результаты.

Самой эффективной структурой данных по времени обработки является хеш-таблица (в 5 раз быстрее, чем ДДП и в 4 раза быстрее, чем сбалансированное). Выигрыш по времени объясняется числом сравнений при поиске (при отсутствии коллизий количество сравнений при поиске слова равно 1). Объем памяти хеш- таблицы меньше в 2 раза, чем объем памяти, выделенной под деревья.

Самой эффективной структурой данных по занимаемому объему памяти является файл (в 5 раз меньше, чем деревья и в 3 раза меньше, чем хеш-таблица). При этом файл является самой неэффективной структурой данных по времени, так как все слова в файле идут последовательно и необходимо пройти все слова, лежащие до искомого.

Сравнение двух реализаций деревьев (ДДП и сбалансированного) показало, что поиск слова в сбалансированном дереве происходит быстрее в 1.2 раза, чем в ДДП, что объясняется меньшей высотой сбалансированного дерева. Объем занимаемой памяти двух реализаций одинаков.