|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство образования и науки 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 №6  
«Работа с деревьями»**

Студент, группа **Пронин А.С. ИУ7-32Б**

*2020 г.*

# Условия задачи

Построить ДДП, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из слов текстового файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного слова в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию.

# Техническое задание

## Входные данные

1) Целое число от 0 до 10 (для ввода номера команды из меню)

2) Строка для ввода искомого слова

2) Строка для ввода имени файла

3) Целое число > 0 для ввода максимального кол-ва сравнений

## Выходные данные

Вывод дерева c его высотой

Вывод информации о слове найденном в дереве и его месте

Вывод удаляемой ветки

Вывод хеш-таблицы и хеша искомого слова

Вывод номера троки в файле искомого слова

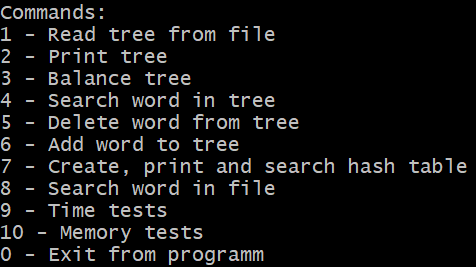
Вывод информации о сравнении времени поиска в разных структурах данных

Вывод информации о сравнении занимаемой памяти

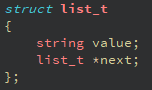
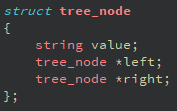
Вывод информации о проверке фрагментации

## Функции программы

Программа представляет из себя консольное приложение исполняющее следующие команды:



## Структуры данных

# Описание алгоритма

Команды:

1 – Считывание дерева из файла

2 – Вывод дерева

3 – Балансировка дерева

4 – Поиск слова в дереве

5 – Удаление ветки из дерева

6 – Добавление слова в дерево и балансировка

7 – Создание, вывод и поиск в хеш-таблице

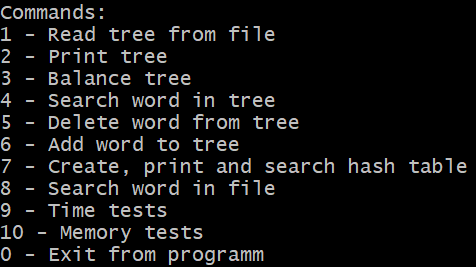
8 – Поиск слова в файле

9 – Тесты на время

10 – Тесты на объём занимаемой памяти

# Тесты

### Вот так выглядит меню:

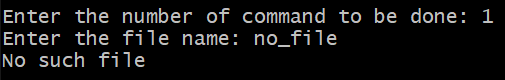


### Некорректный ввод:

1. Некорректная команда

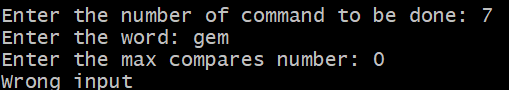




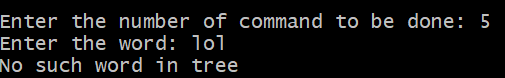
1. Команда 1, 8, 9, 10 (несуществующий файл) 
2. Команда 2, 3, 4, 5, 7 (Пустое дерево)

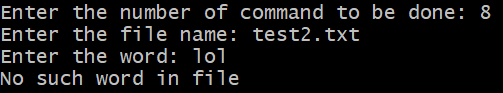


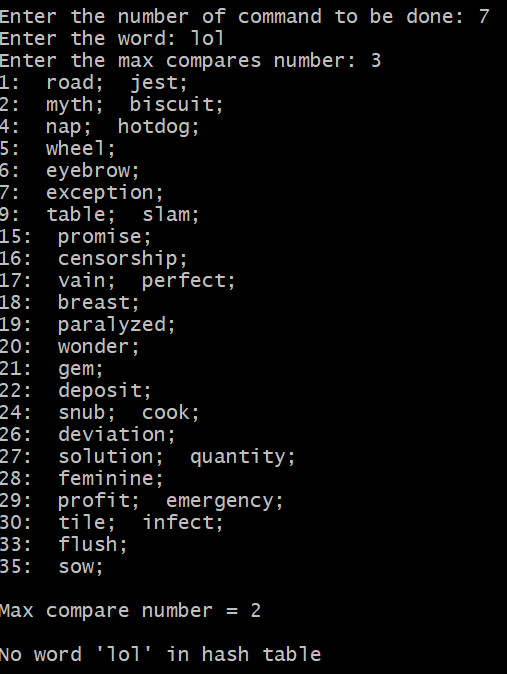
1. Команда 7 (Макс. Кол-во сравнений <= 0)



1. Команда 5, 7, 8 (Не найденное слово)



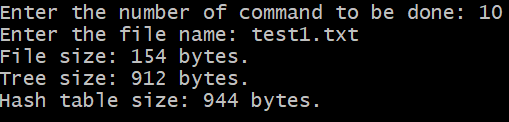




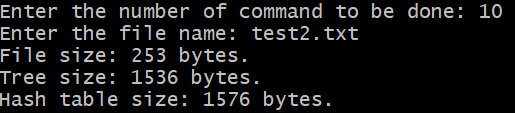
# Оценка эффективности

## Память

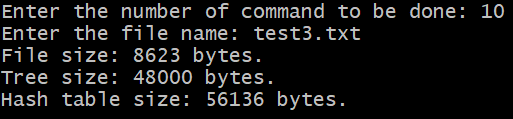
### Test1.txt – 19 слов:



### Test2.txt – 32 слова:



### Test3.txt – 1000 слов:



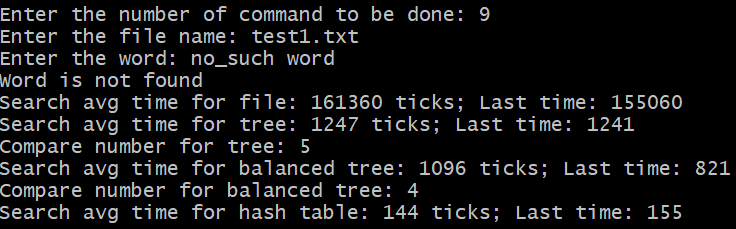
(В качестве строчки использовался тип string размер которого 32байта)

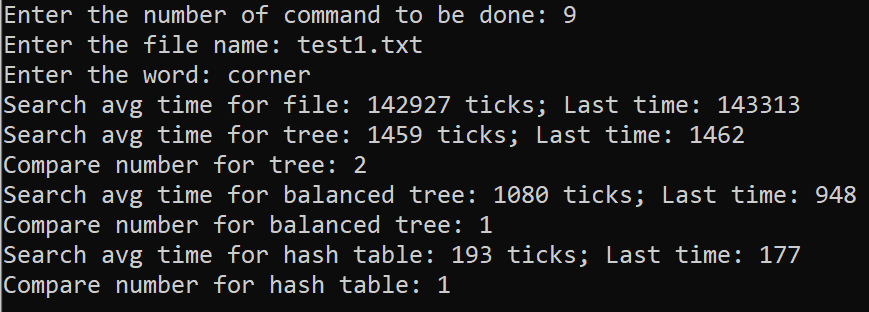
(кол-во сравнений в хеш-таблице не более 3)

## Эффективность

### Test1.txt – 19 слов:

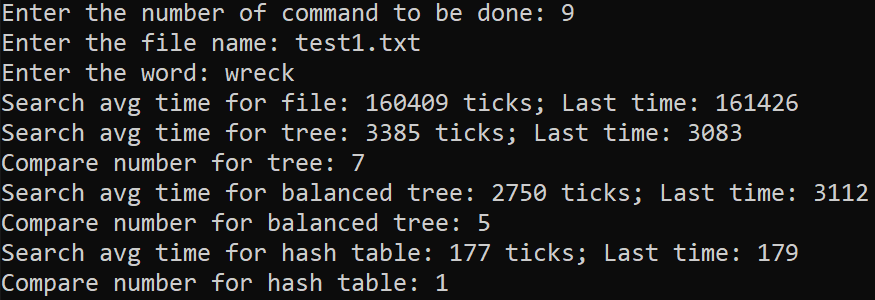
Сначала попробуем найти слово которого нет в файле



Теперь введём случайное слово 

Как мы видим оно находится на высоте 2 в несбалансированном дереве и на высоте 1 в сбалансированном

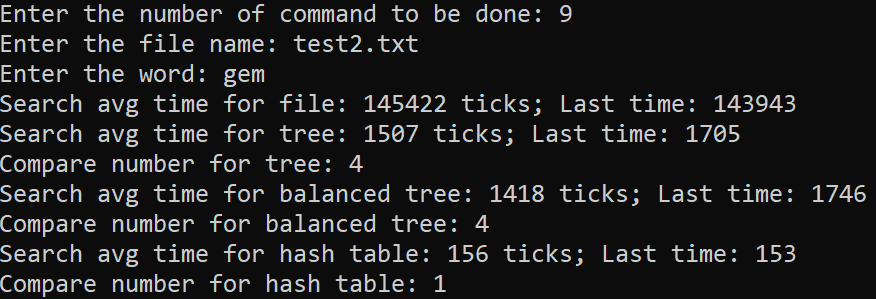
Попробуем слово последнее по алфавиту



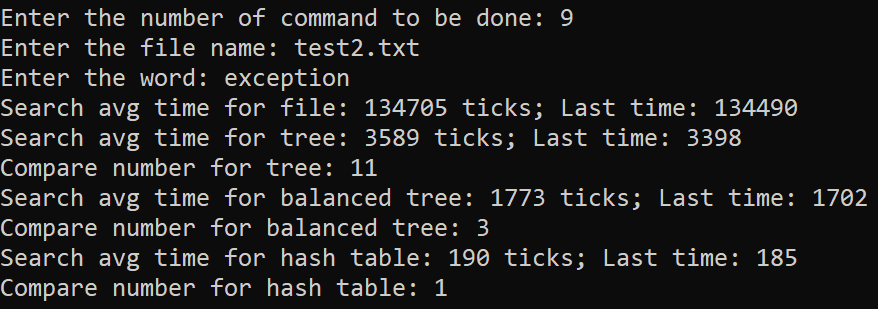
Как мы видим поиск по файлу всегда длится дольше всех, а поиск по хеш-таблице занимает меньше всего времени и почти не зависит от выбранного слова

### Test2.txt – 32 слова:

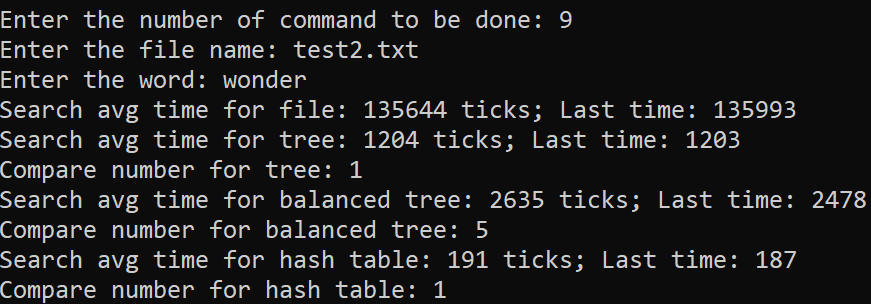
Введём слово, которое находится на одинаковой высоте в обоих деревьях



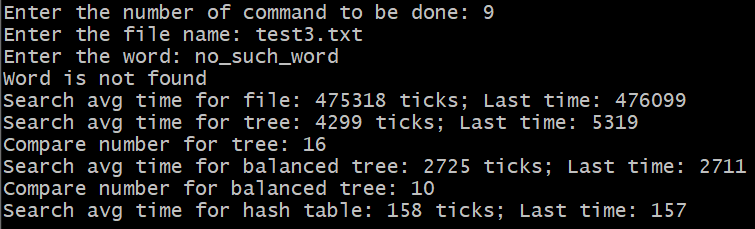
Теперь введём слово, которое находится выше всех в несбалансированном дереве

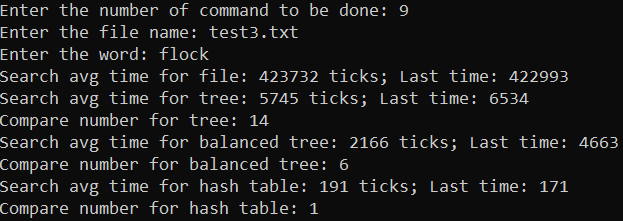


И наконец, слово которое в сбалансированном дереве находится в верхушке а в несбалансированном в корне

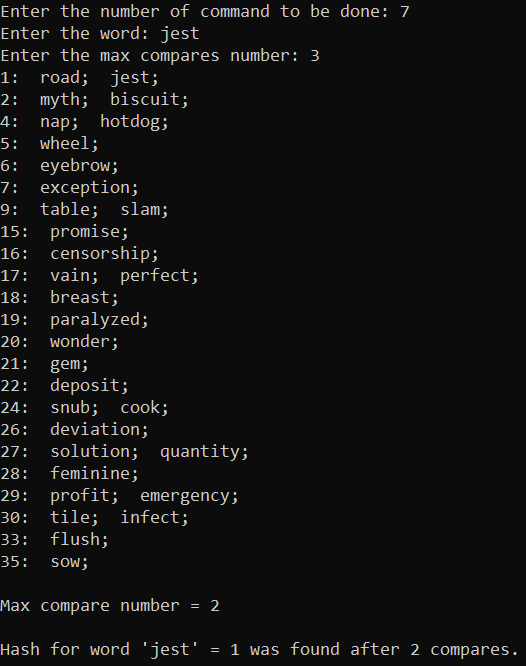


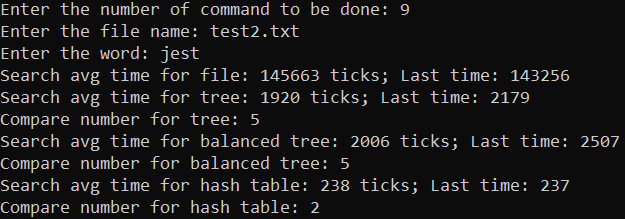
### Test3.txt – 1000 слов:





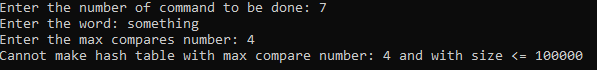
Также можем отследить кол-во сравнений для нахождения слова в хеш-таблице



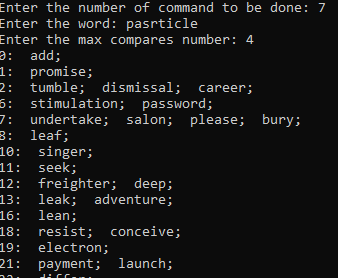
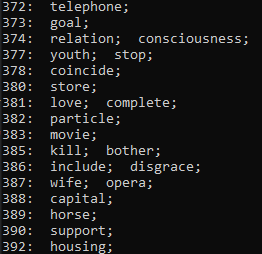


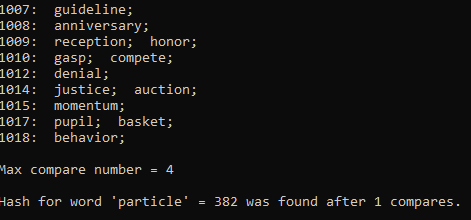
# Хеш-функции

Сначала я написал хеш-функцию которая суммировала коды символов и возвращала остаток от деления. Но она не справлялась с большим кол-вом слов в дереве, так чтобы кол-во сравнений было приемлемым



Поэтому я решил использовать более сложную функцию хеширования



# Контрольные вопросы

**1. Что такое дерево?**

Дерево – это рекурсивная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

**2. Как выделяется память под представление деревьев?**

В виде связного списка с двумя указателями — динамически под каждый узел.

**3. Какие стандартные операции возможны над деревьями?**

Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

**4. Что такое дерево двоичного поиска?**

Двоичное дерево поиска - двоичное дерево, для каждого узла которого сохраняется условие: левый потомок >= родителю, правый потомок строго < родителя (либо наоборот).

**5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?**

У АВЛ дерева для каждой его вершины высота двух её поддеревьев различается не более чем на 1, а у идеально сбалансированного дерева различается количество вершин в каждом поддереве не более чем на 1.

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

Поиск в АВЛ дереве происходит быстрее, чем в ДДП.

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

Хеш-таблицей называется массив, заполненный элементами в порядке, определяемом хеш-функцией. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс. Функция должна быть простой для вычисления, распределять ключи в таблице равномерно и давать минимум коллизий.

**8. Что такое коллизии? Каковы методы их устранения.**

Коллизия – ситуация, когда разным ключам хеш-функция ставит в соответствие один и тот же индекс. Основные методы устранения коллизий: открытое (метод цепочек) и закрытое (открытая адресация) хеширование. При открытом хешировании к ячейке по данному ключу прибавляется связанны список, при закрытом – новый элемент кладется в ближайшую свободную ячейку после данной.

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

Поиск в хеш-таблице становится неэффективен при большом числе коллизий –сложность поиска возрастает по сравнению с О(1). В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции.

**10. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах?**

В хеш-таблице минимальное время поиска О(1). В АВЛ: О(log2n). В дереве двоичного поиска О(h), где h - высота дерева (от log2n до n).

# Вывод

Поиск по хеш-таблице всегда эффективнее по времени, но не всегда эффективно по памяти (в случае хорошей дистрибуции функции распределение будет не плотным), так как требует выделенной памяти под каждый хеш (если отсутствуют коллизии, хорошая дистрибуция). В случае деревьев, АВЛ дерево не всегда выигрывает по времени поиска у несбалансированного дерева, так как порядок вершин при балансировке меняется, но в среднем поиск по сбалансированному дереву эффективней из-за меньшей максимальной высоты его узлов. В случае использования файлов мы выигрываем по памяти, но много времени теряем при поиске.