Невмержицкий 19 вариант 5 задача

## Условие задачи: построить хеш-таблицу по указанным данным. Сравнить эффективность поиска в сбалансированном двоичном дереве и в хеш-таблице. Вывести на экран дерево 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.

## Описание ТЗ: Удалить все слова, начинающиеся на указанную букву в таблице и в сбалансированном дереве, вывести таблицу. Сравнить время поиска, объем памяти и количество сравнений при использовании хеш-таблиц и сбалансированных деревьев.

## Структуры данных:

// Тип элемента дерева

struct TreeType

{

char value[100]; // Слово в дереве

struct TreeType \*left; // Левый потомок

struct TreeType \*right; // Правый потомок

};

// Тип элемента хеш-таблицы

struct TableType

{

char value[100]; // Значение элемента

struct TableType \*next; // Адрес следующего элемента

};

struct TreeType \*tree; // Дерево

struct TableType \*table[100]; // Таблица

## Описание алгоритмов:

## Поиск в дереве:

## Поиск (дерево, элемент)

## начало

## Если элемент дерева = элемент

## Вернуть 1

## Если элемент дерева < элемент

## Поиск(правый потомок, элемент)

## Иначе

## Поиск(левый потомок, элемент)

## Вернуть 0

## Конец

## Поиск в хеш-таблице:

## Поиск (строка таблицы, слово)

## Начало

## Пока (таблица != NULL)

## Если элемент таблицы = элемент

## Вернуть 1

## Перейти на следующий элемент

## Вернуть 0

## Конец

## Представление данных:

## Дерево двоичного поиска

## 

## Сбалансированное дерево

## 

## Хеш-таблица

## 

## Поиск элементов:

## Сбалансированное дерево

## 

## Дерево двоичного поиска

## 

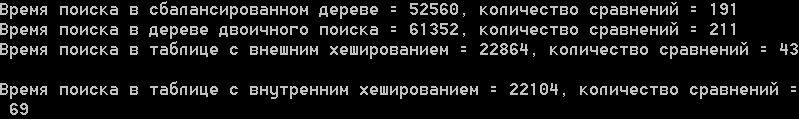
## Хеш-таблица

## 

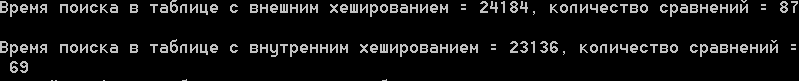
**Проверка производительности:**

Из полученных результатов видно, что хеш-таблица работает быстрее, чем сбалансированное дерево. Количество сравнений в таблице всегда будет меньше, так как номер строки, в которой находится нужный элемент, легко определить, зная хеш-функцию, и не нужно перебирать большое количество элементов. Количество сравнений зависит от максимально допустимого числа коллизий (чем оно больше, тем больше элементов в каждой строке, и больше сравнений, время поиска также повысится). Что касается используемой памяти, здесь хеш-таблицы проигрывают, так как кроме самих элементов нужно хранить n-e количество строк таблицы. Также из результатов видно, что сбалансированное дерево работает несколько быстрее дерева двоичного поиска. Если бы дерево двоичного поиска ветвилось слабо, то разница во времени была бы значительной. Отношение времени в дереве к времени в таблице = 2.5, а количества операций = 5. Это связано с тем, что значительное количество времени тратится на хеширование.

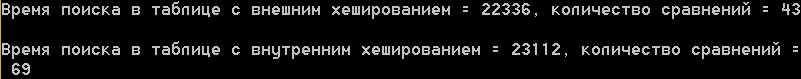
Обычный тест, 25 элементов и 2 коллизии максимум



5 коллизий



2 коллизии



**Контрольные вопросы:**

1. В идеально сбалансированном дереве количество вершин отличается не более, чем на 1, в каждом поддереве, а АВЛ дерево – дерево, высота каждого поддерева которого отличается не более, чем на 1.
2. Сам алгоритм поиска будет аналогичный, однако время поиска в сбалансированном дереве будет меньше, так как его высота будет меньше.
3. Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей. Строится по ключам, определенным хеш-функцией. В данном случае у меня использовалась функция деления.
4. Пусть H – хеш-функция, тогда коллизией называется ситуация H(x1) = H(x2). Устраняются путем выбора другой хеш-функции.
5. Поиск в хеш-таблицах неэффективен, если количество коллизий в строках таблицы большое, т.е. в этом случае количество сравнений будет большим.
6. Поиск в АВЛ-дереве эффективнее, чем в двоичном дереве, так как высота будет меньше. Поиск в хеш-таблице эффективнее, чем в АВЛ-дереве, так как при небольшом количестве коллизий количество сравнений будет небольшим.