



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 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

### “Деревья, хеш-таблицы”

Студент **Зернов Георгий Павлович**

Группа **ИУ7-34Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент \_\_\_\_\_ **Зернов Г.П.**

Преподаватель \_\_\_\_\_ **Никульшина Т.А.**

## Оглавление

Условие задачи.....	3
Техническое задание.....	4
Реализуемая в программе задача.....	4
Входные данные.....	4
Выходные данные.....	6
Способ обращения к программе.....	6
Возможные аварийные ситуации или ошибки пользователя.....	6
Внутренние структуры и типы данных.....	7
Алгоритм программы.....	9
Функции программы.....	12
Тесты.....	16
Анализ эффективности.....	19
Вывод.....	20
Контрольные вопросы:.....	21

## **Условие задачи**

Построить дерево поиска из слов текстового файла, сбалансировать полученное дерево. Вывести его на экран в виде дерева. Удалить все слова, начинающиеся на указанную букву, в исходном и сбалансированном дереве. Сравнить время удаления, объем памяти. Построить хеш-таблицу из слов текстового файла. Вывести построенную таблицу слов на экран. Осуществить поиск и удаление введенного слова, вывести таблицу. Выполнить программу для различных размерностей таблицы и сравнить время удаления, объем памяти и количество сравнений при использовании сбалансированных деревьев и хеш-таблиц

## **Техническое задание**

### **Реализуемая в программе задача**

Освоение работы с хеш-таблицами, сравнение эффективности поиска в сбалансированных (AVL) деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнение эффективности устранения коллизий при внешнем и внутреннем хешировании.

### **Входные данные**

В случае выбора опции в меню целое число, соответствующее одной из указанных опций (опции с 1 по 5 выполняются сразу для бинарного дерева поиска и AVL дерева, опции с 8 по 12 выполняются одновременно на хеш-таблицах с открытой и закрытой адресацией):

- 1 – Загрузить дерево из файла
- 2 – Добавить элемент в дерево
- 3 – Найти элемент в дереве
- 4 – Удалить элемент из дерева
- 5 – Удалить элементы, начинающиеся на заданную букву из дерева
- 6 – Вывести бинарное дерево поиска
- 7 – Вывести AVL дерево
- 8 – Загрузить хеш-таблицу из файла
- 9 – Добавить элемент в хеш-таблицу
- 10 – Найти элемент в хеш-таблице
- 11 – Удалить элемент из хеш-таблицы
- 12 – Удалить элементы, начинающиеся на заданную букву из хеш-таблицы
- 13 – Вывести хеш-таблицу с открытым хешированием

- 14 – Вывести хеш-таблицу с закрытым хешированием
- 15 – Анализ эффективности
- 16 – Выход

В случае загрузки дерева из файла:

- Имя файла – строка.

В случае добавления элемента в дерево:

- Значение элемента – строка.

В случае поиска элемента в дереве:

- Значение элемента – строка.

В случае удаления элемента из дерева:

- Значение элемента – строка.

В случае удаления элементов из дерева по первому символу:

- Символ.

В случае вывода бинарного дерева поиска:

- Имя файла – строка.

В случае вывода АВЛ дерева:

- Имя файла – строка.

В случае загрузки хеш-таблицы из файла:

- Имя файла – строка.

В случае добавления элемента в хеш-таблицу:

- Значение элемента – строка.

В случае поиска элемента в хеш-таблице:

- Значение элемента – строка.

В случае удаления элемента из хеш-таблицы:

- Значение элемента – строка.

В случае удаления элементов из хеш-таблицы по первому символу:

- Символ.

В случае вывода хеш-таблицы с открытым хешированием:

- Имя файла – строка.

В случае вывода хеш-таблицы с закрытым хешированием:

- Имя файла – строка.

### **Выходные данные**

В случае вывода бинарного дерева поиска:

- Файл, содержащий бинарное дерево поиска в .dot формате.

В случае вывода AVL дерева:

- Файл, содержащий AVL дерево в .dot формате.

В случае вывода хеш-таблицы с открытым хешированием:

- Хеш-таблица с открытым хешированием, выведенная в терминал.

В случае вывода хеш-таблицы с закрытым хешированием:

- Хеш-таблица с закрытым хешированием, выведенная в терминал.

### **Способ обращения к программе**

Чтобы обратиться к программе, пользователю необходимо запустить файл app.exe.

### **Возможные аварийные ситуации или ошибки пользователя**

- Введено невалидное значение опции – вывод на экран информирующего сообщения о неверном вводе и предоставление пользователю возможности повторно ввести значение.
- Введено невалидное имя файла - вывод на экран информирующего сообщения о неверном вводе и предоставление пользователю возможности повторно ввести имя файла.
- Ввод из файла, который невозможно открыть на чтение или содержит невалидную информацию – вывод информирующего сообщения.
- Вывод в файл, который невозможно открыть на запись – вывод информирующего сообщения.

- Нехватка памяти при её выделении – соответствующее сообщение об ошибке и остановка текущего процесса с возвратом программы в состояние до его начала.

## **Внутренние структуры и типы данных**

Структура, представляющее тип данных “узел бинарного дерева поиска”.

- struct tree\_node\_t \*left – указатель на левое поддерево.
- struct tree\_node\_t \*right – указатель на правое поддерево.
- char \*data – указатель на значение узла (строку).

Листинг структуры:

```
typedef struct tree_node_t
{
    struct tree_node_t *left;
    struct tree_node_t *right;
    char *data;
} tree_node_t;
```

Структура, представляющее тип данных “узел АВЛ дерева”.

- struct avl\_tree\_node\_t \*left – указатель на левое поддерево.
- struct avl\_tree\_node\_t \*right – указатель на правое поддерево.
- size\_t height – высота узла.
- char \*data – указатель на значение узла (строку).

Листинг структуры:

```
typedef struct avl_tree_node_t
{
    struct avl_tree_node_t *left;
    struct avl_tree_node_t *right;
    size_t height;
    char *data;
} avl_tree_node_t;
```

Структура, представляющее тип данных “узел хеш-таблицы с открытым хешированием”.

- `char *data` – указатель на значение узла (строку).
- `size_t index` – номер узла в списке.
- `struct hash_node_t *next` – указатель на следующий узел.

Листинг структуры:

```
typedef struct hash_node_t
{
    char *data;
    size_t index;
    struct hash_node_t *next;
} hash_node_t;
```

Структура, представляющее тип данных “хеш-таблица с открытым хешированием”.

- `size_t (*hash_func)(char *data, size_t size)` – хеш-функция.
- `size_t size` – размер хеш-таблицы.
- `size_t elem_count` – число элементов в хеш-таблице.
- `hash_node_t **data` – массив данных.

Листинг структуры:

```
typedef struct
{
    size_t (*hash_func)(char *data, size_t size);
    size_t size;
    size_t elem_count;
    hash_node_t **data;
} oh_hash_table_t;
```

Структура, представляющее тип данных “хеш-таблица с закрытым хешированием”.

- `size_t (*hash_func)(char *data, size_t size)` – хеш-функция.



- size\_t size – размер хеш-таблицы.
- size\_t elem\_count – число элементов в хеш-таблице.
- hash\_node\_t \*\*data – массив данных.

Листинг структуры:

```
typedef struct
{
    size_t (*hash_func)(char *data, size_t size);
    size_t size;
    size_t elem_count;
    char **data;
} ch_hash_table_t;
```

### Алгоритм программы

- Общий алгоритм программы:

1. Запуск основного цикла программы.
2. Вывод меню.
3. Запрос у пользователя опции для выполнения.
4. Выполнение выбранной опции.
5. Переход к новой итерации цикла или завершение основного цикла при соответствующем выборе пользователя.

- Алгоритм добавления узла в дерево:

1. Если дерево пустое – возврат узла.
2. Иначе – если текущий элемент меньше вставляемого – рекурсивный запуск алгоритма на правом поддереве, а если больше – на левом.

- Алгоритм удаления элемента из дерева:

1. Если дерево пустое – возврат дерева.

2. Иначе – если текущий элемент меньше искомого – рекурсивный запуск алгоритма на правом поддереве, а если больше – на левом.
3. Иначе если и правое и левое поддерево не пусты – замещаем значение текущего узла, значением минимального узла правого поддерева и удаляем этот узел.
4. Иначе если одно из поддеревьев не пусто заменяем текущий узел этим поддеревом.
5. Иначе удаляем текущий узел.

- Алгоритм поиска элемента в дереве:

1. Если элемент искомый – возврат элемента.
2. Иначе – если текущий элемент меньше искомого – рекурсивный запуск алгоритма на правом поддереве, а если больше – на левом.

- Алгоритм удаления из дерева по первой букве:

1. Если дерево пустое – завершаем работу.
2. Если первая буква текущего элемента больше необходимой – рекурсивный запуск алгоритма на левом поддереве, если меньше – на правом, иначе – на обоих.
3. Если первая буква текущего элемента равна необходимой – удаление узла.

Для АВЛ дерева применяются те же алгоритмы, после каждого изменения дерева производится его балансировка одним из базовых поворотов (левый, правый).

- Алгоритм левого поворота:

1. Текущая вершина дерева становится вершиной левого поддерева. Новой вершиной становится вершина правого поддерева.

- Алгоритм правого поворота:

1. Текущая вершина дерева становится вершиной левого поддеревья. Новой вершиной становится вершина правого поддеревья.

В хеш таблицах используется полиномиальное хеширование строк: для каждого элемента кроме первого к значению хеша прибавляется значение текущего символа переведённое в числовой формат, умноженное на номер предыдущего символа относительно «а». Конечная сумма умножается на длину ключа и берётся по модулю количества ячеек в хеш таблице.

- Алгоритм добавления в хеш-таблицу с открытым хешированием:

1. Вычисление хеша.
2. Добавление значения в список с соответствующим хешем, если длина списка превышает 4, то реструктуризация хеш-таблицы

- Алгоритм удаления элемента из хеш-таблицы с открытым хешированием:

1. Вычисление хеша.
2. Поиск и удаление элемента из списка с соответствующим хешем.

- Алгоритм поиска элемента в хеш-таблице с открытым хешированием:

1. Вычисление хеша.
2. Поиск в списке с соответствующим хешем.

- Алгоритм удаления из хеш-таблицы с открытым хешированием по первой букве:

1. Проход по всем элементам хеш-таблицы с удалением нужных.

- Алгоритм реструктуризации хеш-таблицы с закрытым хешированием:

1. Создание новой хеш-таблицы с расширенным массивом данных.
2. Добавление значений из старой таблицы в новую.
3. Разрушение старой таблицы.

- Алгоритм добавления в хеш-таблицу с закрытым хешированием:

1. Вычисление хеша.
2. Добавление значения в ячейку с соответствующим хешем, если ячейка занята, то проход по ячейкам до первой свободной.

- Алгоритм удаления элемента из хеш-таблицы с закрытым хешированием:

1. Вычисление хеша.
2. Поиск элемента и его удаление.

- Алгоритм поиска элемента в хеш-таблице с закрытым хешированием:

1. Вычисление хеша.
2. Если ячейка не содержит данный элемент, то проход по ячейкам до нужного элемента.

- Алгоритм удаления из хеш-таблицы с закрытым хешированием по первой букве:

1. Проход по всем элементам хеш-таблицы с удалением нужных.

### **Функции программы**

- `tree_node_t* create_node(char *data)` – создаёт узел по содержимому, возвращает указатель на узел.
- `void destroy_tree(tree_node_t *tree)` – освобождает выделенную под дерево память.

- `tree_node_t *minimum(tree_node_t *tree)` – возвращает указатель на минимальный узел в поддереве.
- `tree_node_t *maximum(tree_node_t *tree)` – возвращает указатель на максимальный узел в поддереве.
- `tree_node_t* insert(tree_node_t *tree, tree_node_t *node, int cmp(void *, void *))` – добавляет в бинарное дерево узел по компаратору, возвращает указатель на дерево с добавленным узлом.
- `tree_node_t *delete(tree_node_t *tree, char *data, int cmp(void *, void *))` – удаляет узел из дерева по значению и компаратору, возвращает указатель на изменённое дерево.
- `tree_node_t *search(tree_node_t *tree, void *data, int cmp(void *, void *))` – ищет узел в дереве по значению и компаратору, возвращает указатель на найденный узел.
- `void apply_pre(tree_node_t *tree, void f(tree_node_t*, void*), void *arg)` – применяет функцию к дереву, используя префиксный обход.
- `void apply_in(tree_node_t *tree, void f(tree_node_t*, void*), void *arg)` – применяет функцию к дереву, используя инфиксный обход.
- `void apply_post(tree_node_t *tree, void f(tree_node_t*, void*), void *arg)` – применяет функцию к дереву, используя постфиксный обход.
- `void to_dot(tree_node_t *tree, void *param)` – экспортирует поддерево в dot формат.
- `void export_to_dot(FILE *file, const char *tree_name, tree_node_t *tree)` – экспортирует дерево в dot формат.
- `void delete_by_first_letter(tree_node_t **tree, char smb, int cmp(void *, void *))` – удаляет из дерева узлы по первому символу значения и компаратору.
- `avl_tree_node_t *avl_create_node(char *data)` – создаёт узел АВЛ дерева по содержимому, возвращает указатель на узел.
- `void avl_destroy_tree(avl_tree_node_t *tree)` – освобождает выделенную под АВЛ дерево память.
- `size_t avl_height(avl_tree_node_t *tree)` – возвращает высоту узла АВЛ дерева.

- `avl_tree_node_t *avl_minimum(avl_tree_node_t *tree)` – возвращает указатель на минимальный узел в АВЛ поддереве.
- `avl_tree_node_t *avl_maximum(avl_tree_node_t *tree)` – возвращает указатель на максимальный узел в АВЛ поддереве.
- `avl_tree_node_t *avl_left_rotate(avl_tree_node_t *tree)` – поворачивает АВЛ дерево влево, возвращает указатель на повернутое АВЛ дерево.
- `avl_tree_node_t *avl_right_rotate(avl_tree_node_t *tree)` – поворачивает АВЛ дерево вправо, возвращает указатель на повернутое АВЛ дерево.
- `int avl_get_balance(avl_tree_node_t *tree)` – возвращает разность высот правого и левого АВЛ поддеревьев.
- `avl_tree_node_t *avl_insert(avl_tree_node_t *tree, avl_tree_node_t *node, int cmp(void *, void *))` – добавляет в АВЛ дерево узел по компаратору, возвращает указатель на АВЛ дерево с добавленным узлом.
- `avl_tree_node_t *avl_delete(avl_tree_node_t *tree, char *data, int cmp(void *, void *))` – удаляет узел из АВЛ дерева по значению и компаратору, возвращает указатель на изменённое АВЛ дерево.
- `avl_tree_node_t *avl_search(avl_tree_node_t *tree, void *data, int cmp(void *, void *))` – ищет узел в АВЛ дереве по значению и компаратору, возвращает указатель на найденный узел.
- `void avl_apply_pre(avl_tree_node_t *tree, void f(avl_tree_node_t*, void*), void *arg)` – применяет функцию к АВЛ дереву, используя префиксный обход.
- `void avl_apply_in(avl_tree_node_t *tree, void f(avl_tree_node_t*, void*), void *arg)` – применяет функцию к АВЛ дереву, используя инфиксный обход.
- `void avl_apply_post(avl_tree_node_t *tree, void f(avl_tree_node_t*, void*), void *arg)` – применяет функцию к АВЛ дереву, используя постфиксный обход.
- `void avl_to_dot(avl_tree_node_t *tree, void *param)` – экспортирует АВЛ поддерево в dot формат.
- `void avl_export_to_dot(FILE *file, const char *tree_name, avl_tree_node_t *tree)` – экспортирует АВЛ дерево в dot формат.

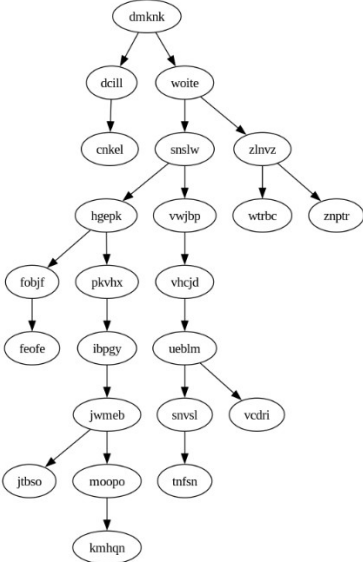
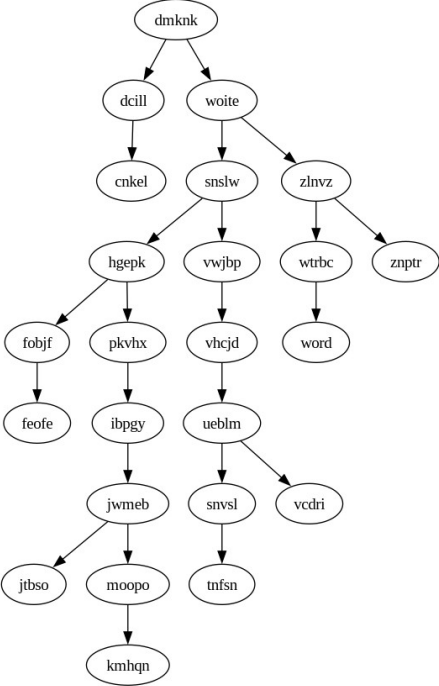
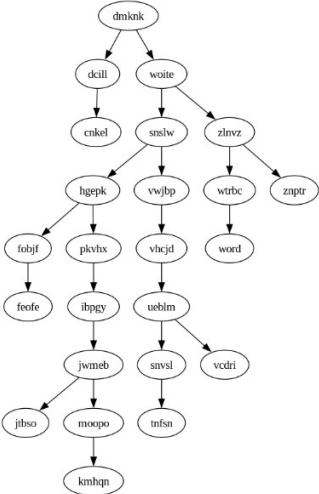
- `void avl_delete_by_first_letter(avl_tree_node_t **tree, char smb, int cmp(void *, void *))` – удаляет из АВЛ дерева узлы по первому символу значения и компаратору.
- `void oh_hash_create(oh_hash_table_t *table, size_t size, size_t hash_func(char *data, size_t size))` – создаёт хеш-таблицу с открытым хешированием.
- `void oh_hash_destroy(oh_hash_table_t *table)` – освобождает выделенную под хеш-таблицу с открытым хешированием память.
- `void oh_hash_restructure(oh_hash_table_t *table)` – реструктурирует хеш-таблицу с открытым хешированием.
- `void oh_hash_insert(oh_hash_table_t *table, char *data)` – добавляет элемент в хеш-таблицу с открытым хешированием.
- `hash_node_t *oh_hash_search(oh_hash_table_t *table, char *data)` – ищет элемент в хеш-таблице с открытым хешированием, возвращает указатель на него.
- `int oh_hash_delete(oh_hash_table_t *table, char *data)` – удаляет элемент из хеш-таблицы с открытым хешированием, возвращает код ошибки.
- `void oh_hash_fprint(FILE *file, oh_hash_table_t *table)` – выводит в файл хеш-таблицу с открытым хешированием.
- `void oh_hash_delete_by_smb(oh_hash_table_t *table, char smb)` – удаляет из хеш-таблицы с открытым хешированием все элементы по первому символу значения.
- `void ch_hash_create(ch_hash_table_t *table, size_t size, size_t hash_func(char *data, size_t size))` – создаёт хеш-таблицу с закрытым хешированием.
- `void ch_hash_destroy(ch_hash_table_t *table)` – освобождает выделенную под хеш-таблицу с закрытым хешированием память.
- `void ch_hash_restructure(ch_hash_table_t *table)` – реструктурирует хеш-таблицу с закрытым хешированием.
- `void ch_hash_insert(ch_hash_table_t *table, char *data)` – добавляет элемент в хеш-таблицу с закрытым хешированием.
- `char *ch_hash_search(ch_hash_table_t *table, char *data)` – ищет элемент в хеш-таблице с закрытым хешированием, возвращает указатель на него.
- `int ch_hash_delete(ch_hash_table_t *table, char *data)` – удаляет элемент из хеш-таблицы с закрытым хешированием, возвращает код ошибки.

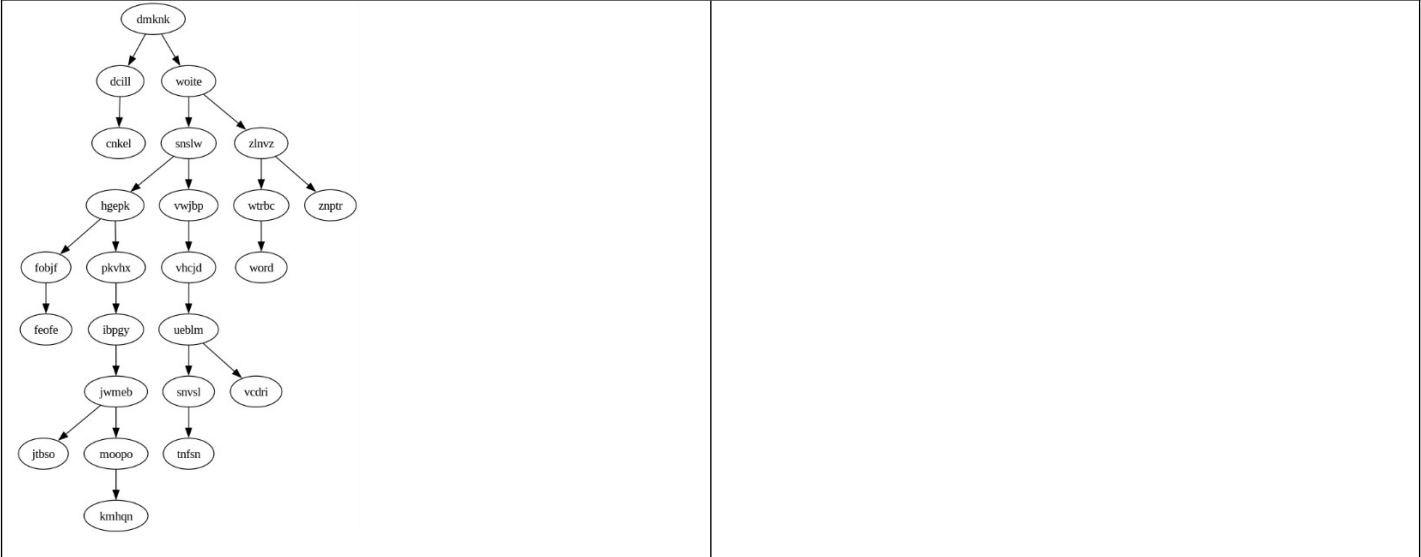
- `void ch_hash_fprint(FILE *file, ch_hash_table_t *table)` – выводит в файл хеш-таблицу с закрытым хешированием.
- `void ch_hash_delete_by_smb(ch_hash_table_t *table, char smb)` – удаляет из хеш-таблицы с закрытым хешированием все элементы по первому символу значения.
- `int time_measurement(void)` – проводит анализ времени, возвращает код ошибки.
- `void print_menu(void)` – выводит меню.
- `int main(void)` – выполняет основной алгоритм программы.

## Тесты

Тестовый случай	Ожидаемый результат
Введена невалидная опция	Соответствующее сообщение об ошибке, возможность снова ввести требуемое значение.
Введено невалидное имя файла ввода	Соответствующее сообщение об ошибке, возможность снова ввести имя файла.
Чтение из файла ввода невозможно	Соответствующее сообщение об ошибке.
Введено невалидное имя файла вывода	Соответствующее сообщение об ошибке, возможность снова ввести имя файла.
Запись в файл вывода невозможна	Соответствующее сообщение об ошибке.
Введено невалидное слово	Соответствующее сообщение об ошибке, возможность снова ввести имя.
Введена невалидная буква	Соответствующее сообщение об ошибке, возможность снова ввести букву.
Попытка добавить в дерево уже существующий элемент.	Соответствующее сообщение об ошибке.
Попытка удалить несуществующий элемент.	Соответствующее сообщение об ошибке.
Открытие файла с последовательностью слов	Успешное открытие файла



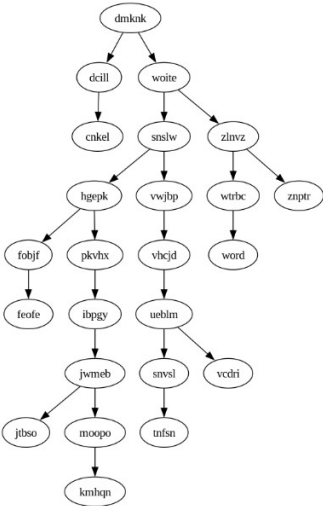
<p>Вывод дерева в файл</p> <p>Добавление слова в дерево</p> <p>Слово: word</p> <p>Дерево:</p> 	<p>Успешный вывод дерева в dot формате</p> <p>Полученное дерево:</p> 
<p>Поиск слова в дереве</p> <p>Слово: word</p> <p>Дерево:</p> 	<p>Слово найдено</p>
<p>Поиск слова в дереве</p> <p>Слово: notaword</p> <p>Дерево:</p>	<p>Слово не найдено</p>



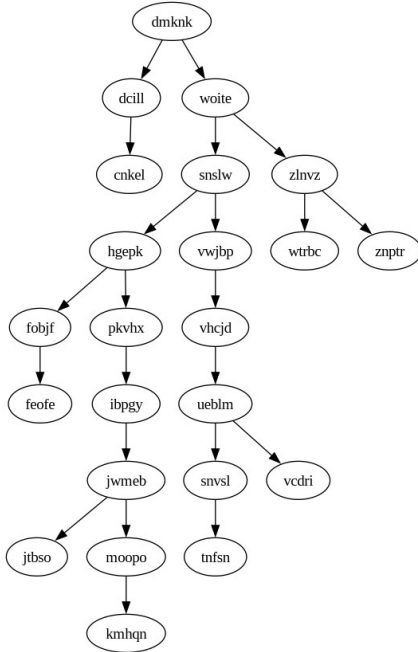
Удаление слова из дерева

Слово: word

Дерево:



Полученное дерево:

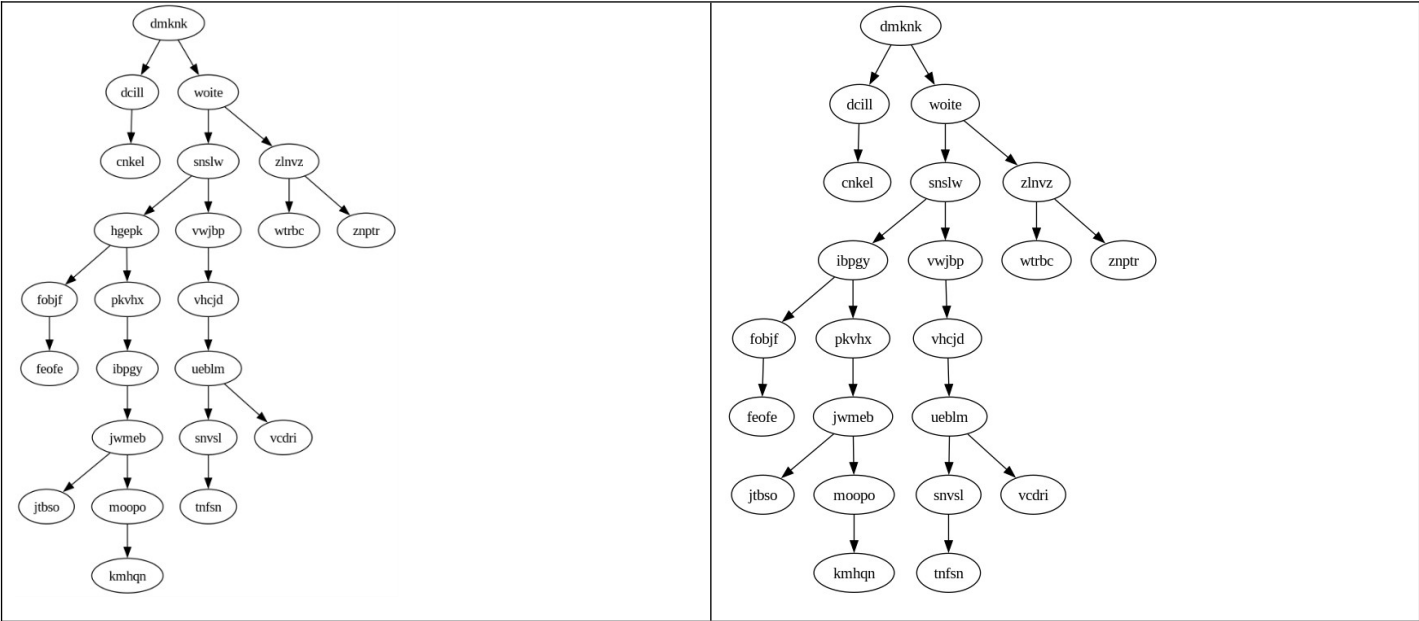


Удаление слов по первой букве из дерева

Буква: h

Дерево:

Полученное дерево:



Анализ эффективности

Анализ эффективности производится на последовательности уникальных слов. Используется полиномиальное хеширование.

Удаление по первому символу

Слов	BST (нс)	AVL (нс)	Открытое хеширован ие (нс)	Закрытое хеширова ние (нс)	BST (байт)	AVL (байт)	Открытое хеширование (байт)	Закрытое хеширование (байт)
100	2696	3344	2894	2853	3000	3800	3832	1432
200	4122	5118	4423	4075	6000	7600	7632	2832
300	5814	6957	8720	8336	9000	11400	11432	4232
400	7367	9048	12930	11535	12000	15200	15232	5632
500	9552	11459	15292	13743	15000	19000	19032	7032

Заметим что удаление из AVL дерева дольше чем из бинарного дерева поиска из-за балансирования, а в хеш таблице с закрытым хешированием удаление быстрее чем в хеш таблице с открытым хешированием так как проход по массиву с освобождением памяти под строки быстрее чем проход по массиву списков и удаление из них.

## Поиск

Слов	BST (нс)	AVL (нс)	Открытое хеширование (нс)	Закрытое хеширование (нс)
100	1990	1808	529	643
200	2278	2080	536	610
300	2476	2239	541	630
400	2695	2351	572	654
500	2887	2438	612	678

Заметим что поиск в хеш таблице выполняется за амортизированную константу, при этом в при закрытом хешировании он медленнее из-за большего числа коллизий. Поиск в АВЛ дереве выполняется за  $\log(O(n))$ , поиск в бинарном дереве поиска несколько медленнее из-за его несбалансированности.

## Вывод

По времени хэш таблицы эффективны для операции поиска, их эффективность для операций вставки и удаления значительно зависит от реализации и данных. Деревья менее эффективны для этих операций, меньше зависят от данных. Аналогично по памяти.

## **Контрольные вопросы:**

### **Чем отличается идеально сбалансированное дерево от AVL дерева?**

Идеально сбалансированное дерево отличается от AVL дерева тем что:

- Оно не является бинарным деревом поиска.
- Имеет более “сильное” условие сбалансированности, в AVL дереве разность высот поддеревьев не превышает 1, а в ИСД разность числа узлов в поддеревьях не превышает 1.

### **Чем отличается поиск в AVL-дереве от поиска в дереве двоичного поиска?**

Поиск в AVL дереве имеет сложность как в лучшем, так и в худшем случае сложность  $O(\log(n))$ . Поиск в бинарном дереве поиска в лучшем случае также имеет сложность  $O(\log(n))$ , но в случае вырожденного дерева деградирует до  $O(n)$ .

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

Хеш-таблица — структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию удаления и операцию поиска пары по ключу. Для хеш таблицы задана хеш функция которая по данным возвращает ключ.

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

Коллизии — это ситуация, когда хеш-функция возвращает одинаковый хеш для двух разных ключей.

Методы устранения:

- Метод цепочек – элементы с одинаковым хешем образуют список

- Открытая адресация – если ячейка занята, то производится поиск свободной.

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

- Неэффективная хеш-функция
- Высокий коэффициент заполнения
- Большое число коллизий
- Неоднородный доступ к данным

**Эффективность поиска в AVL деревьях, в дереве двоичного поиска, в хеш-таблицах и в файле.**

	Лучший случай	Среднее	Худший случай
Дерево двоичного поиска	$O(\log(n))$	$O(\log(n))$	$O(n)$
AVL дерево	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
Хеш-таблица	$O(1)$	$O(1)$	$O(n)$
Файл	$O(n)$	$O(n)$	$O(n)$