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 №6 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

Деревья

Студент **Павлов Д. В.**

Группа **ИУ7-33Б**

Вариант 3

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

Студент _____ **Павлов Д. В.**

Преподаватель _____ **Никульшина Т. А.**

2024

1. Описание условия задачи.

Построить дерево в соответствии со своим вариантом задания.
Вывести его на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления.

2. Описание технического задания.

Построить бинарное дерево поиска, в вершинах которого находятся слова из текстового файла (предполагается, что исходный файл не содержит повторяющихся слов). Вывести его на экран в виде дерева. Удалить все слова, начинающиеся на указанную букву. Сравнить время удаления слов, начинающихся на указанную букву, в дереве и в файле.

Входные данные.

Целое число: пункт меню (от 0 до 7).

```
1) Построить бинарное дерево из файла
2) Добавить слово в бинарное дерево
3) Извлечь элемент из бинарного дерева
4) Найти элемент в дереве.
5) Удалить все слова, начинающиеся на указанную букву
6) Вывести бинарное дерево на экран
7) Вывести замеры эффективности
0) Завершить работу программы
```

Строка: путь к файлу
Строка: вставляемый элемент
Строка: удаляемый элемент
Строка: буква для удаления слов.

Выходные данные.

Сообщение об успехе или ошибке выполнения пункта, изображение графа, таблица замеров по времени.

Способ обращения к программе.

Запускается через терминал командой *./app*

Возможные аварийные ситуации и ошибки пользователя.

Некорректный ввод данных, удаление из пустого дерева. Ошибка выделения памяти.

3. Описание внутренних структур данных.

Структура вершины дерева.

```
typedef struct tree_node
{
   char *str; //Данные
   struct tree_node *parent; // Указатель на родителя
   struct tree_node *left; // Указатель на левого потомка
   struct tree_node *right; // Указатель на правого потомка
} tree_node_t;
```

Интерфейс для работы со структурами.

Для работы с деревьями

```
// Очищение памяти, выделенной под дерево
void free_tree(tree_node_t *tree_node);

// Создание вершины дерева
tree_node_t *create_node(char *str);

// Вставка вершины в дерево
tree_node_t *tree_node_insert(tree_node_t *root, tree_node_t *node);

// Чтение дерева из файла
int tree_read(FILE *file, tree_node_t **root);

// Поиск вершины в дереве
tree_node_t *tree_node_search(tree_node_t *root, const char *str);

// Удаление вершины из дерева
tree_node_t *tree_node_remove(tree_node_t *root, tree_node_t *removed_node);

// Удаление всех вершин, начинающихся на задааную букву
tree_node_t *tree_remove_by_first_letter(tree_node_t *root, char letter, size_t
*removed_elems);

// Вывод дерева на экран
int open_dot_img(const char *file_name, tree_node_t *root);
```

4. Описание алгоритма.

Рассматривается алгоритм удаления слов, начинающихся на заданную букву.

Описание алгоритма:

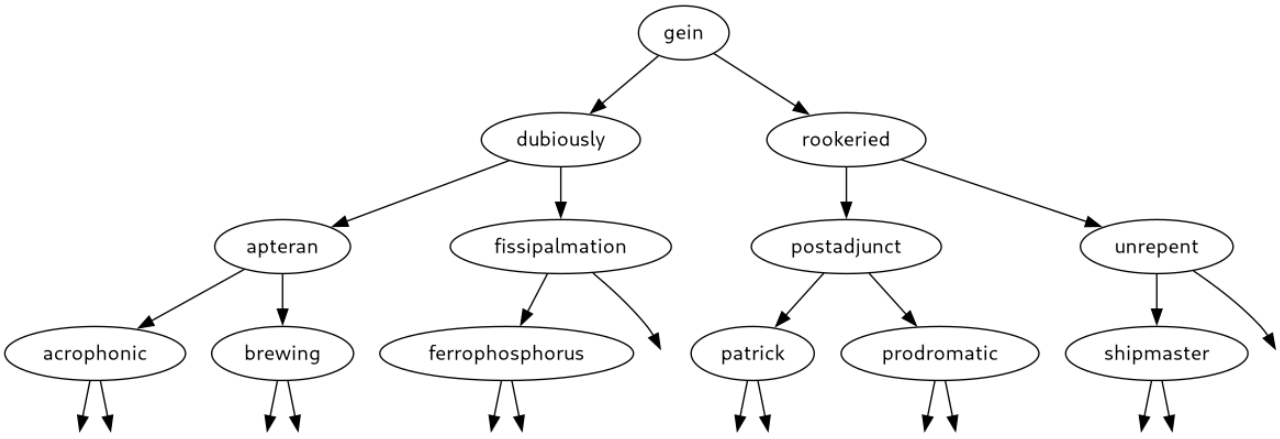
1. Рекурсивно вызывается функция для левого поддерева.
2. Затем рекурсивно вызывается функция для правого поддерева.
3. После того как обработаны оба поддерева, проверяется условие: если строка в текущем узле дерева начинается с буквы, то текущий узел удаляется.

Алгоритм удаления вершины:

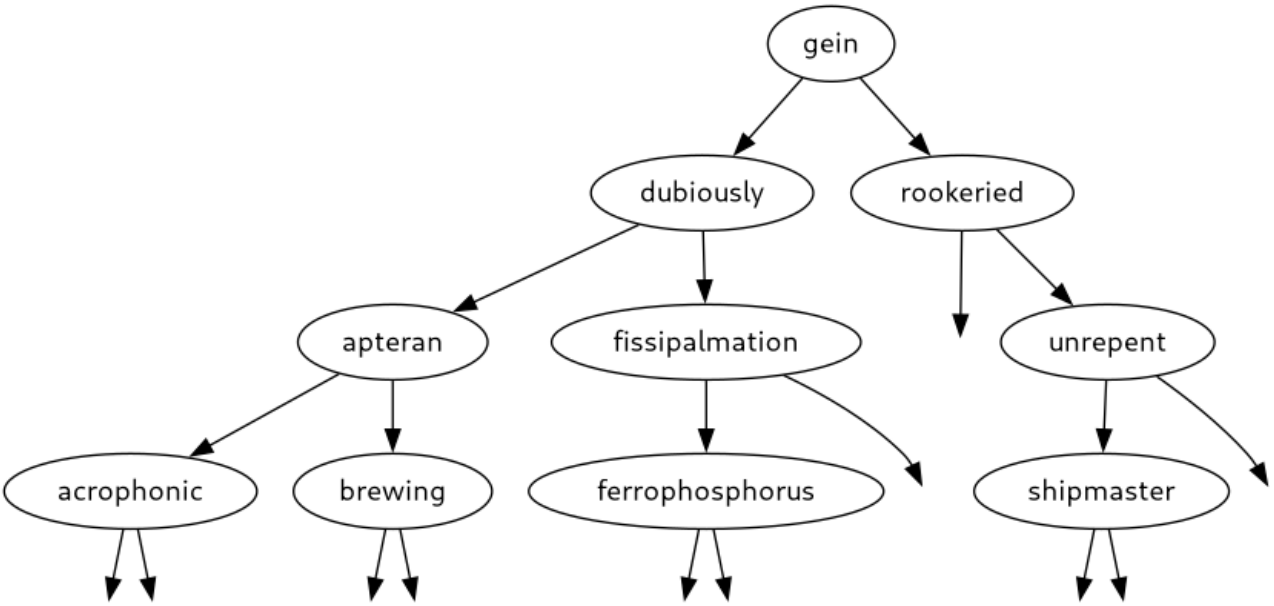
1. Вершина не имеет потомков:
    - Удаляется вершина, и указатель родителя заменяется на null
2. Вершина имеет одного потомка:
    - Удаляется вершина, создаётся новая связь между родителем удаляемого узла и его дочерним узлом. Родитель указывает на дочерний элемент удаляемой вершины.
3. Вершина имеет 2 потомка:
    - Ищется следующий за удаляемой вершиной элемент n (самый левый потомок правого поддерева), его данные копируются в удаляемую вершину, затем удаляется элемент n.

5. Тестирование задания.

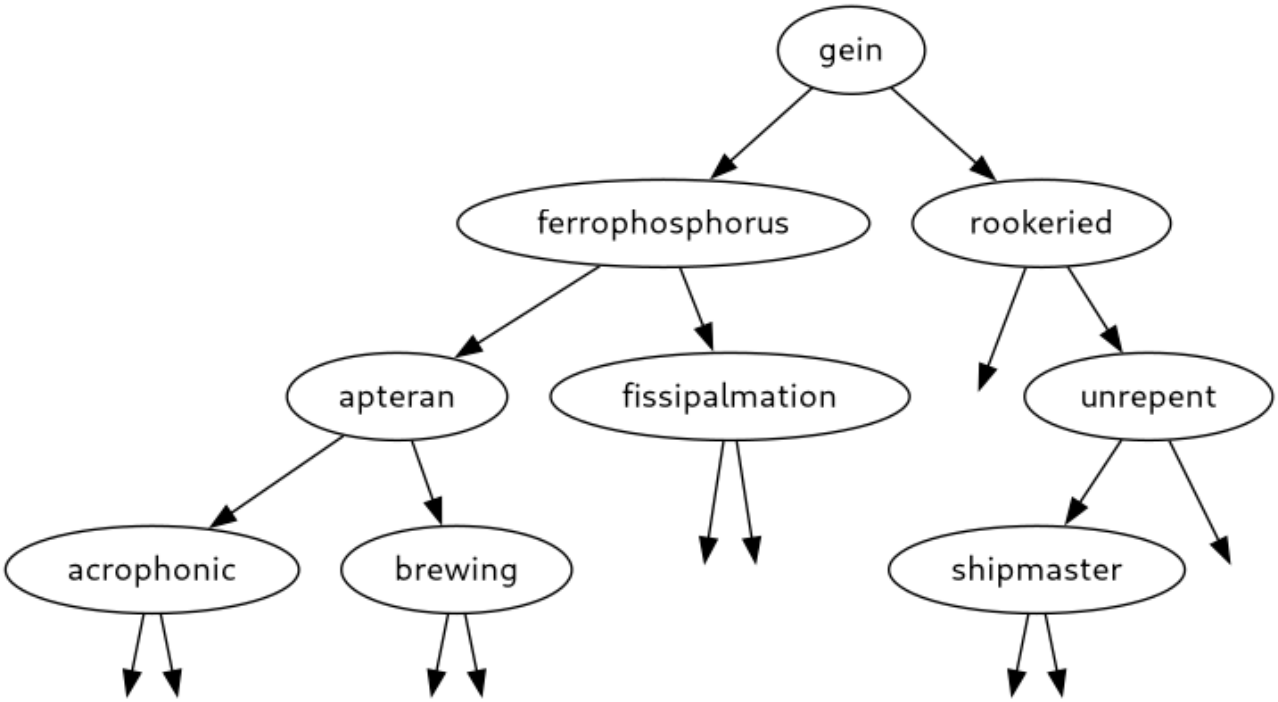
На входе имеем следующий граф:



После удаления всех слов начинающихся на «p» получаем:

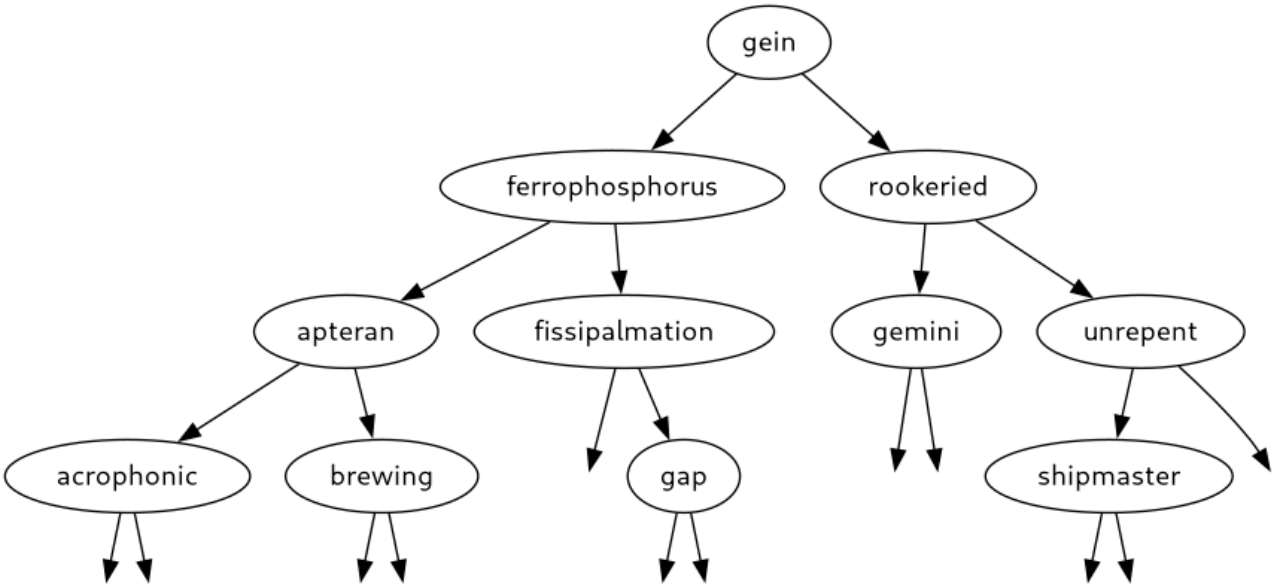


Проверим, правильность удаления вершины с двумя потомками, и удалим слово «dubiously»:



Можно заметить, что структура дерева не изменилась, а значит вершина удалилась правильно.

Добавим слово «gemini» и «gap»:



Вершины встали на правильные места.

6. Замеры.

Количество итераций = 10000.

Количество вершин	Балансировка	Время сортировки	Время поиска	Время удаления в дереве	Время удаления в файле
10	Сбалансировано	0.0357	0.0329	0.1202	2.1722
	Левостороннее	0.0336	0.0416	0.0847	2.1377
20	Сбалансировано	0.0308	0.0304	0.1471	2.1263
	Левостороннее	0.0303	0.0623	0.2405	2.1396
50	Сбалансировано	0.0394	0.0398	0.5019	2.1958
	Левостороннее	0.0354	0.1507	0.5520	2.2023
75	Сбалансировано	0.0316	0.043	0.4486	2.1417
	Левостороннее	0.0327	0.2213	1.0365	2.717
100	Сбалансировано	0.032	0.0471	0.6166	2.5063
	Левостороннее	0.0332	0.3346	1.2072	2.1341

По таблице, можно сделать вывод, что время сортировки не зависит от того сбалансировано ли дерево или нет, так как оно зависит только от количества элементов.

Время поиска, в свою очередь, напрямую зависит от сбалансированности дерева. Это связано с тем, что высота сбалансированного дерева будет равна log n соответсвенно скорость поиска элемента в худшем случае будет равна O(log n), а у левостороннего высота будет n, и скорость обработки будет равна O(n).

Для небольших объёмов данных (10–20 элементов) разница незначительна, но при увеличении набора данных неэффективность односторонних деревьев становится критичной.

7. Выводы по проделанной работе.

Деревья удобно применять, когда нужно упорядочено хранить элементы и когда требуется быстрый доступ к данным. Деревья также полезны для задач, связанных с иерархическим представлением данных, например, в файловых системах. Однако реализация вставки и удаления для деревьев, сложнее чем на массиве или на списке, так как нужно найти куда вставить полученный элемент, и рассмотреть все случаи при удалении элемента из дерева. Как было сказано выше, такое представление высокую эффективность операций(при сбалансированном дереве за O(log n), а также не требует заранее выделенной памяти. При разработке программы нужно тщательно протестировать корректное освобождение памяти, правильность операций, особенно удаления, когда вершина имеет два потомка, а также стоит оценивать сбалансированность дерева, ведь дерево иначе может превратиться в список.

Можно заметить, что удаление данных в дереве, всегда выигрывает удаление непосредственно в самом файле, так как удаление данных в файле требует перезаписи содержимого, что связано с большими затратами на операции ввода-вывода.

Контрольные вопросы.

1. Что такое дерево? Как выделяется память под представление деревьев?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Память под представление деревьев выделяется динамически под каждый узел, который хранит в себе данные, указатели на потомков и (необязательно) указатель на родителя.

2. Какие бывают типы деревьев?

По количеству потомков: бинарные, тернарные и т. д.; по сбалансированности: AVL-деревья, красно-черные, B-деревья.

3. Какие стандартные операции возможны над деревьями?

Вставка, удаление, поиск элементов, обход деревьев.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска – это такое бинарное дерево, в котором все значения левых потомков меньше значения родителя, а всех правых – больше.