

Условие задачи:

Программа строит дерево из букв введенной строки. 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.

Задание по варианту:

Построить двоичное дерево поиска из букв вводимой строки. Вывести его на экран в виде дерева. Выделить цветом все буквы, встречающиеся более одного раза. Удалить из дерева эти буквы. Вывести оставшиеся элементы дерева при постфиксном его обходе. Сравнить время удаления повторяющихся букв из дерева и из строки.

Техническое задание:

1. Входные данные: заглавные и строчные буквы английского алфавита и цифры для выбора команды программы(при вводе первого предложения можно использовать любые символы, но при построении дерева будут игнорироваться символы, которые не являются английскими буквами).

2. Выходные данные: в зависимости от команды данной программе.

Команды:

0. Завершение программы. Сообщение “Программа была завершена.”

1. Переформирование файла с визуализацией дерева, на основе предыдущих изменений. В случае если в памяти нет информации о дереве выводит сообщение “Невозможно построить дерево, отсутствуют его элементы.” Если же дерево можно построить - сообщение “Дерево было успешно построено”.

2. Вставка элемента в дерево. Если буква была введена неправильно – сообщение “Ошибка! Вы неправильно ввели букву для вставки.”, иначе - “Вставка успешно завершена.”.

3. Удаление элемента из дерева. Если буква была введена неправильно – сообщение “Ошибка! Вы неправильно ввели букву для удаления.”, иначе - “Удаление успешно завершено.” **В случае отсутствия элемента в дереве программа никак не меняет дерево и не выводит никаких специальных сообщений.**

4. Поиск узла с буквой. Если буква была введена неправильно – сообщение “Ошибка! Вы неправильно ввели букву для поиска.”, иначе – визуализация узла этой буквы и сообщение “Поиск успешно завершён.” **В случае отсутствия элемента в дереве программа не выводит никаких специальных сообщений.**

5. Выводятся оставшиеся в дереве буквы, в порядке постфиксного прохода.

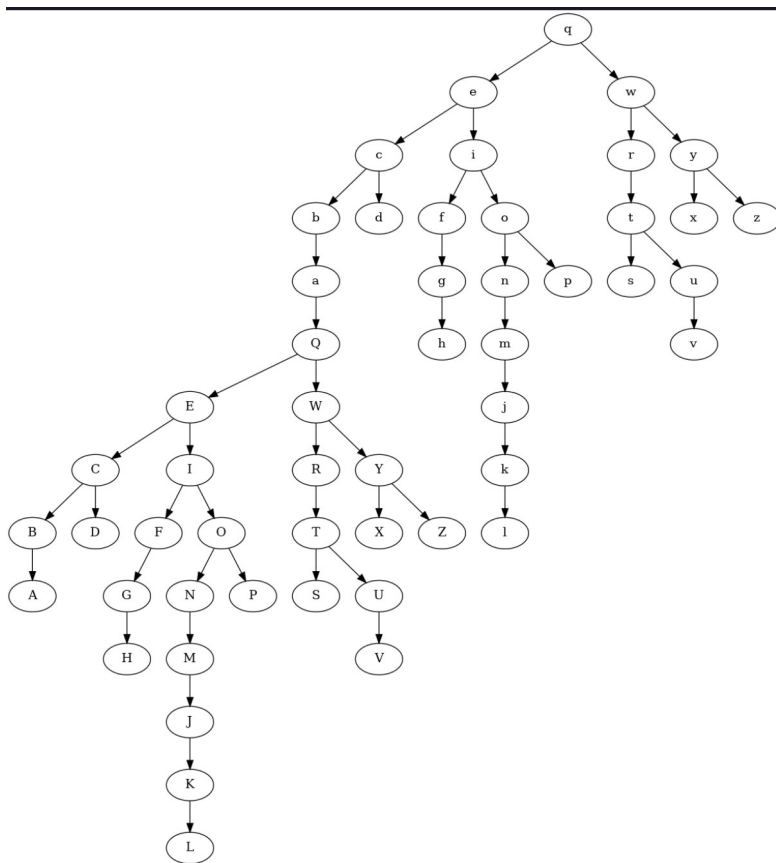
3. Количество элементов в дереве ограничено количеством букв в английском алфавите, умноженным на 2.

Описание структур данных:

```
typedef struct node
{
    bool sign; // стоит ли его раскрашивать при визуализации
    char name; // имя узла
    struct node *left; // левый узел
    struct node *right; // правый узел
    int height; // высота узла
} tree_n;
```

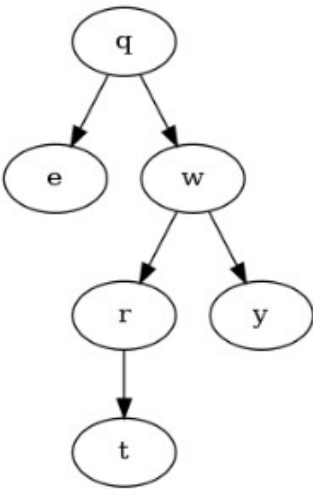
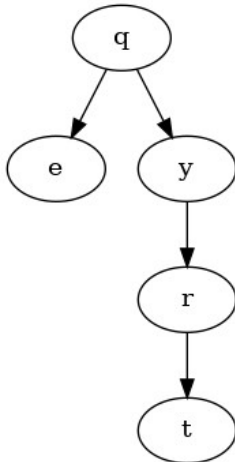
Эта структура – узел бинарного дерева.

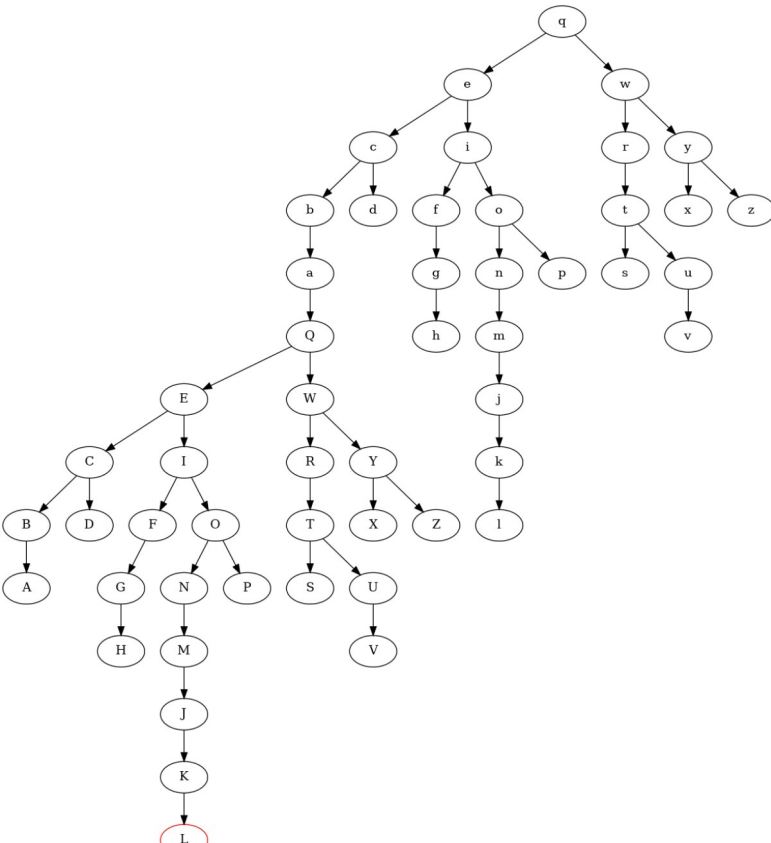
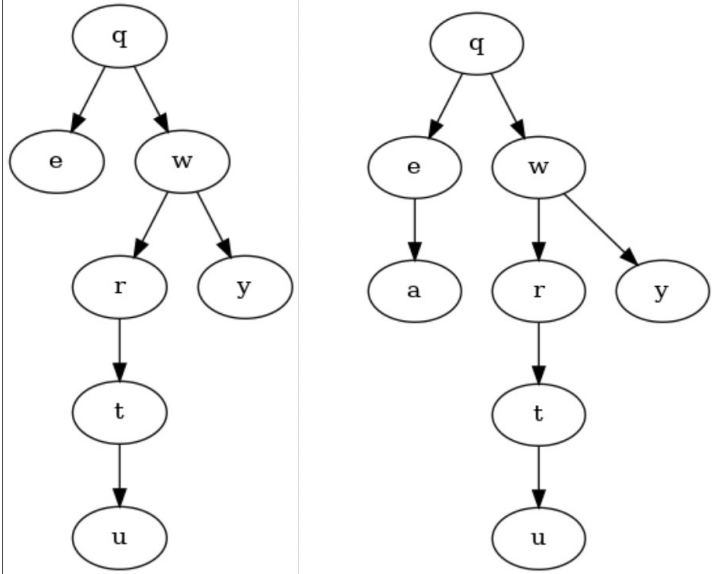
Тесты:

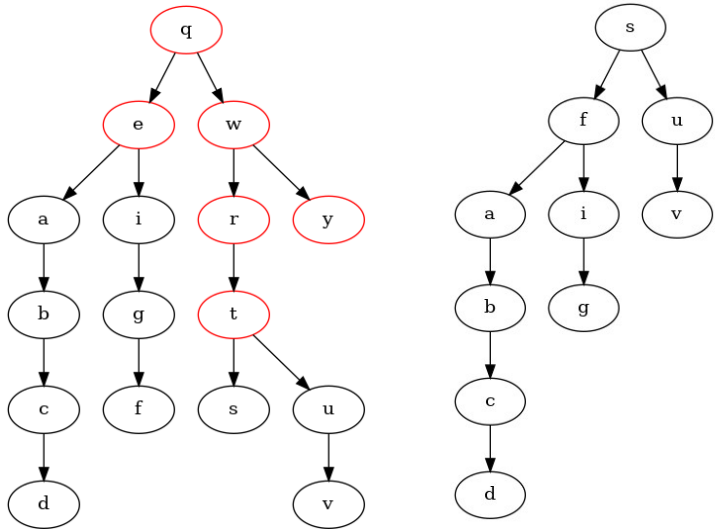
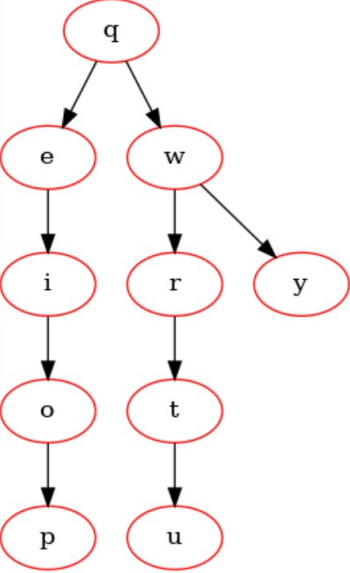


Примерный вид дерева из всех возможных букв (исходное дерево для всех тестов, где не указано исходное дерево)

| Входные данные: | Выходные данные: |
|---|--|
| Попытка построить дерево при отсутствии узлов | Невозможно построить дерево, отсутствуют его элементы. |
| Попытка ввести не букву при вставке/поиске/удалении | Ошибка! Вы неправильно ввели букву для |

| | |
|--|---|
| | вставки/поиска/удаления. |
| Некорректная введенная строка при первом ее запросе | Ошибка! Вы неправильно ввели строку! |
| Вводится строка из всех букв английского алфавита (заглавных и строчных) | Программа при вызове команды 1, составляется дерево из всех букв. |
| Удаление несуществующих узлов | Программа ничего не удаляет |
| Поиск несуществующего узла | Ошибка! Вы неправильно ввели букву для поиска. |
| Поиск существующего узла | <div> Введите номер действия, которое хотите совершить. Чтобы завершить ввод, напишите 0. 4 Введите букву, узел которой хотите вывести: f <pre> f \ g </pre> Поиск успешно завершен. </div> |
| Удаление существующего узла | <p>Удалит узел из памяти программы и сообщение “Удаление успешно завершено. Если удаляемого элемента не было в дереве, дерево не изменилось.”</p> <div> <p>Было, удалили “w”</p>  </div> <div> <p>Стало</p>  </div> |

| | |
|---|---|
| <p>Вставка существующего элемента</p> |  <p>Изменит флаг элемента на true, что будет означать, что элемент повторный, на визуализации элемент выделится красным цветом.</p> |
| <p>Вставка несуществующего элемента</p> | <p>Добавит узел с элементом в память программы При исходном дереве Получится</p>  <p>(Добавляем элемент “а”)</p> |
| <p>Удаление всех повторяющихся букв из дерева</p> | <p>В памяти программы не останется узлов с флагом true</p> |

| | |
|------------------------------|---|
| | <div><p>d c b a g i f v u s – оставшиеся вершины выведены в порядке постфиксного обхода.</p></div> |
| Удаление всех букв из дерева | <div><p>Удалятся все буквы из дерева Исходное дерево</p><p>Конечное дерево будет пустым и при попытке его построения будет выведено сообщение “Ошибка! Вы неправильно ввели строку.”</p></div> |
| Завершение программы | Завершение программы |

Замерный эксперимент для сортировки и поиска в зависимости от количества элементов:

| Количество элементов | Время заполнения дерева в мксек | Количество элементов в дереве | Время поиска в дереве в мксек |
|----------------------|---------------------------------|-------------------------------|-------------------------------|
| 10 | 8 | 5 | 15 |
| 30 | 13 | 10 | 20 |
| 50 | 17 | 20 | 25 |
| 100 | 18 | 30 | 28 |
| 500 | 25 | 40 | 30 |
| 1000 | 37 | 52 | 45 |

Замерный эксперимент зависимости времени удаления повторяющихся букв из дерева и из строки:

| Количество элементов | Время удаления дубликатов из строки, в мксек | Время удаления дубликатов из дерева, в мксек |
|----------------------|--|--|
| 10 | 0.4 | 4 |
| 30 | 0.7 | 7 |
| 50 | 1.2 | 12 |
| 100 | 2.3 | 13 |
| 500 | 2.4 | 15 |
| 1000 | 2.6 | 18 |

Вывод:

В результате выполнения лабораторной работы я выяснила, что в случае удаления элемента из дерева и из строки на небольших количествах обрабатываемых элементов дубликаты быстрее удаляются из строки. Если же количество элементов будет большим (хотя бы 10000), удаление из строки будет сильно медленнее чем удаление из дерева.

Таким образом деревья удобно использовать когда мы в реализации программы чаще ищем элемент дерева, чем модифицируем дерево, то есть если у нас есть статичная структура данных, менять которую нам не надо, мы берем дерево, потому что с ним быстрее работать в статике, если же структуру нужно постоянно модифицировать, легче взять другой тип данных. В тестировке дерева необходимо проверить такие крайние случаи как: удаление всех элементов из дерева, когда в дереве ничего не остается, удаление несуществующих узлов, поиск несуществующих узлов.

Контрольные вопросы:

1. Что такое дерево?

Дерево — нелинейная структура данных, используемая при представлении иерархических связей, имеющих отношение «один ко многим». Также деревом называется совокупность элементов, называемых узлами или вершинами, и отношений («родительских») между ними, образующих иерархическую структуру узлов.

2. Как выделяется память под представление деревьев?

Деревья могут представляться как списком, так и массивом. При реализации списком соответственно память выделяется под каждый элемент (для значения, правого и левого потомков), при реализации массивом выделяется с запасом фиксированная длина. При этом размер массива выбирается исходя из максимально возможного количества уровней двоичного дерева, и чем менее полным является дерево, тем менее рационально используется память.

3. Какие стандартные операции возможны над деревьями

Поиск, добавление, удаление, обход дерева, балансировка.

4. Что такое дерево двоичного поиска?

Двоичным деревом поиска называют дерево, все вершины которого упорядочены, каждая вершина имеет не более двух потомков (назовём их левым и правым), и все вершины, кроме корня, имеют родителя.