**Типы и структуры данных.**

**Лабораторная работа №6**

# Работа с деревьями.

**Выполнил**: Кузнецов Александр

**Группа**: ИУ7-33

**Цель работы** – получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов.

Построить дерево в соответствии с заданным 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 дерева и степени его ветвления.

**Задача 2:**

Построить двоичное дерево поиска, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Определить количество вершин дерева, содержащих слова, начинающиеся на указанную букву. Выделить эти вершины цветом. Сравнить время поиска начинающихся на указанную букву слов в дереве и в файле.

**Входные данные:**

Номер выполнения команды;

Символ для поиска

Ключ удаления

**Пример:**

//ввод команды для дерева

>>1

//ввод ключа удаления для дерева

>>aircraft

//ввод символа для поиска по дереву

>>a

**Выходные данные**

Время поиска элементов в файле и в дереве по первому символу;

Количество найденных слов по символу для поиска

**Функция программы**

Обработка дерева, измерение времени выполнения поиска по файлу и по дереву, поиск слов по первому заданному символу, отрисовка дерева в DOT.

**Аварийные ситуации**

**-** Элемент по заданному ключу не найден

- Неверно введена команда над деревом

**-** Не существует сыновей у заданного подкорня

- Не существует родителя у заданного подкорня

**Описание использованных структур данных**

//Узел дерева

struct Node{

char \*string;

struct Node \*pred;

struct Node \*right;

struct Node \*left;

};

**Алгоритм**

**Обработка ввода:**

1. Вывести интерфейс программы
2. Проверить корректность ввода (Интерфейс)
3. Запустить выбранную команду

-//////- Вывод сообщение об ошибке, если она есть -//////-

Иначе

**Обработка операций над деревом:**

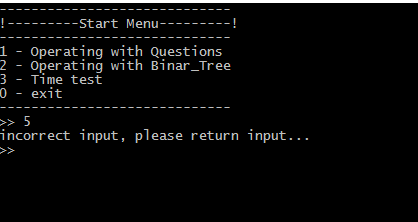
1. Работа с основным заданием
   1. Поиск по заданному символу и вывод количества найденных элементов.
   2. Запись дерева в DOT file
2. Работа с базовыми функциями дерева
   1. Поиск максимума
   2. Поиск минимума
   3. Запись в DOT
   4. Поиск по ключу
   5. Удаление подкорня
   6. Добавление подкорня
3. Расчет времени

**Обработка времени:**

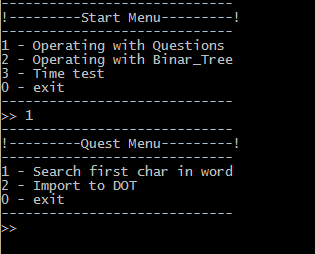
1. Расчет времени для поиска по дерева
2. Расчет времени для поиска по файлу

**Тесты**

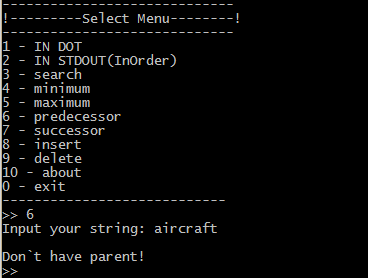
**Тест ввода:**Некорректный ввод команды дерева:



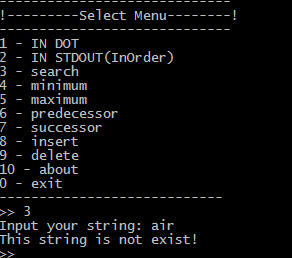
Корректный ввод номера операции над деревом:



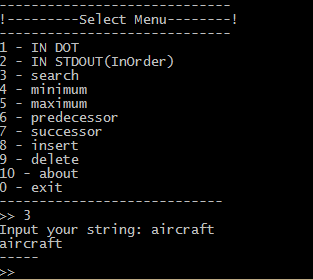
Тест на поиск несуществующего родителя:



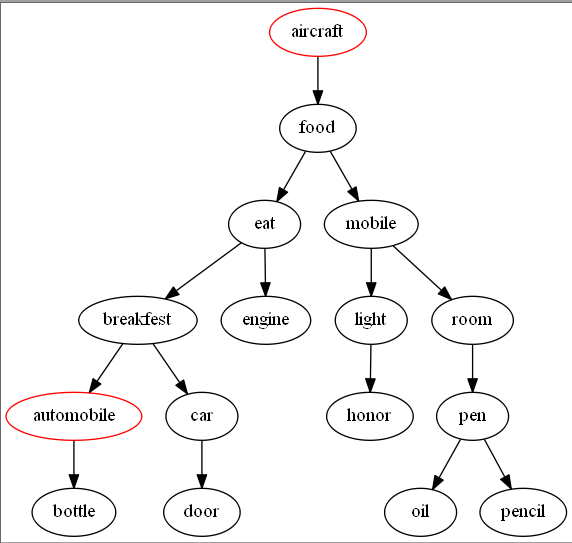
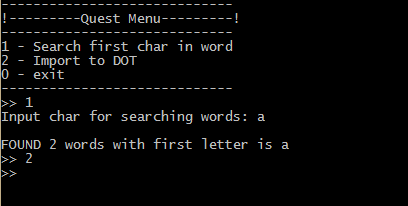
Тест на поиск элемента по ключу (Неуспешный)



Тест на поиск элемента по ключу (Успешный)



**Результат поиска по символу (a):**



**Сравнение методов**

При поиске среди 1000 слов по первому символу дерево выигрывает по времени в 3 раза, относительно чтения из файла.

|  |  |  |
| --- | --- | --- |
| N(элем) | Search\_in\_Tree (Ticks) | Search\_in\_File (Ticks) |
| 1000 | 41 | 122 |

Отсюда видно, что

**Эффективность по памяти**

При поиске по файлу мы храним только буферную переменную, для считывания текущего слова и сравнения его 1го элемента, в то время как

При поиске по дереву мы храним указатели на сыновей (2), указатель на родителя(1) и какую-то информацию в текущем поддереве.

Отсюда следует, что дерево расходует больше памяти, чем чтение из файла, однако поиск по нему работает быстрее, чем по файлу.

**Вопрос – Ответ**

*1. Что такое дерево?*

Дерево – нелинейная структура данных, которая используется для представления иерархических связей «один ко многим». Дерево с базовым типом Т определяется рекурсивно: это либо пустая структура (пустое дерево), либо узел типа Т с конечным числом древовидных структур того же типа – поддеревьев.

*2. Как выделяется память под представление деревьев?*

Выделение памяти под деревья определяется типом их представления. Это может быть таблица связей с предками (№ вершины - № родителя), или связный список сыновей. Оба представления можно реализовать как с помощью матрицы, так и с помощью списков. При динамическом представлении деревьев (когда элементы можно удалять и добавлять) целесообразнее использовать списки – т.е. выделять память под каждый элемент динамически.

*3. Какие бывают типы деревьев?*

АВЛ-деревья, сбалансированные деревья, двоичные, двоичного поиска.

*4. Какие стандартные операции возможны над деревьями?*

Основные операции с деревьями: обход (инфиксный, префиксный, постфиксный), поиск элемента по дереву, включение и исключение элемента из дерева.

*5. Что такое дерево двоичного поиска?*

Дерево двоичного поиска – дерево, в котором все левые потомки «моложе» предка, а все правые – «старше». Это свойство выполняется для любого узла, включая корень.