Лабораторная работа №6
по дисциплине «Типы и структуры данных»
Тема : Обработка деревьев.

Вариант 9 (задание 2)

Условие 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. 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:

Задание: Файл, содержащий слова (набор символов), которые разделяются пробелом или переходом на новую строку. (Количество слов в файле не ограничено.) Длина любого слова в файле не должна превышать 15 символов.

Работа с деревом: Добавлять / удалять / искать элементы в дереве можно с помощью консоли (ввод слова, которое нужно добавить / удалить / найти в дереве. Количество символов в слове не должно превышать 15 символов).

Выходные данные:

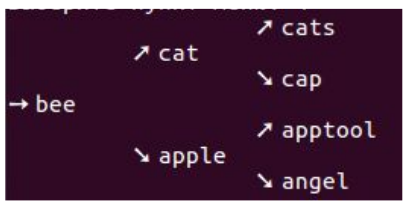
Задание : результатом работы программы является поиск в дереве слов, начинающихся на определенную букву и вывод на экран количества таких слов.
На экран выводится граф, состоящий из слов файла, в котором другим цветом выделены слова на определенную букву.
Также, на экран выводится время, которое потребовалось на поиск таких слов в дереве (в тиках) и в файле (в тиках).

Работа с деревом: результатом работы программы является построенное дерево двоичного поиска, которое можно вывести на экран.
Также выходными данными являются три последовательности узлов дерева, соответствующие трем способам обхода бинарного дерева.
После завершения работы программы с помощью команды в консоли “make graph” можно получить изображение созданного графа в формате .png

Меню работы с программой:

```
Меню работы с бинарным деревом:
(1) Добавить элемент (слово) в дерево
(2) Удалить элемент (слово) из дерева
(3) Поиск узла
(4) Печать дерева
(5) Обход дерева
(6) Выполнение задания лабораторной работы
(0) Завершение работы программы
```

(1) Добавление слова в дерево. Ввод слова после сообщения “Введите информацию для узла (не более 15 символов): ”. В случае успешного добавления - сообщение “Элемент добавлен”.
(2) Удаление узла из дерева. Ввод слова, узел с которым нужно удалить из дерева, после сообщения “Введите информацию для узла: ”. В случае успешного удаления - сообщение “Элемент удален”.
(3)Поиск узла в дереве. Ввод слова, узел с которым нужно найти, после сообщения “Введите информацию, которую нужно найти: ”. Если узел с введенным словом найден, то печатается дерево, где найденное слово выделено другим цветом, в противном случае сообщение “Не найдено”
(4)Печать дерева. Вывод дерева на экран. Если дерево на момент вызова печати дерева пусто - сообщение “Дерево пусто”. Если дерево не пусто - вывод в консоль его графического изображения.



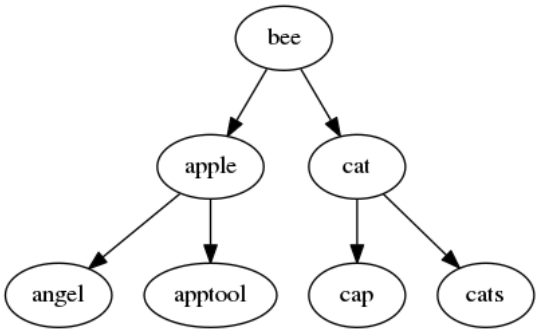
Верхние потомки - больше родителя, нижние потомки - меньше родителя. Какое из слов больше определяется в соответствии с лексикографическим порядком:

Слово *a* предшествует слову *b* ($a < b$), если первые m символов слов совпадают, а $m + 1$ символ слова *a* меньше (относительно отношения порядка, заданного в Σ) $m + 1$ символа слова *b*.

(5) Обход дерева. Вывод на экран три последовательности слов, представляющих собой три способа обхода дерева: прямой, поперечный и обратный. Например, для дерева из пункта (4):

```
Прямой обход графа:
bee apple angel apptool cat cap cats
Поперечный обход графа:
angel apple apptool bee cap cat cats
Обратный обход графа:
angel apptool apple cap cats cat bee
```

(6) Выполнение задания лабораторной работы. Считывание слов из файла и построение дерева с этими словами в вершинах двоичного дерева поиска. Ввод с клавиатуры буквы, с которой начинаются слова, которые нужно найти: сообщение “Поиск слов на определенную букву (введите букву): ”. Подсчет количества слов в дереве, начинающихся на эту букву. Замер времени, потраченного на поиск этих слов в дереве и в файле.



(0) Завершение работы программы. Генерация файла для DOT Graphviz из дерева, созданного с помощью пунктов (1)-(5). Освобождение памяти, занятой деревом.

Допущения: A и a - разные буквы.
1 < 10, но 10 < 2 (в соответствии с лексикографическим порядком)

Структура узла бинарного дерева:

```
struct BinaryTree
{
    char data[N];
    struct BinaryTree *left;
    struct BinaryTree *right;
    int color;
};
```

data - слово в узле
left / right - указатель на левого / правого потомка
color - цвет при печати дерева (0 - белый, 1 - красный)

Функции.

создание нового узла :
struct BinaryTree *add_new(char *data);
вставка узла в дерево :
struct BinaryTree *insert_element(struct BinaryTree *head, struct BinaryTree *new);
удаление узла с data из дерева :
struct BinaryTree *delete_element(struct BinaryTree *head, char *data);
вывод дерева на экран :
void print_tree(struct BinaryTree *head, int down, int lr);
поиск узла, содержащего data :
struct BinaryTree *search_element(struct BinaryTree *head, const char *data);
поиск слов в дереве, начинающихся на букву letter :
void search_words_to_same_letter(struct BinaryTree *head, char letter, int *count);
поиск слов в дереве, начинающихся на букву letter :
void search_words_in_file(FILE *input, char letter, int *count, char *word);
обходы дерева :
void pre_order(struct BinaryTree *def_head);
void in_order(struct BinaryTree *def_head);
void post_order(struct BinaryTree *def_head);
генерация информации для получения графа с помощью DOT :
void make_graph(struct BinaryTree *def_head, struct BinaryTree *parent, FILE *graph);
освобождение памяти, занимаемой деревом :
void free_tree(struct BinaryTree *head);

Время создания двоичного дерева поиска

Количество элементов	Время генерации дерева
10	34 571
25	66 242
100	168 115
500	525 009

Сравнение времени поиска в файле и в дереве.

Количество слов	Время поиска в дереве (в тиках).	Время поиска в файле (в тиках).
10	2 361	19 119
25	4 913	28 850
100	18 419	89 087
500	69 557	232 624

Вывод:

Алгоритм поиска слов в файле имеет линейную сложность O(n) т.е. пропорционален количеству элементов.
Алгоритм поиска слов в бинарном дереве поиска имеет O(log n), где n - количество слов. Поиск в дереве занимает в 7-10 раз меньше времени, чем в файле. Таким образом, дерево – это более подходящая структура для организации поиска, чем структуры данных, в которых для поиска информации требуется просмотреть весь объем данных(линейный список и др.)

Вопросы к лабораторной работе.

1. Что такое дерево?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Дерево с базовым типом T определяется рекурсивно либо как пустая структура (пустое дерево), либо как узел типа T с конечным числом древовидных структур этого же типа, называемых поддеревьями.

2. Как выделяется память под представление деревьев?

Способ выделения памяти под деревья определяется способом их представления в программе. Например с помощью матрицы или списка может быть реализована таблица связей с предками или связный список сыновей. Целесообразно использовать списки для упрощенной работы с данными, когда

элементы требуется добавлять и удалять, т. е. выделять память под каждый элемент отдельно.

3. Какие бывают типы деревьев?

Типы деревьев: АВЛ-деревья, сбалансированные деревья, двоичные деревья, деревья двоичного поиска.

4. Какие стандартные операции возможны над деревьями?

Основные операции над деревьями - обход дерева (инфиксный, префиксный, постфиксный), поиск элемента в дереве, добавление и удаление элемента в дерево.

5. Что такое дерево двоичного поиска?

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