Лабораторная работа №6
Обработка деревьев

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

Задание (Вариант 2):

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.

Требования к входным данным:

Программа предназначена для работы с текстовым файлом. Файл может содержать произвольное количество слов. Словом считается любая последовательность печатных символов (т.е. слова разделяются пробелами или символами перевода строк). Максимально возможная длина слова - 256.

Также, слова в дерево поиска можно добавлять через консоль. (Требования аналогичные.)

Выходные данные:

По требованию пользователя программа печатает содержимое дерева (можно выбрать способы обхода, а также способ отображения целого дерева). Также, программа определяет, есть ли в дереве искомое слово. Также она осуществляет поиск по дереву слов, начинающихся с определенной буквы (после этого она печатает дерево на экран, выделяя найденные слова красным цветом).При необходимости, программа производит сравнение времени поиска слова в файле и в дереве. Примеры см. Интерфейс программы.

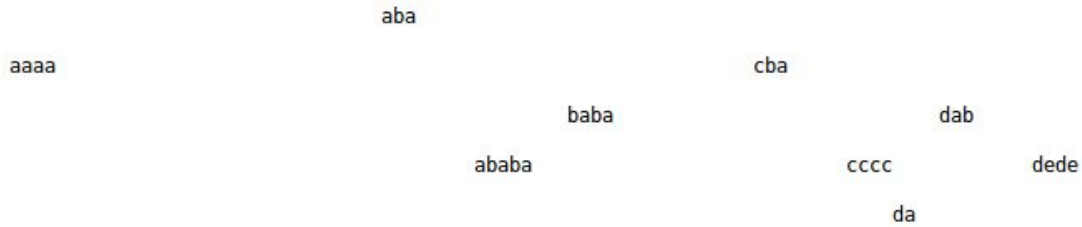
Интерфейс программы:

```
************************************************************
Выберите одно из следующих действий:
0: Добавление в дерево слов из файла
1: Отобразить дерево (1 способ)
2: Отобразить дерево (2 способ)
3: Добавить слово в дерево
4: Удалить слово из дерева
5: Поиск слова
6: Найти кол-во слов, начинающихся на одну букву, выделить их
7: Сравнить время поиска слов из файла и в дереве
8: Очистить дерево
9: Обход дерева
10: Закончить работу
```

0 - Добавление в дерево слов, заранее подготовленных и записанных в файле data.txt. Пример содержания файла:

aba cba
dab baba
ababa dede
aaaa cccc da

1 - Печать дерева первым способом. Ограничение: программа может распечатать дерево в таком виде, если длина его слов не больше 6. При этом производится печать первых 5 уровней (это связано с наглядностью расположения эл-тов в дереве)



2 - Печать дерева вторым способом. Сыновья расположены правее родителей. Верхний сын является левым сыном, нижний - правым сыном для каждого элемента. Для удобства, если у какого-либо элемента есть только один потомок, на месте второго печатается прочерк (-----).

```
aba
|----aaaa
|----cba
|    |----baba
|    |    |----ababa
|    |    |---------
|    |----dab
|    |    |----cccc
|    |    |    |---------
|    |    |    |----da
|    |    |----dede
```

3 - Добавление слова в дерево. Программа предложит ввести слово в строке консоли.
4 - Удаление слова. Если дерево не содержит указанного слова, программа выдает соответствующее сообщение, иначе - сообщение об успешном удалении.
5 - Поиск слова. Программа запрашивает у пользователя слова и выдает результат: содержится ли указанное слово в дереве, или нет.

6 - Поиск слов, начинающихся на указанную букву, их печать. Подсчет их количества, а также печать дерева с выделением найденных слов.

```
Введите букву: a
Искомые слова: aba aaaa ababa
Количество слов: 3
aba
|----aaaa
|----cba
|    |----baba
|    |    |----ababa
|    |    |---------
|    |----dab
|    |    |----cccc
|    |    |    |---------
|    |    |    |----da
|    |    |----dede
```

7 - Печать времени поиска слова в файле (bigdata.txt) и в дереве.
8 - Удаляет все элементы дерева
9 - Обход дерева - печать всех его элементов в заданном порядке

```
1: Сверху вниз (корень, левый, правый) - префиксный обход
2: Слева направо (левый, корень, правый) - инфиксный обход
3: Сверху вниз (левый, правый, корень) - постфиксный обход
```

9.1 Префиксный обход. Пример:
aba aaaa ababa baba cba cccc da dab dede
9.2 Инфиксный обход. Пример:
aaaa aba ababa baba cba cccc da dab dede
9.3 Постфиксный обход. Пример:
aaaa ababa baba cba cccc da dab dede aba
10 - Закончить работу

Обращение к программе:
Через консоль

Внутренние структура данных:
Структура элемента дерева

```
struct element {
  char word[LEN_WORD];
  element *left = NULL;
  element *right = NULL;
  element(char *str) {
     assert(strlen(str) < LEN_WORD);
     strcpy(word, str);
     left = NULL;
     right = NULL;
  }

};
//Binary search tree
class BST {
private:
```

```
    element *head = NULL;
    bool printable = true;
    void print(element* tmp, int deep, bool flag, char c = '\0');
    void delete_tree(element* tmp);
    int for_find_letter(element* tmp, char c = '\0');
    void operator_copy(element **head, element *tmp);
    void print_tree(element *h, char c = '\0');
    void RootLR(element *tmp);
    void LRootR(element *tmp);
    void LRRoot(element *tmp);
    void delete_remove(element* prev, element *tmp);
public:
    BST(const BST &obj);
    BST();
    int find_letter(char c);

    BST& operator=(const BST& obj);
    void insert(char *str);
    bool remove(char *str);
    ~BST();
    void show(char c = '\0');
    void show_as_tree(char c = '\0');

    bool find(char *str);
    void RootLeftRight();
    void LeftRootRight();
    void LeftRightRoot();

};
```

Сравнение времени поиска в файле и в дереве:
Время поиска в файле имеет линейную сложность, т.е. пропорционально количеству слов в файле. При этом, в лучшем случае, искомое слово будет находится в начале файла, т.е. время его поиска будет составлять O(1). Поэтому будем рассматривать худший вариант: искомое слово расположено в конце файла.
Сложность поиска слова в дереве O(logn), где n - количество слов.

№	Количество слов	Время поиска в дереве(мкс)	Время поиска в файле(мкс)
1	100	3	51
2	300	7	81
3	1000	20	300

Вывод:

Основным преимуществом двоичного дерева поиска перед другими структурами данных является возможная высокая эффективность реализации

основанных на нём алгоритмов поиска и сортировки.

Хранение и обработка дерева требует аккуратного обращение с памятью. Поэтому для этой структуры данных нужно особенно тщательно тестировать удаление элементов, а также добавление элементов.

Теоретическая часть:

1. Что такое дерево?

Дерево – нелинейная структура данных, которая используется для представления иерархических связей «один ко многим». Дерево с базовым типом T определяется рекурсивно: это либо пустая структура (пустое дерево), либо узел типа T 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.