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

Студент **Простев Тимофей Александрович**

Группа **ИУ7-33Б**

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

Студент _____ **Простев Т. А.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

Оценка _____

2023 г.

1. Описание условия задачи.

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.

2. Техническое задание.

Исходные данные и правила их ввода:

- Номер пункта меню: целое число от 1 до 9:
  - 1 - Считать дерево из файла.
  - 2 - Вывести дерево на экран.
  - 3 - Посчитать количество слов, начинающихся на символ.
  - 4 - Вывести на экран дерево с окрашенными словами, начинающиеся на символ.
  - 5 - Добавить слово в дерево.
  - 6 - Удалить слово из дерева.
  - 7 - Присутствует ли слово в дереве.
  - 8 - Провести исследование.
  - 9 - Выход из программы.
- Имя файла: стока, не больше 255 символов.
- Символ для поиска слов: 1 символ.
- Слово для добавления, удаления, поиска: строка, не больше 255 символов.

Выходные данные:

- При выводе дерева: png-изображения на экран.
- При выводе количества слов, начинающихся на символ: целое положительное число.
- При поиске слова в дереве: сообщение пристутствии или отсутствии слова в дереве.
- При проведении исследования: таблица с результатами исследования.

Способ обращения к программе:

Запуск исполняемого файла. В рабочей директории выполнить команду ./app.exe.

Аварийные ситуации:

• Ввод некорректных данных.

• Ошибка выделения памяти.

В случае аварийной ситуации выводится сообщение об ошибке.

3. Описание внутренних структур данных.

Структура для хранения узла дерева:

```
struct tree_node {
      Char *data;
      tree_node_t *smaller;
      tree_node_t *bigger;
};
```

Структура содержит 3 поля:

- data - указатель на строку.
- smaller - указатель на меньший узел.
- bigger - указатель на больший узел.

Структура для хранения массива слов:

```
struct words {
    char **words;
    size_t size;
};
```

Структура содержит два поля:

- words - массив строк.
- size - количество слов в массиве - беззнаковое целое типа size_t.

Структура данных исследования:

```
typedef struct research {
      size_t words_count;
      double average_search_tree;
      double average_search_file;
} research_t;
```

Структура содержит 3 поля:

- words_count - количество слов в файле и в дереве.
- average_search_tree - среднее время поиска всех слов, начинающихся на символ в дереве.
- average_search_file - среднее время поиска всех слов, начинающихся на символ в файле.

Для работы со структурами используются следующие функции:

`search_tree_t *search_tree_new();` - Создание дерева.

`void search_tree_delete(search_tree_t *tree);` - Удаление дерева с освобождением памяти.
`void search_tree_for_each(search_tree_t *tree, void (*function)(void *data));` - Применить функцию к каждому элементу в дереве.
`int search_tree_insert(search_tree_t *tree, char *value);` - Вставить слово в дерево.
`int search_tree_remove(search_tree_t *tree, char *value);` - Удалить слово из дерева.
`int search_tree_search(search_tree_t *tree, char *value,);` - Присутствует ли слово в дереве.
`size_t search_tree_first_letter_count(search_tree_t *tree, char letter);` - Посчитать количество слов в дереве, начинающихся на символ.
`void search_tree_to_dot(FILE *stream, search_tree_t *tree);` - Перевести дерево в dot формат в файл.
`void search_tree_to_dot_first_letter(FILE *stream, search_tree_t *tree, char letter);` - Перевести дерево в dot формат с подстветкой слов, начинающихся на символ.
words_t* words_new(size_t size); - Создание массива слов.
void words_delete(words_t *words); - Освобождение памяти, выделенной под массив слов.
int words_file_scan(words_t *words, size_t count, FILE *file); - Считать слова из файла.
int words_file_scan(words_t *words, size_t count, FILE *file); - Записать слова из массива в бинарное дерево поиска.

4. Описание алгоритма.

Выводится меню программы. Пользователь вводит номер команды, после чего выполняется действие.

При добавлении слова в дерево временный указатель проходит по узлам дерева, пока не дойдёт до листа. Проход выполняется при помощи цикла. Данные в узле сравниваются со вставляемыми, после чего указатель меняется на указатель на меньший или больший узел. После того, как было найдено место для вставки, создается новый узел, который подставляется на место указателя.

При поиске слова в дереве слово сравниается со словом в вершине дерева. Если слово больше, алгоритм вызывается рекурсивно для правого наследника, если меньше - для левого. Алгоритм работает до того, как слово будет найдено или пока не дойдёт до неподходящего листа дерева.

При удалении слова из дерева, слово сравнивается со словом в вершине. Если слово оказалось больше, чем слово в вершине, алгоритм рекурсивно вызвается для левого потомка, если меньше - для правого. Если слово совпало со словом вершине, проверяется наличие всех потомков. Если один из потомков отсутствует, на место данной вершины встает второй, а данная вершина удаляется. Если существет оба потомка, в правом поддереве производится поиск наименьшего элемента, слово которого встает на место слова в данной верщине, после чего удаляется найденный наименьший элемент правого поддерева.

При поиске всех слов, начинающихся на символ, алгоритм обходит все дерево при помощи инфиксного обхода.

5. Оценка эффективности.

При проведении исследования по 100 раз проводился поиск слов, начинающихся на символ ‘a’ в файле и в сгенерированном на основе этого же файла бинарном дереве. Исследование проводилось над 20 файлами со случайно сгенерированными наборами слов в количестве от 10000 до 100000.

Количество слов в файле.	Время поиска в файле. мс.	Время поиска в дереве. мс.
500	46	4
1000	90	10
10000	872	232
15000	1306	371
20000	1742	503
25000	2191	698
30000	2652	864
35000	3134	1166
40000	3537	1480
45000	4029	1807
50000	4414	2042
55000	4877	2223
60000	5244	2476
65000	5687	2828
70000	6128	3220
75000	6537	3613
80000	7083	3998
85000	7397	4294
90000	7902	4798
95000	8288	5063

Время поиска в бинарном дереве всегда меньше, чем время поиска в файле.

6. Вывод.

Использования бинарного дерева для хранения данных выгодно, когда в наборе данных нужно производить много операция поиска, так как поиск в бинарном дереве поиска происходит в среднем за O(log n). При этом вставка и удаление из дерева производятся так же за O(log n). В худшем случае все эти операции могут производится за O(n). Время выполнения сильно зависит от порядка вставки элементов в дерево.

7. Контрольные вопросы.

1. Что такое дерево?

Дерево — это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

2. Как выделяется память под представление деревьев?

Дерево представляется нелинейная структура, состоящая из узлов, в которых хранятся указатели на данные и на предков, поэтому память выделяется под каждый узел.

3. Какие бывают типы деревьев?

Бинарные деревья, двоичные деревья поиска, АВЛ-деревья, красно-черные деревья.

4. Какие стандартные операции возможны над деревьями?

Поиск в дереве, добавление в дерево, удаление из дерева, обход дерева.

5. Что такое дерево двоичного поиска?

Дерево двоичного поиска - дерево, в котором все левые потомки меньше предка, а все правые - больше.