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
«ОБРАБОТКА ДЕРЕВЬЕВ»

Студент Равашдех Фадей Хешамович

Группа ИУ7 – 35Б

Преподаватель Никульшина Т. А.

2023

Оглавление

Описание условия 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 частотный словарь (слово – количество повторений) из слов текстового файла в виде дерева двоичного поиска. Вывести его на экран в виде дерева. Осуществить поиск указанного слова в дереве и в файле. Если слова нет, то (по желанию пользователя) добавить его в дерево и, соответственно, в файл. Сравнить время поиска слова в дереве и в файле.

Описание технического задания

Входные данные: Строка пути к файлу с данными, слова для их поиска и добавления в дерево.

Выходные данные: Считаное из файла бинарное дерево, результат поиска в дереве и сравнение количества сравнений, времени поиска и занимаемой памяти между файлом и деревом.

Обращение к программе:

Запускается через терминал командой: ./app.exe.

Функции меню программы:

0. Информация о программе
1. Считать частотный словарь из файла
2. Вывести частотный словарь в виде дерева
3. Найти слово в дереве и в файле
4. Сравнение времени поиска слова в дереве и в файле
5. Выйти

Аварийные ситуации:

1. Ошибка выделения памяти;
2. Прекращение ввода (EOF);
3. Некорректный ввод;
4. Некорректный выбор действия меню программы;
5. Пустая строка;
6. Некорректный файл.

НАБОР ТЕСТОВ

№	Название теста	Пользовательский ввод	Вывод
1	Некорректнрое действие	10	Некорректный номер пункта меню. Действия не существует. Ошибка: Действие с введенным пунктом отсутствует.
2	Некорректный ввод файла	1 \n	Ошибка: Некорректный ввод.
3	Работа с некорректным файлом	1 not_existing_file.txt 4	Ошибка: некорректный файл.
4	Некорректный ввод	1 data/10 \n	Ошибка: Некорректный ввод.
5	Корректный ввод и вывод дерева	1 data/10 2	Вывод дерева: '0' - 6 ├── NULL └── NULL

Описание структур данных

Основные структуры дерева

```
// Структура данных узла дерева: слово и его частота в файле
typedef struct
{
    char *key;
    int value;
} pair_t;

// Структура узла дерева
struct bintree
{
    pair_t data;
    bin_tree_t left;
    bin_tree_t right;
};
```

Вспомогательные структуры для обхода по дереву

```
// Структура для замера времени поиска
struct measure_times
{
    double tree_timer;
    double file_timer;
    char *filename;
    bin_tree_t tree;
};
// Структура для списка для стека для вывода дерева
typedef struct list *list_t;
struct list
{
    char ch;
    list_t next;
};
```

Описание функций программы

Функции основного файла программы

```
int controller(int option, bin_tree_t *tree, char **filename);

int main(void);
```

Функции бинарного дерева

```
void bin_tree_clear(bin_tree_t *tree);

bt_rc_t bin_tree_insert(bin_tree_t *tree, const char *key);

bt_rc_t bin_tree_find(const bin_tree_t tree, const char *key,
int **num, int *cmp_count);

bt_rc_t bin_tree_remove(bin_tree_t *tree, const char **key);

void bin_tree_each(const bin_tree_t tree, void (*action)(const
char *key, int *num, void *param), void *param);
```

Функции-контроллеры

```
int read_freq_list(bin_tree_t *tree, char **filename);

void print_freq_list(bin_tree_t tree);

int find_in_freq_list(bin_tree_t *tree, char *filename);

int find_in_file(char *filename, char *str, int *res, int
*cmp_count);
```

Функции сравнения

```
int compare(char *filename, bin_tree_t tree);


int file_find(char *filename, char *str);


void measure_search(const char *key, int *value, void *n);
```

```
int compare_time(char *filename, bin_tree_t tree);

int compare_cmps(char *filename, bin_tree_t tree);

void count_mem(const char *key, int *value, void *n);

int compare_mem(char *filename, bin_tree_t tree);
```

Функции ввода/вывода

```
// Вывод меню программы
void print_menu(void);

// Вывод информации о программе
void print_info(void);

int get_input(char **str, FILE *f);

// Чтение строки определенной длины из стандартного потока ввода
int get_n_input(char pch[], size_t n);

// Получение выбранного пользователем пункта
int get_option(size_t *option);

// Вывод сообщения, поясняющего ошибку
void print_error_msg(int rc);

// Вывод сообщения, поясняющего ошибку, и возврат того же кода возврата
int print_error(int rc);
```

Оценка среднего количества сравнений

Количество узлов дерева	Файл	Дерево
1	1.000000	1.000000
10	7.600000	2.971429
100	80.633838	7.512626
1000	733.199754	10.462162
10000	7626.300193	15.448150

Можно заметить, что рост количества сравнений в файле зависит от количества элементов линейно, а рост количества сравнений в дереве зависит от количества узлов примерно логарифмически. Т.е. в дереве делается меньше сравнений чем в файле.

Оценка эффективности поиска (с)

Количество узлов дерева	Файл	Дерево
1	0.000013	0.000001
10	0.000115	0.000007
100	0.001585	0.000046
1000	0.024919	0.000187
10000	2.349096	0.002066

Можно заметить, что рост времени поиска в файле зависит от количества элементов линейно. Поиск в дереве гораздо быстрее чем поиск в файле. Рост времени поиска в файле больше роста времени поиска в дереве.

Оценка памяти (байт)

Количество узлов дерева	Файл	Дерево
1	12	30
10	70	300
100	1158	3090
1000	15840	31890
10000	195407	328890

Можно заметить, что рост затрачиваемой памяти в файле и в дереве зависит от количества элементов линейно. Но при этом на малых данных дерево занимает больше памяти, чем файл. Также можно заметить, что рост затрачиваемой памяти в файле выше, чем рост затрачиваемой памяти в дереве и если средняя частота слова > 1 (в данном случае примерно равна 4), то при достаточном количестве элементов память дерева сравняется с памятью файла. Память в файле равна P*N*(log2(N) + 1), а память дерева равна N*(log2(N) + 29), где N – количество элементов, а P средняя частота повторения слова.

Описание алгоритма

1. Поиск в дереве двоичного поиска: сравнение искомого слова со словом данного узла. Если искомое слово равно, то слово найдено, если меньше, то вызывается функция поиска от левой вершины, иначе от правой. Если в итоге узел пуст, то слово не найдено.
2. Вставка элемента: проход рекурсивно по дереву как при поиске, но если находит искомый узел, то увеличивает его чатоту на единицу, а если узел пуст, то создает элемент с переданным словом и вставляет его на место пустого.
3. Удаление элемента: Если узел пуст, то возврат. Если узел не имеет потомков, то удаление узла. Если у узла один потомок, то узел удаляется, а на его место встает его потомок. Если у узла два потомка, то находится самый правый потомок у левого потомка и данные заменяются на найденные, а найденый потомок удаляется.
4. Обход дерева: Обход выполняется префиксно, сначала обрабатывается узел дерева с помощью переданной функции и переменной возврата, потом рекурсивный обход левого и правого потомков.

Ответы на контрольные вопросы

1. Что такое дерево? Как выделяется память под представление деревьев?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Память выделяется под каждый узел дерева с учетом хранения в нем данных узла и указателей на потомков узла.

2. Какие бывают типы деревьев?

Двоичное (бинарное) дерево, дерево двоичного поиска, сбалансиррованное дерево.

3. Какие стандартные операции возможны над деревьями?

Поиск в дереве, обход дерева с применением к узлам переданной функции, добавление и удаление элемента в дереве.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше.

Вывод

Среднее количество сравнений в двоичном дереве поиска примерно равно log2(N), что подтвержает теоетические расчеты.

Таким образом с помощью двоичного дерева поиска можно быстро искать, вставлять и удалять данные, т. к. асимптотика поиска равна log2(N), поэтому поиск в дереве занимает меньше времени, чем поиск в файле.

Частотный словарь на дереве затрачивает больше памяти, чем обычное хранение данных в файле на малых данных, т. к. помимо слова хранится еще и ссылкии на потомков дерева. Но если элементов будет слишком много или слова будут повторятся слишком часто, то частотный словарь будет эффективнее по памяти, т. к. не хранит повторяющиеся строки.