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 № 2

«Записи с вариантами, обработка таблиц»

Студент Маслова Марина Дмитриевна
фамилия, имя, отчество

Группа ИУ7-33Б

2020 г.

Оглавление

Техническое задание

Условие задачи

Создать таблицу, содержащую не менее 40-ка записей с вариантами (объединениями). Упорядочить данные в ней по возрастанию ключей, двумя алгоритмами сортировки, где ключ – любое невариантное поле (по выбору программиста), используя: а) саму таблицу, б) массив ключей. Реализовать возможность добавления и удаления записей в ручном режиме. Осуществить поиск информации по варианту.

Ввести список машин, имеющихся в автомагазине, содержащий: марку автомобиля, страну-производитель, цену, цвет и состояние: новый – гарантия (в годах); нет - год выпуска, пробег, количество ремонтов, количество собственников. Вывести цены не новых машин указанной марки с одним предыдущим собственником, отсутствием ремонта в указанном диапазоне цен.

Входные данные

Файл с данными в формате csv (важно представление данных, а не расширение файла). Первая строка в файле содержит заголовки полей, все последующие строки содержат значения полей, разделенные запятой «,», в случае отсутствия поля у данной записи на месте данных должен стоять знак минус «-».

Целое число, задающее выбор пункта меню.

Строковые и целочисленные данные при добавлении, удалении и поиске записей.

Выходные данные

Таблица, выгруженная из файла.

Текущее состояние таблицы до/после добавления, удаления из нее записей.

Исходная таблица или таблица ключей в отсортированном виде.

Результат поиска по таблице.

Количественная характеристика сравнения эффективности вариантов сортировки.

Задачи, реализуемые программой

1. Выгрузка данных из файла.

2. Просмотр таблицы.

3. Добавление информации о новой машине.

4. Удаление информации о машинах с заданной ценой.

5. Просмотр отсортированную по цене таблицы ключей.

6. Просмотр упорядоченной по цене таблицы.

7. Просмотр исходной таблицы в упорядоченном по цене виде по упорядоченной таблице ключей.

8. Сравнение эффективности сортировки способами 6 и 7.

9. Сравнение времени работы сортировок разной сложности (пирамидальной O(n*log(n)) и вставками $O(n^2)$.

10. Поиск не новых машин указанной марки с одним предыдущим собственником, отсутствием ремонта в указанном диапазоне цен.

Способ обращения к программе

Через терминал.

Возможные аварийные ситуации и ошибки пользователя

Некорректный ввод пункта меню.

Некорректный ввод строковых данных (превышение длины строки).

Некорректный ввод целочисленных данных.

Попытка просмотра, сортировки, удаления и поиска данных в пустой таблице.

Описание внутренних структур данных

Для хранения каждой записи таблицы использована структура:

```
typedef struct
{
   char brand[MAX_BRAND_LEN + 1];
   char country[MAX_COUNTRY_LEN + 1];
   char color[MAX_COLOR_LEN + 1];
   unsigned int price;
   condition_t condition;
   info_t more_info;
} car_t;
```

Поля структуры **cat_t**:

brand – строка, хранящая название марки автомобиля;

coutry – строка, хранящая название страны-произоводителя;

color – строка, хранящая цвет автомобиля;

price – беззнаковое целое, хранящее цену автомобиля;

condition – переменная типа condition_t описанного ниже, хранящая состояние автомоблия;

more_info – вариантная часть записи для хранения информации, зависящей от состояния автомобиля (структура info_t описана ниже).

Объединение для хранения вариантной части записи:

```
typedef union
{
   new_car_t new_car;
   used_car_t used_car;
} info_t;
```

Поля структуры **info_t**:

new_car – часть объединения, отвечающая за хранение информации о новых автомобилях (структура new_cat_t описана ниже);

used_car – часть объединения, отвечающая за хранение информации о б/у автомобилях (структура used_cat_t описана ниже);

Структура для хранения дополнительной информации о новых автомобилях:

```
typedef struct
{
   short unsigned int warranty;
} new_car_t;
```

Короткое беззнаковое поле *warranty* используется для хранения гарантии на новый автомобиль.

Структура для хранения дополнительной информации о б/у автомобилях:

```
typedef struct
{
   short unsigned int year;
   unsigned int mileage;
   short unsigned int repairs_num;
   short unsigned int owners_num;
} used_car_t;
```

Поля структуры **used_car_t**:

year – короткое беззнаковое целое, хранящее год выпуска автомобиля;

mileage – беззнаковое целое, хранящее пробег автомобиля;

repairs_num – короткое беззнаковое целое, хранящее количество ремонтов;

owners_num – короткое беззнаковое целое, хранящее количество собственников

Структура для представления в памяти таблицы, содержащая саму таблицу в виде массива типа car_t и её длину:

```
typedef struct
{
   car_t table[MAX_TABLE_LEN];
   size_t len;
} car_table_t;
```

Структуры для хранения таблицы ключей и её записей содержат поля, названия которых определяют то, что в них хранится:

```
typedef struct
{
   size_t num;
   unsigned int price;
} car_key_t;

typedef struct
{
   car_key_t table[MAX_TABLE_LEN];
   size_t len;
} car_key_table_t;
```

Для хранения состояния использован перечисляемый тип:

```
typedef enum condition
{
   NEW,
   USED
} condition_t;
```

Описание функций

```
/*
* Функция, загружающая данные из csv файла.
* Принимает указатель на таблицу, в которую необходимо
* считать данные, и возвращаает код ошибки.
*/
int upload_from_file(car_table_t *table);


/*
* Функция, выводящая таблицу на экран.
* Принимает указатель на таблицу, которую необходимо вывести
*/
void print_cars(car_table_t *table);
```

```
/*
* Функция, считывающая запись для добавление.
* Принимает указатель на таблицу, в которую необходимо добавить
* запись.
*/
int read_record(car_table_t *table);

/*
* Функция, удаляющая запись по цене.
* Принимает указатель на таблицу, из которой необходимо удалить
* запись.
*/
int delete_record(car_table_t *table);

/*
* Функция, создающая отсортированную таблицу ключей.
* Принимает указатель на таблицу, из которой необходимо создать
* таблицу ключей, и указатель на эту таблицу ключей.
*/
void create_sort_keys_table(car_table_t *table, car_key_table_t *keys);

/*
* Функция, печатающая таблицу ключей.
* Принимает указатель на таблицу ключей.
*/
void print_cars_keys(car_key_table_t *table);
```

```
/*
* Функция, реализующая пирамидальную сортировку.
* Принимает указатель на первый элемент в массиве, размер массива,
* размер элемента массива и функцию сравнения.
*/
void heapsort(void *first, size_t number, size_t size, int (*comparator) (const void *, const void *))

/*
* Функция, реализующая сортировку вставками.
* Принимает указатель на первый элемент в массиве, размер массива,
* размер элемента массива и функцию сравнения.
*/
void insertionsort(void *first, size_t number, size_t size, int (*comparator) (const void *, const void *))

/*
* Функция, выводящая количественную характеристику времени
* работы и используемой памяти при пирамидальной сортировке
* исходной таблицы и таблицы ключей.
* Принимает указатель на исходную таблицу и таблицу ключей.
*/
void compare_heapsorts(car_table_t *table, car_key_table_t *keys);

/* Функция, выводящая количественную характеристику времени
* работы и используемой памяти при пирамидальной сортировке
* исходной таблицы и таблицы ключей.
* Принимает указатель на исходную таблицу и таблицу ключей.
*/
void compare_sorts_types(car_table_t *table, car_key_table_t *keys);
```

Описание алгоритма

Пользователю предлагается выбор одного из пунктов меню до тех пор пока не будет выбран выход из программы.

Для сортировки используется алгоритм пирамидальной сортировки. На начальном этапе на массиве (представленном в виде структуры) строится бинарное дерево, в котором каждый родитель больше двух своих соседей. Далее наибольший элемент, оказавшийся в корне дерева, который хранится как первый элемент массива, переставляется с последним. Алгоритм повторяется каждый раз работая с массивом с длиной на 1 меньше, чем на предыдущем шаге.

Также используется алгоритм сортировки вставками, где левая часть массива считается отсортированной и в неё вставляется новый элемент из правой части так, чтобы порядок невозрастания или неубывания сохранился.

Для нахождения информации использован алгоритм линейного поиска.

Тесты

№	Что проверяется	Пользовательский ввод	Результат
1	Некорректный ввод пункта меню	jk	Введенный код не соответствует ни одному действию. Попробуйте ещё раз.
2	Некорректный ввод пункта меню (длинная строка)	hjkl	Некорректный ввод кода действия. Попробуйте ещё раз.
3	Открытие несуществующего файла	1 file	Ошибка при открытии файла!
4	Пустой файл	1 ./data/empty.txt	Пустой файл!
5	Файл в неверном формате	1 ./data/incorrect.csv (в файле две строки «abracadabra»)	Ошибка чтения файла в 2-й строке. Неверный формат данных в файле!
6	Верный формат файла	1 ./data/cars.csv	Данные успешно загружены!

7	Вывод таблицы при загруженном файле или добавленных записях	2	Таблица данных в удобном для чтения формате (не приводится в силу недостатка места)
8	Попытка вывода таблицы, когда в ней нет данных	2	В таблице нет данных!
9	Превышение длины строки марки при вводе поля или чтения из файла	3 аааааааааааааа (или чтение файла)	Слишком длинная строка марки!
10	Превышение длины строки страны при вводе поля или чтения из файла	3 Volvo аааааааааааааааааааа (или чтение файла)	Слишком длинная строка страны!
11	Превышение длины строки цвета при вводе поля или чтения из файла	3 Volvo РФ аааааааааааааааааааа (или чтение из файла)	Слишком длинная строка цвета!
12	Превышение длины строки цены при вводе поля или чтения из файла	3 Volvo РФ фиолетовый аааааааааааааааааааа (или чтение из файла)	Слишком длинная строка цены!
13	Нецелая цена	3 Volvo РФ фиолетовый 123.4 (или чтение из файла)	Цена -- целое число!
14	Отрицательная цена	3 Volvo РФ фиолетовый -123 (или чтение из файла)	Введенная цена выходит за допустимый диапазон значений!
15	Цена, выходящая за unsigned int	3 Volvo РФ	Введенная цена выходит за допустимый диапазон значений!

		фиолетовый 9999999999 (или чтение из файла)	
16	Неверно заданное состояние	3 Volvo РФ фиолетовый 400 k (или чтение из файла)	Состояние задано некорректно (в файле только new или used, в строке -- n/N или u/U)!
17	Неверно заданная гарантия	3 Volvo РФ фиолетовый 400 n aaaa (или чтение из файла)	Слишком длинная строка гарантии!
18	Неверно заданный год выпуска	3 Volvo РФ фиолетовый 400 u a (или чтение из файла)	Год -- целое число!
19	Неверно заданный пробег	3 Volvo РФ фиолетовый 400 u 2001 a (или чтение из файла)	Пробег -- целое число!
20	Неверно заданное количество ремонтов	3 Volvo РФ фиолетовый 400 u 2001	Количество ремонтов -- целое число!

		200000 a (или чтение из файла)	
21	Неверно заданное количество собственников	3 Volvo РФ фиолетовый 400 u 2001 200000 1 a (или чтение из файла)	Количество собственников -- целое число!
22	Верные данные при добавлении	3 Volvo РФ фиолетовый 400 u 2001 200000 1 2	Данные об автомобиле успешно добавлены
23	Неверно заданный диапазон цен при поиске	10 Volvo 9000 1000	Цена "от" должна быть ниже цены "до"!
24	Данные по поиску не найдены	10 Volvo 1 10	По заданному запросу ничего не найдено!
25	Данные для удаления не найдены	4 1	Машин с такой ценой нет!
26	Верные данные для удаления	4 6719	Данные успешно удалены!

Оценка эффективности

Время работы методов сортировок в микросекундах при различных количествах записей.

Пирамидальная сортировка

Количество записей	Сортировка исходной таблицы	Сортировка таблицы ключей
50	360	71
150	1239	237
300	3063	548

Сортировка вставками

Количество записей	Сортировка исходной таблицы	Сортировка таблицы ключей
50	830	127
150	5168	403
300	11574	1449

Используемая память (байт)

Количество записей	Сортировка исходной таблицы	Сортировка таблицы ключей
50	6200	7000 (+13%)
150	18600	21000 (+13%)
300	37200	42000 (+13%)

Контрольные вопросы

Как выделяется память под вариантную часть записи?

В вариантной части записи память для всех полей является общей, поэтому её размер равен максимальному по длине полю вариантной части.

Что будет, если в вариантную часть ввести данные, несоответствующие описанным?

Тип данных в вариативной части не проверяется при компиляции, поэтому поведение программы при неверных данных не определено.

Кто должен следить за правильностью выполнения операций с вариантной частью записи?

Контроль за правильностью выполнения операций с вариантной частью записи целиком возлагается на программиста.

Что представляет собой таблица ключей, зачем она нужна?

Таблица ключей включает в себя два столбца: индекс элемента в исходной таблице и выбранный ключ (одно из полей исходной таблицы). Она нужна для оптимизации сортировки при работе с большими данными.

В каких случаях эффективнее обрабатывать данные в самой таблице, когда использовать таблицу ключей?

При сортировке по таблице ключей мы экономим время на перестановке записей, однако задействуем большее количество памяти, также не стоит забывать, что затрачивается время и на выбор данных из основной таблицы, поэтому при работе с таблицей, количество полей в которой невелико, эффективнее обрабатывать данные в самой таблице. Аналогично необходимо поступать, если при реализации программы требуется экономия памяти, а время не так важно.

Какие способы сортировки предпочтительнее для обработки таблиц и почему?

При обработке таблицы с большим количеством полей предпочтительнее использовать алгоритмы сортировки, которые требуют наименьшее количество операций перестановки, у которых наименьшая сложность работы, так как в этом случае затрачивается минимальное время для обработки данных в таблице.

Вывод

Как следует из результатов измерения времени работы и используемой памяти, при увеличении объема памяти на 13% время при количестве записей,

равном 50, время пирамидальной сортировки уменьшилось в 4 раза, вставками – 7 раз, – равном 150, пирамидальной – в 5 раз, вставками – в 13 раз, – равном 300, пирамидальной – в 6 раз, вставками – в 10 раз.

Таким образом, чем больше размер исходной таблице и чем больше количество и размер полей каждой записи этой таблицы, тем более эффективен метод сортировки таблицы ключей, однако и при небольшом количестве записей этот метод оказывается довольно эффективен. Но, стоит заметить, что при небольших размерах полей и небольшом их количестве, алгоритм сортировки таблицы ключей не эффективен, так как требует существенного увеличения памяти.

Очевидно, что алгоритм сортировки сложности $O(n^2)$ работает медленнее алгоритма сортировки сложности O(n*log(n)), но первый алгоритм более прост в своей реализации. Таким образом, если в программе необходима скорость работы, то лучше реализовать более быстрый алгоритм, если же программа используется для сортировки небольшого количества данных, то для повышения продуктивности работы программиста может быть использована и более простая в реализации, но более медленная в работе сортировка.