Московский государственный технический университет им. Н.Э. Баумана

(национальный исследовательский университет)

**Отчет по лабораторной работе №2**

по курсу “Типы и структуры данных”

Выполнила: студентка группы ИУ7-35Б

Голикова С.М.

Москва, 2021

Задание

1. Условие задачи

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.

Оценить относительную эффективность программы (в процентах) по времени и по используемому объему памяти в зависимости от используемого алгоритма и от объема сортируемой информации.

1. Внешняя спецификация
2. Исходные данные и результаты

Исходные данные:

* Номер команды из меню (от 0 до 9), текстовый файл (по желанию пользователя)
* В зависимости от команды может потребоваться ввод дополнительных параметров (например, марки автомобиля, цены или пробега)

Допущения:

1. Максимальное число автомобилей в таблице - 999
2. Максимальная длина строки - 50 символов
3. Каждое поле в файле записано на отдельной строке, после каждой структуры - пустая строка

Выходные данные:

Таблица автомобилей либо результат выполнения действия согласно выбранному пункту меню.

1. Задача, реализуемая программой

Задачи, которые может решать программа:

* Вывести список автомобилей
* Добавить информацию об автомобиле в конец таблицы
* Удалить данные из таблицы по полю
* Вывести отсортированную (по ценам) таблицу ключей при несортированной исходной таблице
* Вывести отсортированную (по ценам) таблицу
* Вывести отсортированную (по ценам) таблицу, используя отсортированный массив ключей
* Вывести результаты сравнения эффективности работы программы при сортировке данных в исходной таблице и таблице ключей
* Вывести цены не новых машин указанной марки с одним предыдущим собственником и отсутствием ремонта, в указанном диапазоне цен
* Выбрать новую исходную таблицу
* Выйти из программы

1. Способ обращения к программе

Программа может быть вызвана через консоль (./app.exe) или запущена через любую среду разработки С.

Пользователь взаимодействует с программой через меню, выбирая нужный ему пункт и вводя необходимые данные.

1. Возможные аварийные ситуации и ошибки пользователя

Аварийные ситуации и ошибки при вводе: слишком длинная строка, некорректный ввод на месте целого положительного числа, некорректный выбор пункта меню. В случае аварийной ситуации пользователю выдается сообщение о том, в каком формате следует вводить значение, и предлагается ввести его еще раз.

3. Внутренние структуры данных

Для реализации задачи были созданы следующие типы и структуры:

typedef enum { FALSE, TRUE } mybool;

typedef struct {

int index;

int field;

} key\_t;

typedef struct

{

int warranty;

} new\_t;

typedef struct

{

int release\_year;

int mileage;

int repairs\_num;

int owners\_num;

} used\_t;

typedef union

{

new\_t new\_car;

used\_t used\_car;

} condition\_t;

typedef struct

{

char brand[MAX\_STR\_LEN];

char country[MAX\_STR\_LEN];

int price;

char color[MAX\_STR\_LEN];

mybool is\_new;

condition\_t condition;

} car\_t;

Основная структура с полями, содержащими общее описание автомобиля: car\_t.

Вариативное поле, содержащее конкретную информацию об автомобиле с учетом его состояния (новый или старый): condition.

4. Описание алгоритма

Сортировка пузырьком: Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются n - 1 раз, где n - длина массива.

Быстрая сортировка: Рекурсивный метод сортировки, на каждом этапе рекурсии в массиве выбирается “опорный” элемент, который находится в середине массива. Элементы массива перераспределяются таким образом, что все элементы, меньшие опорного, оказываются в левой половине относительно опорного, а остальные - справа. Дальше сортировке подвергаются две полученные части по отдельности. База рекурсии - массив единичной длины.

Таблица ключей формируется на основе исходной таблицы путем выбора из этой таблицы индекса и ключа (в нашей задаче ключ - цена автомобиля).

5. Основные функции, используемые в программе

| Имя | int choose\_action(void) |
| --- | --- |
| Функция | Выбор действия в меню |
|  | |
| Имя | int add\_car(char \*file\_name, car\_t \*cars, int \*n) |
| Функция | Добавление машины в конец таблицы |
|  | |
| Имя | int delete\_cars(char \*file\_name, car\_t \*cars, int \*n) |
| Функция | Удаление машин по выбранному полю |
|  | |
| Имя | int download\_table(char \*file\_name, car\_t \*cars, int \*n) |
| Функция | Загрузка таблицы из файла |
|  | |
| Имя | int find\_in\_table(car\_t \*cars, int n) |
| Функция | Поиск в таблице по варианту (не новых машин указанной марки с одним собственником и отсутствием ремонта, в указанном диапазоне цен) |
|  | |
| Имя | void form\_keys\_table(car\_t \*cars, key\_t \*keys, int n) |
| Функция | Формирование таблицы ключей на основе исходной |
|  | |
| Имя | int bubble\_sort\_keys(key\_t \*keys, int n)  int qsort\_keys(key\_t \*keys, int n)  int bubble\_sort\_table(car\_t \*cars, int n)  int qsort\_table(car\_t \*cars, int n) |
| Функция | Сортировки |

5. Тесты

| Действия | Результат |
| --- | --- |
| При предложении выбрать исходную таблицу выбирается таблица по умолчанию | Загрузка существующего текстового файла с информацией о 120 автомобилях |
| Выбор пункта 1 меню | Вывод таблицы с информацией обо всех автомобилях |
| Выбор пункта 2 меню, затем - ввод следующей информации:  BMW  Germany  70000  black  1 (при ответе на вопрос, новая ли машина)  10 | Добавление в конец таблицы введенной информации |
| Выбор пункта 3 меню, затем - 1 при выборе поля, по которому производится удаление, затем - снова 1 | Удаление первой машины из таблицы |
| Выбор пункта 4 меню | Вывод отсортированной таблицы ключей |
| Выбор пункта 5 меню | Вывод отсортированной таблицы |
| Выбор пункта 6 меню | Вывод отсортированной таблицы ключей и упорядоченной с их помощью исходной таблицы |
| Выбор пункта 7 меню | Вывод замеров времени и памяти, а также их анализа для разных типов сортировок и методов |
| Выбор пункта 8 меню, затем:  BMW  50000  100000 | Для таблицы по умолчанию вывод:  62000  92000 |
| Выбор пункта 0 меню | Завершение программы |
| Выбор некорректного пункта меню | “Ошибка: должно быть введено целое неотрицательное число  Повторите ввод: ” |
| Ввод слишком длинной строки | “Ошибка: превышена максимальная длина строки  Повторите ввод: “ |
| Ввод строки или отрицательного числа на месте целого неотрицательного | “Ошибка: должно быть введено целое неотрицательное число  Повторите ввод: “ |
| Файл, название которого введено пользователем, не существует | “Указанный файл не найден. Хотите ли вы использовать уже существующий  файл с информацией об автомобилях "data/cars\_table.txt"?  1 - Да  2 - Нет  Ваш выбор: “ |

6. Анализ эффективности работы программы

Время сортировки (в секундах, округление до 6 знаков после запятой)

| Количество записей | Сортировка пузырьком | | Быстрая сортировка | |
| --- | --- | --- | --- | --- |
| Таблица | Массив ключей | Таблица | Массив ключей |
| 10 | 0.000004 | 0.000002 | 0.000002 | 0.000002 |
| 50 | 0.000066 | 0.000024 | 0.000014 | 0.000010 |
| 100 | 0.000276 | 0.000100 | 0.000031 | 0.000018 |
| 200 | 0.001256 | 0.000406 | 0.000065 | 0.000037 |
| 500 | 0.004687 | 0.001286 | 0.000050 | 0.000024 |

Объем занимаемой памяти (в байтах)

| Количество записей | Таблица | Массив ключей |
| --- | --- | --- |
| 10 | 1760 | 80 |
| 50 | 8800 | 400 |
| 100 | 17600 | 800 |
| 200 | 35200 | 1600 |
| 500 | 88000 | 4000 |

Эффективность по разным параметрам (в процентах)

| Количество записей | % памяти, занимаемый таблицей ключей относительно исходной таблицы | % роста скорости сортировки исходной таблицы по сравнению с сортировкой таблицы ключей  (пузырек) | % роста скорости сортировки исходной таблицы по сравнению с сортировкой таблицы ключей  (быстрая сортировка) |
| --- | --- | --- | --- |
| 10 | 4.55% | 229.81% | 157.01% |
| 50 | 4.55% | 274.64% | 140.04% |
| 100 | 4.55% | 276.10% | 167.82% |
| 200 | 4.55% | 309.22% | 175.65% |
| 500 | 4.55% | 364.56% | 209.01% |

7. Выводы

Чем больше размер исходной таблицы, тем более эффективной становится сортировка массива ключей. Однако за скорость приходится платить памятью, выделяемой под массив ключей. В моем случае размер таблицы ключей составил ~4.55% от исходной, что не так много, т.к. выбранный ключ представлял собой целочисленное значение. Когда ключ является строкой, на его хранение и обработку уходит гораздо больше ресурсов, поэтому обработка массива ключей становится не такой эффективной по сравнению с обработкой исходной таблицы. Таким образом, массив ключей выгодно использовать, когда затраты на память малы по сравнению с увеличением скорости сортировки.

8. Контрольные вопросы

**1. Как выделяется память под вариантную часть записи?**

В языке Си вариантная часть структуры реализована с помощью union. Выделяется один блок памяти, который будут разделять вариантные переменные. Размер области памяти, выделяемый под вариантную часть, равен максимальному из размеров вариантных полей.

**2. Что будет, если в вариантную часть ввести данные, не соответствующие описанным?**

Неопределенное поведение: результат будет системно зависимым и трудно предсказуемым.

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

Ответственность за правильность выполнения операций с вариантной частью записи целиком и полностью лежит на программисте.

**4. Что представляет собой таблица ключей, зачем она нужна?**

Таблица ключей представляет собой таблицу, в которой находится два столбца: номер ячейки в исходной таблице и значение выбранного поля исходной таблицы для этой ячейки.

Она нужна для оптимизации сортировки посредством сокращения количества перестановок в исходной таблице, которые являются ресурсозатратными.

**5. В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?**

Таблицу ключей эффективнее использовать, когда затраты на выделение памяти под эту таблицу значительно меньше увеличения скорости работы программы. Этот выигрыш в скорости достигается за счет того, что снижается количество перестановок в исходной таблице и программа производит перестановку всего лишь одного ключа (в отличие от целой записи в случае сортировки исходной таблицы). Наиболее эффективно применять таблицу ключей, когда выбранное поле является целочисленным типом данных. Если в качестве выбранного ключа выступает строковый тип данных, выигрыш по времени значительно снижается, т.к. обработка строк занимает больше ресурсов.

**6. Какие способы сортировки предпочтительнее для обработки таблиц и почему?**

В случае сортировки таблицы с большим количеством записей выгоднее использовать стандартные и устойчивые способы сортировки, в которых количество перестановок и сложность минимальны: O(n\*log n) (quicksort, mergesort и т.д.). Если же в таблице не так много записей, можно использовать простые алгоритмы сортировки, например, сортировку пузырьком.