Лабораторная работа №2

по дисциплине

“Типы и структуры данных”

Соколов Ефим

ИУ7-33Б

Москва, 2019 г.

**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.

**Техническое задание:**

Приобрести навыки работы с типом данных «запись» (структура), содержащим вариантную часть – объединение (union); произвести сравнительный анализ реализации алгоритмов сортировки с данными, хранящимися в таблице; произвести сравнительный анализ реализации алгоритмов сортировки и поиск информации в таблице, при использовании записей с большим числом полей, и тех же алгоритмов при использовании таблицы ключей; оценить эффективность программы по времени и по используемому объему памяти при использовании различных структур и эффективность использования различных алгоритмов сортировок.

**Входные данные**

* Имя файла с записями автомобилей
* Команда меню (число от 0 до 12 включительно)
* Параметра добавляемой/удаляемой записи

**Выходные данные**

* Печать таблицы записей (ключей)
* Результат поиска (удаления) записей
* Сравнение эффективности по времени алгоритмов сортировки

**Возможные аварийные ситуации**

* Неизвестная команда меню
* Файл с записями пуст или не существует
* Некорректный ввод информации о новой записи
* Попытка обработки (вывода) таблицы записей, не импортировав её из исходного файла

**Структура даннных**

// Состояние автомобиля

typedef enum

{

    new = 1, // новая

    used // с пробегом

} condition\_t;

// Поля машины с пробегом

typedef struct

{

    int year; // год выпуска

    int mileage; // пробег

    int repairs; // количество ремонтов

    int owners; // количество владельцев

} driven\_t;

// Объединение полей новой и старой машины

typedef union

{

    int guarantee;  // новая (гарантия)

    driven\_t used;  // с пробегом

} variable\_t;

// Информация об одной записи таблицы (автомобиля)

typedef struct

{

    char brand[MAX\_FIELD\_LEN]; // Марка (MAX\_FIELD\_LEN = 256)

    char country[MAX\_FIELD\_LEN]; // Страна производитель (MAX\_FIELD\_LEN = 256)

    int price; // Цена автомобиля

    char color[MAX\_FIELD\_LEN]; // Цвет автомобиля (MAX\_FIELD\_LEN = 256)

    condition\_t car\_type; // Состояние автомобиля

    variable\_t add\_info; // Доп. инфо (зависит от состояния: новая ли машина или нет)

} car\_t;

// Таблица ключей

typedef struct

{

    int car\_table\_index; // Индекс записи

    int car\_price; // Цена автомобиля

} car\_key\_t;

// Вся информация

typedef struct

{

    car\_t \*main\_table; // таблица записей

    car\_key\_t \*key\_table; // таблица ключей

    int size; // количество записей

} data\_t;

**Оценки эффективности**

*Замеры памяти (в байтах)*

|  |  |  |
| --- | --- | --- |
| *Количество записей* | *Размер таблицы записей* | *Размер таблицы ключей* |
| *10* | *1760* | *80* |
| *50* | *8800* | *400* |
| *100* | *17600* | *800* |
| *500* | *88000* | *4000* |
| *1000* | *176000* | *8000* |

*Замеры времени выполнения сортировки*

*(в миллисекундах\*)*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Количество записей* | *Таблица записей (bubble sort)* | *Таблица записей (qsort)* | *Таблица ключей (bubble sort)* | *Таблица ключей (qsort)* |
| *10* | *1* | *1* | *1* | *1* |
| *50* | *17* | *6* | *14* | *3* |
| *100* | *61* | *15* | *42* | *4* |
| *500* | *1732* | *77* | *1480* | *19* |
| *1000* | *7825* | *224* | *6033* | *43* |

*(\*) Время сортировки: среднее время выполнения сортировки на 1000 повторениях*

*Сравнение алгоритмов сортировки*

*(сортировки таблицы записей)*

|  |  |
| --- | --- |
| *Количество записей* | *Соотношение превосходства скорости алгоритма qsort над bubble sort* |
| *10* | *в ~ 1.0 раз* |
| *50* | *в ~ 2.8 раза* |
| *100* | *в ~ 4.1 раз* |
| *500* | *в ~ 22.5 раза* |
| *1000* | *в ~ 35.0 раз* |

*Сравнение времени сортировки*

*таблицы ключей и таблицы ключей*

|  |  |
| --- | --- |
| *Количество записей* | *Соотношение превосходства скорости сортировки таблицы ключей над таблицей записей* |
| *10* | *в ~ 1.0 раз* |
| *50* | *в ~ 2.0 раза* |
| *100* | *в ~ 3.8 раз* |
| *500* | *в ~ 4.1 раза* |
| *1000* | *в ~ 5.2 раза* |

**Вывод**

Чем больше размер таблицы, тем эффективнее сортировка массива ключей (но даже на маленьких размерах таблицы, сортировка массива ключей происходит быстрее, чем сортировка самой таблицы). Для хранения моего массива ключей 1000 записей понадобилось ~4.5% памяти занимаемой таблицей записей, но это обеспечили выигрыш во времени более чем в 5 раз. Следовательно сортировка таблицы через массив ключей эффективнее всегда.

**Ответы на контрольные вопросы**

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

**Ответ:** в Си, вариативная часть структуры реализована с помощью “union”. Память выделяется в одном “куске” памяти, имеющий размер, который способен вместить наибольшее поле из указанных.

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

**Ответ:** результат будет зависеть от ОС. Предсказать результат будет очень трудно(но один из возможных вариантов – приведение типов).

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

**Ответ:** ответственность за правильность проведения операций целиком и полностью лежит на программисте.

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

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

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

**Ответ:** обрабатывать данные в самой таблице записей эффективнее (по памяти), когда время обработки не так важно, как задействованная память. Использование же таблицы ключей, эффективно, когда нужно сократить время обработки и не так важна дополнительная задействованная память (эффективнее по времени). Стоит отметить, что использование таблицы ключей будет неэффективно, когда исходная таблица состоит из малого числа полей. Тогда таблица ключей будет лишь занимать дополнительное место в памяти и не даст значительной выгоды во времени.

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

**Ответ:** для таблиц из большого количества записей предпочтительно использовать алгоритмы сортировки с относительно низкой вычислительной сложностью (в моей лабораторной работе – qsort со сложностью O(n\*log(n)). Если же в таблица имеет небольшое количество записей, то лучше использовать простые алгоритмы сортировки, например, сортировку пузырьком или сортировку вставками.