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

Студент **Звягин Даниил Олегович**

Группа **ИУ7-33Б**

Название дисциплины **Типы и структуры данных**
Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Звягин Д. О.**

Преподаватель _____ **Барышникова М. Ю.**

Оценка _____

2023 г.

Описание условия задачи

Ввести репертуар театров, содержащий:

- название театра
- спектакль
- диапазон цены билета
- тип спектакля
    - 1. пьеса
    - 2. драма
    - 3. комедия
    - 4. сказка
        - возраст (3+, 10+,16+)
    - 5. музыкальный
        - композитор
        - страна
        - тип
            - балет
            - опера
            - мюзикл
        - возраст (3+, 10+,16+)
        - продолжительность

Вывести список всех балетов для детей указанного возраста с продолжительностью меньше указанной.

Техническое задание

Разработать программу для работы с типом данных «таблица», состоящего структур типа «запись», каждая из которых содержит данные о конкретном выступлении некоторого театра, а также вариантную часть (в нашем случае - union) для хранения атрибутов некоторых из типов выступлений.

Произвести сравнительный анализ реализации алгоритмов сортировки информации в таблицах, при использовании таблиц с большим числом записей, и тех же алгоритмов, при использовании таблицы ключей.

Оценить эффективность программы по времени и по используемому объему памяти при использовании различных структур, а также эффективность различных алгоритмов сортировок.

Входные данные

1. В качестве аргумента командной строки прописывается путь к файлу с базой данных. (Если такого файла не существует, программа предложит пользователю его создать)
2. Число - Пункт меню (см. далее)

```
Введите режим работы:
1: Вывести таблицу
2: Добавить элемент в таблицу
3: Отсортировать таблицу
4: Вывести проходящие по фильтру (из задания) записи
5: Удалить элемент из таблицы
6: Вывести таблицу эффективности сортировки
0: Выход
```

В соответствии с каждым пунктом меню, вам может понадобиться (или не понадобиться) ввод дополнительных данных, его содержание можно понять из названия пунктов меню. (Так, например, для того, чтобы добавить запись в таблицу, вам будет необходимо ввести каждое поле этой записи)

Выходные данные

В зависимости от пункта меню:

Состояние таблицы в данный момент.

Отсортированная таблица.

Состояние таблицы ключей с содержанием основной таблицы (для анализа).

Результаты замеров времени сортировок.

Сравнение времени и памяти, требуемой для сортировок.

«Отфильтрованная» таблица (с балетами по условию из постановки задания).
Сообщения об ошибках.

Меню.

Сообщение о необходимости передачи аргумента командной строки в случае его отсутствия.

Способы обращения к программе

1. Программа не имеет графического интерфейса, поэтому вызывается из консоли (терминала) вызовом исполняемого файла (например app.exe). Необходимо передать название файла базы данных для начала работы программы. Входные данные передаются через стандартный поток ввода консоли (ввод с клавиатуры или перенаправление потоков), а выходные – через стандартный поток вывода (вывод в консоль или в файл с помощью перенаправления потока).

Возможные аварийные ситуации

1. Некорректный ввод данных
2. Отсутствие имени файла БД в качестве аргумента КС при вызове программы
3. Недостаток памяти для записи (или создания копии) таблицы
4. Невозможность создания файла БД в случае его отсутствия.
5. Ошибки записи или чтения из файла (например в случае отсутствия необходимых прав доступа)

Описание внутренних структур данных

Перечисление режимов работы программы (для меню)

```
enum modes
{
    EXIT,
    PRINT_TABLE,
    ADD_EL,
    SORT_TABLE,
    FILTER_TABLE,
    DELETE_EL,
    TIME_TABLE,
    N_MODES,
};
```

Перечисление типов музыкальных выступлений

```
enum musical_types
{
    BALLET = 1,
    OPERA,
    MUSICAL,
    MUSICAL_TYPE_N,
};
```

Структура для атрибутов музыкального выступления

```
typedef struct
{
    unsigned char age_restriction;
    name_t composer;
    name_t country;
    unsigned char type;
    short length;
} musical_attr_t;
```

Структура для диапазона цен на выступление

```
typedef struct
{
    int32_t min_price;
    int32_t max_price;
} price_t;
```

Перечисляемый тип типов выступлений

```
enum play_type
{
    PLAY = 1,
    DRAMA,
    COMEDY,
    FAIRYTALE,
    MUSIC,
    PLAY_TYPE_N,
};
```

Структура записи о выступлении в театре

```
typedef struct
{
    name_t theatre_name;
    name_t play_name;
    price_t ticket_prices;
    unsigned char play_type;
    union {
        unsigned char age_restriction;
        musical_attr_t musical_attrs;
    } play_attrs;
} record_t;
```

Перечисление возможных ошибок при вводе записи в таблицу

```
enum record_input_errors
{
    NO_THEATRE_NAME = 1,
    THEATRE_NAME_TOO_LONG,
    NO_PLAY_NAME,
    PLAY_NAME_TOO_LONG,
    PRICE_LOWER_THAN_ZERO,
    PRICE_MINMAX_MISMATCH,
    PLAY_TYPE_OUT_OF_BOUNDS,
    BAD_AGE_RESTRICTION,
    NO_COMPOSER,
    COMPOSER_TOO_LONG,
    NO_COUNTRY,
    COUNTRY_TOO_LONG,
    MUSICAL_TYPE_OUT_OF_BOUNDS,
    BAD_PLAY_LENGTH,
    STRING_FIELDS_OVERLAP,
};
```

Перечисление возможных ошибок считывания конкретных значений

```
enum scan_errors
{
    STRING_TOO_LONG = 1,
    NO_STRING,
    FGETS_ERR,
    BAD_NUMBER,
};
```

Структура таблицы записей – содержит в себе кол-во элементов, кол-во зарезервированной памяти (чтобы знать на какое количество элементов выделена память, так как я выделяю её динамически) и указатель на массив записей (типа record_t)

```
typedef struct
```

```
{
    size_t el_count;
    size_t reserved_el_count;
    record_t *dataptr;
} record_table_t;
```

Перечисление ошибок с инициализацией и считыванием таблицы

```
enum table_errors
{
    NOT_ENOUGH_MEMORY = 1,
    BAD_FILE_SIZE,
    CANT_WRITE,
    BAD_FILENAME,
};
```

Тип данных для одной записи в таблице ключей

```
typedef struct
{
    size_t main_index;
    union {
        char *text;
        int32_t num;
    } value;
} key_t;
```

Перечисление возможных ключей для сортировки (во время замеров времени используется только сортировка по минимальной цене билета, так как сортировка по названиям театров крайне нестабильна даже при одинаковых условиях тестирования)

```
enum KEYS
{
    T_NAME = 1,
```

```
    MIN_PRICE,
    KEY_N,
};
```

Описание алгоритмов сортировок

1. Сортировка пузырьком – если при сравнении текущий элемент «больше» следующего по ключу, они меняются местами
2. Сортировка выбором – находится «максимальный» по ключу элемент в массиве и ставится в «конец» массива, размер «исследуемого» массива уменьшается на 1

Замерный эксперимент

В таблице приводится среднее время сортировки таблицы с некоторым количеством записей за некоторое количество итераций (которое означает количество запусков сортировки, а соответственно и замеров времени).

Время приведено в наносекундах:

Кол-во записей	Количество итераций	Пузырек		Вставки	
		Таблица	Ключи	Таблица	Ключи
20	10000	2097	1660	1001	892
100	1000	56454	46430	23410	19849
1000	1000	5194947	3997638	1489292	1192386
10000	100	612833730	492925382	125878425	110636885

Объем занимаемой памяти:

Кол-во записей	Таблица	Ключи
20	2000	2320
100	10000	11600
1000	100000	116000
10000	1000000	1160000

Анализ полученных результатов:

В результате исследования выяснилось, что сортировка таблицы ключей увеличивает скорость сортировки в среднем на 25% для сортировки пузырьком

и 10-15% для сортировки методом выбора. Я связываю такое различие эффективности с тем, что во время сортировки пузырьком, мы делаем гораздо больше перестановок элементов, чем во время сортировки вставками. Это также одна из причин, почему результаты сортировки выбором нестабильны при небольшом количестве элементов (сортировка таблицы ключей то лучше, то хуже) – бОльшую роль играет сравнение элементов, а не перестановка.

Касаемо объёма занимаемой памяти: моя структура record_t (очень удачно) занимает ровно 100 байт, поэтому объём занимаемой памяти – такие красивые числа, а тип key_t занимает 16 байт, то есть для создания таблицы ключей нам требуется дополнительно 16% от объёма памяти, занятого основной таблицей.

Выводы по проделанной работе

Чем больше элементов мы добавляем в таблицу, тем эффективнее становится сортировка массива ключей, на малых таблицах разница не сильно заметна между сортировкой самой таблицы и таблицы ключей, а результаты порой бывают хаотичны. Также прирост эффективности сильно зависит от количества перестановок в самой сортировке. Однако, при выборе между сортировкой таблицы и таблицы ключей, нельзя забывать о объёме памяти, занимаемой самой таблицей ключей, ведь чем больше исходная таблица, тем больше таблица ключей. Я также научился пользоваться вариантной структурой, или «объединением» в реалиях языка Си. Это позволило мне сэкономить память для хранения атрибутов выступлений.

Контрольные вопросы

1. Как выделяется память под вариантную часть записи?

Выделяется память для наибольшего по объёму элемента объединения (так, если в объединении будут определены значения типа char (1Б) и short (2Б), объединение займёт 2 байта – наибольшее из двух)

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

Согласно стандарту C99 поведение не определено

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

Ответственность полностью лежит на программисте (что следует из предыдущего вопроса)

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

Таблица ключей – это таблица, содержащая номер ячейки в исходной таблице и значение необходимого для определённой цели (в нашем случае - сортировки) поля исходной таблицы.

Нужна она для того, чтобы обрабатывать меньшее количество памяти, когда работа с памятью критична (занимает значительную часть времени выполнения программы).

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

Когда время работы с памятью занимает большУю часть работы программы. Например, когда нужно много раз переставлять элементы, как, например, в сортировке.

Также прирост эффективности от использования таблицы ключей увеличивается с увеличением количества столбцов в исходной таблице. (Если их изначально не так много, то особого смысла использовать таблицу ключей нет)

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

Следует выбирать сортировки, которые требуют наименьшее количество перестановок и дополнительной памяти. Это можно наглядно увидеть из моего замерного эксперимента, а стандартная функцию qsort сортировала миллион элементов за 0.25 секунды (когда для сортировки 100000 элементов мне приходится ждать по несколько минут).