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)

Дисциплина “Типы и структуры данных”

Лабораторный практикум №2

по теме: «Записи с вариантами, обработка таблиц»

Выполнил студент: __*Клименко Алексей Константинович*_____

фамилия, имя, отчество

Группа: _____*ИУ7-35Б*_____

Проверил, к.п.н.: _____

подпись, дата

Оценка _____ Дата _____

2020 г.

цель работы

Приобрести навыки работы с типом данных «запись» (структура), содержащим вариантную часть (объединение, смесь), и с данными, хранящимися в таблицах, произвести сравнительный анализ реализации алгоритмов сортировки и поиска информации в таблицах, 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.

описание условия задачи

Необходимо создать таблицу, содержащую не менее 40 записей с вариантной частью. Произвести поиск информации по вариантному полю. Упорядочить таблицу, по возрастанию ключей (где ключ – любое невариантное поле по выбору программиста), используя:

а) исходную таблицу

б) массив ключей, двумя алгоритмами сортировки

Оценить эффективность выбранных алгоритмов (по времени и по используемому объему памяти) при различной реализации программы, то есть, в случаях а) и б). Обосновать выбор алгоритмов сортировки. Оценка эффективности должна быть относительной (в %).

техническое задание

исходные данные

Исходными данными является таблица квартир, считываемая из файла, а также добавляемые в процессе работы программы квартиры.

Программное меню

При запуске, пользователю предлагается ввести имя файла с данными таблицы. После успешного считывания данных из файла, перед пользователем отображается меню, в котором он может выбирать опции по своему желанию.

В главном меню пользователь может:

- Добавить квартиру в таблицу
- Посмотреть таблицу
- Удалить квартиру по ID
- Найти квартиру по параметрам
- Перейти в меню сортировки

В меню сортировки пользователь может:

- Посмотреть исходную таблицу
- Посмотреть таблицу по ключам

- Посмотреть таблицу ключей
- Отсортировать таблицу ключей

Множетво допустимых значений

Для данной СД для хранения информации о квартирах допустимые значения для полей структуры таковы: // не нужно здесь

Адрес (address) – строка до 30 символов (без символа «;»)

Площадь комнат (area) – (0.0f, 830.0f]

Количество комнат (rooms_amount) – [1, 10]

Цена за квадратный метр (price_per_m2) – (0.0f, 1.0e+6]

Тип жилья (type) – { PRIMARY, SECONDARY }

Наличие отделки (has_trim) – { 0, 1 }

Были ли домашние животные (was_pets) – { 0, 1 }

Время постройки (build_time) – [1880, 2020]

Кол-во предыдущих владельцев – [0, 100]

Кол-во предыдущих жильцов – [0, 20]

выходные данные

Выходными данными является таблица ключей, отсортированная таблица и таблица по ключам.

результат

Результатом работы программы является сортировка таблицы; поиск, добавление и удаление квартиры.

описание задачи, реализуемой программой

Программа при запуске запрашивает имя файла с данными, после чего считывает данные из файла и запускает пользовательское меню.

способы обращения к программе

Вызов программы происходит любым возможным способом запуска программы **app.exe**. Для корректной работы программы при её вызове не требуются дополнительные аргументы командной строки.

возможные аварийные ситуации и ошибки пользователя

При неправильном вводе пользователя программа возвращается в меню и предлагает повторить ввод еще раз.

Структуры данных

Для хранения информации об одной квартире мной используется структура **`flat_t`**:

```
typedef enum
{
   PRIMARY,  // Первичная квартира
   SECONDARY // Вторичная квартира
} flat_type_t;

typedef struct
{
   unsigned int id;              // ID квартиры (в таблице)
   char address[MAX_ADDRESS_SIZE]; // адрес квартиры (30 символов)
   float area;                   // общая площадь
   uint8_t rooms_amount;         // кол-во комнат
   float price_per_m2;           // цена за квадратный метр
   uint8_t type;                 // PRIMARY или SECONDARY
   union
   {
      uint8_t has_trim;          // Отделка. 0 или 1
      uint8_t was_pets;          // Наличие животных. 0 или 1
   } type_data;
   time_t build_time;            // время постройки
   uint8_t prev_owners_amount;   // предыдущие владельцы
   uint8_t prev_lodgers_amount;  // предыдущие жильцы
} flat_t;
```

Объём памяти необходимый для хранения одной записи:

```
sizeof(flat_t) == 72
```

Структуры для хранения и обработки таблицы с квартирами:

```
typedef struct
{
   uint32_t id;
   uint8_t rooms_amount;
} flat_key_t; // Структура для хранения одного ключа

typedef struct
{
   size_t size;
   flat_t *flats_array;
} flat_table_t; // Таблица квартир
```

```
typedef struct
{
    size_t size;
    flat_key_t *keys;
} keys_table_t; // Таблица ключей
```

Набор функций

```
// Инициализация структуры квартиры нулевыми значениями
flat_t create_flat(void);

// Создание копии структуры
flat_t clone_flat(flat_t *original);

// Преобразование информации из строки в структуру
int sread_flat(const char *str, flat_t *flat);

// Печать данных структуры на экран
void printf_flat(flat_t *flat);

// Создание пустой таблицы квартир
flat_table_t ft_create(void);

// Копирование таблицы квартир
flat_table_t ft_clone(flat_table_t *original);

// Очищение памяти и обнуление таблицы
void ft_free(flat_table_t *flat_table);

// Чтение таблицы из файла
int ft_read(FILE *file, flat_table_t *flat_table);

// Добавление квартиры в конец таблицы
int ft_append_flat(flat_table_t *flat_table, flat_t *flat);
// Удаление квартиры по ID
int ft_delete_flat(flat_table_t *table,
        unsigned int id, flat_t *deleted_flat);

// Заполнение таблицы ключей по исходной таблице
void ft_gen_keys(flat_table_t *table, flat_t *keys);

// 4 метода сортировки.
// a = с таблицей ключей, b = без ключей
// slow = обменом, fast = вставками
size_t ft_sort_a_fast(flat_table_t *table,
        key_table_t *keys, sort_params_t params);
size_t ft_sort_a_slow(flat_table_t *table,
        key_table_t *keys, sort_params_t params);
size_t ft_sort_b_fast(flat_table_t *table,
```

```
                key_table_t *keys, sort_params_t params);
size_t ft_sort_b_slow(flat_table_t *table,
                key_table_t *keys, sort_params_t params);
```

Описание алгоритмов обработки данных

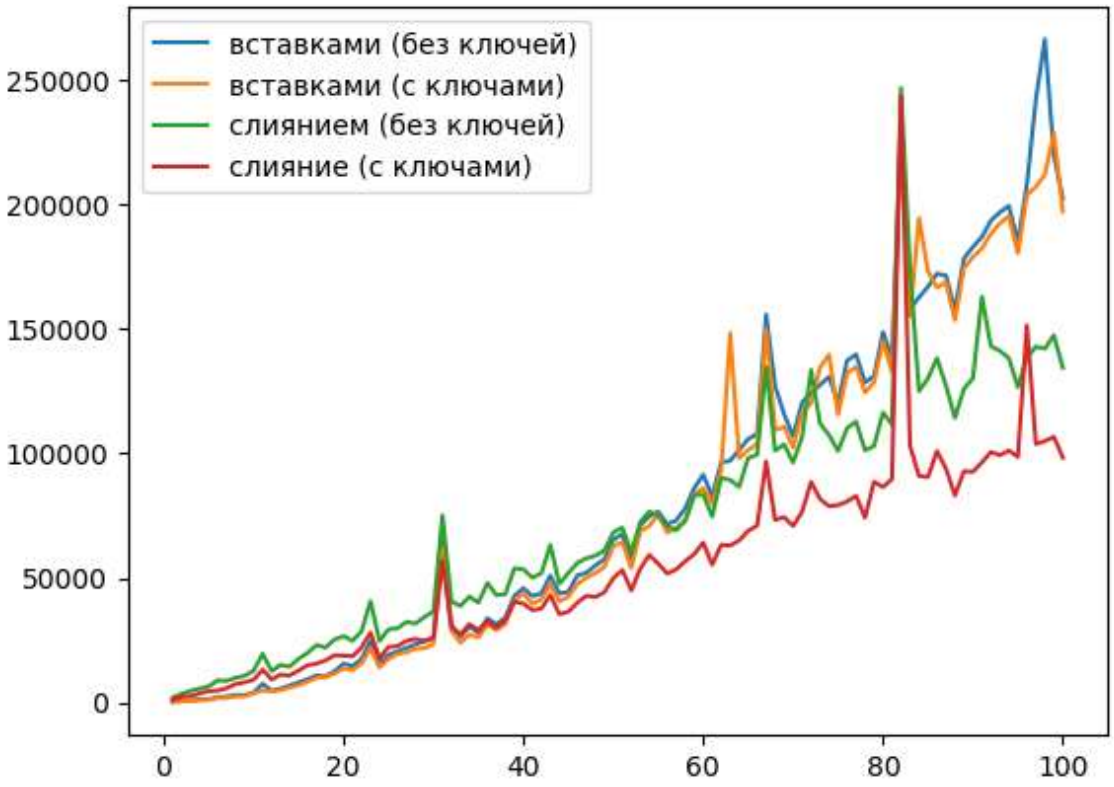
Для реализации сортировки мною будут рассмотрены такие методы как сортировка обменом и сортировка слиянием как представители более медленной и более быстрой сортировок.

Приведём сложность упомянутых алгоритмов:

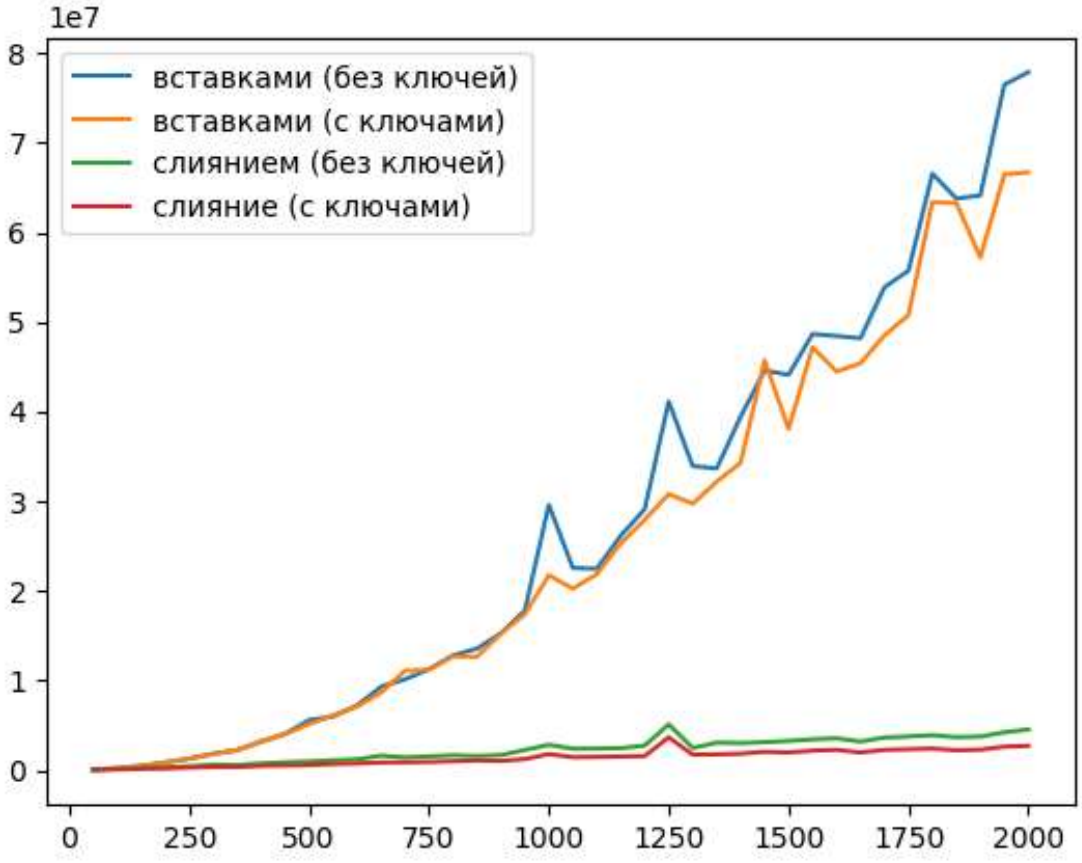
Сортировка вставками: **$O(N^2)$**

Сортировка слиянием: **O(N*log(N))**

Результаты тестирования в виде графиков:



Для большего числа элементов:



Относительная эффективность по времени:

```
вставками/слиянием (без ключей): 33.55%
вставками/слиянием (с ключами): 50.12%
с/без ключей (слияние): 26.86%
```

набор тестов

№	Описание теста	Входные данные	Выходные данные
1	Вывод таблицы	`in.txt` `3`	Форматированная таблица с данными
2	Ошибка при чтении файла	`Null`	Сообщение об ошибке. Завершение работы
3	Неверный выбор опции меню	`in.txt` `a`	Отображение сообщения об ошибке и повторный запуск меню
4	Добавление новой записи	`in.txt` `1` *<flat data>* `3`	Добавление записи в конец таблицы
5	Ошибки при добавлении новой записи	`in.txt` `1` *<invalid data>*	Прерывание операции, возврат в меню

6	Удаление записи	in.txt 5 *<flat id>* 3	Удаление записи с указанным ID из таблицы
7	Ошибки при удалении записи	in.txt 5 *<not flat id>*	Отображение сообщения об ошибке и переход в меню
8	Сортировка таблицы по ключу	in.txt 2 *<1,2,3,4>* *<1,2,3>* *<0,1>*	Сортировка таблицы с последующим её отображением
9	Сортировка пустой таблицы	in.txt 2 *<1,2,3,4>* *<1,2,3>* *<0,1>*	Сообщение о пустой таблице и переход в главное меню
10	Ошибки при сортировке таблицы	in.txt 2 *<invalid vals>*	Отображение сообщения об ошибке и переход в меню
11	Поиск записей по условию	in.txt 4 *<min price>* *<max price>*	Вывод на экран всех подходящих под условия поиска записей или сообщения, что подходящих записей нет
12	Неверное условие при поиске записей	in.txt 4 *<wrong prices>*	Вывод на экран сообщения об ошибке и переход в меню

Выводы по проделанной работе

По окончании работы мне удалось на практике сравнить эффективность двух различных алгоритмов сортировки, а также подтвердить практически их асимптотическую сложность, рассчитанную теоритически.

Судя по полученным результатам, сортировать таблицу с применением дополнительных массивов, оказывается немного более эффективным решением с точки зрения времени выполнения (приблизительно на **26.9%** для алгоритма сортировки слиянием).

Таким образом, при разработке программ необходимо выбирать структуры данных и алгоритмы по их обработке основываясь на имеющихся ресурсах для того, чтобы сделать разрабатываемый продукт наиболее эффективным.

Контрольные вопросы

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

Память под вариантную часть записи выделяется единым блоком, который по своему объему может уместить максимальный тип из используемых. При этом остальные типы используют ту же область памяти,

из-за чего могут быть логические ошибки при неверном интерпретировании имеющихся в вариантой части данных.

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

В лучшем случае произойдет ошибка компиляции. В худшем — введённые данные будут неправильно интерпретироваться в дальнейшем и в какой-то момент приведут к более серьёзным последствиям.

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

За правильностью выполнения операций с вариантной частью должен следить сам программист.

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

Таблица ключей представляет собой массив из упрощенных моделей обычных записей, которые включают в себя минимально возможную информацию для однозначного сопоставления их с исходными записями.

Таблица ключей нужна для сокращения времени работы с исходной таблицей при необходимости частой модификации структуры таблицы, но не самих записей в ней. Например, такой модификацией можно считать сортировку записей, вставку новой записи с сохранением упорядоченности таблицы.

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

В случаях, когда память является более весомым критерием эффективности, следует обрабатывать данные непосредственно на месте, а когда на первом месте стоит время, то конечно стоит использовать таблицу ключей.

Также, если в самой таблице не очень много данных, и они не часто обрабатываются, то перебарщивать с оптимизацией не нужно — в большинстве случаев прирост производительности будет неоправданным (если вообще будет).

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

Для обработки таблиц предпочтительнее использовать способы сортировки не требующие большого количества проходов по всему объему данных, так как таблицы зачастую хранят довольно большие объемы информации и такие «обходы» могут очень дорого обойтись, когда речь зайдёт об эффективности алгоритмов сортировки.