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»

Отчёт

по лабораторной работе №2

Название «Записи с вариантами, обработка таблиц»

Дисциплина «Типы и Структуры Данных»

Вариант 3

Студент ИУ7-31Б _____ Корниенко К. Ю.
(подпись, дата) (Фамилия И.О.)

Преподователь _____ Силантьева А. В.
(подпись, дата) (Фамилия И.О.)

Москва, 2021

Содержание

Введение

Цель лабораторной 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.

Задачи:

— Реализовать программу для работа с типом данных «запись» и с данными, хранящимися в таблицах.

— Провести сравнительный анализ реализации алгоритмов сортировки в таблицах с большим числом полей и тех же алгоритмов, но при использовании таблицы ключей.

— Сравнить эффективность протестированных алгоритмов.

1 Постановка задачи

1.1 Описание задачи

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, используя 2 разных алгоритма сортировки (простой, ускоренный). Оценить эффективность этих 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 %).

1.2 Техническое задание

Имеются описания: Type жилье = (дом, общежитие); Данные: Фамилия, имя, группа, пол (м, ж), возраст, средний балл за сессию, дата поступления адрес: дом: (улица, No. дома, No. кв); общежитие: (No. общ., No. комн.);

Ввести общий список студентов. Вывести список студентов, указанного года поступления, живущих в общежитии.

Сортировки:

В данной лабораторной работе для упорядочивания таблицы будут применены следующие алгоритмы:

— Сортировка выбором (Selection sort) – временная сложность $O(n^2)$

— Сортировка слиянием (Merge sort) – временная сложность $O(nlogn)$

2 Конструкторский раздел

В данном разделе представлены листинги кодов используемых структур данных.

2.1 Описание Структур Данных

В листингах 2.1 - 2.7 приведены используемые в программе структуры данных. Применение типа «запись» с вариантной частью обоснована тем, что при хранении информации о месте проживания студента, целесообразно хранить либо информацию об общежитии, либо информацию о квартире.

Листинг 2.1 — Структура Таблицы

```
typedef struct studtable
{
    student_t* data;
    unsigned int size;
} studtable_t;
```

Листинг 2.2 — Структура таблицы ключей

```
typedef struct keytable
{
    _key_t* data;
    size_t size;
} keytable_t;
```

Листинг 2.3 — Структура ключа

```
typedef struct key
{
    size_t id;
    double avg;
} _key_t;
```

Листинг 2.4 — Структура студента

```
typedef struct
{
    char surname[MAX_STR];
    char name[MAX_STR];
    char group[MAX_STR];
    gender_t gender;
    uint16_t age;
    double avg_score;
    date_t enroll_date;
    housing_t house;
    union housing
    {
        dorm_t dorm;
```

```
14          appartment_t appartment;
15      } housing;
16  } student_t;
```

Листинг 2.5 — Перечисляемый тип пола

```
1  typedef enum
2  {
3      UNDEFINED,
4      MALE,
5      FEMALE
6  } gender_t;
```

Листинг 2.6 — Структура даты

```
1  typedef struct
2  {
3      uint16_t year;
4      uint16_t month;
5      uint16_t day;
6  } date_t;
```

Листинг 2.7 — Перечисляемый тип жилья

```
1  typedef enum
2  {
3      UNKNOWN,
4      DORM,
5      APPARTMENT
6  } housing_t;
```

2.2 Вывод

Результатом данного раздела являются спроектированные структуры данных, используемые для создания таблицы студентов и таблицы ключей.

3 Технологический раздел

К разрабатываемому программному обеспечению предъявляются следующие требования:

3.1 Требования к ПО

Требования ко вводу:

1) На вход программе подается пункт меню (число от 0 до 9).
2) На вход подается файл с данными студентов в заданном формате:

<Фамилия>;<Имя>;<Группа>;<Пол>;<Возраст>;<Средний балл>;<Дата поступления>;<Где проживает>;[...]; В зависимости от поля *<где проживает> [...]:*

dorm : *<номер общежития>,<номер комнаты>;*

appartment : *<улица>,<дом>,<квартира>;*

3) ПО должно:
   - Выводить таблицу студентов.
   - Выводить список студентов указанного года поступления, живущих в общежитии.
   - Сортировать исходную таблицу.
   - Выводить таблицу ключей.
   - Сортировать таблицу ключей.
   - Выводить отсортированную исходную таблицу по таблице ключей.
   - Добавлять студентов в таблицу (ввод с клавиатуры).
   - Удалять студентов из таблицы по ключу (id).
   - Проводить анализ временной сложности алгоритмов сортировки.
4) ПО должно выводить потраченную память и время.

3.2 Возможные аварийные ситуации

- Некорректный ввод данных.
- Пустой файл.

3.3 Набор основных функций

Ниже представлены листинги основных функций для работы с таблицей (Листинг 3.1) и для работы со структурой студента (Листинг 3.2)

Листинг 3.1 — Набор функций для работы с таблицей

```
1 // Функция возвращает пустую таблицу
2 studtable_t stable_empty();
3
4 // Функция резервирует память под таблицу
5 studtable_t stable_reserve(unsigned int size);
6
7 // Функция освобождает ресурсы
```

```
8  void stable_destroy(studtable_t* table);
9
10 // Функция проверки таблицы на валидность
11 bool stable_valid(const studtable_t* table);
12
13 // Функция загрузки таблицы по имени файла
14 status_t stable_load(studtable_t* table, const char* filename);
15
16 // Функция загрузки таблицы по файловому указателю
17 status_t stable_loadf(studtable_t* table, FILE* file);
18
19 // Функция добавления студента
20 status_t stable_push_back(studtable_t* table, const student_t* stud);
21
22 // Функция удаления студента
23 status_t stable_remove(studtable_t* table, size_t id);
24
25 // Функция вывода таблицы студентов
26 void stable_print(const studtable_t* table);
27
28 // Функция вывода студентов указанного года поступления, живущих в общежитии
29 void stable_cond_print(studtable_t* table, uint16_t year);
30
31 // Функция вывода студентов по таблице ключей
32 void stable_print_key(const studtable_t* table, const keytable_t* kt);
33
34 // Функция сортировки таблицы
35 void stable_sort(studtable_t* table, sort_t sort_fn);
```

Листинг 3.2 — Набор функций для работы со структурой студента

```
1  // Проверка структуры на валидность
2  bool stud_valid(const student_t* stud);
3
4  // Чтение структуры по строке
5  status_t read_student(student_t* stud, const char* str);
6
7  // Чтение структуры из файла
8  status_t read_student_f(student_t* stud, FILE* file);
9
10 // Ввод структуры с клавиатуры
11 status_t input_student(student_t* stud);
```

3.4 Реализация алгоритмов

В листингах 3.3, 3.5 представлены вспомогательные функции для сортировок.

В листингах 3.4, 3.6 представлены реализации алгоритмов сортировки Выбором и Слиянием.

Листинг 3.3 — Вспомогательные функции

```
static void swap(void* a, void* b, size_t size)
{
    for (size_t i = 0; i < size; i++)
    {
        *((char*)a + i) ^= *((char*)b + i);
        *((char*)b + i) ^= *((char*)a + i);
        *((char*)a + i) ^= *((char*)b + i);
    }
}

static size_t min_index(const void* base, size_t nitems, size_t size, compare_t
    cmp)
{
    size_t res = 0;
    const void* min_el = base;

    for (size_t i = 1; i < nitems; i++)
    {
        const void* elem = (const char*)base + i * size;
        if (cmp(elem, min_el) < 0)
        {
            min_el = elem;
            res = i;
        }
    }

    return res;
}
```

Листинг 3.4 — Сортировка выбором

```
void selection_sort(void* base, size_t nitems, size_t size, compare_t cmp)
{
    for (size_t i = 0; i < nitems; i++)
    {
        size_t min_i = min_index((char*)base + i * size, nitems - i, size, cmp);
        if (min_i != 0)
            swap((char*)base + i * size, (char*)base + (i + min_i) * size, size);
    }
}
```

Листинг 3.5 — Слияние двух массивов

```
static void merge(void* base, size_t size, size_t mid, size_t right, compare_t
    cmp)
{
    char tmp[size * mid];
```

```
    memmove(tmp, base, size * mid);

    size_t i = 0, j = mid;
    for (size_t k = 0; k < right; k++)
    {
        if (j == right || (i < mid && cmp(tmp + i * size, (char*)base + j *
            size) < 0))
        {
            memmove((char*)base + k * size, tmp + i * size, size);
            i++;
        }
        else
        {
            memmove((char*)base + k * size, (char*)base + j * size, size);
            j++;
        }
    }
}
```

Листинг 3.6 — Сортировка слиянием

```
void merge_sort(void* base, size_t nitems, size_t size, compare_t cmp)
{
    if (nitems < 2)
        return;

    if (nitems == 2)
    {
        if (cmp(base, (char*)base + size) > 0)
        swap(base, (char*)base + size, size);
        return;
    }

    size_t mid = nitems >> 1;

    merge_sort(base, mid, size, cmp);
    merge_sort((char*)base + mid * size, nitems - mid, size, cmp);
    merge(base, size, mid, nitems, cmp);
}
```

3.5 Тестовые данные

В таблице 3.1 указаны сценарии тестов для функционала реализуемой программы.

Таблица 3.1 — Тестовые данные

№	Описание теста	Входные данные	Выходные данные
1	Вывод таблицы	data.txt 6	Форматированная таблица с данными
2	Ошибка при чтении	NULL	Сообщение об ошибке. Завершение работы
3	Неверный ввод опции меня	data.txt a	Завершение работы
4	Добавление новой записи	data.txt 4 <student data>	Добавление записи в конец таблицы
5	Ошибки во время добавления записи	data.txt 4 <invalid student data>	Сообщение об ошибке. Возврат в меню.
6	Удаление записи	data.txt 3 <student id>	Удаление записи с указанным ID
7	Ошибка при удалении записи	data.txt 3 <not student id>	Сообщение об ошибке. Возврат в меню
8	Сортировка таблицы по ключу	data.txt 2 2	Вывод информации о временной сложности сортировки и затраченной памяти. Отсортированная таблица ключей
9	Загрузка пустой таблицы	data.txt	Сообщение об ошибке. Завершение программы.
10	Поиск записей по условию	data.txt 5 <year>	Вывод таблицы студентов указанного года поступления, живущих в общежитии
11	Неверное условие при поиске записей	data.txt 5 <bad year>	Сообщение об ошибке. Возврат в меню.

3.6 Вывод

В данном разделе были разработаны исходные коды алгоритмов сортировок, а также выделены тестовые данные. Все тесты пройдены успешно.

4 Исследовательский раздел

4.1 Технические характеристики

Ниже были приведены технические характеристики устройства, на котором было произведено тестирование ПО:

— Операционная система: Ubuntu (Linux) 20.04 LTS 64-bit.
— Оперативная память: 16 GB.
— Процессор: AMD Ryzen 5 3500U @ 8x 2,1GHz

4.2 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось при помощи функции *clock()* из библиотеки *time.h*

Таблица 4.1 — Сравнение быстродействия алгоритмов сортировки с ключом и без ключа

N	Выбором, мкс	Слиянием, мкс	Выбором/ключ, мкс	Слияние/ключ, мкс
100	1800	479	84	31
200	3822	401	189	67
300	2758	1569	846	102
400	7009	1916	920	133
500	8481	2820	1395	175
600	10269	1928	2072	214
700	11297	3162	3524	265
800	13791	2038	4583	358
900	15396	2365	5608	237
1000	14242	5560	5273	379
1100	16930	3199	4442	587
1200	21750	5216	5936	486
1300	20576	5029	6992	645
1400	22221	5574	10749	392
1500	22694	5946	10927	617
1600	31442	6474	12145	677
1700	26806	7070	13073	905
1800	27186	7949	14929	507
1900	34171	6021	16445	573

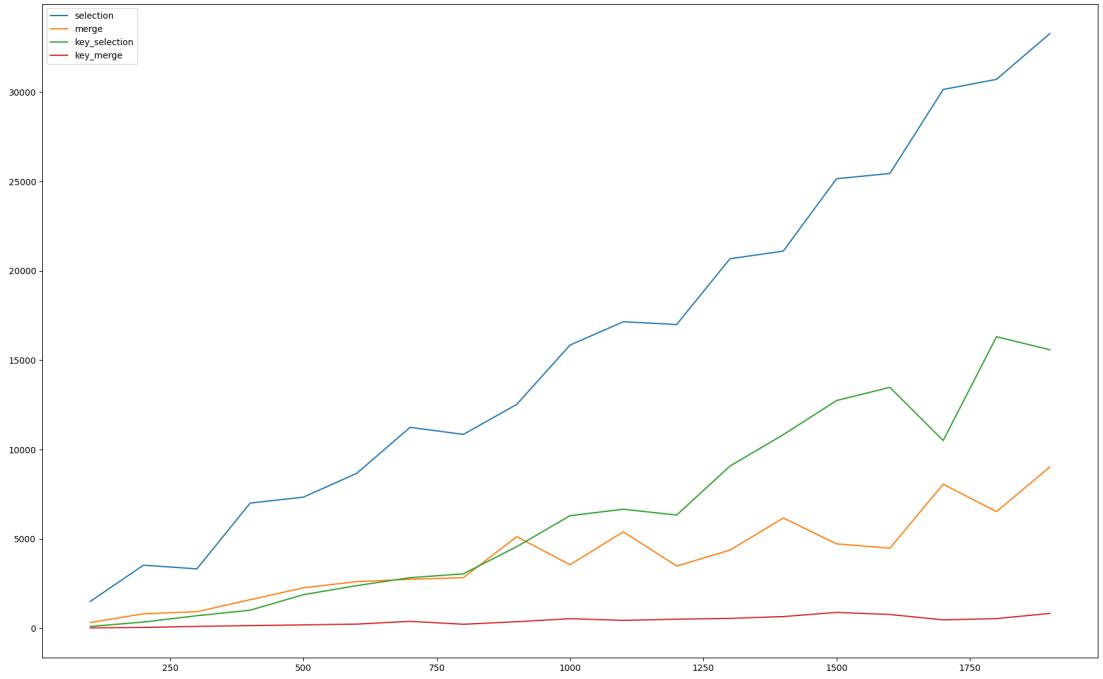

Рисунок 4.1 — График зависимости времени выполнения сортировки от размерности сортируемой таблицы

4.3 Относительная эффективность по времени

Проведем расчет эффективности по времени изучаемых алгоритмов, согласно формуле (4.1):

$$f = \frac{t_1 - t_2}{t_1} * 100\% \tag{4.1}$$

1) Вставками/слиянием (без ключей) – 73%
2) Вставками/слиянием (с ключами) – 95%
3) С ключами/без ключей (слияние) – 90%
4) С ключами/без ключей (выбором) – 53%

4.4 Вывод

Реализация алгоритма сортировки слиянием выполняется примерно в 4 раза быстрее, чем реализации алгоритма сортировки вствками. Сортировка вставками по ключам увеличивает эффективность по времени еще в 10 раз. Таким образом использование таблицы ключей при сортировке больших таблиц увеличивает эффективность по времени, но в то же время требует дополнительных затрат памяти на хранение самой таблицы.

5 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.

Заключение

В ходе выполнения лабораторной работы было выявлено, что использование таблицы ключей при сортировке таблицы делает процесс сортировки более эффективным по времени, но также менее эффективным по памяти. Это связано с тем, что нет необходимости работать с массивной структурой, а достаточно просто отсортировать таблицу, состоящую из двух полей. Однако в таком случае затрачивается дополнительная память на хранение этой таблицы.

Стоит отметить, что использование таблицы ключей неэффективно при небольших размерах исходной таблицы. Разница во времени сортировки по ключам и без них несущественна – в таком случае лучше использовать сортировку самой таблицы, сократив при этом объемы потребляемой памяти.