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

ДИСЦИПЛИНА Типы и структуры данных

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 «ЗАПИСИ С ВАРИАНТАМИ. ОБРАБОТКА ТАБЛИЦ».

Студент Иванов Павел Александрович

Группа ИУ7 – 35Б

Вариант №6

Проверил

2021 г.

Оглавление

Цель работы 2
Описание условия задачи 2
Описание технического задания 4
Описание структур данных 6
Описание алгоритма 8
Функции программы 8
Набор тестов 9
Оценка эффективности 11
Ответы на контрольные вопросы 14
Вывод 15

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

Создать таблицу, содержащую не менее 40 записей с вариантной частью (тип – запись с вариантами (объединениями)). 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 %).

Ввести список литературы, содержащий фамилию автора, название книги, издательство, количество страниц, вид литературы

(1: техническая – отрасль, отечественная, переводная, год издания;
2: художественная – роман, пьеса, стихи;
3: детская – сказки, стихи).

Вывести список отечественной технической литературы по указанной отрасли указанного год.

Интерфейс программы должен быть понятен неподготовленному пользователю. При разработке интерфейса программы следует предусмотреть:

- указание формата и диапазона данных при вводе и (или) добавлении записей; указание операций, производимых программой;
- наличие пояснений при выводе результата;
- возможность добавления записей в конец таблицы и удаления записи по значению указанного поля.
- просмотр отсортированной таблицы ключей при несортированной исходной таблице;
- вывод упорядоченной исходной таблицы;
- вывод исходной таблицы в упорядоченном виде, используя упорядоченную таблицу ключей;
- вывод результатов сравнения эффективности работы программы при обработке данных в исходной таблице и в таблице ключей;
- вывод результатов использования различных алгоритмов сортировок.

Одним из результатов работы программы должна быть количественная информация (лучше представить в виде таблицы) с указанием времени, затраченного на обработку исходной таблицы и таблицы ключей двумя алгоритмами сортировки (при этом, не забыть оценить так же время выборки данных из основной таблицы с использованием таблицы ключей), а также — объем занимаемой при этом оперативной памяти.

При тестировании программы необходимо:

- проверить правильность ввода и вывода данных (в том числе, отследить попытки ввода неверных по типу данных в вариантную часть записи);
- обеспечить вывод сообщений при отсутствии входных данных («пустой ввод»);
- проверить правильность выполнения операций;

- отследить переполнение таблицы.

- При хранении исходных данных в файлах (что приветствуется) необходимо также проверить наличие файла и изменения информации в нем при удалении и добавлении данных в таблицу.

ОПИСАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Входные данные:

Текстовый файл:

Файл, содержащий записи таблицы. Каждая запись располагается на новой строке, поля записей разделяются **табуляцией**[1].Записей в файле должно быть не менее 40, но и не более 1000. Повторение записей допускается. Каждая запись должна иметь следующее представление:
<фамилия автора> \t <название книги> \t <издательство> \t <количество страниц> \t <вид литературы> \t + в зависимости от вида:

- Если техническая:
  <отрасль> \t <отечественная / переводная (или, или)> \t <год>
- Художественная
  <роман / пьеса / стихи (или, или, или)>
- Детская
  <Сказки / стихи>

В файле допустимо использование **только латиницы**. Причем вид литературы может быть только три слова, поэтому его перевод может быть только: *technical* (техническая), *fiction* (художественная), *children's* (детская). Также **недопустимы иные переводы**, кроме
отечественная — *state*
переводная — *translated*
роман — *novel*
пьеса — *play*
стихи — *poetry*
сказки — *fairy tails*
Пустые поля в файле недопустимы. Также существуют ограничения на входные данные:

- Любая строка не может быть длиннее 20 символов (учитывая пробелы и знаки препинания)
- Год не может быть отрицательным или превосходить 2021
- Количество страниц не превышает 30 000

[1] Использование табуляции обосновано тем, что так будет удобно вставлять таблицу в привычные программы для их обработки по типу MS Excel

- Файл не может заканчиваться новой пустой строкой.

Пункт меню:
Целое число.
Дополнительные данные для меню:
Строка (только если требуется ввести ключ)

Выходные данные:

a) Таблица данных (отсортированная или неотсортированная в зависимости от выбранного пункта меню)
b) Данные о сравнении эффективности способов сортировки
c) Таблица ключей
d) Результат поиска

Действие программы:

Обработка таблицы данных, прочитанной из файла, которая включает в себя добавление новых записей, сортировку разными способами. Кроме этого измеряется и сравнивается скорость сортировок и их затраты по памяти.

Функции программы:
0 – Выход;
1 – Загрузить данные из файла;
2 – Добавить запись в конец;
3 – Удалить запись по значению указанного поля;
4 – Поиск отечественной технической литературы по указанной отрасли указанного года;
5 – Просмотр упорядоченной таблицы ключей по полю количества страниц;
6 – Упорядочивание таблицы по возрастанию ключей (по полю количества страниц), используя основную таблицу;
7 – Упорядочивание таблицы по возрастанию ключей (по полю количества страниц) с помощью таблицы ключей
8 – Вывод результатов сравнения эффективности работы программы при обработке данных в исходной таблице и в таблице ключей;
9 – Вывод результатов использования различных алгоритмов сортировок.

Обращение к программе:

Запускается .exe файл. Если его нет, из консоли следует вызывать make run.

Аварийные ситуации:

- Подан пустой файл

Сообщение: “Something wrong in file or you already have one!”

- Данные в файле поданы в неверном формате
  Сообщение: «Something wrong in file or you already have one!»
- Файл с данными не существует
  Сообщение: «Something wrong in file or you already have one!»
- Добавленная запись некорректна
  Сообщение: «Incorrect input! Try again! »
- Запись для удаления не существует
  Сообщение: «Required information doesn’t exist!»
- Запись для поиска не существует
  Сообщение: «Required information doesn’t exist!»

ОПИСАНИЕ СТРУКТУР ДАННЫХ

Тип книги описывается как перечисляемый тип. Таким образом программа легче читается и хранение более удобное.

```
typedef enum { TECHNICAL, FICTION, CHILDREN } book_t;
```

Аналогично тип художественной литературы:

```
typedef enum { NOVEL, PLAY, FICTION_POETRY }
fiction_info_t;
```

Детской литературы:

```
typedef enum { FAIRYTAILS, CHILDREN_POETRY }
children_info_t;
```

И технической литературы:

```
typedef enum { STATE, TRANSLATED } tech_pub_t;
```

Для хранения информации о технической литературе используется структура tech_info_t. Поля структуры: строка area (область исследований – не более 20 символов), тип технической литературы (перечисляемый тип выше) и целое число типа short (год не может быть более 2021).

```
typedef struct
{
  char area[MAX_FIELD_LEN + 1];
  tech_pub_t type;
  short int year;
} tech_info_t;
```

Для хранения одной книги используется следующая структура:

```
typedef struct
{
```

```
  char name[MAX_FIELD_LEN + 1];
  char title[MAX_FIELD_LEN + 1];
  char publisher[MAX_FIELD_LEN + 1];
  int pages;
  book_t type;
  union spec_u
    {
        tech_info_t tech_info;
        fiction_info_t fiction_info;
        children_info_t children_info;
    } spec;
} book;
```

Её поля: строка name (не более 20 символов), строка title (не более 20 символов), строка publisher (не более 20 символов), целое число – количество страниц, которое хранится в int. Выбор данного типа обоснован тем, что число страниц может быть до 30 000. Далее идёт перечислимый тип книги и вариативная часть — в зависимости от вида книги она содержит разное число полей. Это может быть информация о технической (структура tech_info_t), художественной (перечисляемый тип fiction_info_t) и детской литературе (перечисляемый тип children_info_type).

Использование вариативной части обоснованно тем, что с помощью её мы выделяем меньше памяти. Если бы это была структура, то выделилось бы sizeof(tech_info_t) + sizeof(fiction_info_t) + sizeof(children_info_t). В случае же объединения выделится max(sizeof(tech_info_t), sizeof(fiction_info_t, sizeof(children_info_t)), что очевидно меньше. Таким образом, экономится память.

Для хранения всей таблицы используется структура book_table, в которой два поля: data (статический массив из не более чем MAX_BOOKS_NUM (=100) элементов) и quantity (short число элементов в массиве). Поле quantity необходимо, чтобы избежать выход за пределы массива.

```
typedef struct
{
    book data[MAX_BOOKS_NUM];
    short int quantity;
} book_table;
```

Для таблице ключей же используется структура pahe_key_table, которая хранит число элементов в таблице как число типа short (оно не больше 100), а также таблицу ключей в виде матрицы элементов short. Первый столбец матрицы — номер в исходной таблице, второй — номер после сортировки, третий — ключ.

```
typedef struct
{
```

```
    short int n;
    /* | 0 - old | 1 - new | 2 - value | */
    short int pages[MAX_BOOKS_NUM][3];
} page_key_table;
```

ОПИСАНИЕ АЛГОРИТМА

1. Вывод меню программы;
2. Ввод данных: проверка данных, добавление в структуры;
3. Сортировка: либо пузырьком («камнем вниз»), либо qsort;
4. Замер времени, памяти;
5. Выход, если введён ноль.

ФУНКЦИИ ПРОГРАММЫ

- добавление записи

```
void add_line(book_table *table);
```

- удаление записи

```
void delete_line(book_table *table);
```

- поиск записи по полю

```
void find_tech_data_by_area(book_table *table);
```

- создание таблицы ключей

```
int create_page_key_table(book_table *table,
page_key_table *key_table);
```

- сортировка таблицы ключей

```
void sort_key_table(page_key_table *key_table,
int64_t *bubble_time_key, int64_t *qsort_time_key,
int64_t elapsed_time);
```

- сравнение строк таблицы ключей

```
int compare_key_lines(const void *one, const void
*two);
```

- сортировка пузырьком таблицы ключей

```
void bubble_sort_key_table(page_key_table
*key_table);
```

- перестановка строк таблицы ключей

```
void swap_lines(page_key_table *key_table, short int
line1, short int line2);
```

- вывод статистики сортировки

```
void output_sort_stats(book_table table);
```

- вывод сравнения сортировок

```
void output_efficiency_results(book_table table);
```

НАБОР ТЕСТОВ

№	Тестовый случай	Пользовательский ввод	Результат
1	Неверный ввод команды	99	No such input!
2	Пустой файл	1, empty.txt	Something wrong in file or you already have one!
3	Файла нет	1, nofile.txt	Something wrong in file or you already have one!
4	Некорректный ввод в файле (неполная строка / переполнение поля / введено не число)	1, broken.txt	Something wrong in file or you already have one!
5	Переполнение при добавлении	1, 100.txt, 2	Table overflow!
6	Недостаточно записей	1, 39.txt	Something wrong in file or you already have one!

7	Слишком много записей	1, 101.txt	Something wrong in file or you already have one!
8	Добавление корректной записи в конец	1, 40.txt, 2, корректная строка	Добавлена строка в конец таблицы
9	Добавление некорректной записи в конец	1, 40.txt, 2, некорректная строка	Строка не добавлена
8	Удаление несуществующих записей	1, 40.txt, 3, несуществующая строка	Строка не удалена
9	Удаление существующих записей	1, 40.txt, 3, существующая запись	Удалены все существующие записи по заданному полю
10	Поиск существующей записи по полю	1, 40.txt, 4, существующее поле и ключ	Выведены на экран все подходящие записи
11	Поиск несуществующей записи по полю	1, 40.txt, 4, несуществующая запись	Выведено сообщение об ошибке
12	Открытие таблицы ключей	1, 40.txt, 5	Выведена таблица ключей, выбран алгоритм, выведена
13	Сортировка главной таблицы	1, 40.txt, 6	Выведена главная таблица, отсортированная по количеству страниц

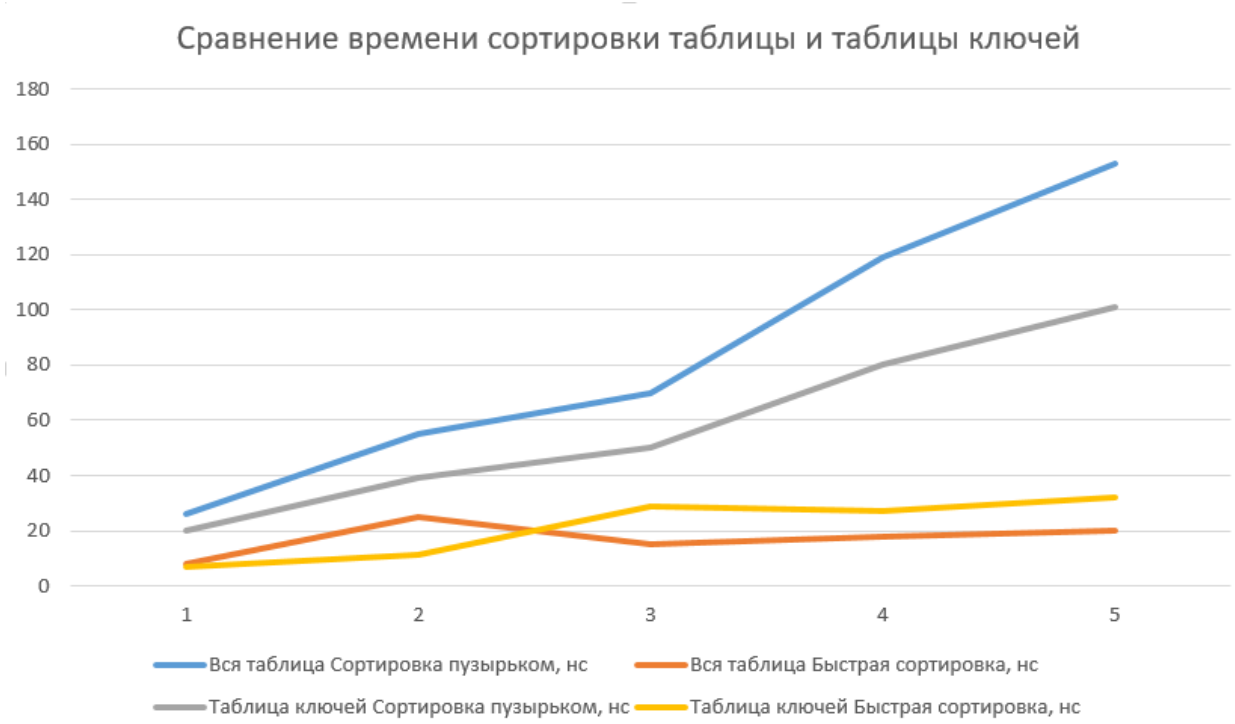
14	Сортировка таблицы ключей	1, 40.txt, 7	Выведена главная таблица, отсортированная по количеству страниц
15	Удаление в пустой таблице	1, 40.txt, 3	Сообщение об ошибки
16	Открытие результатов эффективности	1, 40.txt, 8	Выведена таблица сравнений по времени и памяти для конкретной таблицы
17	Открытие результатов сравнения алгоритмов	1, 40.txt, 9	Выведена таблица сравнений по времени для конкретной таблицы

ОЦЕНКА ЭФФЕКТИВНОСТИ

Произведу замеры по времени для разных размеров таблиц. Время печати таблицы не измеряется. Асимптотическая сложность сортировки пузырьком — O(n2), быстрой — O(nlogn). В измерение времени сортировки таблицы ключей включено время создания таблицы ключей.

N	Вся таблица		Таблица ключей	
	Сортировка пузырьком, нс	Быстрая сортировка, нс	Сортировка пузырьком, нс	Быстрая сортировка, нс
40	26	8	20	7
60	55	25	39	11
70	70	15	50	29

90	119	18	80	27
100	153	20	101	32



Особого внимания требует время быстрой сортировки таблицы ключей, которое в среднем больше, чем время быстрой сортировки всей таблицы. Такая разница следует из того, что в данное время было включено время создания таблицы ключей.

Сравнение по памяти:

N	Таблица ключей, байты	Вся таблица, байты
40	4162	240
60	6242	360
70	7282	420

90	9362	540
100	10402	600

Важно отметить, что измерялась фактическая (занятая не мусором) память. В моей же реализации используется статическая память, которая выделяется по максимуму. Получается, что во всех строках таблицы должны быть такие же значения, как в строке при N = 100. При динамической же реализации будут актуальны данные, представленные в моей таблице.

Во всяком случае, как оказалось при анализе, при любом N таблица ключей будет занимать не более 6% от исходной таблицы, поэтому подход к измерению памяти (измерения занятой или выделенной) не так важен.

Сравнение в процентах:

N	Дополнительная память, необходимая под таблицу ключей, процент от исходной таблицы	Превосходство сортировки пузырьком на таблице ключей над сортировкой пузырьком на всей таблице	Превосходство быстрой сортировки на таблице ключей над быстрой на всей таблице
40	5,7%	77%	—14%
60	5,7%	70%	44%
70	5,7%	71%	—28%
90	5,7%	67%	—32%
100	5,7%	66%	—33%

Конечно же, быстрая сортировка сортирует таблицу ключей быстрее всей таблицы. Но по моим данным получилось немного наоборот. Это связано с

тем, что я учитывал время создания таблицы ключей (чтобы это сделать, нужно пройтись по всей таблице). Исходя из этого, можно считать результаты соответствующими ожиданиям.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1.Как выделяется память под вариантную часть записи?

Выделяется максимальное количество памяти, которое может занимать вариантное поле. Так, например, если вариантное поле может быть целым числом типа int (4 байта) или double (8 байт), то всегда будет выделяться 8 байтов памяти.

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

Тип данных в вариативной части при компиляции не проверяется, поэтому проверка соответствия типа данных лежит на программисте. Если попытаться ввести в вариантное поле данные, не соответствующие описанным, программа будет либо неправильно работать, либо это может привести к непредсказуемому поведению.

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

Поскольку тип данных в вариативной части не проверяется, за правильностью выполнения операций с ней должен следить программист.

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

Каждая запись в таблице разделена на поля. Как правило, каждому полю придается смысл и у него есть название. Значение определённой записи по определенному полю называется ключом. Таблица ключей — таблица, содержащая только ключи записей по конкретному полю и их номера в изначальной таблице. Такую структуру данных удобно использовать для сортировки по полю. Использование такой таблицы сокращает затраты по времени и не портит изначальную таблицу.

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

Использование таблицы ключей требует затраты по памяти. Иногда экономия по времени, получаемая вследствие экономии по памяти, настолько невелика, что нет смысла о ней задумываться. Например, при обработке таблиц относительно небольшого размера, где каждая запись не располагает большим количеством полей. Когда задача подразумевает либо неизменность изначальной таблицы, либо большие массивы данных, которые записаны в таблицу, необходимо использовать таблицу ключей.

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

Выбор алгоритма сортировки зависит от выбора способа обработки таблицы. Так, если по заданию используется таблица целиком, то лучше использовать алгоритмы, требующие наименьшее количество перестановок. Иначе может не хватить памяти или постоянное копирование длинных записей займёт продолжительное время. Если же используется таблица ключей, можно использовать алгоритмы, которые являются более эффективными по времени.

Вывод

В ходе данной работы я разобрался с типом данных «таблица» и методами её обработки. В ходе работы я как раз научился использовать таблицу ключей. Как оказалось, у неё есть как свои плюсы, так и свои минусы.

Из плюсов таблицы ключей могу отметить:

- Исходная таблица не меняется
- Медленные алгоритмы сортировки (в моей работе — с асимптотической сложность $O(n^2)$ и требующие большого числа перестановок) работают быстрее на таблице ключей

Минусы таблицы ключей:

- Необходима дополнительная память
- Требуется время на создание таблицы (целый проход по таблице — если таблица большая, то это может занять длительное время)

Из опыта, полученного в лабораторной работе могу заключить, что если в программе используется медленный алгоритм сортировки, то лучше

использовать таблицу ключей. Прирост по памяти в 5,7% компенсируется приростом по времени в 70%. С быстрой же сортировкой дела обстоят по-другому. Из-за времени, которое требуется на создание таблицы ключей, программа в целом работает немного дольше. Также разница по времени может быть связана с тем, что быстрая сортировка не требует такого большого количества перестановок, как сортировка пузырьком.

Что касается структур данных, я на примере данного задания разобрался с вариативной частью структуры. Её использование сильно экономит память, хотя и требует дополнительной ответственности (все поля разделяют одну область памяти).

Таким образом, подбор структуры данных играет большую роль, так как правильно выбранная структура может сэкономить память, но тем не менее может требовать дополнительного внимания. А выбор способа сортировки таблицы должен исходить из выбора алгоритма сортировки и размера таблицы.