|  |  |
| --- | --- |
|  | **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 предмета: Типы и структуры данных

Студент: Гриценко Алексей Михайлович

Группа: ИУ7-31Б

*2020г.*

1. **Описание условия задачи**

Создать таблицу, содержащую не менее 40-ка записей (тип – запись с вариантами (объединениями)). Упорядочить данные в ней по возрастанию ключей, двумя алгоритмами сортировки, где ключ – любое невариантное поле (по выбору программиста), используя: а) саму таблицу, б) массив ключей. (Возможность добавления и удаления записей в ручном режиме обязательна). Осуществить поиск информации по варианту.

1. **Техническое задание**
2. ***Описание исходных данных***

Текстовый файл, представляющий таблицу записей с вариантной частью.

В текстовом файле хранятся записи абонентов, информация идёт через запятую, когда все поля заполнены, новый абонент и информация о нём хранится в новой строке

Формат ввода:

1. Добавить абонента в таблицу

2. Вывести таблицу абонентов

3. Удалить абонента из таблицы

4. Считать таблицу абонентов из файла

5. Записать таблицу абонентов в файл

6. Отсортировать таблицу абонентов по фамилии

7. Вывести таблицу ключей

8. Отсортировать таблицу ключей по фамилии

9. Результат сравнения эффективности обработки данных

10. Результат сравнения сортировок

11. Вывести список абонентов, у которых день рождения в ближайшую неделю

0. Выход

1. ***Описание результата программы***

Список записей, удовлетворяющих условиям (все друзья, которых необходимо поздравить с днем рождения в ближайшую неделю). Вывод получившейся таблицы в другой текстовый файл. Отсортированные таблицы ключей. Время работы для различных сортировок. Сравнение эффективности обработки данных

1. ***Задача, реализуемая программой***

Программа позволяет пользователю работать с таблицей

1. ***Обращения к программе***

Способ обращения к программе происходит через консоль.

1. ***Описание возможных аварийных ситуаций и ошибок пользователя***

* Открытие несуществующего файла;
* Открытие файла с некорректным форматом данных;
* Работа с пустым списком

Во всех указанных случаях программа завершится корректно сообщит об ошибке, а также её типе.

1. **Описание внутренних структур данных**

Основная структура listp, которая хранит в себе фамилию, имя, телефон, адрес где живет человек, далее идет статус абонента (личный/рабочий), и в зависимости от статуса либо личные данные, либо рабочие. Структура data, хранящая в себе дату, месяц и год рождения абонента. Структура office, хранящая в себе должность и организацию абонента

**Абонент**

typedef struct

{

char surname[30];

char name[20];

int phone;

char adres[40];

enum

{

personal,

work

}st;

union

{

struct data dofb;

struct office ofc;

} type;

} listp;

**Ключ**

typedef struct

{

int index\_src;

char surname[30];

} key\_listp;

**Ключ – статус «личный»**

struct data

{

int d;

int m;

int y;

};

**Ключ – статус «служебный»**

struct office

{

char position[20];

char organization[20];

};

1. **Описание алгоритма**
2. Данные из файла „in.txt’’ читаются в таблицу элементов, далее создается таблица ключей по прочитанной таблице абонентов.
3. Читается команда пользователя и выполняется.
4. При печати таблицы производится печать каждого ее элемента на новой строке, с номером и пояснением что есть что.
5. При добавлении в таблицу элемента производится проверка размера таблицы, чтение элемента. Если размер позволяет добавить абонента и он прочитан корректно, то он добавляется в конец таблицы абонентов, создается ключ по этому абоненту, который добавляется в конец таблицы ключей, и размер таблицы увеличивается на 1.
6. При удалении элемента из таблицы производится чтение индекса элемента, который нужно удалить. Если индекс является валидным, то все элементы в таблице абонентов, стоящие за ним, сдвигаются на 1 влево, производится поиск ключа в таблице ключей с введенным индексом и аналогичное смещение, размер таблицы уменьшается на 1.
7. При поиске абонентов, у которых день рождения будет в ближайшую неделю, происходит обход всех абонентов из таблицы абонентов. Для каждого абонента создается копия даты его рождения, в которой в качестве года выступает текущей год. Полученная дата сравнивается с текущей и если разница между ними меньше или равна 7 дням, то происходит печать абонента на экран.
8. Сортировка таблиц ключей и абонентов происходит с помощью алгоритмов сортировки шейкером и быстрой сортировки. Если сортировка происходит с использованием таблицы ключей, то сортируется эта таблица, а после печатается таблица абонентов в порядке индексов ключей, полученных после сортировки.
9. Подсчета времени, потраченного на сортировку. Засекается время, которое требуется на копирование таблиц элементов и ключей. Для каждого метода и алгоритма сортировки производится подсчет времени в следующем цикле: копирование, выполнение сортировки. Из полученного времени, деленного на количество итераций цикла, вычитается время, потраченное на копирование.

**Функции, использующиеся в программе**

void init\_list(void) – функция инициализации изначального списка

void init\_key\_list(void) – функция инициализации списка ключей

int find\_free(void) – функция нахождения первого свободного элемента, либо, если все слоты заняты, функция возвратит код ошибки

int find\_not\_free\_1(void) – функция нахождения первого не свободного слота в изначальном списке

int find\_not\_free\_2(void) - функция нахождения первого не свободного слота в списке ключей

int find\_free(void) – функция нахождения первого свободного слота в изначальном списке

void del(void) – функция удаления слота из списка

int count\_fields(const char \*str) – функция разбиения строки на элементы

void fscan\_list(void) – функция чтения таблицы из файла

void fprint\_list(void) – функция печати нашего списка

void scan\_key\_list(void) – функция сканирования данных для ключа

void print\_key\_list(void) – функция печати данных ключа

int type(void) – функция выбора типа информации (персональная/служебная)

void scan\_list(void) – функция добавления в таблица абонента, данные которого вводит пользователь

int compare\_list(const void \*p, const void \*q) – функция сравнения для сортировки для списка абонентов

void sort\_list(void) – функция сортировка списка методом qsort

int compare\_key\_list(const void \*p, const void \*q) – функция сравнения для сортировки для ключа

void sort\_key\_list(void) – функция сортировки ключа методом qsort

void swap(void \*x, void \*y, int size) – функция замены местами данных

void mysort(void \*a, int n, int sizem, int (\*compare)(const void \*, const void \*)) – функция сортировки методом шейк

void spec(void) – функция нахождения всех друзей, у которых день рождения в ближайшую неделю

void check\_ef(void) – функция cравнения эффективности обработки данных

void check\_sort(void) – функция сравнения сортировок

1. **Тестирование**

**Позитивные тесты:**

|  |  |  |
| --- | --- | --- |
| Входные данные | Описание теста | Результат |
| 1  А А 111111 адрес 1 2002 04 03 | Добавление элемента | Элемент с указанными полями добавляется в конец таблицы ключей и таблицы абонентов |
| 2 | Печать таблицы абонентов | Таблица элементов в читаемом формате |
| 3  1 | Удаление | Из таблицы абонентов удален элемент с индексом 1, из таблицы ключей удален ключ c индексом 1, а индексы всех остальных ключей уменьшены на 1 |
| 4  table5.txt | Считывание из файла таблицы | В таблицу абонентов добавлены записи из текстового файла |
| 5  table6.txt | Вывод таблицы в файл | Записывает таблицу в текстовый файл |
| 6 | Сортировка исходной таблицы по фамилии | Исходная таблица отсортирована по фамилиям абонентов. Таблица ключей переписана под исходную |
| 7 | Вывод таблицы ключей | Выводится таблица ключей |
| 8 | Сортировка с использованием таблицы ключей | Таблица ключей отсортирована по ключу, исходная таблица не изменена |
| 9 | Информация об эффективности обработки данных с помощью таблицы ключей | Исходная таблица и таблица ключей не изменены, на экран выведены результаты времени работы сортировок |
| 10 | Информация о времени выполнения сортировок | Исходная таблица и таблица ключей не изменены, на экран выведены результаты времени работы сортировок |
| 11 | Информация об абонентах в таблице, у которых день рождения в ближайшую неделю | Фамилии абонентов |

1. **Исследование**

Результаты времени сортировок для 100 элементов:

|  |  |  |
| --- | --- | --- |
|  | сортировка | память |
| table | 96723212 |  |
| keys | 22774608 | Всегда дополнительно 17% |

В данном случае сортировать таблицу ключей на 77% быстрее, чем исходную

Результаты времени сортировок для 500 элементов:

|  |  |  |
| --- | --- | --- |
|  | сортировка | память |
| table | 109447744 |  |
| keys | 34969408 | Всегда дополнительно 17% |

В данном случае сортировать таблицу ключей на 69% быстрее, чем исходную

Результаты времени сортировок для 1000 элементов:

|  |  |  |
| --- | --- | --- |
|  | сортировка | память |
| table | 124163354 |  |
| keys | 45271290 | Всегда дополнительно 17% |

В данном случае сортировать таблицу ключей на 64% быстрее, чем исходную

Если мы будем использовать ключ – телефон, то в таком случае дополнительная память будет больше на 3%.

1. **Выводы**

Благодаря использованию таблицы ключей мы получили сильный выигрыш во времени, в среднем составляет 70%. При обработке таблиц небольших размерностей выигрыш во времени получается максимальным. Но даже с увеличением количества элементов таблицы эффективность падает несущественно. Если взять ключ как поле ‘Фамилия’, то наш ключ будет занимать дополнительно 17%. Если же мы возьмем ключ как поле ‘Телефон’, то ключ будет занимать дополнительно 3%, это происходит из-за того, что в созданной нами структурой данных телефон – число типа int.

1. **Ответы на вопросы**

*Как выделяется память под вариантную часть записи?* – Выделяется область памяти, равная размеру максимального по длине поля вариантной части.

*Что будет, если в вариантную часть ввести данные, несоответствующие описанным?* – Тип данных в вариантной части при компиляции не проверяется, все проверки необходимо осуществлять самостоятельно.

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

*Что представляет собой таблица ключей, зачем она нужна?* – Таблица ключей содержит индекс элемента в исходной таблице и выбранный ключ. За счет небольших дополнительных затрат памяти позволяет ускорить процесс поиска и сортировки элементов исходной таблицы.

*В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?* – Использовать таблицу ключей эффективнее в случае большого количества записей или большого размера памяти, необходимой для хранения каждой записи. --

*Какие способы сортировки предпочтительнее для обработки таблиц и почему?* – При сортировке таблиц эффективнее использовать таблицу ключей. При этом быстрая сортировка работает быстрее, чем сортировка шейкер.

[tnikulshina@bmstu.ru](mailto:tnikulshina@bmstu.ru)

добавить описание функций