МГТУ им. Н. Э. Баумана

**Дисциплина типы и структуры данных**

**Лабораторный практикум №2**

**по теме: «Записи с вариантами. Обработка таблиц»**

**Вариант №2**

Работу выполнил:

студент группы ИУ7-35

Платонова Ольга

Работу проверил:

Москва, 2019 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.

***Условие задачи***

Требуется создать таблицу, содержащую не менее 40 записей с вариантной частью. Произвести поиск информации по вариантному полю. Упорядочить таблицу, по возрастанию ключей, используя:

а) исходную таблицу,

б) массив ключей,

применяя 2 разных алгоритма сортировки. Оценить эффективность этих алгоритмов. Обосновать выбор алгоритма сортировки.

***Техническое задание***

Имеются описания:

Type жилье = (дом, общежитие);

Данные:

Фамилия, имя, пол (м, ж), возраст, средний балл за сессию, дата поступления

адрес:

дом: (улица, №дома, №кв.);

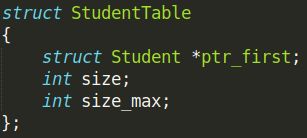
общежитие: (№общ., №комн.);

Вывести общий список студентов группы.

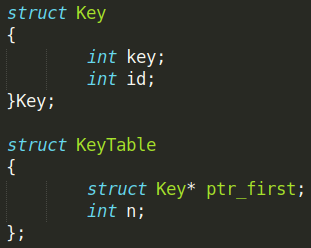
Вывести список студентов, живущих в общежитии указанного года поступления.

Программист должен рассмотреть возможный формат ввода данных, ошибки, связанные с ошибочным вводом, ошибочной работой с файлом. Должен осуществить две сортировки как исходной таблицы, так и таблицы ключей. Также необходимо рассмотреть аварийные ситуации, включающие неверное обращение к файлу, невозможность выгрузки таблицы в ОП, сортировку таблицы ключей, которую не удалось создать, поиск по ключу, выходящему за пределы.

***Структура данных***

Изображение выглядит как текст, сидит

Автоматически созданное описание



Для хранения данных об одном студенте воспользуемся структурой с объединением Student. Для хранения таблицы, содержащей всех студентов, воспользуемся структурой StudentTable. Аналогично будем хранить таблицу ключей. Поскольку ключ имеет тип int, пользователь задает любой из ключей типа int.

Исходные данные храним в файле. Для сортировки исходную информацию подгружаем в оперативную память, что позволяет осуществить повторную сортировку (таблица в файле остается несортированной).

Входные параметры храним в виде числа или массива символов.

***Описание алгоритма***

Изображение выглядит как текст

Автоматически созданное описание

1. Выгружаем таблицу из файла в ОП.

2. Осуществляем поиск по возрасту, проходя всю таблицу по одному разу. В случае совпадения выводим студента на экран.

3. Сортируем таблицу «пузырьком». Проходим таблицу (кол-во студентов – 1) раз, каждого студента сравниваем через компаратор.

4. Сортируем таблицу быстрой сортировкой, используя встроенную функцию qsort().

5. Формируем таблицу ключей, динамически выделяя память ключу каждого студента. Используем структуру {ключ, порядковый номер студента}

6.-7. Сортировка аналогично пунктам 3-4.

8. Проверяем, что таблица ключей создана. В цикле проходим исходную таблицу, осуществляем поиск по ключу. В случае совпадения выводим студента на экран.

9. Считываем из консоли строковые и числовые значения соответственно. Записываем значения в переменную struct Student, расширяем таблицу на размер переменной, увеличиваем число студентов на 1.

10. Считываем порядковый номер. Проходим таблицу. начиная с момента совпадения, сдвигаем всех студентов на единицу, освобождаем память последнего студента.

11. В цикле от 1 до 100 на каждой итерации загружаем таблицу, сортируем выбранным способом. В случае с ключами создаем таблицу ключей и сортируем ее. Полученное время каждой сортировки (сортировки + создания таблицы ключей) суммируем, а затем делим на количество итераций в цикле (100).

***Тесты***

1. Пустой файл

2. Поиск по году поступления: неверный формат ввода года, введенный год выходит за рамки граничных значений, ни одного студента не найдено.

3. Добавление студента: имя содержит иные символы, длина превышает допустимый размер, неверно указан пол, место проживания, возраст, средний балл, год поступления выходит за границы.

Во всех перечисленных случаях программа выдает соответствующее сообщение об ошибке на экран и завершает работу.

***Вывод***

На практике мы убедились, насколько целесообразно хранить тип «запись» с вариантной частью в случае большого количества этих записей. В одну структуру мы смогли уместить полную информацию об одном студенте, иначе нам пришлось бы хранить много массивов, что приводит к потере памяти.

Таблица 1.

Количество записей — 40. Количество итераций — 100.

|  |  |  |
| --- | --- | --- |
|  | Сортировка исходной таблицы | Сортировка с помощью таблицы ключей |
| «Пузырек» | 125315 | 44166 |
| «Быстрая сортировка» | 15769 | 3928 |

Отметим, что пузырек в обоих случаях занимает много времени, поскольку требуется неоднократный проход по всей таблице. Быстрая сортировка быстрее пузырька почти на 80%. В тоже время, быстрая сортировка с использованием таблицы ключей на 75% быстрее сортировки исходной таблицы. А «пузырек» по ключам – на 65%.

Что касается памяти, структура, содержащая информацию по ключам занимает 24 бита, в то время как исходная таблица 16. То есть, в случае сортировки по ключам программист проигрывает в памяти в 1.5 раза.

1. Как выделяется память под вариантную часть записи?

Память для записи с вариантами выделяется как сумма объемов памяти фиксированного размера и максимальный объем памяти поля вариантной часть.

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

На этапе компиляции контроль за соответствие типов в вариантной части не осуществляется, поэтому программист самостоятельно должен отслеживать корректность данных.

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

Программист.

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

Таблица ключей представляет собой массив, содержащий индекс элемента в исходной таблице и выбранный ключ. Таблица ключей позволяет существенно сократить время поиска, сортировки при больших размерах исходной таблицы.

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

В случае, если программист имеет ограниченно малую область памяти, или если в качестве ключа используется символьное поле записи, требующее длительной обработки удобнее обрабатывать записи в исходной таблице. В случае, когда исходная таблица очень большая, а ключи подобраны правильно, удобнее использовать массив ключей.

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

Для обработки таблиц с большим числом записей выгодней использовать такие сортировки, как «быстрая» или «куча», поскольку они не требуют многократного прохода по всей таблице, а выполняют все за один проход.