|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 №3**

**по теме: «обработка разреженных матриц»**

Выполнил студент: \_\_*Клименко Алексей Константинович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*фамилия, имя, отчество*

Группа: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*ИУ7-35Б*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил, к.п.н.: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*подпись, дата*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2020 г.*

**цель работы**

Реализовать алгоритмы обработки разреженных матриц, сравнить эффективность использования этих алгоритмов (по времени выполнения и по требуемой памяти) со стандартными алгоритмами обработки матриц при различном процентном заполнении матриц ненулевыми значениями и при различных размерах матриц.

**описание условия задачи**

Разработать программу умножения или сложения разреженных матриц. Предусмотреть возможность ввода данных, как с клавиатуры, так и использования заранее подготовленных данных. Матрицы хранятся и выводятся в форме трех объектов. Для небольших матриц можно дополнительно вывести матрицу в виде матрицы. Величина матриц - любая (допустим, 1000\*1000). Сравнить эффективность (по памяти и по времени выполнения) стандартных алгоритмов обработки матриц с алгоритмами обработки разреженных матриц при различной степени разреженности матриц и различной размерности матриц.

**указания к выполнению работы**

При разработке интерфейса программы следует предусмотреть:

* указание формата и диапазона вводимых данных,
* указание операции, производимой программой,
* наличие пояснений при выводе результата,
* указание формата выводимых данных
* возможность заполнения разреженных матриц вручную (даже при большой размерности, например, 1000\*1000) и автоматически с разным процентом разреженности.

При тестировании программы необходимо:

* проверить правильность ввода
* проконтролировать правильность вывода данных (т.е. их соответствиетребуемому формату);
* проверить правильность выполнения операций;
* обеспечить вывод сообщений при отсутствии входных данных («пустой ввод»);
* обеспечить вывод сообщений при нулевых результате или вывод нулевого результата при ненулевом входе;
* обеспечить возможность ввода данных и вывода результата как при малых матрицах, так и при больших (например, 1000 \* 1000).
* сравнить время выполнения стандартного алгоритма обработки матриц и алгоритма обработки разреженных матриц при различной заполненности матриц (от 1 элемента до того количества нулей (в %), при котором становится неэффективно использование алгоритма сокращенного умножения).
* сравнить объем требуемой памяти для реализации стандартного алгоритма обработки матриц и алгоритма обработки разреженных матриц при различном проценте заполнения матриц и при различном их размере.

Следует также протестировать программу при полной загрузке системы, то есть при полном заполнении матриц. Программа должна адекватно реагировать на неверный ввод, пустой ввод и выход за границы матрицы или вектора. Необходимо тщательно следить за освобождением динамической памяти (если она используется) при окончании программы.

**техническое задание**

**исходные данные**

Исходными данными являются матрицы, запрашиваемые у пользователя в программе с графическим интерфейсом, и считываемые из текстовых файлов при тестировании эффективности различных алгоритмов.

Текстовый файл с данными для ввода матрицы имеет следующую структуру: в первой строке находится единственное число — номер используемого фармата.

Если номер формата равен 1, то на следующей строке через пробел указывается размер матрицы (кол-во строк и столбцов), а далее в последующих строках указываются элементы матрицы.

Если же номер формата — 2, то на следующей строке помимо числа строк и столбцов указывается число ненулевых элементов (или 0, если ненулевых элементов нет). А далее для каждого ненулевого элемента на строке располагаются индексы строки и столбца и значение элемента.

*Примеры файлов, использующих разные форматы для представления одной и той же матрицы:*

|  |  |
| --- | --- |
| 1  3 7  0 0 0 -2 0 0 0  0 5 0 0 0 0 0  0 0 8 0 0 1 0 | 2  3 7 4  1 1 5  0 3 -2  2 2 8  2 5 1 |

Использование двух типов формата аргументировано тем, что для более плотных матриц бывает компактнее представить их непосредственно в матричном виде. Если представить себе, что нам нужно представить матрицу с процентом заполнения **P**, то для случаев **P > 33%** использовать координатный формат становится грамоздко и неудобно.

**результат**

Результатом работы программы с пользовательским интерфейсом является демонстрация работоспособности разработанных алгоритмов обработки разреженных матриц.

Для тестирующей производительность алгоритмов программы результатом считаются временные показатели, полученные экспериментально для двух разных реализаций функции обработки матриц, а также словестное подытоживание полученных результатов — сообщение о том, при каких обстоятельствах и на сколько один алгоритм эффективнее другого по времени.

**описание задачи, реализуемой программой**

Программа с пользовательским интерфейсом реализует 3 главные задачи — интерактивную демонстрацию корректности ввода матрицы, умножения матрицы на вектор и умножение матрицы на другую матрицу.

Программа тестирующая эффективность, что очевидно, тестирует два алгоритма и определяет их относительную эффективность.

**способы обращения к программе**

Для запуска программы в сборке выпуска предусматривается наличие исполняемого **\*.exe** файла, который запускает программу с пользовательским интерфейсом. Далее все должно быть интуитивно понятно.

Для запуска тестирования интерфейс еще не продуман до конца. Так что пока без тестирования эффективности.

**возможные аварийные ситуации и ошибки пользователя**

При работе программы с пользовательским интерфейсом наиболее очевидные ошибки пользователя, такие как ввод неположительных размеров матриц и векторов, незамедлительно пресекаются. (например, невозможно ввести отрицательное число в поле для ввода размера). Однако даже если что-то пойдет не так, в программе предусмотрены окошки с сообщениями об ошибках.

При вводе значений матрицы, выходящих за пределы допустимых, числа в текстовых полях для ввода обрезаются без уведомления об этом пользователя.

**Структуры данных**

Из условия задачи:

*Разреженная матрица хранится в форме 3-х обьектов:*

* *вектор* ***A*** *содержит значения ненулевых элементов*
* *вектор* ***IA*** *содержит номера строк для элементов вектора* ***A***
* *связный список* ***JA****, в элементе* ***Nk*** *которого находится номер компонент в* ***A*** *и* ***IA****, с которых начинается описание столбца* ***Nk*** *матрицы* ***A***

Реализация структуры разреженной матрицы в решении:

**typedef int mat\_elem\_t;**

**typedef struct sparse\_matrix\_t**

**{**

**size\_t rows\_size;** // кол-во строк матрицы

**size\_t cols\_size;** // кол-во столбцов матрицы

**size\_t nonzero\_size;**  // кол-во ненулевых элементов

**size\_t \_\_alloc\_nz\_sz;** // кол-во байт, выделенных для ненулевых эл-тов

**size\_t \_\_alloc\_cl\_sz;** // кол-во байт, выделенное для массива JA

**size\_t \*cols;** // JA — массив индексов эл-тов для столбцов

**size\_t \*rows;** // IA — массив индексов строк соотв-щих ненулевых эл-тов

**mat\_elem\_t \*nonzero\_array;** // A — массив ненулевых эл-тов

**} sparse\_matrix\_t;**

Вычислим объём памяти необходимый для хранения матрицы размером (**M, N**) и с **K** ненулевыми элементами:

**sizeof(sparse\_matrix) ==** **5 \* 4Б + 3 \* 8Б (+ N \* 4Б) + 2 \* K \* (4Б + 4Б)**

**== (44 + 4N + 16K) Б**

А теперь для сравнения приведём расчёт объема занимаемой памяти для обычного метода хранения матрицы размером (**N+D, N**), где **D** – положительное число (при обратном — можно хранить матрицу как транспонированную):

**sizeof(dense\_matrix) == 8Б + (N + D) \* N \* 4Б == 8Б + N2 \* 4Б + N \* D \* 4Б**

Видно, что зависимость объема памяти для хранения плотной матрицы имеет квадратичную зависимость от **N**, в то время как объем памяти разреженной — линейную.

**Множетво допустимых значений**

Для данной СД допустимыми являются все возможные для хранения данные, а именно:

количество строк и столбцов — от 1 до 4 294 967 295 включительно

количество ненулевых элементов матрицы — от 1 до 4 294 967 295 включительно

**Набор функций**

Над структурой, представляющей собой запись в таблице, допустимо совершать следующие действия:

* Инициализация структуры квартиры нулевыми значениями
* Освобождение памяти, выделенное для данной структуры
* Преобразование информации из структуры в строку и наоборот
* Печать данных структуры в файл или на экран
* Считывание данных структуры из файла или консоли

Над таблицей данных разрешено выполнять следующие действия:

* Инициализация таблицы нулевым количеством квартир
* Освобождение памяти, выделенной для данной таблицы
* Сортировка таблицы по одному из предложенных невариантных полей: адрес, площадь комнат, количество комнат
* Добавление новой записи в конец таблицы
* Считывание данных таблицы из файла
* Вывод данных таблицы на экран консоли
* Создание копии таблицы
* Инициализация массива ключей для сортировки

**Описание алгоритма обработки данных**

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

Для пункта (а), в котором таблица хранит в себе только массив структур, обе сортировки будут оперировать этим массивом данных непосредственно, т. е. будут производить *обмен элементами на месте*.

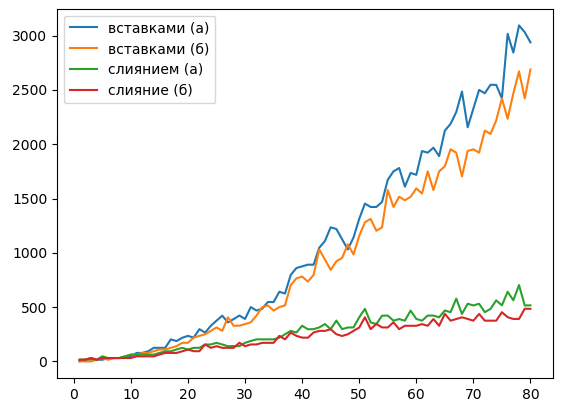
Для пункта (б), в котором помимо массива с данными будет и массив указателей на данные, сортироваться будет массив указателей, а это значит, что объем обрабатываемой при сортировке памяти будет меньше.

Приведём сложность упомянутых алгоритмов:

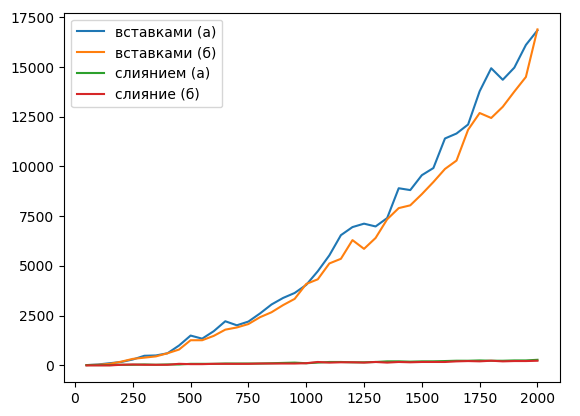
Сортировка вставками: **O(N2)**

Сортировка слиянием: **O(N\*log(N))**

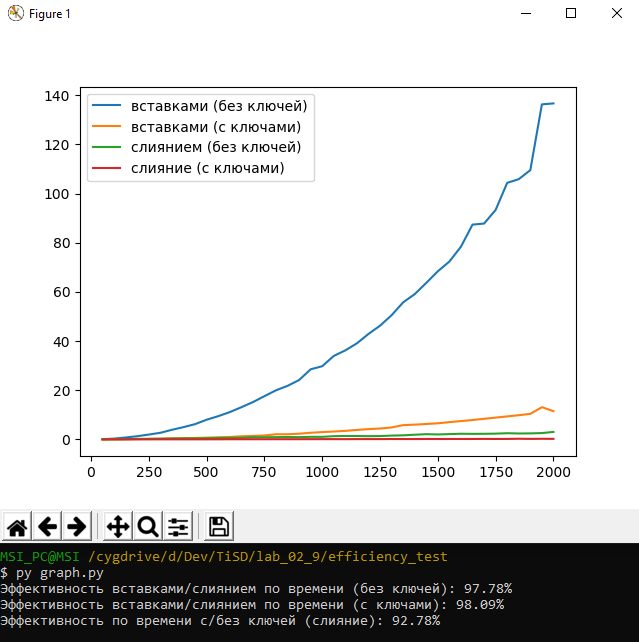
Результаты тестирования в виде графика (с учетом времени на инициализацию новой таблицы для сортировки):



Для большего числа элементов:



Без учёта времени инициализации копии таблицы графики преобретают иной вид:



Расчеты, проведённые в программе MathCAD для наиболее точного расчёта относительной эффективности двух алгоритмов при использовании дополнительной таблицы ключей:

Отностительная эффективность по времени (с учётом инициализаций):

**двух медленных сортировок: 3.1%**

**двух быстрых сортировок: 14.5%**

**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 оперции, 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 | Ошибки при сортировке таблицы | in.txt  2  *<invalid vals>* | Отображение сообщения об ошибке и переход в меню |
| 10 | 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 |
| 11 | 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 **14.5%** для алгоритма сортировки слиянием). Однако уменьшение скорости оборачивается увеличением необходимого для работы объема памяти (приблизительно на **14.28%**).

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. Для облегчения отслеживания текущей интерпретации данных допускается использование дополнительного флагового поля, которое будет показывать, какой тип данных используется в данный момент. (Конечно же, это поле не должно храниться внутри вариантного поля).

*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, вставка 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.