|  |  |
| --- | --- |
| **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 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. 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.

**техническое задание**

**исходные данные**

Исходными данными являются матрицы, запрашиваемые у пользователя в программе, и генерируемые автоматически при тестировании эффективности различных алгоритмов.

Ввод матрицы с клавиатуры имеет следующий формат: 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.

Если же номер формата – 3, то на следующей строке через пробел указывается размер матрицы (кол-во строк и столбцов). На следующей строке необходимо указать вещественное число (можно и целое) в интервале от 0 до 100 не включительно – процент автоматического заполнения матриц. Минимальный и максимальный элементы устанавливаются в -100 и 100 соответственно.

*Примеры ввода, с использованием разных форматов 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.

*Пример ввода для автоматического заполнения матрицы размером 100 на 200 элементов:*

|  |  |
| --- | --- |
| 3  100 200  24.3 | 3  100 200  10 |

**результат**

Результатом работы программы является вычисление произведения матриц.

Для тестирующей производительность алгоритмов программы результатом считаются показатели эффективности, полученные экспериментально для двух разных реализаций функции обработки матриц.

**описание задачи, реализуемой программой**

Умножение матрицы на вектор и умножение матрицы на другую матрицу, а также тестирование двух алгоритмов и определение их относительной эффективности.

**способы обращения к программе**

Для запуска программы в сборке выпуска предусматривается наличие исполняемого **app.exe** файла, который запускает программу. После запуска пользователь выбирает команды из меню и вводит соответствующие им цифры.

**возможные аварийные ситуации и ошибки пользователя**

При вводе неверных данных для текущей опции меню выполнение данной опции прекращается и пользователь возвращается в главное меню программы.

**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;**

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

**sizeof(sparse\_matrix) == (44 + 4N + 16K)Б**

А теперь для сравнения приведём расчёт объема занимаемой памяти для обычного метода хранения матрицы размером (**N, N**):

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

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

**Набор функций**

**// NULL-матрица.**

**sparse\_matrix\_t sp\_null\_matrix(void);**

**// Создание пустой матрицы с заданным кол-вом строк и столбцов.**

**sparse\_matrix\_t sp\_create(uint32\_t rows, uint32\_t cols);**

**// Перераспределение элементов под заданный размер.**

**int sp\_recreate(sparse\_matrix\_t \*matrix, uint32\_t rows, uint32\_t cols);**

**// Очищение памяти и обнуление матрицы.**

**void sp\_free(sparse\_matrix\_t \*matrix);**

**// Создание глубокой копии матрицы.**

**sparse\_matrix\_t sp\_copy(const sparse\_matrix\_t \*matrix);**

**// Проверка матрицы на корректность.**

**bool sp\_mat\_is\_null(const sparse\_matrix\_t \*matrix);**

**// Считывание элемента по индексам строки и столбца.**

**mat\_elem\_t sp\_get(const sparse\_matrix\_t \*matrix, uint32\_t row, uint32\_t col);**

**// Запись элемента по индексам строки и столбца.**

**void sp\_set(sparse\_matrix\_t \*matrix, uint32\_t row, uint32\_t col, mat\_elem\_t value);**

**// Сжатие матрицы, путём удаления нулевых элементов.**

**void sp\_compress(sparse\_matrix\_t \*matrix);**

**// Обнуление матрицы.**

**void sp\_clear(sparse\_matrix\_t \*matrix);**

**// Заполнение случайными числами.**

**void sp\_randomize(sparse\_matrix\_t \*matrix, float nz\_percent);**

**// Транспонирует матрицу.**

**void sp\_transpose(sparse\_matrix\_t \*matrix);**

**// Умножает две матрицы, вторая из которых транспонирована.**

**int sp\_mult\_matrix(const sparse\_matrix\_t \*matrix\_1, const sparse\_matrix\_t \*matrix\_2,**

**sparse\_matrix\_t \*out);**

**// Печатает подробную информацию о структуре матрицы.**

**void sp\_print\_info(const sparse\_matrix\_t \*matrix);**

**// Печатает матрицу в виде матрицы.**

**void sp\_print(const sparse\_matrix\_t \*matrix);**

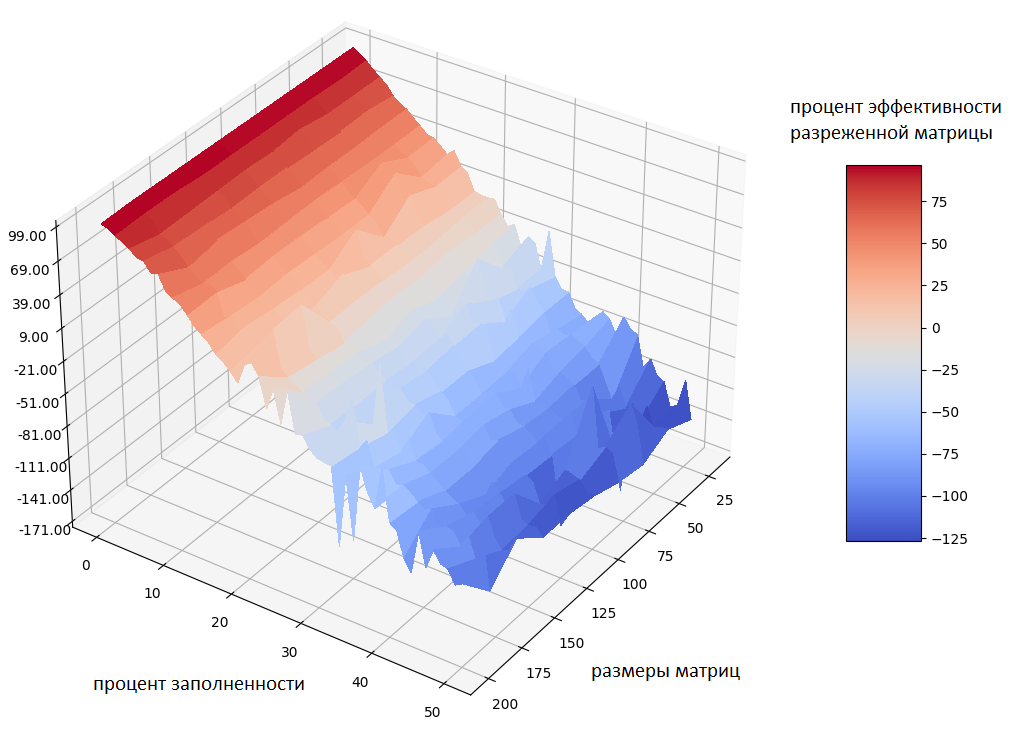
**Описание алгоритма обработки данных**

Ключевая идея алгоритма умножения двух разреженных матриц состоит в том, чтобы умножать только ненулевые элементы матриц.

Для упрощения и ускорения работы алгоритма необходимо предварительно транспонировать вторую матрицу.

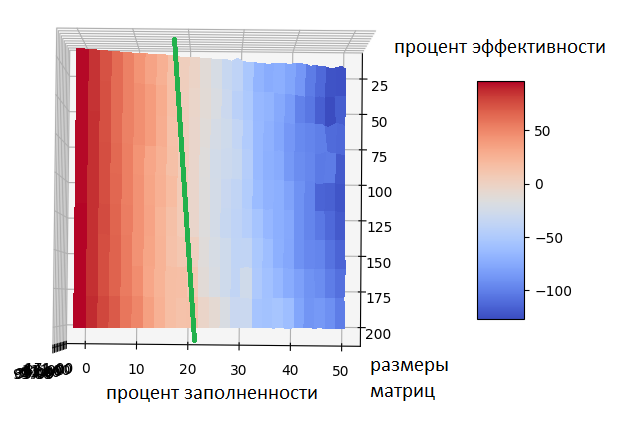
Далее - пройти по каждому столбцу в перемножаемых матрицах и рассчитать произведения элементов только на тех позициях, где оба элемента ненулевые. Полученные произведения записать в соответствующую позицию в результирующей матрице.

Результаты тестирования скорости в виде графика (без учёта времени на транспонирование матрицы):

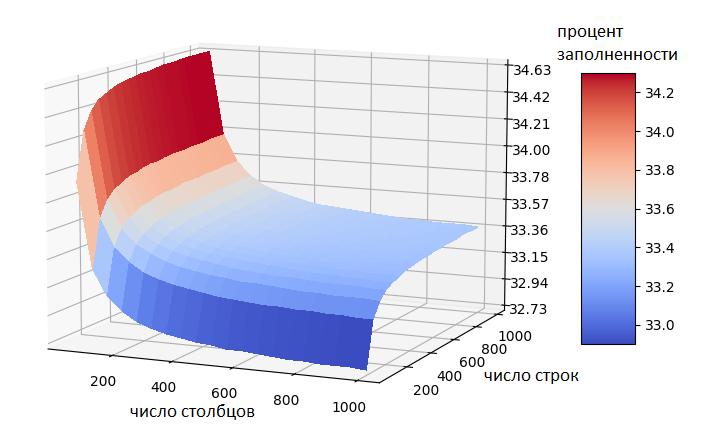


Как видно, при достижении отметки в ~20% оба алгоритма умножения становятся одинаково эффективными. При заполненности матрицы менее 20% эффективнее разреженный алгоритм, а при заполнении матрицы более чем на 20% более эффективным становится обычный алгоритм умножения.

Также можно заметить, что с ростом размеров матриц, разреженная версия матрицы обрабатывается чуть-чуть хуже.

****

Сравнение эффективности по памяти для переменных размеров матриц:

 По данным тестирования можно утверждать, что выбранное решение хранения данных разреженных матриц является эффективнее обычного метода в среднем при проценте заполненности матрицы менее чем 33-34%.

При большем проценте заполненности хранить матрицу в разреженной форме становится менее эффективно.

**набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Описание теста** | **Входные данные** | **Выходные данные** |
| 1 | неверный выбор опции меню | a | Повторное отображение главного меню |
| 2 | Неверный формат при вводе матрицы | 1  4 | Сообщение об ошибке. Переход в главное меню |
| 3 | Неверный ввод данных матрицы | 1  1  one three | Отображение сообщения об ошибке. Переход в главное меню |
| 4 | Неверный ввод размеров матриц при авто тестировании | 3  10  -20 | Сообщение об ошибке и переход в главное меню |
| 5 | Указание неверного процента | 4  -10 | Сообщение об ошибке и переход в главное меню |
| 6 | Умножение матрицы на вектор | 1  <ввод матрицы>  <ввод вектора> | Вывод введённых данных и результата умножения |
| 7 | Умножение матрицы на матрицу | 2  <ввод матрицы>  <ввод матрицы> | Вывод введённых данных и результата умножения |
| 8 | Автоматическое заполнение по заданным размерам матриц | in.txt  3  100  120  150 | Отображение таблицы эффективности для заданной размерности матриц |
| 9 | Автоматическое заполнение по заданному проценту | 4  20 | Отображение таблицы эффективности для данного процента заполненности матриц |
| 10 | Сравнительная характеристика | 5 | Вывод на экран таблицу со значениями эффективности как функции от двух параметров |
| 11 | Графическое отображение | 6 | Выводит на экран окошко с графиком указанным в алгоритмической части данного отчета |

**Выводы по проделанной работе**

В ходе работы я познакомился с формой хранения разреженных матриц и самостоятельно реализовал алгоритмы по их обработке.

Разреженная матрица показывает себя с лучшей стороны при не слишком больших размерах матриц и при проценте заполнения ненулевыми элементами не более 20%. При данных значениях алгоритмы обработки разреженных матриц работают эффективнее как по времени так и по памяти в сравнении с классическими алгоритмами.

Однако как только процент заполненности превышает значение в 20%, а размеры матриц становятся довольно большими, обрабатывать разреженные матрицы становится неэффективно по времени, но все ещё эффективно по памяти.

Но уже после 34% заполненности ненулевыми элементами от разреженных матриц в среднем нет никакой пользы. Они начинают уступать плотным матрицам как по времени, так и по памяти.

**Контрольные вопросы**

1. *Что такое разреженная матрица, какие схемы хранения таких матриц Вы знаете?*

Разреженная матрица это структура данных, в которой хранятся только ненулевые элементы матрицы, и информация об их позиции в матрице. Такой информацией может быть, например явное указание строки и столбца (координатная форма), а может быть только индекс строки, но вместе с ненулевыми элементами тогда хранится список индексов элементов с которых начинается тот или иной столбец в матрице (Йельский формат).

Также, в ряде случаев работа происходит только с симетричными матрицами. Тогда нам достаточно хранить только половину от всех ненулевых элементов матрицы.

Существует и множество других форматов, которые разрабатывались для определённой конфигурации матриц, и подходящие для очень узкого круга задач, например, можно хранить матрицу блоками.

2. *Каким образом и сколько памяти выделяется под хранение разреженной и обычной матрицы?*

Для хранения матрицы в обычном представлении память выделяется сразу под все элементы матрицы.

Для хранения разреженной матрицы память выделяется по мере необходимости и только для ненулевых элементов матрицы.

При этом, для хранения одного элемента в разреженном формате требуется больше памяти, чем в обычном. Тем не менее, при малой заполненности матрицы хранение только ненулевых элементов становится выгоднее.

3. *Каков принцип обработки разреженной матрицы?*

Принцип обработки разреженной матрицы заключается в том, чтобы обходить только ненулевые элементы матрицы, а не все возможные, тем самым облегчая сложность алгоритма с O(N2) до O(K) где N - размерность матрицы, а K - число ненулевых элементов в ней.

4. *В каком случае для матриц эффективнее применять стандартные алгоритмы обработки матриц? От чего это зависит?*

Это зависит от выбранного формата хранения разреженной матрицы, а также в неменьшей степени от процента заполненности матрицы. Чем он меньше (разреженность выше), тем эффективнее использование алгоритмов, работающих с разреженными матрицами.

Однако, если процент заполненности матрицы превосходит 20-30% то стандартные алгоритмы обработки оказываются не только проще, но и эффективнее нестандартных.