Отчёт по лабораторной работе №3

Вариант 5

Выполнил: Соколов Ефим Маркович (группа ИУ7-33Б)

### ***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.

Техническое задание

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов:

- вектор A содержит значения ненулевых элементов;

- вектор JA содержит номера столбцов для элементов вектора A;

- вектор IA, в элементе Nk которого находится номер компонент

в A и JA, с которых начинается описание строки Nk матрицы A.

1. Смоделировать операцию умножения вектора-строки и матрицы,

хранящихся в этой форме, с получением результата в той же форме.

2. Произвести операцию умножения, применяя стандартный алгоритм работы с

матрицами.

3. Сравнить время выполнения операций и объем памяти при использовании

этих 2-х алгоритмов при различном проценте заполнения матриц.

Типы входных и выходных данных

***Входные данные:***

* Количество строк и столбцов
* Вектор-строка, заданная поэлементно или с заданием отдельных элементов (ввод “по координатам”)
* Матрица, заданная поэлементно или с заданием отдельных значений определённых элементов (ввод “по координатам”)
* Процент заполнения ненулевыми элементами сгенерированной матрицы (и сами случайные элементы)

**Выходные данные:**

* Результат произведения вектора-строки на матрицу в определённом формате
* Результаты замеров времени выполнения двух алгоритмов
* Результат замеров затраченной памяти для двух алгоритмов

***Возможные аварийные ситуации:***

* Некорректный ввод данных.

Структуры данных

Для представления матриц в стандартном и разреженном по строкам формате используются следующие структуры данных:

#define MAX\_DIME 150 // Максимальная размерность матрицы

***Матрица в разреженном представлении:***

typedef struct

{

int csr\_values[MAX\_DIME \* MAX\_DIME]; // массив всех ненулевых элементов

int col\_idx[MAX\_DIME \* MAX\_DIME]; // столбцы всех ненулевых элементов

// Массив указателей на элементы, с которых начинаются новые строки

int row\_ptr[MAX\_DIME + 1];

} sparseMatrix\_t;

***Матрица в стандартном представлении:***

typedef struct

{

int matrix[MAX\_DIME][MAX\_DIME]; // Элементы матрицы

int rows; // количество строк матрицы

int cols; // количество столбцов матрицы

} stdMatrix\_t;

Алгоритм умножения

***Для стандартного умножения:***

Реализован “классический” (насколько здесь уместно сказать) алгоритм умножения матриц “строки на столбец”.

***Для разреженного умножения:***  
Матрица транспонируется и передаётся в функцию, перемножающую строку на матрицу построчно.  
Главный цикл происходит по количеству ненулевых элементов матрицы. Пробегом по массиву столбцов (JA) вектора мы находим значения, равные по столбцу матрицы и добавляем произведения их элементов в сумму, если такие нашлись. Далее, пробегом по размеру AI массива матрицы, находим элементы матрицы, являющиеся элементами следующей строки, и по этим элементам (“индексам-сигнализаторам”) добавляем сумму в итоговый массив A и JA. Если сумма является ненулевой – мы вносим её, иначе – пропускаем столбец итогового вектора. Также на этом моменте проверяется, является ли строка матрицы пустой. Если пустая – пропуск строки.

Оценка эффективности

Время перемножения\* (clock\_t):

***10% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***10*** | ***110*** |
| *100 на 100 элементов* | ***1150*** | ***10852*** |
| *150 на 150 элементов* | ***2829*** | ***24099*** |

***20% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***41*** | ***110*** |
| *100 на 100 элементов* | ***3979*** | ***10809*** |
| *150 на 150 элементов* | ***10410*** | ***25194*** |

***30% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***90*** | ***110*** |
| *100 на 100 элементов* | ***9469*** | ***10820*** |
| *150 на 150 элементов* | ***22800*** | ***24267*** |

***40% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***150*** | ***110*** |
| *100 на 100 элементов* | ***15917*** | ***11030*** |
| *150 на 150 элементов* | ***49608*** | ***23711*** |

***50% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***230*** | ***120*** |
| *100 на 100 элементов* | ***25079*** | ***10932*** |
| *150 на 150 элементов* | ***63449*** | ***24201*** |

***100% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***930*** | ***110*** |
| *100 на 100 элементов* | ***101674*** | ***10830*** |
| *150 на 150 элементов* | ***278363*** | ***24867*** |

\*время транспонирование матрицы не учитывается

Занимаемая память (байт):

***10% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***120*** | ***400*** |
| *100 на 100 элементов* | ***12000*** | ***40000*** |
| *150 на 150 элементов* | ***27000*** | ***60000*** |

***20% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***240*** | ***400*** |
| *100 на 100 элементов* | ***24000*** | ***40000*** |
| *150 на 150 элементов* | ***54000*** | ***60000*** |
|  |  |  |

***30% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***360*** | ***400*** |
| *100 на 100 элементов* | ***36000*** | ***40000*** |
| *150 на 150 элементов* | ***81000*** | ***60000*** |
|  |  |  |

***40% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***480*** | ***400*** |
| *100 на 100 элементов* | ***48000*** | ***40000*** |
| *150 на 150 элементов* | ***108000*** | ***60000*** |
|  |  |  |

***50% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***600*** | ***400*** |
| *100 на 100 элементов* | ***60000*** | ***40000*** |
| *150 на 150 элементов* | ***135000*** | ***60000*** |

***100% заполнения***

|  |  |  |
| --- | --- | --- |
| *Размеры матрицы* | *Разреженный формат* | *Классический формат* |
| *10 на 10 элементов* | ***1200*** | ***400*** |
| *100 на 100 элементов* | ***120000*** | ***40000*** |
| *150 на 150 элементов* | ***270000*** | ***60000*** |
|  |  |  |

Вывод

Использование алгоритмов хранения и обработки разреженных матриц по строкам (CSR) выгодно при степени заполненности до ~30%. В таком случае алгоритм выигрывает как по скорости, так и по памяти. При заполненности более 30% алгоритм для разреженного умножения начинает проигрывать и по памяти, и по скорости. Следовательно, после 30% смысла использовать разреженную матрицу нет, так как классический способ хранения даст лучший результат по скорости и меньшие затраты по памяти.

Важно отметить, что вывод этот применим только к моему случаю – умножению вектора-строки на матрицу.

Ответы на контрольные вопросы

1. ***Что такое разреженная матрица, какие способы хранения вы знаете?***

Разреженная матрица – это матрица, содержащая большое количество нулей. Способы хранения: связная схема хранения, строчный формат, линейный связный список, кольцевой связный список, двунаправленные стеки и очереди.

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

Под обычную матрицу выделяет N \* M ячеек памяти, где N – строки, а M – столбцы. Для разреженной матрицы – зависит от способа. В случае разреженного формата, требуется (2 \* K + N + 1) ячеек памяти, где K – количество ненулевых элементов.

1. ***Каков принцип обработки разреженной матрицы?***

Алгоритмы обработки разреженных матриц предусматривают действие только с ненулевыми элементами, и, таким образом, количество операций будет пропорционально количеству ненулевых элементов.

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

Стандартные алгоритмы обработки матриц эффективнее применять при большом количестве ненулевых элементов (от 30% для моего случая). Также, если в программе происходит действия именно над матрицами, может случиться так, что при проигрыше по памяти мы всё ещё будем иметь выигрыш по времени. Так, если нам не так важна память, но важна скорость – можно использовать строчный формат. Иначе же, если нам важна память, но не скорость, то мы можем использовать на малых разреженностях классический формат, а в больших разреженностях – строчный форма.