|  |  |
| --- | --- |
|  | **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»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 ПО ДИСЦИПЛИНЕ ТИПЫ И СТРУКТУРЫ ДАННЫХ**

Студент **Паншин Сергей Константинович**

Группа **ИУ7-33Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

|  |  |
| --- | --- |
| Студент | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Паншин С. К.** |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Никульшина Т. А.** |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Барышникова М. Ю.** |

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2023 г.*

**Условие задачи**

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 матрицы и вектора-столбца, хранящегося в форме вектора A и вектора, содержащего номера строк ненулевых элементов, с получением результата в форме хранения вектора-столбца.

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.

**Техническое задание**

Разработать 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.

**Входные данные:**

Пункт меню (число от 0 до 6):

Menu:

Menu:

1) Read matrix

2) Read vector

3) Print matrix

4) Print vector

5) Multiplication matrix and vector

6) Print results of multiplication using various matrix types

0) Quit program

Также размеры матрицы и ее элементы (если не автоматическое заполнение).

**Выходные данные:**

Данные матрицы, данные вектора, результат перемножения матрицы на вектор, сравнение эффективности алгоритмов.

**Возможные аварийные ситуации:**

Некорректный ввод: пункта меню, данные матрицы и вектора.

**Способ обращения к программе**

В папке с программой запустить команду *make run*.

**Структуры данных**

// Структура разреженной матрицы

typedef struct

{

size\_t n;

size\_t m;

size\_t nums\_count;

int \*a;

size\_t \*ja;

size\_t \*ia;

} sparse\_matrix\_t;

// Структура обычной матрицы

typedef struct std\_matr

{

int \*\*matrix;

size\_t n;

size\_t m;

} matrix\_t;

// Структура вектора столбца

typedef struct

{

size\_t row\_count;

int \*a;

size\_t \*ia;

size\_t len;

} vector\_t;

Используемые константы

#define ITER\_COUNT\_TIME 50

#define MAX\_PRINT\_MAT 10

#define ERROR\_INVALID\_NUM 1

#define ERROR\_INVALID\_ROW\_COUNT 2

#define ERROR\_INVALID\_COL\_COUNT 3

#define ERROR\_MEMORY\_ALLOC 4

#define ERROR\_INVALID\_NUMS\_COUNT 5

#define ERROR\_SIZE\_COHERENCE 6

#define ERROR\_WRONG\_MENU\_ITEM 7

Интерфейсы программы

// умножение матрицы на вектор

int matrix\_mul\_vec(matrix\_t \*mat1, vector\_t \*vec, vector\_t \*dst);

// умножение разреженной матрицы на вектор

void sparse\_matrix\_mul\_vec(sparse\_matrix\_t \*mat, vector\_t \*vec, vector\_t \*vec\_res);

// чтение матрицы

int read\_matrix(matrix\_t \*mat, size\_t \*nums\_count, int is\_rand, FILE \*in);

// печать матрицы

void print\_matrix(matrix\_t \*mat, FILE \*out);

// рандомное заполнение матрицы

void matrix\_random\_fill(matrix\_t \*mat, size\_t \*nums\_count);

// выделение памяти под матрицу

int matrix\_alloc(matrix\_t \*const matrix);

void free\_matrix(matrix\_t \*const matrix); // освобождение памяти

// матрица в csr формат хранения

int matrix\_to\_sparse\_matrix(matrix\_t \*mat, sparse\_matrix\_t \*s\_mat, size\_t \*nums\_count);

// csr формат хранения матрицы в обычную матрицу

void sparse\_matrix\_to\_matrix(sparse\_matrix\_t \*s\_mat, matrix\_t \*mat);

// вывод csr формата хранения матрицы

void print\_sparse\_matrix(sparse\_matrix\_t \*s\_mat, FILE \*out);

// выделение памяти под csr формата хранения матрицы

int sparse\_matrix\_alloc(sparse\_matrix\_t \*s\_mat);

// освобождение памяти из под csr формата хранения матрицы

void free\_sparse\_matrix(sparse\_matrix\_t \*s\_mat);

// выделение памяти под вектор

int vector\_alloc(vector\_t \*vec);

// освобождения памяти из под вектор

void free\_vector(vector\_t \*vec);

//чтение вектора

int read\_vector(vector\_t \*vec, int is\_rand, FILE \*in);

// печать вектора

void print\_vector(vector\_t \*vec, FILE \*out);

// заполение вектора рандомными числами

void vector\_random\_fill(vector\_t \*vec);

// получение элемента вектора

int get\_element(vector\_t \*vec, size\_t i);

**Замеры**

Производиться 50 итераций при замерах.

Время выполнения умножения в наносекундах:

10% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрицы** | **Разреженная матрица** |
| 10x10 | 7780 | 840 |
| 100x100 | 1046640 | 90140 |
| 500x500 | 106744020 | 10226220 |

25% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрицы** | **Разреженная матрица** |
| 10x10 | 1320 | 260 |
| 100x100 | 963400 | 215960 |
| 500x500 | 106435300 | 23551740 |

50% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрицы** | **Разреженная матрица** |
| 10x10 | 1260 | 680 |
| 100x100 | 957340 | 386440 |
| 500x500 | 106687240 | 42029360 |

85% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрицы** | **Разреженная матрица** |
| 10x10 | 1320 | 1080 |
| 100x100 | 969800 | 544960 |
| 500x500 | 106887700 | 61165060 |

100% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрицы** | **Разреженная матрица** |
| 10x10 | 1240 | 940 |
| 100x100 | 962480 | 606480 |
| 500x500 | 106860900 | 67618880 |

Объем занимаемой памяти в байтах:

10% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрица** | **Разреженная матрица** |
| 10x10 | 400 | 208 |
| 100x100 | 40000 | 12808 |
| 500x500 | 1000000 | 304008 |

25% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрица** | **Разреженная матрица** |
| 10x10 | 400 | 388 |
| 100x100 | 40000 | 30808 |
| 500x500 | 1000000 | 754008 |

50% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрица** | **Разреженная матрица** |
| 10x10 | 400 | 688 |
| 100x100 | 40000 | 60808 |
| 500x500 | 1000000 | 1504008 |

85% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрица** | **Разреженная матрица** |
| 10x10 | 400 | 1108 |
| 100x100 | 40000 | 102808 |
| 500x500 | 1000000 | 2554008 |

100% заполнения:

|  |  |  |
| --- | --- | --- |
| **Размер матрицы (n x m)** | **Обычная матрицы** | **Разреженная матрица** |
| 10x10 | 400 | 1288 |
| 100x100 | 40000 | 120808 |
| 500x500 | 1000000 | 3004008 |

**Выводы по проделанной работе**

Использование алгоритмов хранения и обработки разреженных матриц выгодно при большом количестве элементов, примерно до 50% заполненности матриц. В таком случае, алгоритм выигрывает, как и в размерах занимаемой памяти, так и в скорости обработки. Но при заполненности более чем 50%, алгоритм обработки и хранения разреженных матриц начинает проигрывать по памяти, но все еще выигрывает во времени. Даже при 100% заполненности матриц этот алгоритм выигрывает во времени у обычного, но занимает места в памяти больше.

**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.

*2. 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 строчного формата, требуется 2 \* K + N + 1 ячеек памяти, где K – количество ненулевых элементов, N – количество строк.

*3. 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.

*4. 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 50%). Стоит отметить, что если не так важна память, занимаемая матрицами, но важно время, то лучше использовать алгоритм обработки разреженных матриц, так как он, хоть и немного, но выигрывает во времени даже на больших % заполнения матриц (80% - 100%)