**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%)