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

Обработка разреженных матриц

Студент **Павлов Д. В.**

Группа **ИУ7-33Б**

Вариант 3

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

Студент _____**Павлов Д. В.**

Преподаватель _____**Барышникова М. Ю.**

2024

1. Описание условия 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.

Вектор-столбец хранится в 2х объектах:
- вектор B, содержащий значения ненулевых элементов
- вектор IB, параллельный вектору B, содержащий индексы ненулевых элементов

1.Смоделировать операцию умножения матрицы и вектора-столбца, хранящихся в приведенных выше форматах, с получением результата в форме хранения вектора.
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.

2. Описание технического задания.

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.

Входные данные.

Целое число: пункт меню (от 0 до 6)
Меню:

```
Программа для умножения матрицы на вектор-столбец.
Пункты меню:
1) Ввести матрицу
2) Ввести вектор-столбец
3) Вывести матрицу
4) Вывести вектор-столбец
5) Умножить матрицу на вектор-столбец
6) Получить результат сравнения алгоритмов при различном проценте заполнения
0) Завершить работу программы
```

Целое число: размеры матрицы, элементы матрицы.

Выходные данные.

Успех или ошибка выполнения какого-либо пункта, матрица в стандартном формате, матрица в форме CSR, таблицы сравнения алгоритмов умножения матриц при разных процентах заполнения.

Способ обращения к программе.

Запускается через кнопку «run» в среде разработки «Clion»

Возможные аварийные ситуации и ошибки пользователя.

Некорректный ввод данных. Ошибка выделения памяти.

3. Описание внутренних структур данных.

Структура для обычной матрицы.

```
typedef struct
{
    int **data; //Элементы массива
    size_t rows; // Количество строк
    size_t cols; // Количество столбцов
} matrix_t;
```

Структура для разреженной матрицы.

```
typedef struct
{
    size_t rows; // Кол-во строк
    size_t cols; // Кол-во столбцов
    size_t num_non_zeros; // Кол-во ненулевых элементов

    int *a; // Вектор значения ненулевых элементов
    size_t *ja; // Номера столбцов вектора a
    size_t *ia; // Вектор индексов начала строк
} sparse_matrix_t;
```

Структура для вектора-столбца.

```
typedef struct
{
    int *data; // вектор B, содержащий значения ненулевых
элементов
    size_t *ib; // вектор IB, содержащий индексы ненулевых
элементов
    size_t num_non_zeros; // Количество ненулевых элементов
```

```
    size_t rows; // Максимальное количество элементов
} vector_t;
```

Интерфейс для работы со структурами.

Для работы с матрицами

```
// Выделение памяти под стандартную матрицу
int matrix_alloc(matrix_t *matrix);

// Очищение памяти для стандартной матрицы
void matrix_free(matrix_t *matrix);

//Выделение памяти под разреженную матрицу
int sparse_matrix_alloc(sparse_matrix_t *sparse_matrix);

// Очищение памяти для разреженной матрицы
void sparse_matrix_free(sparse_matrix_t *sparse_matrix);

// Чтение размера матриц
void read_matrix_size(size_t *rows, size_t *cols, size_t
*num_non_zeros, int is_vector_readed, size_t vector_rows);

// Заполнение матрицы, путем ввода координатным способом
void fill_matrix_with_coords(matrix_t *matrix, const size_t
*num_non_zeros);

// Заполнение матрицы рандомными элементами в рандомные позиции,
соответствующий кол-ву элементов
void fill_matrix_with_rand_elems(matrix_t *matrix, size_t const
*num_non_zeros);

// Перевод из стандартной матрицы в разреженную
void std_matrix_to_sparse(matrix_t matrix, sparse_matrix_t
*sparse_matrix);

// Перевод из разреженной в стандартную:
void sparse_matrix_to_std(sparse_matrix_t sparse_matrix, matrix_t
*matrix);

// Вывод матрицы в стандартном виде
void print_matrix(matrix_t matrix);

// Вывод матрицы в разреженном виде
void print_sparse_matrix(sparse_matrix_t sparse_matrix);
```

Для работы с векторами.

```
// Выделение памяти для вектора
void vector_free(vector_t *vector);
```

```
// Очищение памяти из под вектора
int vector_alloc(vector_t *vector);

// Чтение размеров для вектора
void read_vector_sizes(size_t *rows, size_t *len, int
is_matrix_readed, size_t matrix_cols);

// Чтение данных вектора
void read_vector(vector_t *vector);

// Заполнение вектора рандомными элементами
void fill_vector_rand(vector_t *vector);

// Вывод вектора в стандартном виде
void print_vector_std(vector_t vector);

// Вывод вектора в разреженном виде
void print_sparse_vector(vector_t vector);

// Получение элемента вектора по индексу
int get_vector_element(const vector_t *vector, size_t index_j);

// Удаление нулевых элементов вектора
void del_zero_elements(vector_t *vector);
```

Для операций над вектором и матрицей.

```
// Стандартное умножение матриц
void matrix_mul_vector(const matrix_t *matrix, const vector_t
*vector, vector_t *result);

// Умножение матриц в сокращенном виде
void sparse_matrix_mul_vector(const sparse_matrix_t
*sparse_matrix, const vector_t *vector, vector_t *result);

// Вывод замеров
void print_measurements(void);
```

4. Описание алгоритма.

1. «Традиционное» умножение матрицы на столбец по алгоритму умножения обычных матриц – «строка на столбец».

2. Умножение разреженной матрицы на столбец
   2.1.Пустые строки исходной матрицы пропускаются
   2.2.Для непустых строк определяется конечный индекс в массиве A
   2.3.Производится умножение ненулевых элементов, индексы в столбце определяются с помощью массива JA

2.4.Сумма записывается в соответствующую ячейку результирующей матрицы.

5. Замеры.

Временные замеры.

NUM_OF_ITERATIONS = 20.

Размер матрицы	**Процент заполнения**	**Время (мс) для стандартного умножения**	**Время (мс) для сокращенного умножения**
10	10	1.13	0.63
	25	0.91	0.86
	50	0.89	1.03
	75	0.92	2.08
	100	0.98	1.80
50	10	20.94	3.62
	25	20.32	6.92
	50	20.56	21.72
	75	22.07	31.42
	100	20.22	52.17
100	10	89.41	8.83
	25	84.11	31.91
	50	81.35	66.37
	75	80.54	71.53
	100	80.20	103.00
500	10	1396.14	221.91
	25	1403.16	803.78
	50	1348.65	2431.86
	75	1345.11	2626.54
	100	1345.12	2156.30

Объем занимаемой памяти (байт)

Размер матрицы	Процент заполнения	Количество байт для обычной матрицы	Количество байт для разрежнной матрицы
10	10	400	208
	25	400	388
	50	400	688
	75	400	988
	100	400	1288
50	10	10000	3408
	25	10000	7908
	50	10000	15408
	75	10000	22908
	100	10000	30408
100	10	40000	12808
	25	40000	30808
	50	40000	60808
	75	40000	90808
	100	40000	120808
500	10	1000000	304008
	25	1000000	754008
	50	1000000	1504008
	75	1000000	2254008
	100	1000000	3004008

6. Выводы по проделанной работе.

Стандартное (плотное) хранение матрицы целесообразно использовать при высоком проценте заполнения (75-100%), поскольку оно требует меньше памяти и обеспечивает более быструю обработку данных. В этом случае

плотное хранение выигрывает как по объему памяти, так и по времени выполнения операций.

Разреженное хранение, наоборот, более эффективно для матриц с низким процентом заполнения (до 30%). Оно существенно экономит память, так как хранит только ненулевые элементы, и ускоряет вычисления, особенно для больших матриц с малым количеством ненулевых элементов.

Таким образом, выбор метода хранения зависит от плотности матрицы: для матриц с низким процентом заполнения более эффективны разреженные алгоритмы, а для матриц с высокой плотностью — стандартные алгоритмы.

Контрольные вопросы.

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

Разреженная матрица — это матрица, в которой большая часть элементов равна нулю.
Я знаю следующие представления:

- Сжатое представление по строкам (Compressed Sparse Row, CSR): Хранятся три массива: значения ненулевых элементов, индексы столбцов для этих значений и указатели на начало каждой строки в массиве значений.
- Сжатое представление по столбцам (Compressed Sparse Column, CSC): Аналогично CSR, но хранение данных ориентировано на столбцы. Хранятся значения ненулевых элементов, индексы строк и указатели на начало каждого столбца.
- Диагональное хранение (Diagonal Storage): Используется для разреженных матриц с доминирующими ненулевыми элементами, расположенными вдоль нескольких диагоналей. Хранятся только диагонали, где находятся ненулевые элементы.

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

Под обычную матрицу выделяется n * m ячеек памяти, где n — количество строк матрицы, m — количество столбцов.
Под разреженную матрицу моего способа выделяется 2 * k + n + 1, где k — количество ненулевых элементов матрицы, n — количество строк матрицы.

3. Каков принцип обработки разреженной матрицы?

Принцип обработки разреженной матрицы заключается в работе только с ненулевыми элементами матрицы, следовательно количество операций в данном алгоритме пропорционально числу ненулевых элементов.

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

Когда большинство элементов матрицы ненулевые, стандартные алгоритмы более эффективны, так как они оперируют всей матрицей, без необходимости проверки на нулевые элементы. В разреженных алгоритмах обработка большого числа ненулевых элементов может увеличить время вычислений из-за дополнительных структур данных для индексации.