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
Лабораторная работа №3
«Обработка разреженных матриц»

Группа ИУ7-34Б

Дисциплина Типы и структуры данных

Вариант 14

Студент Козлитин Максим Александрович

Преподаватель Силантьева Александра Васильевна

2022г.

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.

Условие 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. Величина матриц - до 500x500. 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.

Описание исходных данных

1. Выбор действия:

   Целое число от 0 до 15.

2. Разреженная матрица:

   Полные размеры матрицы - Количество_Строк(0-500)
   Количество_Столбцов(0-500) (через пробел).
   Далее номер строки (до неправильного ввода).
   Затем попарно вводятся - первая строка: ненулевые элементы в строке (через пробел) и вторая строка: номера столбцов этих элементов (через пробел).
   При вводе ненулевых элементов следует вводить действительно ненулевые элементы.

3. Матрица (С клавиатуры):

   Полные размеры матрицы - Количество_Строк Количество_Столбцов (через пробел).
   По полным размерам таблицы далее вводятся строки матрицы - числа (через пробел).
   Размер строк матрицы должен совпадать с количество столбцов матрицы и быть равным для всех. Количество введенных строк также должно совпадать

с указанной размерностью матрицы.

4. Матрица (С файла):

   Первая строка определяет количество столбцов.
   Строки считываются до неправильного ввода.

   Элементы - числа (через пробел). Каждая строка матрицы вводится на отдельной строчке.

5. Вектор-столбец:

   Числа в строчку (через пробел). Размер должен совпадать с количеством столбцов матрицы (первый множитель).

6. Название файла:

   Строка размером с максимальным размером 255 символов.

Способ обращения к программе

Работа с программой осуществляется с помощью консоли.

Программа запускается с помощью команды: **./app.exe**

Далее пользователь выполняет взаимодействие с помощью меню.
Меню выводится в начале и, далее, каждые 3 введенные команды.

Меню и приглашение ввода команды:

```
~      0 - Завершить.


~     1 - Ввести разреженную матрицу.
~     2 - Ввести разреженную матрицу. <ФАЙЛОВЫЙ ВВОД>


~     3 - Ввести матрицу.
~     4 - Ввести матрицу. <ФАЙЛОВЫЙ ВВОД>


~     5 - Напечатать матрицу.
~     6 - Напечатать матрицу. <ФОРМАТИРОВАННЫЙ ВЫВОД>
```

```
~      7 - Напечатать разреженную матрицу.

~      8 - Напечатать разреженную матрицу. <ФОРМАТИРОВАННЫЙ ВЫВОД>


~      9 - Напечатать матрицу. <ФАЙЛОВЫЙ ВЫВОД>

~     10 - Напечатать матрицу. <ФОРМАТИРОВАННЫЙ ВЫВОД> <ФАЙЛОВЫЙ ВЫВОД>


~     11 - Умножить матрицу на вектор столбец.

~     12 - Вывести произведение матрицы на вектор столбец. <БЕЗ ИЗМЕНЕНИЯ>
<РАЗРЕЖЕННАЯ МАТРИЦА>


~     13 - Сравнить время выполнения стандартного алгоритма

~          и алгоритма обработки разреженных матриц.


~     14 - Сравнить объем требуемой памяти для реализации стандартного
алгоритма

~          обработки матриц и алгоритма обработки разреженных матриц.


~     15 - Сгенерировать матрицу с заданной разреженностью.


 > Выполнить команду №:
```

Описания возможных аварийных ситуаций и ошибок пользователя

1. Ввод не совпадающий с форматом входных данных (описано в Описание исходных данных).
2. Ввод целых чисел превышающих максимальное допустимое значение типа данных int.

Описание внутренних структур данных

Вектор-столбец, вектор-строка, матрица:

```
#define ROWS_SIZE 500 // Максимальный размер строки
#define COLS_SIZE 500 // Максимальный размер столбца
```

```
typedef struct
{
    size_t size; // размер вектора-столбца
    int data[COLS_SIZE]; // числа вектора-столбца
} vector_col_t;

typedef struct
{
    size_t size; // размер вектора-строки
    int data[ROWS_SIZE]; // числа вектора-строки
} vector_row_t;

typedef struct
{
    size_t row_size; // размер строки матрицы
    size_t col_size; // размер столбца матрицы
    int data[COLS_SIZE][ROWS_SIZE]; // числа матрицы
} matrix_t;
```

size_t *используется для задания размеров наших массивов, так как является особым макросом, значение которого позволяет адресовать массив любой теоретически возможной длины для данной машины.*

int *- тип данных знакового целого числа.*

Таблица 3

```
#define SIGNIFICANT_SIZE (ROWS_SIZE * COLS_SIZE) //
максимальное количество ненулевых элементов

typedef struct
{
    size_t size; // Размер массива ненулевых элементов
```

```
    int data[SIGNIFICANT_SIZE]; // значения элементов
} sign_elems_t; // AN - ненулевые элементы

typedef struct
{
    size_t size; // Размер массива столбцовых индексов ненулевых элементов
    int data[SIGNIFICANT_SIZE]; // Массив столбцовых индексов
} cols_inds_t; // JA - соответствующие ненулевым элементам столбцовые индексы

typedef struct
{
    size_t size; // размер массива
    int data[ROWS_SIZE]; // индексы массиво JA и AN
} row_relation_t; // IA_k - массив, задающий описание строки, отмечая индексы массива JA и AN

typedef struct
{
    size_t size; // количество заданных строк, содержащих описание
    row_relation_t rows[COLS_SIZE]; // описания всех строк матрица
} rows_relations_t; // IA - массив, задающий описание ненулевых строк

typedef struct
{
    size_t row_size; // размер строк полной матрицы
    size_t col_size; // размер столбцов полной матрицы
```

```
    sign_elems_t values; // ненулевые элементы
    cols_inds_t cols_inds; // столбовые индексы
    rows_relations_t rows_rels; // описания ненулевых
строк
} sparse_matrix_t; // разреженная матрица, в виде трех
объектов: AN, JA, IA
```

size_t *используется для задания размеров наших массивов, так как является особым макросом, значение которого позволяет адресовать массив любой теоретически возможной длины для данной машины.*

int *- тип данных знакового целого числа.*

Описание алгоритма

Умножение разреженной матрицы на вектор-столбец - **sparsem_column_prod**:

1. Результат, в виде разреженной матрицы, обнуляется.

2. Итерируясь по массиву, задающему описание строк получаем для текущего элемента его столбцовый индекс и значение.

3. Добавляем в сумму произведение значения на элемент вектор-столбца под индексом, соответствующим столбовому индексу эл-та.

4. Сумму записываем в ответ.
   Добавляем значение в массив значений.
   Добавляем столбцовый индекс в массив индексов, для данного умножения - 1.
   Добавляем в соответствующую строку, описание для данного элемента, указывающее на индекс в массиве значений/столбцовых индексов.

Оценка эффективности

Для оценки эффективности предварительно был сгенерирован набор матриц с различным процентом заполненной нулевыми элементами и размером. (Программа также позволяет генерировать матрицы с определённым размером и заполненностью нулевыми элементами)

Алгоритм измерения времени:

1. Измеряется текущее время - начало.
2. Запускается выполнение функции.
3. Измеряется текущее время - конец.
4. Вычисляется количество наносекунд прошедших от начала и до конца.

5. Полученное значение добавляется в сумму.

6. Данное измерение производится 100 раз, вычисляя итоговую сумму всех запусков.
7. Ответ - среднее арифметическое.

Эксперимент 1 [все нули сконцентрированы в начала матрицы]:

Размеры: [10x10] [100x100]
Процент нулевых элементов: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Метод	Размер	Заполненность нулевыми элементами										
		0	10	20	30	40	50	60	70	80	90	100
Разреженная	**10**	753	599	614	514	523	494	472	480	496	472	74
Обычная	**10**	120	118	118	118	118	118	118	118	118	119	118
Разреженная	**100**	15362	11847	10436	9003	7684	5465	4754	3373	2344	1268	206
Обычная	**100**	6692	6821	6873	6511	6545	6541	6552	6556	6551	6552	6573

Время, нс

В данной таблице промежутки измерений, где метод с разреженной матрицей уступает стандартному методу по эффективности, обведены жирной линией.

Анализ размер 10:

По отношению к методу с обычной матрицей (размер: 10):

	Заполненность нулевыми элементами										
	0	10	20	30	40	50	60	70	80	90	100
Разница	-633	-481	-496	-396	-405	-376	-354	-362	-378	-353	44
Процент	-528	-408	-420	-336	-343	-319	-300	-307	-320	-297	37

Можем заметить, что алгоритм с использованием разреженной матрицы уступает в производительности, в промежутке от 297% до 528% (353-633, нс), по отношению к методу с использованием обычной матрицы.

Это связано с тем, что функция выполняющая метод умножения с разреженной матрицей (далее метод 2), возвращает результат в виде разреженной матрицы, которая предварительно должна быть очищена, что и ухудшает производительность.

Функция, реализующая метод умножения с обычной матрицей (далее метод 1), возвращает результат в виде вектора-столбца, который также должен быть очищен, так как размерность этого объекта меньше, то и затраты на очищение меньше.

За счет того что при 0% размеренности матрицы никаких действий практически не выполняется, то и эффективность возросла на 37%, по отношению к обычной матрице.

Также, стоит отметить, что метод 1 почти не варьируется по времени выполнения, так как время выполнения зависит лишь от размера матрицы.
Метод 2, в свою очередь, с увеличением процента наполненности нулевыми элементами - ускоряется. Так как становится меньше ненулевых элементов, которые влияют на итоговую производительность.

Анализ размер 100:

По отношению к методу с обычной матрицей (размер: 100):

	Заполненность нулевыми элементами										
	0	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Разница	-8670	-5026	-3563	-2492	-1139	1076	1798	3183	4207	5284	6367
Процент	-130	-74	-52	-38	-17	16	27	49	64	81	97

Можем заметить, что алгоритм с использованием разреженной матрицы уступает в производительности до 50% наполненности матрицы нулевыми элементами, в промежутке от 16% до 97%, по отношению к методу с использованием обычной матрицы.

Начиная с 80% наполненности нулевыми элементами метод 2 показывает более лучший результат, нежели метод 1. При матрице полностью заполненной нулями заметим, что производительность увеличивается на 97%.

Также, стоит отметить, что метод 1 почти не варьируется по времени выполнения, так как время выполнения зависит лишь от размера матрицы.
Метод 2, в свою очередь, с увеличением процента наполненности нулевыми элементами - ускоряется. Так как становится меньше ненулевых элементов, которые влияют на итоговую производительность.

Эксперимент 2 [все нули случайно распределены по матрице]:

Размеры: [10x10] [100x100]
Процент нулевых элементов: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Метод	Размер	Заполненность нулевыми элементами											
		0	10	20	30	40	50	60	70	80	90	100	
Разреженная	**10**	827	792	777	763	711	743	730	719	652	524	74	Время, нс
Обычная	**10**	120	118	118	118	118	118	118	118	118	119	118	
Разреженная	**100**	16726	14277	17039	12259	10919	9232	7902	7122	6226	5166	165	
Обычная	**100**	6692	6821	6873	6511	6545	6541	6552	6556	6551	6552	6573	

В данной таблице промежутки измерений, где метод с разреженной матрицей уступает стандартному методу по эффективности, обведены жирной линией.

Анализ размер 10:

По отношению к методу с обычной матрицей (размер: 10):-1

	Заполненность нулевыми элементами										
	0	10	20	30	40	50	60	70	80	90	100
Разница	-707	-674	-659	-645	-593	-625	-612	-601	-534	-405	44
Процент	-589	-571	-558	-547	-503	-530	-519	-509	-453	-340	37

Можем заметить, что алгоритм с использованием разреженной матрицы уступает в производительности, в промежутке от 340% до 633% (334-633, нс), по отношению к методу с использованием обычной матрицы.

Здесь прослеживается, схожий с предыдущим экспериментом результат, что может говорить о том, что для маленьких матриц распределение нулей по матрице не влияет.

Анализ размер 100:

По отношению к методу с обычной матрицей (размер: 100):-1

	Заполненность нулевыми элементами										
	0	10	20	30	40	50	60	70	80	90	100
Разница	-10034	-7456	-10166	-5748	-4374	-2691	-1350	-566	325	1386	6408
Процент	-150	-109	-148	-88	-67	-41	-21	-9	5	21	97

Можем заметить, что алгоритм с использованием разреженной матрицы уступает в производительности до 80% наполненности матрицы нулевыми элементами, в промежутке от 21% до 150%, по отношению к методу с использованием обычной матрицы.

Начиная с 80% наполненности нулевыми элементами метод 2 показывает более лучший результат, нежели метод 1. При матрице полностью заполненной нулями заметим, что производительность увеличивается на 97%.

Здесь прослеживаются, различия с предыдущим экспериментом результат, что может говорить о том, что для матриц более значимого размера распределение нулей по матрице влияет на итоговую эффективность.
Так как в прошлом эксперимент при тех же исходных данных, но с другим распределением, разреженная матрица показывала большую эффективность, начиная с 50%, а в данном эксперименте лишь с 80%, то можем заключить - распределения нулей по матрице влияет на итоговую эффективность алгоритма с использование разреженных матриц.

Сравнение памяти:

	Разреженная матрица	**Обычная матрица**	**Разница**
Память, байт	3004040	1000016	2004024
Процент			200

По данной таблице можем заметить, что разреженная матрица занимает на 200% больше памяти чем обычная, что, несомненно, является большой разницей.

Такую большую разницу можно объяснить тем, что реализация данной структуры данных основана на статической памяти, а не динамической. Соотвественно невозможно достичь уменьшения используемой памяти разряженной матрицы, в зависимости от наполненности нулевыми элементами.

Вывод

В ходе проделанной работы, я приобрел навыки обработки разреженных матриц. Было произведено сравнение эффективности использования алгоритмов обработки разреженных матриц (по времени выполнения и по требуемой памяти) со стандартными алгоритмами обработки матриц при различном процентном заполнении матриц ненулевыми значениями и при различных размерах матриц.

По результатам, проведенной оценки эффективности, было выявлено, что разреженная матрица занимает на 200% больше памяти чем обычная.
В зависимости от распределения нулей по матрице, алгоритм с использованием разреженной матрицы показывает разные результаты эффективности.

Распределение нулей - случайно: Более эффективная производительность впервые достигается при 80% наполненности матрицы нулями для размера матрицы 100x100 элементов.
Распределение нулей - концентрация в начале матрицы: Более эффективная производительность впервые достигается при 50% наполненности матрицы нулями для размера матрицы 100x100 элементов.

Большая разница по памяти обусловлена использованием статической памяти при реализации структуры данных разреженная таблица. То есть память не изменяется в зависимости от наполненности матрицы нулями.
Той же причиной обусловлена низкая производительность данного алгоритма.

Подробные результаты и методы оценки эффективности программы можно изучить в пункте «Оценка эффективности».

Контрольные вопросы

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

   Разреженная матрица – матрица с преимущественно нулевыми элементами. Число ненулевых элементов в матрице порядка n может выражаться как n^(1+g), где g < 1, где значения g лежат в интервале 0.2 ... 0.5.

   Схемы хранения матриц:
   1. Линейный связанный список
   2. Кольцевой связанный список
   3. Разреженный строчный формат
   4. Разреженный столбцовый формат
   5. Хранение с помощью двунаправленного стека и очереди

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

   Под хранение обычной матрицы, как было представлено выше, требуется один и тот же объём памяти, поскольку она состоит из массива массивов фиксированной длины (n * m * sizeof(element)). У разреженных матриц объём вычисляется по-другому: суммарная память, выделяемая под разреженную матрицу, равна сумме всех массивов, использованных для её хранения, а также дополнительных переменных (**K * sizeof(elem) + (n + 1) * sizeof(element)**), где K – количество ненулевых элементов.

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

Принцип обработки разреженной матрицы заключается в работе исключительно с ненулевыми элементами.

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

   Используемый алгоритм следует выбирать, исходя из степени разреженности матрицы. Экспериментально было установлено, что граница для времени находится примерно в 80%. То есть, если разреженность ниже, то следует использовать алгоритм умножения для разреженных матриц, а если выше – обычный алгоритм умножения.