



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

## **ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3** **ОБРАБОТКА РАЗРЕЖЕННЫХ МАТРИЦ**

Название предмета: Типы и структуры данных

Студент: Варламова Екатерина Алексеевна

Группа: ИУ7-31Б

2020г.

## **I. Описание условия задачи**

Разработать программу умножения разреженного вектора на разреженную матрицу. Предусмотреть возможность ввода данных, как с клавиатуры, так и с использованием заранее подготовленных данных. Решить задачу при стандартном хранении матриц и при следующем:

- вектор A содержит значения ненулевых элементов;
- вектор JA содержит номера столбцов для элементов вектора A;
- связный список IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки Nk матрицы A.).

Сравнить время выполнения операций и объем памяти при использовании 2-х алгоритмов при различном проценте заполнения матриц.

### **Техническое задание**

#### ***1. Описание исходных данных***

Программа ожидает:

- размеры матрицы n и m (n считается и длиной вектора-строки).
- номер действия (подробнее о том, какой номер соответствует определённому действию, будет выведено при запуске программы). При этом возможны следующие варианты ввода матрицы и вектора:

- ввод матрицы и вектора полностью;
- частичный ввод матрицы и вектора (с указанием только ненулевых элементов);

#### **Формат ввода:**

##### **1. При вводе полностью**

- для матрицы 3 на 3:

2 3 0

0 2 0

1 2 0

- для вектора из 5 элементов:

0 1 2 0 2

2. При частичном вводе (ввод заканчивается 0):

- Для матрицы 3 на 3:

row of element: 1

column of element: 3

element: 2

row of element: 2

column of element: 1

element: 8

row of element: 3

column of element: 1

element: 5

row of element: 3

column of element: 3

element: 5

row of element: 0

ИТОГ:

0 0 2

8 0 0

5 0 5

- Для вектора из 5 элементов (ввод заканчивается 0):

column of element: 1

element: 5

column of element: 2

element: 1

column of element: 4

element: 10

column of element: 0

ИТОГ:

5 1 0 10 0

### 3. Описание результата программы

Результатом работы программы могут являться (в зависимости от введённого действия):

1. Вектор-столбец, являющийся произведением вектор-строки на матрицу (в зависимости от алгоритма будет выведен в различных форматах).
2. Статистика по времени выполнения и объёму памяти при использовании 2-х алгоритмов при различном проценте заполнения матрицы и вектора.

#### Формат вывода:

1. Вектор-столбец будет выведен полностью (перечислены все элементы с 1 по последний), если матрица и вектор хранятся стандартным образом. Размерности матрицы не превышают 30.

Пример для 5 элементов:

0 1 4 0 2

2. Вектор-столбец будет выведен, если матрица и вектор хранятся стандартным образом. Размерности матрицы превышают 30.

Пример для 31 элемента:

column of element: 2

element: 7

column of element: 4

element: 6

column of element: 19

element: 10

3. Вектор-столбец будет выведен частично (только ненулевые элементы и номера строк, в которых расположены ненулевые элементы). Пример для 5 элементов:

data

1 4 2

rows

2 3 5

4. Статистика в форме таблицы.

#### **4. Описание задачи, реализуемой программой**

Программа в зависимости от запросов пользователя производит ряд действий: получает матрицу и вектор выбранным способом, считает их произведение при любом из двух способов хранения и выводит результат в зависимости от способа хранения. Кроме того, выводит статистику.

#### **5. Способ обращения к программе**

Способ обращения к программе - консольный. Дальнейшие инструкции будут выведены после запуска.

#### **6. Описание возможных аварийных ситуаций и ошибок пользователя**

- ошибки ввода размерностей матрицы и вектора (отрицательные), а также значений, которыми они заполняются (недостаточно данных);
- отказ операционной системы выделить запрашиваемую память;

Во всех указанных случаях программа сообщит об ошибке.

## **II. Описание внутренних структур данных**

### **1. Основная структура данных**

Основной структурой данных в программе является матрица, которая хранится следующим образом:

- вектор A содержит значения ненулевых элементов;
- вектор JA содержит номера столбцов для элементов вектора A;
- связный список IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки Nk матрицы A.). Описание структуры на языке C выглядит следующим образом:

```
typedef struct
{
    vector_t data;
    vector_t rows;
    vector_t columns;
    size_t n;
    size_t m;
} sparse_matrix_t;
```

A также основной структурой является разреженный вектор:

```
typedef struct
{
    vector_t data;
```

```
vector_t columns;  
size_t n;  
} sparse_vector_t;
```

### *Вспомогательная структура данных*

Вспомогательной структурой данных является вектор. Данная структура динамически расширяется (умножением на 2) при необходимости, изначально её размер = 20 элементов.

Её описание представлено на языке C:

```
typedef struct  
{  
    void *front;  
    size_t size_of_element;  
    size_t size;  
    size_t alloc_size;  
} vector_t;
```

## **III. Описание алгоритма**

Для решения задачи вычислить произведение вектора-строки на матрицу используется два подхода:

### 1. Стандартный

Матрица и вектор хранятся в памяти полностью (содержат как нулевые, так и ненулевые элементы). Для вычисления произведения матрица просматривается по столбцам и элементы столбцов поэлементно перемножаются с элементами вектора-строки. Результат записывается в вектор-столбец. Таким образом, оценка алгоритма  $O(n*m)$ .

### 2. Сжатый способ хранения

При данном подходе:

1. Заполняется массив storage длины, равной количеству строк в матрице. Просматривая разреженный вектор (сложность зависит от количества ненулевых элементов), находим столбец  $k$  ( $O(1)$ ), в котором располагается текущий элемент. В массиве storage на место  $k$  ставим номер текущего элемента в разреженном векторе.

2. Для каждой строки просматриваем элементы и добавляем в соответствующую ячейку результата значение произведения. При этом произведение ищем с помощью массива storage.

### ***Оценка по времени***

Таким образом, сложность алгоритма зависит от количества ненулевых элементов в векторе и матрице (не считая заполнение нулями массив storage). То есть, если количество ненулевых элементов в матрице  $= n_1$ , а в векторе  $n_2$ , то сложность составит  $O(n_1 + n_2)$ , что при  $n_1 \ll n * m$  и  $n_2 \ll n$  будет в разы быстрее  $O(n * m)$ .

### ***Оценка по памяти***

Поскольку в основе структуры разреженной матрицы и разреженного вектора лежит СД вектор, которая динамически расширяется при необходимости, то при сжатом способе хранения размер памяти под вектор и под матрицу зависят от количества ненулевых элементов. Пусть количество ненулевых элементов в матрице  $n_1$ , количество ненулевых в векторе  $n_2$ , размер элементов и размер индексов  $= k$ , тогда:

- при стандартном способе хранения общее количество памяти под вектор и матрицу  $= (n * m * k) + (n * k) = n * k(m + 1)$ .
- при сжатом способе хранения общее количество памяти под вектор и под матрицу  $= (n_1 * k + n_1 * k + m * k) + (n_2 * k + n_2 * k) = k(2 * n_1 + 2 * n_2 + m)$ . При этом дополнительно необходим массив, размер которого:  $n * k$ .  
Суммарно:  $k(n + m + 2 * n_1 + 2 * n_2)$ .

Очевидно, что при  $n_1 \ll n * m$  и  $n_2 \ll n$  памяти при сжатом способе хранения требуется меньше, чем при стандартном.

### ***Итог***

Из приведённых выше рассуждений очевидно, что при маленьком проценте заполнения матрицы (при большом количестве нулевых элементов) выгоднее и памяти и по времени использовать сжатый способ хранения. При достаточно заполненной матрице при сжатом способе хранения возникают лишние

затраты по времени и памяти, поэтому разумно использовать стандартный способ хранения.

### *Проверка выводов экспериментально*

Возьмём матрицу размером 1000\*1000 и вектор из 1000 элементов.

Посмотрим, какой алгоритм выгоднее использовать при умножении вектора на матрицу по памяти и по времени при разном проценте заполнения объектов (первый столбец) и проверим наш теоретический вывод экспериментально.

В таблице время указано в миллисекундах (во 2 и 3 столбце), память в байтах (в 4 и 5 столбцах).

percent	ordinary	sparse	size_or	size_sp
1	11081.00	146.00	8008000	176168
6	9776.00	740.00	8008000	976968
11	9557.00	1274.00	8008000	1777768
16	8960.00	1988.00	8008000	2578568
21	9719.00	3187.00	8008000	3379368
26	8737.00	3510.00	8008000	4180168
31	8966.00	4214.00	8008000	4980968
36	9049.00	5504.00	8008000	5781768
41	8667.00	6481.00	8008000	6582568
46	8559.00	7046.00	8008000	7383368
51	9043.00	8675.00	8008000	8184168
56	9209.00	9482.00	8008000	8984968





	26	161.00	79.00	80800	43624
x-----x-----x-----x-----x-----x-----x-----x					
	31	101.00	94.00	80800	51704
x-----x-----x-----x-----x-----x-----x-----x					
	36	100.00	115.00	80800	59784
x-----x-----x-----x-----x-----x-----x-----x					
	41	85.00	115.00	80800	67864
x-----x-----x-----x-----x-----x-----x-----x					
	46	86.00	153.00	80800	75944
x-----x-----x-----x-----x-----x-----x-----x					
	51	86.00	160.00	80800	84024
x-----x-----x-----x-----x-----x-----x-----x					
	56	87.00	183.00	80800	92104
x-----x-----x-----x-----x-----x-----x-----x					
	61	84.00	211.00	80800	100184
x-----x-----x-----x-----x-----x-----x-----x					
	66	66.00	181.00	80800	108264
x-----x-----x-----x-----x-----x-----x-----x					
	71	66.00	198.00	80800	116344
x-----x-----x-----x-----x-----x-----x-----x					
	76	67.00	221.00	80800	124424
x-----x-----x-----x-----x-----x-----x-----x					
	81	69.00	245.00	80800	132504
x-----x-----x-----x-----x-----x-----x-----x					
	86	68.00	269.00	80800	140584
x-----x-----x-----x-----x-----x-----x-----x					
	91	67.00	294.00	80800	148664
x-----x-----x-----x-----x-----x-----x-----x					
	96	77.00	334.00	80800	156744
x-----x-----x-----x-----x-----x-----x-----x					

Видим, что уже при 36% сжатый способ хранения проигрывает по времени. То есть для небольших размеров объектов в общем случае разумнее использовать стандартный способ хранения.

#### IV. Тестирование

##### 1. Позитивные тесты

Входные данные	Описание теста	Результат
0 0 4 10 6 0 0 0 0 0 3 0 0 4 0 8  0 8 0 0	Проверка умножения при стандартном хранении	48 0 0 0
data: 4 10 6 3 4 8 columns: 2 3 0 2 1 3 pointers: 0 2 3 4 6  data: 8 columns: 2	Проверка умножения при сжатом хранении	data: 48 columns: 1
5 4 2 2 6 4 3 3 9 3 8 10 7 9 9 5 6 9 5 2 7 4 2 7 10  5 0 0 0 0	Проверка умножения при стандартном хранении	25 20 10 10 30
data: 5 4 2 2 6 4 3 3 9 3 8 10 7 9 9 5 6 9 5 2 7 4 2 7 10 columns: 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 pointers: 0 5 10 15 20 25  data: 5 columns: 1	Проверка умножения при сжатом хранении	data: 25 20 10 10 30 columns: 1 2 3 4 5

## **V. Выводы по проделанной работе**

В ходе работы был проведён сравнительный анализ подходов к хранению и обработке разреженных матриц и векторов (на примере операции умножения), в результате которого установилось, что при маленьком проценте заполнения объектов (при большом количестве нулевых элементов) выгоднее по памяти и по времени использовать сжатый способ хранения. При достаточно заполненных объектах при сжатом способе хранения возникают лишние затраты по времени и памяти, поэтому разумно использовать стандартный способ хранения. Для небольших размеров объектов в общем случае разумнее использовать стандартный способ хранения.

## **VI. Ответы на вопросы**

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

Разреженная матрица – матрица, содержащая преимущественно нули.

Хранить можно стандартно (все элементы) и с помощью следующей схемы:

- вектор  $A$  содержит значения ненулевых элементов;
- вектор  $JA$  содержит номера столбцов для элементов вектора  $A$ ;
- связный список  $IA$ , в элементе  $N_k$  которого находится номер компонент в  $A$  и  $JA$ , с которых начинается описание строки  $N_k$  матрицы  $A$ ).

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

См. пункт “Оценка по памяти” в описании алгоритмов.

3. Каков принцип обработки разреженной матрицы?

См. пункт “Сжатый способ хранения” в описании алгоритмов.

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

См. пункт “Итог” в описании алгоритмов.