Отчёт лабораторной работы №3

по курсу «Типы и структуры данных»

Отчёт выполнил:

Вариант 23

г. Москва 2019

Отчет

### 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 (№ 6)**:

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов: - вектор A содержит значения ненулевых элементов;  
- вектор IA содержит номера строк для элементов вектора A;  
- связный список JA, в элементе Nk которого находится номер компонент

в A и IA, с которых начинается описание столбца Nk матрицы A.

1. Смоделировать операцию умножения вектора-строки и матрицы, хранящихся в этой форме, с получением результата в той же форме.

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.

**Описание ТЗ:**

**1.Введение:**

**1.1 Наименование программы:**

Мультипликатор и обработчик разреженных матриц.

### 1.2. Срок выполнения работы:

2 недели с момента выдачи задачи.

**1.3 Краткая характеристика области применения:**

Программа позволяет обрабатывать и оценивать эффективность, время работы умножения матрицы на вектор-строку стандартными алгоритмами обработки матриц при различном процентном заполнении матриц и векторов-строк ненулевыми значениями и при различных размерах матриц и векторов-строк.

**2. Основания для разработки:**

### 2.1. Заказчик

Силантьева Александра Васильевна

### 2.2. Исполнитель

Исполнитель – Савинов Егор Дмитриевич, студент группы ИУ7-34б.

### 2.3. Основания для разработки

Учебная программа 3 семестра программы ИУ7.

### 3. Назначение разработки

### 3.1 Общая концепция системы:

Программа позволяет обрабатывать и оценивать эффективность, время работы умножения матрицы на вектор-строку стандартными алгоритмами обработки матриц при различном процентном заполнении матриц и векторов-строк ненулевыми значениями и при различных размерах матриц и векторов-строк.

### 3.2 Описание функциональности системы:

Заполнение матрицы и вектора-строки, умножение, оценка эффективности и вывод время обработки умножения матрицы на вектор-строку.

### 4. Требования к программе:

### 4.1 Требования к информационным структурам:

Программа должна уметь отрабатывать любой ввод пользователя. При корректных данных следует выводить меню, таблицу эффективности или данные, иначе нужно выдавать сообщение об ошибке и после этого программа должна завершить свою работу.

### 4.2. Требования к функциональным характеристикам:

Программа должна выполнять:

1. Корректно отрабатывать команды меню.
2. Автоматическое заполнение данных.
3. Разреженный ввод данных (в том числе и для больших размеров).
4. Классический ввод данных.
5. Сравнение эффективности по скорости обработки умножения и занимаемой памяти.

### 4.3. К надежности:

Программа должна уметь обрабатывать любой ввод, также при введении некорректных данных не должна завершаться аварийно.

### 4.4 Требования к составу аппаратного обеспечения:

У клиента интегрированная среда разработки должна поддерживать файлы с расширением .с. Программа не требует поддержки со стороны пользователя.

### 5. Стадии и этапы разработки:

*Стадии:*

1. Начальный этап разработки (ввод чисел и защита от некорректного ввода)
2. Тестирование заполнения матрицы и вектора-строки, умножения, оценки эффективности и вывода время обработки умножения матрицы на вектор-строку.
3. Разработка вывода программы (в виде отформатированной таблицы, вывода меню, начальных данных, результата умножения).
4. Тестирование программы.
5. Доработка на основе тестирования программы и сдача окончательного варианта программы лабораторной работы.

*Этапы:*

Получение от преподавателя лабораторной работы и сроков сдачи программы, написание тестов для программы, создание программы, тестирование, написание отчета по проделанной работе, сдача программы и отчета.

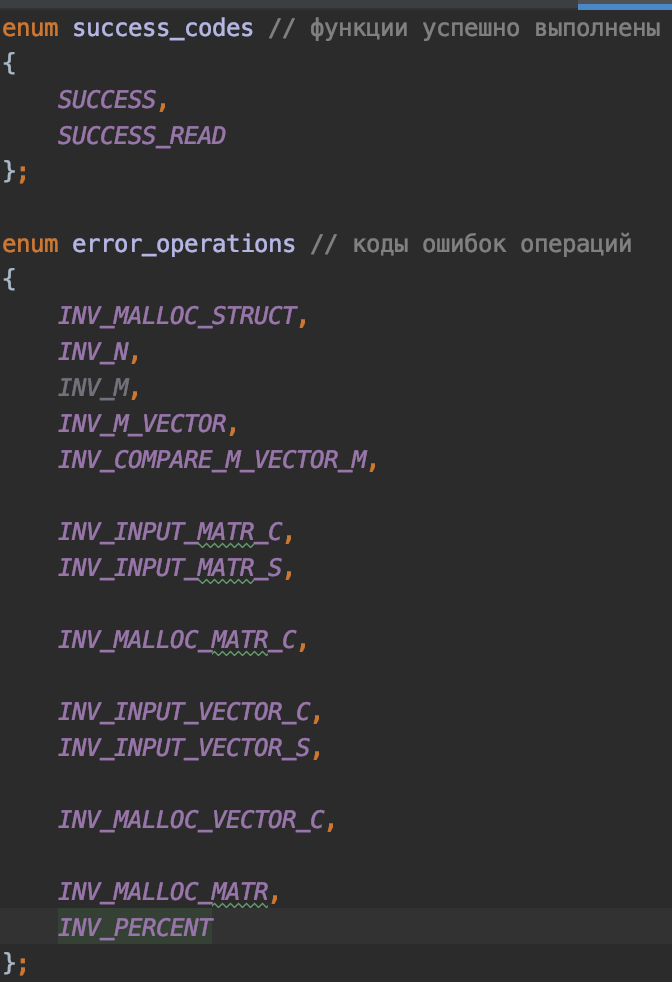
**Алгоритм работы программы:**

1. Ввод размера матрицы и вектора-строки.
2. Заполнение значениями матрицы и вектора-строки.
3. Вывод умножения вектора-строки на матрицу.
4. Вывод таблицы эффективности умножения.

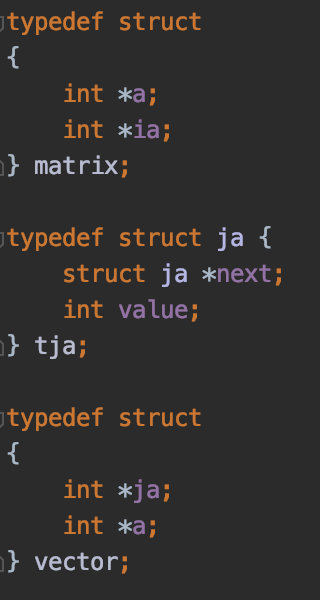
**Константы:**



**Коды выполнения программы:**

****

**Структура данных:**



**Способ обращения к программе**:

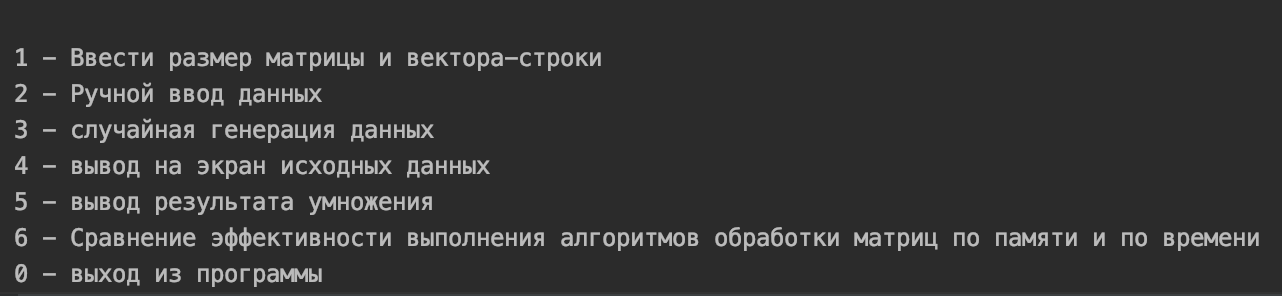
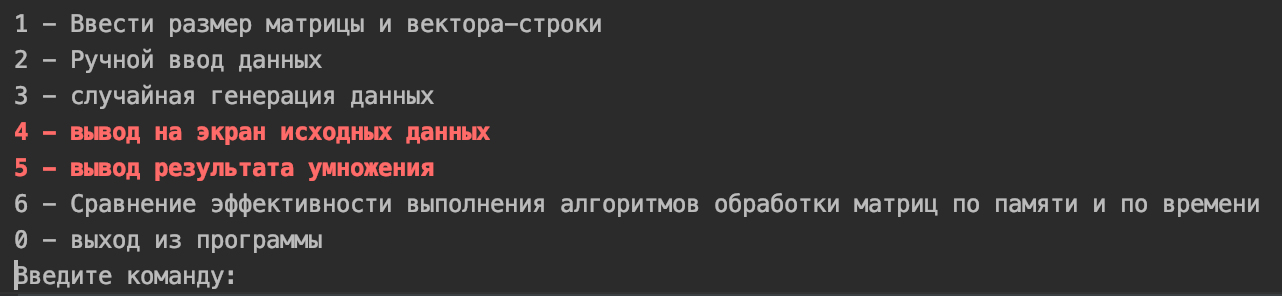
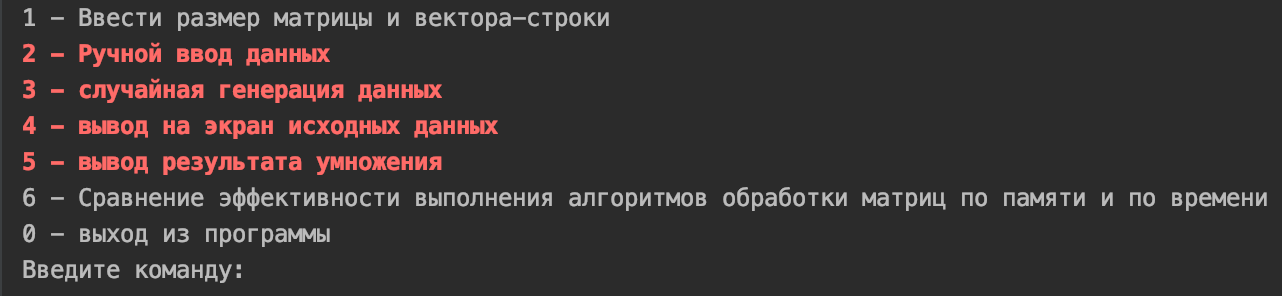
Программа состоит из шести исходных файлов main.c input.с memory.с output.с tool.с menu.c. Для запуска программы необходим исполняемый файл с расширением .exe. Запуск возможен с консоли или из любого другого проводника ОС.

**Входные данные:**

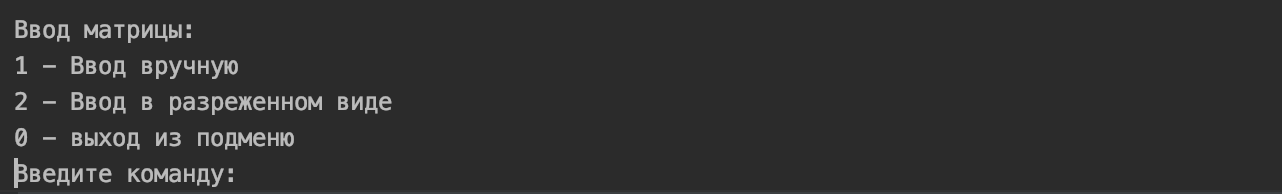
Программа ничего не получает на вход, для корректной работы необходимо пользователю ввести размеры матрицы, вектора-строки и необходимо заполнить их значениями

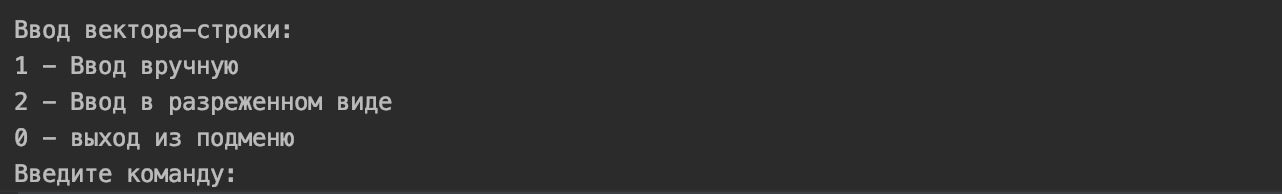
Для пользователя предусмотрено меню действий:

Красным отмечены меню действий, которые недоступны (т.к. не введены размеры).



Если ввести «2» в главном меню, то выведется подменю для выбора заполнения:





Если нажать 1:

Нужно будет ввести размер матрицы и вектора-строки.

Если нажать 2:

Нужно будет вручную ввести данные.

Если нажать 3:

Будет предложено ввести % разреженности матрицы и вектора-строки и будет произведена генерация данных.

Если нажать 4:

Вывод на экран исходных данных.

Если нажать 5:

Вывод результата умножения.

Если нажать 0:

Программа завершит свою работу

Обращение к пунктам меню:

Набор соответствующих цифр через консоль.

**Выходные данные:**

Выходными данными являются:

* Исходные данные
* Результат умножения
* Оценка эффективности и времени умножения вектора-строки на матрицу.

**Функции:**

int input\_matrix\_classic(matrix \*\*, tja \*\*, int \*\*, int, int, int \*, int \*);

Вход: структура данных для матрицы, кол-во строк, столбцов, кол-во ненулевых элементов.

Выход: код выполнения работы функции.

int input\_matrix\_sparse(matrix \*\*, tja \*\*, int \*\*, int, int, int \*, int \*);

Вход: структура данных для матрицы, кол-во строк, столбцов, кол-во ненулевых элементов.

Выход: код выполнения работы функции.

int input\_vector\_classic(vector \*\*, int \*\*, int, int \*);

Вход: структура данных для вектора-строки, кол-во столбцов, кол-во ненулевых элементов.

Выход: код выполнения работы функции.

int input\_vector\_sparse(vector \*\*, int \*\*, int, int \*);

Вход: структура данных для вектора-строки, кол-во столбцов, кол-во ненулевых элементов.

Выход: код выполнения работы функции.

int input\_size(int \*, int \*, int \*);

Вход: кол-во строк, столбцов.

Выход: код выполнения работы функции.

int generate(matrix \*\*, vector \*\*, tja \*\*, int \*\*, int \*\*, int, int, int, int \*, int \*, double, double, int \*);

Вход: структура данных для матрицы и вектора-строки, кол-во строк, столбцов, кол-во ненулевых элементов, проценты разреженности для матрицы и вектора-строки

Выход: код выполнения работы функции.

int fill\_percent(double \*, double \*);

Вход: проценты разреженности для матрицы и вектора-строки.

Выход: код выполнения работы функции.

int malloc\_struct\_matrix(matrix \*\*, int);

Вход: адрес структуры матрицы, размер.

Выход: код выполнения работы функции.

int malloc\_struct\_vector(vector \*\*, int);

Вход: адрес структуры вектора.

Выход: код выполнения работы функции.

int mem\_alloc\_matrix\_classic(int \*\*\*, int, int);

Вход: адрес структуры матрицы, размеры.

Выход: код выполнения работы функции.

void free\_matrix\_classic(int \*\*\*, int);

Вход: адрес структуры матрицы, размер.

int mem\_alloc\_vector\_classic(int \*\*\*, int);

Вход: адрес структуры вектора, размер.

Выход: код выполнения работы функции.

void free\_vector\_classic(int \*\*\*, int);

Вход: адрес структуры вектора, размер.

void free\_struct\_matrix(matrix \*\*);

Вход: адрес структуры матрицы.

void free\_struct\_vector(vector \*\*);

Вход: адрес структуры вектора.

int add\_elem(tja \*\*, int);

Вход: адрес головы списка, добавляемый элемент.

Выход: код выполнения работы функции.

void free\_head(tja \*\*);

Вход: адрес головы списка.

void read\_head(tja \*, int);

Вход: голова списка, элемент, кол-во до нужного элемента.

void print\_red\_menu();

void print\_green\_menu();

void print\_matrix\_menu();

void print\_vector\_menu();

void print\_red\_uninitial\_menu();

void get\_choice(int, int, int \*);

void print\_initial\_data(matrix \*, vector \*, tja \*\*, int \*\*, int \*, int, int, int, int);

Вход: структура данных для матрицы и вектора-строки, классическая матрица и вектор-строка, размеры матрицы и вектора-строки, кол-во ненулевых элементов.

void result\_print(int \*\*, int , int, matrix \*\*, tja \*\*);

Вход: матрица результата, ее размер, структура данных для матрицы, кол-во ненулевых элементов.

double multiplication\_matrix\_vector\_sparse(matrix \*, tja \*, int \*\*, int \*\*\*, int , int);

Вход: структура данных для матрицы, классический вектор-строка, матрица результата, размер вектора-строки, кол-во ненулевых элементов.

Выход: время умножения разреженной матрицы и вектора-строки.

double multiplication\_matrix\_vector\_classic(int \*\*, int \*, int \*\*\*, int);

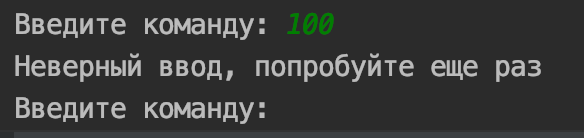
void test\_memory();

Вход: классическая матрица и вектор-строка, матрица результата, размеры матрицы, вектора-строки.

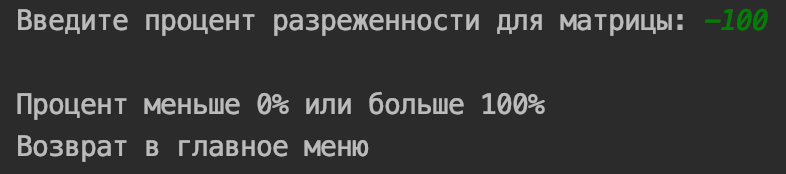
Выход: время умножения стандарной матрицы и вектора-строки.

**Аварийные ситуации:**

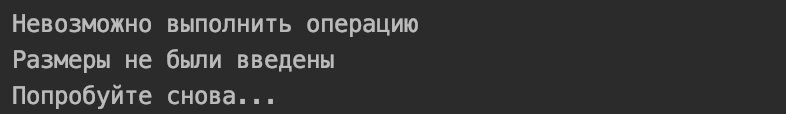
1. Ввод несуществующего пункта меню:



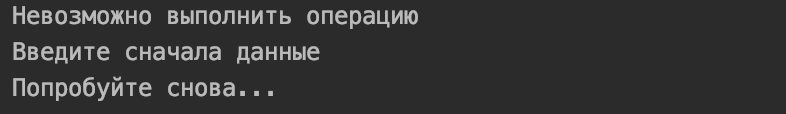
1. Ввод значения разреженности меньше 0 и больше 100:



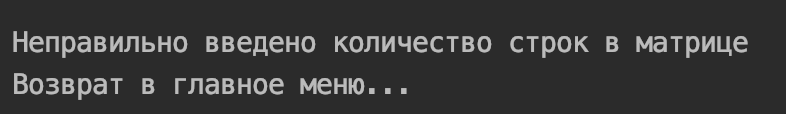
1. Попытка заполнить, сгенерировать данные, вывод на экран исходных данных, данных результата умножения, если размеры еще не были введены.

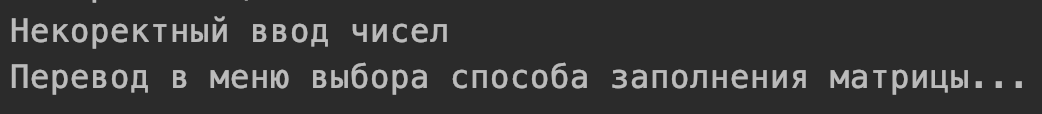
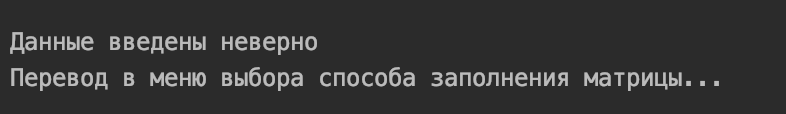


4. Попытка вывода на экран исходных данных, данных результата умножения, если данные еще не были введены.



5.Попытка ввода недопустимых значений при ручном вводе данных, при вводе размеров матрицы, вектора-строки:





**Тесты**

1. Вывод на экран исходных данных

размер 3x3 1x3:

Исходная матрица:

Классический вид:

-18 -67 -69

-68 -34 -48

-20 -93 -63

Исходная матрица:

Разреженный вид:

A: -18 -68 -20 -67 -34 -93 -69 -48 -63

IA: 1 2 3 1 2 3 1 2 3

JA: 3 6 9

Исходный вектор-строка:

Классический вид:

-36 -76 -24

Исходный вектор-строка:

Разреженный вид:

A: -36 -76 -24

JA: 1 2 3

2. Вывод на экран результата умножения.

Стандартный вид:

6296 7228 7644

Разреженный вид:

A: 6296 7228 7644

IA: 1 2 3

JA: 3

1. При нулевом векторе-строке или матрице получим в результате нулевую матрицу.

Исходная матрица:

Классический вид:

0 0 0

0 0 0

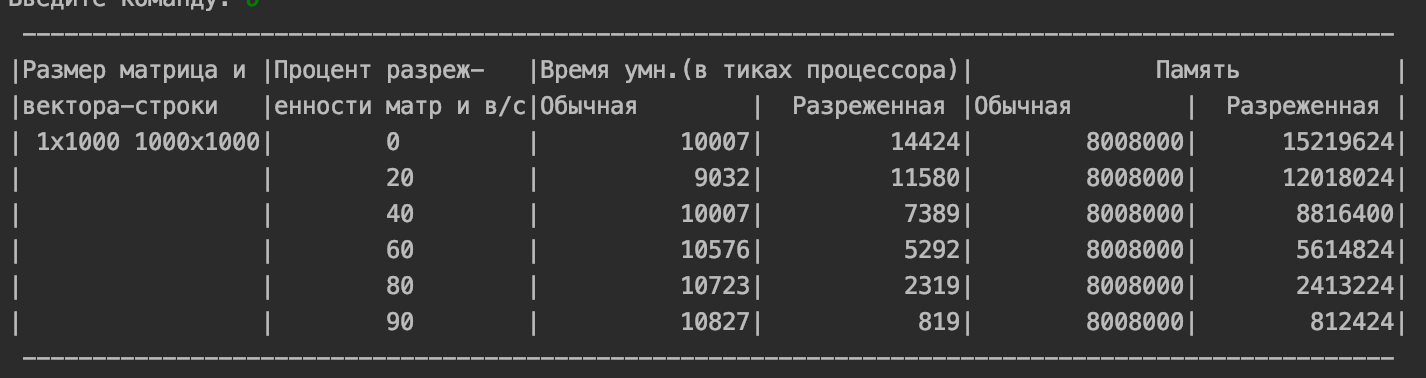
0 0 0

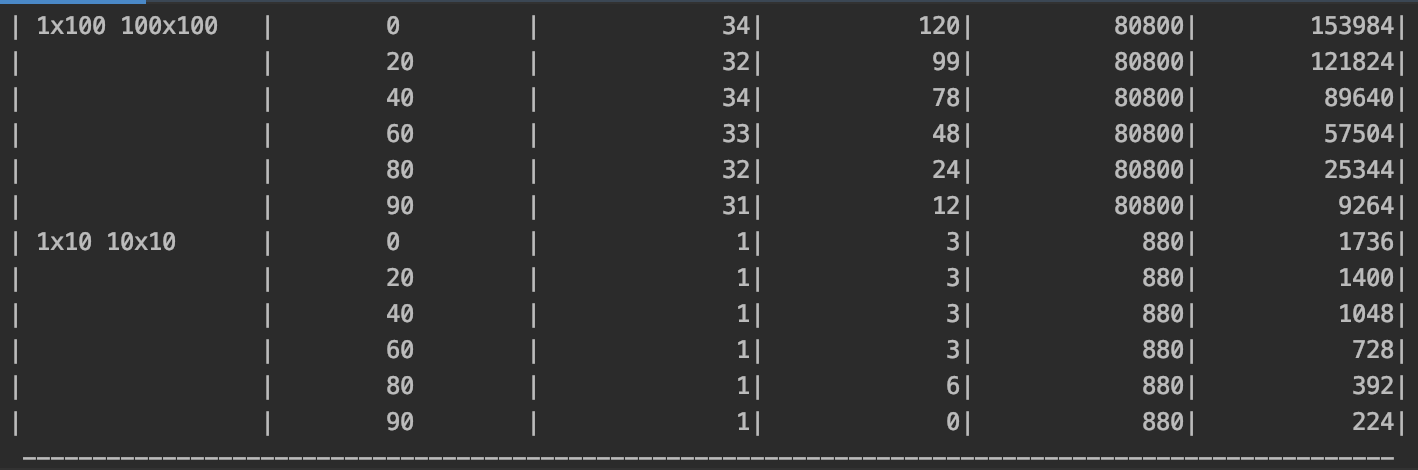
Исходный вектор-строка:

Классический вид:

0 0 0

**Оценка эффективности**

****

****

От 40% до 100% разреженности матрицы и размерах больших 1000x1000 выгоднее использовать алгоритм умножения разреженных матриц (экономия времени и памяти).

При малом размере матрицы (от 100x100 до 1000x1000) и от 20% до 100% разреженности матрицы выгоднее использовать алгоритм умножения разреженных матриц. В остальных случаях выгоднее использовать стандартный алгоритм обработки.

Самые маленькие матрицы (меньше 100x100) при любом % разреженности лучше обрабатывать стандартным алгоритмом.

**Вывод:**

Разреженный алгоритм работает быстрее при заполненности матрицы нулями от 40% и эффективнее по памяти при заполненности нулями от 60%.

**Ответы на вопросы.**

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

Разреженная матрица – матрица с преимущественно нулевыми элементами. Матрицы могут быть достаточно большие (больше 1010−20 элементов), а число ненулевых элементов при матрице порядка **n** может выражаться как **n**1+g **,** где **g < 1**.  
В подобных матричных задачах значения **g** лежат в интервале **0.2** ... **0.5**, т.е. матрица разрежена. Но разреженность матрицы следует учитывать только в том случае, когда из этого можно извлечь выгоду за счет игнорирования нулевых элементов. На самом деле разреженную матрицу можно обрабатывать так же как плотную, и наоборот, плотную матрицу можно обрабатывать так же как разреженную. В обоих случаях получаются правильные числовые результаты, но вычислительные затраты в том или другом случае могут существенно различаться.

Существуют различные методы хранения элементов матрицы в памяти. Например, линейный связный список, т.е. последовательность ячеек, связанных в определенном порядке. Каждая ячейка списка содержит элемент списка и указатель на положение следующей ячейки. Можно хранить матрицу, используя кольцевой связный список, двунаправленные стеки и очереди. Существует диагональная схема хранения симметричных матриц, а также - связные схемы разреженного хранения. Связная схема хранения матриц, предложенная Кнутом, предлагает хранить в массиве (например, в AN) в произвольном порядке сами элементы, индексы строк и столбцов соответствующих элементов (например, в массивах I и J), номер (из массива AN) следующего ненулевого элемента, расположенного в матрице по строке (NR) и по столбцу (NC), а также номера элементов, с которых начинается строка (указатели для входа в строку – JR) и номера элементов, с которых начинается столбец (указатели для входа в столбец - JC). Данная схема хранения избыточна, но позволяет легко осуществлять любые операции с элементами матрицы.

Наиболее широко используемая схема хранения разреженных матриц - это схема, предложенная Чангом и Густавсоном, называемая: "разреженный строчный формат". Эта схема предъявляет минимальные требования к памяти и очень удобна при выполнении операций сложения, умножения матриц, перестановок строк и столбцов, транспонирования, решения систем линейных уравнений, при хранении коэффициентов в разреженных матрицах и т.п.

В этом случае значения ненулевых элементов хранятся в массиве AN, соответствующие им столбцовые индексы - в массиве JA. Кроме того, используется массив указателей, например, IA, отмечающих позиции AN и JA, с которых начинаются описание очередной строки. Дополнительная компонента в IA содержит указатель первой свободной позиции в JA и AN.

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

Под хранение обычной матрицы выделяется n \* m \* sizeof(тип элементов матрицы).

Под хранение разреженной матрицы выделяется различное кол-во памяти. Количество памяти зависит от способа хранения и разреженности матрицы.

3. Каков принцип обработки разреженной матрицы?

Принцип обработки разреженных матриц состоит в том, чтобы работать только с ненулевыми элементами матрицы.

4. В каком случае для матриц эффективнее применять стандартные алгоритмы обработки матриц? От чего это зависит?Если матрица содержит большое кол-во ненулевых элементов, то эффективней применять стандартные способы обработки матриц.

Выбор способа применения зависит от размера матрицы, процента разреженности матрицы и реализации способа обработки.