Условие задачи

Реализовать алгоритмы обработки разреженных матриц, сравнить эффективность использования этих алгоритмов (по времени выполнения и требуемой памяти) со стандартным алгоритмами обработки матриц при различном процентном заполнения матриц ненулевыми значениями и при различных размерах матриц.

Техническое задание

Смоделировать операцию умножения вектора строки и матрицы хранящихся

а) в разреженном виде; б) в стандартном виде. Сравнить время выполнения и объем занимаемой памяти при использовании 2-х этих способов хранения.

Разреженная матрица должна хранится в виде:
- 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.

Входные данные

Количество строк и столбцов матрицы, количество ненулевых элементов матриц и вектора строки, их индексы в матрице или векторе, вариант заполнения матриц (случайный или с клавиатуры).

Выходные данные

Результат умножения вектора строки на матрицу, результаты сравнения 2-х алгоритмов обработки.

Возможные аварийные ситуации

Некорректный ввод.

Структуры данных

Структура разреженной матрицы:

```
typedef struct
{
   int xsize;
   int ysize;
   int curr_size;

   int *A;
   int *JA;
   list_t IA;
} matrix_t;;
```

xsize и *ysize* — количество столбцов и строк матрицы;
curr_size – количество ненулевых элементов;
A — указатель на массив содержащий все ненулевых элементы;

JA — указатель на массив содержащий номера столбцов для каждого элемента массива A;
IA – связанный список, который содержит для каждого первого элемента i-ой строки его индекс в массивах A и JA.

Структура связанного списка:

```
typedef struct node
{
   int start_row_ind;
   struct node *next_element;
} node_t;
```

node_t — сама структура очередного элемента списка.
start_row_ind — индекс первого элемента i-ой строки матрицы в массивах A и JA;
next_element — указатель на следующий элемент связанного списка.

Структура стандартной матрицы:

```
typedef struct std_matr
{
   int **matrix;
   int xsize;
   int ysize;
} std_matrix_t;
```

matrix — указатель на матрицу;
xsize и *ysize* — количество столбцов и строк матрицы.

Алгоритм

Матрица транспонируется. Далее, умножение происходит вектор на вектор столько раз, сколько в *исходной* матрице столбцов. Для этого, я прохожу по массиву ненулвых элементов, и пользуясь индексами столбцов этих элементов из массива IA, последовательно перемножаю каждый элемент текущий строки на нужный элемент вектора-строки.

Тесты

Время

5% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	1487	4439
100x100	21843	316411
500x500	374378	8060332

10% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	2425	4777
100x100	36291	345510
500x500	626417	7629614

20% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	2703	4516
100x100	57357	315133
500x500	1205327	7598031

30% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	2946	4550
100x100	85709	330380
500x500	1606860	7657904

40% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	3316	4678
100x100	104210	315015
500x500	2106324	7154358

50% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	3696	4670
100x100	135812	330841
500x500	2918496	7315635

100% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	4125	4893
100x100	258291	329299
500x500	4540212	6215850

Занимаемая память

1% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	168	400
100x100	2400	40000
500x500	28000	1000000

10% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	240	400
100x100	9600	40000
500x500	208000	1000000

50% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	560	400
100x100	41600	40000
500x500	1008000	1000000

100% заполнения

Размер матрицы	Разреженная матрица	Обычная матрица
10x10	960	400
100x100	81600	40000
500x500	2008000	1000000

Выводы по проделанной работе

Использование алгоритмов хранения и обработки разреженных матриц выгодно при большом количестве элементов, примерно до 40-45% 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 во времени, но все еще выигрывает во времени. Даже при 100% заполненности матриц этот алгоритм выигрывает во времени у обычного, но занимает места в памяти в более чем в 2 раза 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 формата, требуется 3 * K ячеек памяти, где K – количество ненулевых элементов.

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 можно лучше использовать алгоритм обработки разреженных матриц, так как он, хоть и немного, но выигрывает во времени даже на больших % заполненостях матриц (80% - 100%)