|  |  |
| --- | --- |
|  | **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)** |

Отчет по лабораторной работе № 3

«Разреженные матрицы»

Работу выполнил:

студент группы ИУ7-32Б

Богатырев И.С.

Работу проверил:

Барышникова М.Ю.

Никульшина Т.А.

Москва, 2020 г.

**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 sparse\_matrix

{

int rows;

int columns;

long int curr\_size;

int \*A;

int \*JA;

linked\_list\_t IA;

} sparse\_matrix\_t;

rows и columns — количество строк и столбцов 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 А;

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\_ptr;

} node\_t;

node\_t — сама структура очередного элемента списка.

start\_row\_ind — индекс первого элемента i-ой строки матрицы в массивах А и JA;

next\_element\_ptr — указатель на следующий элемент связанного списка.

**Структура стандартной матрицы**

typedef struct matrix

{

int \*\*matrix;

int rows;

int columns;

} matrix\_t;

matrix — указатель на матрицу;

rows и columns — количество строк и столбцов 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**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10 | 1487 | 4439 |
| 100 | 21843 | 316411 |
| 1000 | 374378 | 8060332 |

**10% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 2425 | 4777 |
| 100x100 | 36291 | 345510 |
| 500x500 | 626417 | 7629614 |

**20% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 2703 | 4516 |
| 100x100 | 57357 | 315133 |
| 500x500 | 1205327 | 7598031 |

**30% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 2946 | 4550 |
| 100x100 | 85709 | 330380 |
| 500x500 | 1606860 | 7657904 |

**40% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 3316 | 4678 |
| 100x100 | 104210 | 315015 |
| 500x500 | 2106324 | 7154358 |

**50% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 4096 | 4670 |
| 100x100 | 135812 | 330841 |
| 500x500 | 2918496 | 7315635 |

**100% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 4725 | 4893 |
| 100x100 | 308291 | 329299 |
| 500x500 | 5840212 | 6215850 |

**Занимаемая память**

**1% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 168 | 400 |
| 100x100 | 2400 | 40000 |
| 500x500 | 28000 | 1000000 |

**10% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 240 | 400 |
| 100x100 | 9600 | 40000 |
| 500x500 | 208000 | 1000000 |

**50% заполнения**

| **Размеры матрицы** | **Разреженная матрица** | **Обычная матрица** |
| --- | --- | --- |
| 10x10 | 560 | 400 |
| 100x100 | 41600 | 40000 |
| 500x500 | 1008000 | 1000000 |

**100% заполнения**

| **Размеры 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 по памяти, 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 моем случае 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%)