**Типы и структуры данных**

**Лабораторная работа №5**

**Обработка разреженных матриц**

**Выполнил**: Кузнецов Александр

**Группа**: ИУ7-33

**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.

Разработать программу умножения или сложения разреженных матриц. Предусмотреть возможность ввода данных, как с клавиатуры, так и использование заранее подготовленных данных. Матрицы хранятся и выводятся в форме трех объектов. Для небольших матриц можно дополнительно вывести матрицу в виде матрицы. Величина матриц - любая (допустим, 1000\*1000). Сравнить эффективность (по памяти и по времени выполнения) стандартных алгоритмов обработки матриц с алгоритмами обработки разреженных матриц при различной степени разреженности матриц.

**Вариант 13 (задача 2)**

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 сложения двух матриц, хранящихся в этой форме, с получением результата в той же форме.

2. Произвести операцию сложения, 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.

**Входные данные:**

Номер выполнения команды;

Целые числа элементы матрицы;

**Пример:**

//ввод команды для матрицы

>>1

//ввод целого числа для матрицы

(int) >> 13

**Выходные данные**

Стандартный алгоритм и алгоритм на списках: Матрица А, Матрица B, сумма матриц А + B;

Сравнение времени работы алгоритмов.

**Функция программы**

Сложение двух разреженных матриц двумя алгоритмами (стандартное сложение и сложение с помощью списка) и сравнение времени работы этих алгоритмов.

**Аварийные ситуации**

**-** Превышена максимальная ширина и высота матрицы.

**-** Вводимые элементы матрицы отличны от типа Integer.

**Описание использованных структур данных**

Узел

struct Node{

int n;

struct Node \*next;

};

Список

typedef struct{

struct Node \*first;

struct Node \*last;

} List;

Матрица

typedef struct{

int \*A

int \*IA;

List JA;

} Matrix;

**Алгоритм**

**Обработка ввода:**

1. Вывести интерфейс программы
2. Проверить корректность ввода(Интерфейс)
3. Запустить выбранную команду

-//////- Вывод сообщение об ошибке, если она есть -//////-

Иначе

**Обработка операций над матрицей:**

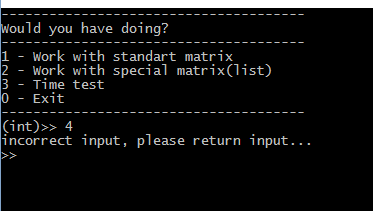
1. Стандартный алгоритм сложения матриц
   1. Ввод данных из файла
   2. Ввод данных с клавиатуры
2. Алгоритм с использованием списка
   1. Ввод данных из файла
   2. Ввод данных с клавиатуры
3. Расчет времени

**Обработка времени:**

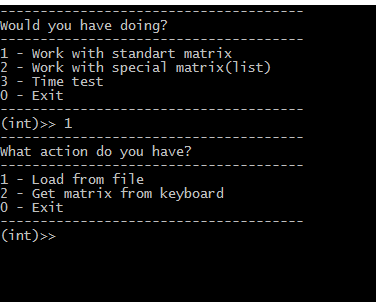
1. Расчет среднего времени для стандартного алгоритма
2. Расчет среднего времени для алгоритма, основанного на списках

**Тесты**

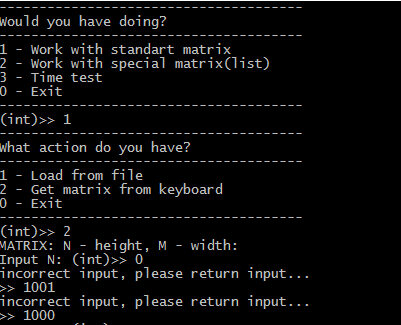
**Тест ввода:**Некорректный ввод команды матрицы:



Корректный ввод номера операции над матрицей:



Тестирование границ матрицы:



**Сравнение методов**

Измерение времени при фиксированном размере(1000x1000) и изменяемом проценте заполнения матрицы:

|  |  |  |
| --- | --- | --- |
| Elements != 0 (%) | Standard method (ms) | Special method (ms) |
| 10 | 13.6 | 11.8 |
| 20 | 14.7 | 24.4 |
| 30 | 13.6 | 35.1 |
| 40 | 14.1 | 45.0 |
| 50 | 14.9 | 55.7 |
| 60 | 14.5 | 61.8 |
| 70 | 14.8 | 66.09 |
| 80 | 14.8 | 70.2 |
| 90 | 13.7 | 73.0 |
| 100 | 14.0 | 75.5 |

Измерение времени при изменяемом размере и фиксированном проценте заполнения матрицы (50%):

|  |  |  |
| --- | --- | --- |
| Size of matrix (n x m) | Standard method (ms) | Special method (ms) |
| 10x10 | 0 | 0 |
| 50x50 | 0 | 0 |
| 100x100 | 0 | 0.21 |
| 200x200 | 1 | 1.1 |
| 400x400 | 1 | 11.58 |
| 800x800 | 8.61 | 38.4 |
| 1000x1000 | 13.29 | 57.8 |

**Вывод:**

Отсюда видно, что метод для разреженных матриц работает быстрее обычного метода при большом количестве нулей в матрице, но медленнее при малом количестве нулей, т.к. создается список и 2 вектора большой длины, указывающие на каждый элемент матрицы, т.е. расход по памяти получается большой поэтому для разреженных матриц работает медленно. То есть, допустим при 100% заполненной матрице 10x10 в обычном методе хранится 100 элементов, а в методе разреженной матрицы 300. Отсюда видно, что метод разреженных матриц хорошо работает только при большом количестве нулей, т.к. будет хранить наименьшее количество значений.

**Вопрос – Ответ**

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

**Разрежённая матрица** — это **матрица** с преимущественно нулевыми элементами. В противном случае, если большая часть элементов **матрицы** ненулевые, **матрица** считается плотной. Среди специалистов нет единства в определении того, какое именно количество ненулевых элементов делает **матрицу разрежённой**.

Можно хранить матрицу, используя кольцевой связный список, двунаправленные стеки и очереди. Существует диагональная схема хранения симметричных матриц, а также связные схемы разреженного хранения.

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.

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

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.

Под обычную матрицу выделяется больше элементов, чем под разряженную, т.к.

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

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. 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.

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

При вычислении стандартным способом нужно N2 просмотров массива. Для сокращения алгебраических операций удобно во время работы хранить расширенный (по размерности массивов **a** и **b**) массив указателей IP (его начальное состояние - нулевое). Этот массив заполняется путем одного просмотра массива JA

Количество операций в алгоритме разряженной матрицы пропорционально числу ненулевых элементов, не считая засылки N нулей в массив IP.

Применение этого алгоритма особенно эффективно в ситуации, когда вектор нужно скалярно умножить на несколько векторов. В этом случае массив IP заполняется только один раз и затем используется для вычисления всех требуемых скалярных произведений. Эта ситуация возникает при необходимости перемножить разреженную матрицу и разреженный вектор.