Тюпов Артем, ИУ7-32

Типы и структуры данных

Лабораторная работа 5

Работа с разреженными матрицами

Вариант

**Техническое задание**

**Задача**: 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.

Разреженная матрица хранится в форме 3-х объектов:

- вектор A содержит значения ненулевых элементов;

- вектор JA содержит номера столбцов для элементов вектора A;

- связный список IA, в элементе N которого находится номер компонент в A и JA, с которых начинается описание N-й строки матрицы.

**Входные данные**: две матрицы указанных размеров, заполненные вручную пользователем или случайным образом.

**Выходные данные**: матрица, являющаяся результатом сложения первых двух; время, затраченное на сложение.

**Функция программы**: реализация сложения разреженных матриц; отображение их в заданном виде. Измерение времени на сложение матриц «классическим» и «разреженным» спообом.

**Аварийные ситуации**

1) Попытка сложить две матрицы разных размерностей

Будет выведено соответствующее сообщение

2) Попытка добавить элемент с индексами бОльшими, чем размер матрицы

Будет выведено соответствующее сообщение.

3) Попытка добавить элемент, не являющийся числом.

Будет выведено соответствующее сообщение.

**Используемые структуры**

float \***A** – вектор, в котором находятся значения ненулевых элементов

float \***JA** – вектор, в котором находятся номера строк для элементов вектора А;

struct IA // Список

{

int i; // Номер столбца

int Nk; // Номер компоненты

struct IA \*next; // Указатель на следующий элемент

};

**Функции и алгоритмы**

У пользователя запрашиваются размеры массивов, под них выделяется память. В случае, если размеры матриц не равны, выводится сообщение. После этого пользователь, по желанию, может заполнить матрицу вручную. Если нет, то она заполняется автоматически.

После задания матриц происходит суммирование двумя методами. Идет перебор ячеек массива, и они суммируются. При суммировании вторым методом идет переборка по списку структур и они суммируются.

После этого на экран выводится результирующая матрица. У пользователя запрашивается, нужно ли делать тест на время, если да, то на экран выводятся результаты теста.

После этого происходит высвобождение памяти, программа завершается.

int matrInput(int n, int m, float \*Matr, float \*A, int \*JA, struct IA \*IA, int \*lenA)

int create\_matrix(float \*matr, int n, int m, int fill, int \*count)

void matrDefault(float \*Matr, int n, int m,float \*A, int \*JA, struct IA \*IA, int \*lenA)

void print(int n, int m, float \*Matr, float \*A, int \*JA, struct IA \*IA, int lenA)

int printmatrix(float \*matr, int n, int m)

int Plus\_R(int n, int m, float \*A1, float \*A2, int \*JA1, int \*JA2, struct IA \*IA1, struct IA \*IA2, int lenA1, int lenA2, float \*A3, int \*JA3, struct IA \*IA3, int \*lenA3)

int summtrix(float \*matr1, float \*matr2, float \*matr3,int n,int m)

int time\_test(int fill)

**Тесты**

* Попытка сложить две матрицы разных размерностей – сообщение об ошибке: **Diffrent size of matrixes**.
* Попытка добавить элемент с индексами большими, чем размер матрицы – сообщение об ошибке: **Index out of range! Try again!**.
* Попытка добавить элемент, не являющийся числом – сообщение об ошибке **It isnt number! Try again!**.

**Сравнение времени сложения**

При использовании хотя бы 10.000 х 10.000 матриц, на их заполнение уходит почти столько же времени, сколько и на сложение – поэтому исследование проводилось на 1000 х 1000 матрицах.

По результатам исследования, получилось следующее соотношение % заполнения, сложения «классическим» способом и сложения «разреженным» способом:

Время работы алгоритмов (в тиках) сложения при заполненности матрицы в 5%:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10х10 | 100х100 | 1000х1000 |
| Разреженный способ | 11550 | 233524 | 11288540 |
| Стандартный способ | 1736 | 161252 | 12983924 |

Время работы алгоритмов (в тиках) сложения при заполненности матрицы в 15%:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10х10 | 100х100 | 1000х1000 |
| Разреженный способ | 9240 | 247272 | 12706668 |
| Стандартный способ | 5006 | 119756 | 11674410 |

Время работы алгоритмов (в тиках) сложения при заполненности матрицы в 25%:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10х10 | 100х100 | 1000х1000 |
| Разреженный способ | 10942 | 265524 | 14817048 |
| Стандартный способ | 3322 | 121452 | 11628182 |

Время работы алгоритмов (в тиках) сложения при заполненности матрицы в 40%:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10х10 | 100х100 | 1000х1000 |
| Разреженный способ | 10956 | 233348 | 17684658 |
| Стандартный способ | 4528 | 141620 | 11920678 |

Увеличение времени на «классическое» сложение можно обосновать внутренней оптимизацией вычислений Си.

Таким образом, использование схемы разреженного построчного хранения матрицы даёт выигрыш во времени до двух раз.

Рассмотрим матрицу M x N элементов размером S каждый. Пусть в матрице P << M\*N элементов. Схема разреженного построчного хранения предполагает использование двух массивов из P элементов по S байт каждый (массивы AN и JA), и одного массива M положительных целых чисел (массив IA). «Классическая» матрица будет занимать M \* N \* S байт; разреженная: (M \* размер целого) \* 2 \* P.

Для примера, рассмотрим матрицу 4байтовых чисел размерами 3000 х 1000 (3.000.000 элементов), в которой лишь 1% (30.000 элементов) ненулевых; целое число для разреженной матрицы занимает 2 байта.

В «классическом» представлении матрица будет занимать 3.000.000 \* 4 байт =  
12.000.000 байт = 12 Мбайт.

В разреженном же, (3000 \* 2) \* 2 \* 30.000 \* 4 байт = 1.440.000 байт.

Таким образом, использование схемы разреженного хранения может давать выигрыш как по памяти, так и по времени при низком (<10%) заполнении матрицы. Тем не менее, с увеличением процента заполнения эффективность алгоритма может резко снижаться: так, в вышеописанном примере уже при 10% заполнения разреженная матрица будет занимать уже 14.4 Мбайт, и 28.8 Мбайт при использовании 4байтовых целых для хранения массива IA.

**Контрольные вопросы**

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

Разреженная – матрица, содержащая достаточно большое количество элементов, из которых лишь малая часть является ненулевыми (n^(1+g) для матрицы размерности n, g<1).

Простейшая схема хранения разреженной матрицы: хранить массив ненулевых элементов (AN), и два массива их «координат» (I, J) - номера столбцов и строк, в которых они расположены.

Кнут предложил дополнить эту схему также массивами NR (содержит номер из AN следующего ненулевогj элемента, расположенного в матрице по строке) и NC (номера –‘’- по столбцу), а также массивы JR и JC (указатели для входа в строку и столбец). Данная схема хранения избыточна, но позволяет легко осуществлять все матричные операции.

Чанг и Густавсон предложили схему разреженного строчного формата: хранятся массивы AN и J, а массив IA содержит номера (в AN) элементов, с которых начинается очередная строка матрицы.

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

Количество памяти, выделяемой под хранение обычной матрицы, определяется количеством её элементов (включая нулевые) и размером одного элемента, M \* N \* S. Память же под разреженную матрицу выдляется в зависимости от типа хранения (см. пример в тестах). В то время как при формировании обычной матрицы выделяется один блок памяти на всю матрицу, при формировании разреженной памяти память выделяется по мере наполнения её ненулевыми элементами.

*3. Каков принцип обработки разреженной матрицы?*

Обработка разреженной матрицы предполагает работу только с ненулевыми элементами и зависит от схемы хранения матрицы, и типа операции.

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

Эффективность стандартных\разреженных алгоритмов обратно пропорциональна проценту «наполненности» матрицы. Чем больше в матриц ненулевых элементов, тем меньше выигрыш во времени и памяти; при превышении определённого уровня «наполненности» разреженные алгоритмы начинают давать даже более худшие результаты, нежели стандартные (см. пример).