Лабораторная работа №5

Обработка разреженных матриц

Цель работы: реализация алгоритмов обработки разреженных матриц, сравнение этих алгоритмов со стандартными алгоритмами обработки матриц.

Задание (Вариант 5):

Разреженная (содержащая много нулей) 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 Nk которого находится номер компонент в ***A*** и ***JA***, с которых начинается описание строки 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.

Требования к входным данным:

Программа предназначена для работы с произвольными матрицами, размерность которых меньше 1000*1000. Элементы матриц - целые сичла, не превосходящие 1000 (ограничение сделано в целях избежания переполнения типа при умножении).

Интерфейс программы:

```
************************************************************
Выберите одно из следующих действий:
0: Работа с матрицами
1: Сравнение скорости работы стандартного алгоритма и работы с разряженными матрицами
2: Exit
```

0 - Работа с матрицами предполагает ввод/вывод двух матриц, их умножение и просмотр результата

Сначала пользователь должен ввести вектор-строк, затем матрицу. Каждый раз у него есть возможность выбрать ручной и автоматический режим заполнения.

Сначала необходимо ввести кол-во столбцов.

В случае выбора автозаполнения необходимо будет выбрать процент заполнения (вещественное число от 0 до 100), и диапазон значений (целые числа от -1000 до 1000), при этом первое число должно быть не больше второго.

При выборе ручного заполнения необходимо вводить 3 числа: индекс строки и столбца (начиная с нуля). Программа отслеживает попытку записать вне области выделенной памяти (выход за предел по кол-ву строк/столбцов).

```
************************************************************
Данная программа умножает вектор-строку на матрицу:
Ввод вектора-строки
Введите кол-во столбцов [1, 1000]: 500
Выберите ввод ручной(1) или автоматический(2)
2
Введите процент заполнения(вещественное число) [0, 100]: 34
Выберите диапазон значений [-1000, 1000]:
Начало диапазона значений:
-100
Конец диапазона значений:
100
Выберите ввод ручной(1) или автоматический(2)
1
Введите по 3 числа для каждого ненулевого эл-та (номер строки, столбца начиная с нуля, значение)
 2 4 3
Продолжить (1-нет)? 0
6 4 5
Продолжить (1-нет)? 0
7 2 4
Продолжить (1-нет)? 0
9 6 334
Продолжить (1-нет)? 1
```

Далее программа предложит проанализировать введенные данные и вычислить результат умножения матриц.

```
0: Показать первую матрицу в разреженном формате
1: Показать вторую матрицу в разреженном формате
2: Показать результат умножения в разреженном формате
3: Показать первую матрицу в стандартном формате
4: Показать вторую матрицу в стандартном формате
5: Показать результат в стандартном формате
6: Показать результат работы стандартного алгоритма
7: Закончить работу с введенной парой матриц
```

0,1,2 - показать матрицы в разреженном формате. Например, матрица, введенная выше в ручном режиме будет иметь вид:

```
1
n m : 35 34
A : 3 5 4 334
JA: 4 4 2 6
IA_list: 2147483647 2147483647 0 1 1 1 1 2 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

Так как первые две строки нулевые, то первые значения списка заполняются значением бесконечность.

3,4,5 - показать матрицы в стандартном формате. Пример:

```
0 0 0 0 0
0 0 0 0 0
0 3 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
5 0 0 0 0
0 4 0 0 8
```

6 - Показать результат работы стандартного алгоритма (для визуального сравнение полученного результата).

7 - Вернуться в исходное меню

1 - Сравнить время выполнения стандартного алгоритма и работы сч разреженными матрицами

```
Введите кол-во строк и столбцов первой матрицы и кол-во столбцов второй [1, 1000]: 100 300 150
Введите процент заполнения(вещественное число) [0, 100]: 45
Время умножения в разреженном виде: 75030
Время умножения в стандартном виде: 31570
Память, необходимая для хранения стандартной матрицы 300000
Память, необходимая для зранения разряженной матрицы 30150
```

Программа предлагает ввести размеры матриц (кол-во строк второй матрицы равно кол-во столбцов первой). Все размеры - целые числа, лежащие в диапазоне от 1 до 1000. Далее необходимо ввести процент заполнения для данных матриц (вещественное число от 0 до 100)

В качестве результата программа выдает время работы различных алгоритмов в мкс, а также память в байтах, которая потребовалась для хранения результата.

Обращение к программе:

Через консоль

Внутренние структура данных:

Структура списка:

```
typedef struct Node {
   request value;
   Node* next;
} my_list;
```

Функции для работы со списками:

```
my_list *create_node(int x);
my_list *copy_list(my_list *head);
my_list *add_to_list(my_list *tmp, my_list *head);
void free_all(my_list *head);
void print_list(my_list *head);
```

Класс разреженной матрицы

```
class CSparse_matrix {
private:
   vector <int> A;
   vector <int> JA;
   int NA;
   int n;
   int m;
   my_list* IA_head = NULL;
public:
   CSparse_matrix();
   //создание разреженной матрицы из стандартной
   CSparse_matrix(int narr, int marr, int **arr);
   CSparse_matrix(const CSparse_matrix &obj);
   void transposition(); // траспонирование матриц
   CSparse_matrix operator* (CSparse_matrix &obj);
   CSparse_matrix& operator=(const CSparse_matrix& obj);
   ~CSparse_matrix();
   void show();
   int memory(); // Определение памяти, выделяемой под матрицу
   int** to_standart(); // приведение матрицы к станлартному виду
};
```

Сравнение эффективности реализаций:

При тестировании производилось умножение двух квадратных матриц размера n.

№	n - размер матриц n*n	Процент заполнения	Время работы, мкс		Объем задействованной памяти, байт	
			стандарт	разреж.	стандарт	разреж.
1	100	0	9030	132	40000	400
2		25	7294	14634		17984
3		50	7300	18809		31928
4		75	7337	29295		42640
5		100	7757	28475		50576
6	300	0	228261	1697	360000	1200
7		10	216344	153535		70040
8		20	222996	232040		134464
9	500	0.01	1327499	1544	1000000	2200
10		0.05	1408246	3212		3000
11		0.25	2497101	11056		7000
12		1.25	2001818	104873		26880
13		6.25	1300078	422129		123448

Вывод:

Стандартный алгоритм хранения матриц предполагает постоянное количество выделенной памяти, тогда как память, выделенная для разряженного способа хранения напрямую зависит от кол-ва ненулевых эл-тов. Тесты показали, что при проценте заполнения матрицы до 50 % разряженный способ позволяет значительно уменьшить кол-во память. При увеличении процента заполнения он уступает стандартной реализации, так как предполагает хранение дополнительно индексы столбцов.

Время выполнение стандартного алгоритма также постоянно и линейно зависит от кол-ва эл-тов в матрице. Разряженный алгоритм показывает более высокую скорость работы при проценте заполнения 15% и ниже. Низкая скорость работы по сравнению со стандартной реализаций для заполненных матриц обусловлена необходимостью перемещения по списку строк (двух матриц одновременно), что усложняет доступ к эл-ту по индексу.

Таким образом, при заполнении матриц не более чем на 15 % целесообразней использовать хранение и проводить операции в разреженном формате. Это поволит повысить эффективность работы программы и по времени, и по объему

используемой памяти. В противном случае, следует использовать стандартные способы обработки матриц.

Теоретическая часть:

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 ненулевого j 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 иJ, 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 ее элементов (включая нулевые) и размером одного элемента, M * N * Size_of_elem. Память же под разреженную матрицу выделяется в зависимости от используемой схемы хранения. При формировании разреженной памяти память выделяется по мере наполнения ее ненулевыми элементами и напрямую зависит от их количества.

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 определенного уровня «наполненности» разреженные алгоритмы начинают давать даже более худшие результаты, нежели стандартные.