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

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

Название предмета: Типы и структуры данных

Студент: Варламова Екатерина Алексеевнa

Группа: ИУ7-31Б

*2020г.*

1. **Описание условия задачи**

Разработать программу умножения разреженнной матрицы на вектор-строку. Предусмотреть возможность ввода данных, как с клавиатуры, так и использования заранее подготовленных данных. Решить задачу при стандартном хранении матриц и при следующем:

- 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.).

Сравнить время выполнения операций и объем памяти при использовании 2-х алгоритмов при различном проценте заполнения матриц.

**Техническое задание**

1. ***Описание исходных данных***

Программа ожидает:

- размеры матрицы n и m (n считается и длиной вектора-строки).

- номер действия (подробнее о том, какой номер соответствует определённому действию, будет выведено при запуске программы). При этом возможны следующие варианты ввода матрицы и вектора:

* ввод матрицы и вектора полностью;
* частичный ввод матрицы и вектора (с указанием только ненулевых элементов);

Формат ввода:

1. При вводе полностью

* для матрицы 3 на 3:

2 3 0

0 2 0

1 2 0

* для вектора из 5 элементов:

0 1 2 0 2

1. При частичном вводе (ввод заканчивается 0):

* Для матрицы 3 на 3:

row of element: 1

column of element: 3

element: 2

row of element: 2

column of element: 1

element: 8

row of element: 3

column of element: 1

element: 5

row of element: 3

column of element: 3

element: 5

row of element: 0

ИТОГ:

0 0 2

8 0 0

5 0 5

* Для вектора из 5 элементов (ввод заканчивается 0):

column of element: 1

element: 5

column of element: 2

element: 1

column of element: 4

element: 10

column of element: 0

ИТОГ:

5 1 0 10 0

1. ***Описание результата программы***

Результатом работы программы могут являться (в зависимости от введённого действия):

1. Вектор-столбец, являющийся произведением матрицы на вектор-строку (в зависимости от алгоритма будет выведен в различных форматах).
2. Статистика по времени выполнения и объёму памяти при использовании 2-х алгоритмов при различном проценте заполнения матриц.

Формат вывода:

1. Вектор-столбец будет выведен полностью (перечислены все элементы с 1 по последний), если матрица и вектор хранятся стандартным образом. Размерности матрицы не превышают 30.

Пример для 5 элементов:

0 1 4 0 2

1. Вектор-столбец будет выведен, если матрица и вектор хранятся стандартным образом. Размерности матрицы превышают 30.

Пример для 31 элемента:

column of element: 2

element: 7

column of element: 4

element: 6

column of element: 19

element: 10

1. Вектор-столбец будет выведен частично (только ненулевые элементы и номера строк, в которых расположены ненулевые элементы). Пример для 5 элементов:

data

1 4 2

rows

2 3 5

1. Статистика в форме таблицы.
2. ***Описание задачи, реализуемой программой***

Программа в зависимости от запросов пользователя производит ряд действий: получает матрицу и вектор выбранным способом, считает их произведение при любом из двух способов хранения и выводит результат в зависимости от способа хранения. Кроме того, выводит статистику.

1. ***Способ обращения к программе***

Способ обращения к программе - консольный. Дальнейшие инструкции будут выведены после запуска.

1. ***Описание возможных аварийных ситуаций и ошибок пользователя***

* ошибки ввода размерностей матрицы и вектора (отрицательные), а также значений, которыми они заполняются (недостаточно данных);
* отказ операционной системы выделить запрашиваемую память;

Во всех указанных случаях программа сообщит об ошибке.

1. **Описание внутренних структур данных**
2. *Основная структура данных*

Основной структурой данных в программе является матрица, которая хранится следующим образом:

- 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.). Описание структуры на языке C выглядит следующим образом:

typedef struct

{

vector\_t data;

vector\_t rows;

vector\_t columns;

size\_t n;

size\_t m;

} sparse\_matrix\_t;

А также основной структурой является разреженный вектор:

typedef struct

{

vector\_t data;

vector\_t columns;

size\_t n;

} sparse\_vector\_t;

*Вспомогательная структура данных*

Вспомогательной структурой данных является вектор. Данная структура динамически расширяется (умножением на 2) при необходимости, изначально её размер = 20 элементов.

Её описание представлено на языке C:

typedef struct

{

void \*front;

size\_t size\_of\_element;

size\_t size;

size\_t alloc\_size;

} vector\_t;

1. **Описание алгоритма**

Для решения задачи вычислить произведение вектора-строки на матрицу используется два подхода:

1. Стандартный

Матрица и вектор хранятся в памяти полностью (содержат как нулевые, так и ненулевые элементы). Для вычисления произведения матрица просматривается по столбцам и элементы столбцов поэлементно перемножаются с элементами вектора-строки. Результат записывается в вектор-столбец. Таким образом, оценка алгоритма O(n\*m).

1. Сжатый способ хранения

При данном подходе:

1. Заполняется массив storage длины, равной количеству строк в матрице. Просматривая разряженный вектор (сложность зависит от количества ненулевых элементов), находим столбец k (O(1)), в котором располагается текущий элемент. В массиве storage на место k ставим номер текущего элемента в разряженном векторе.
2. Для каждой строки просматриваем элементы и добавляем в соответствующую ячейку результата значение произведения. При этом произведение ищем с помощью массива storage.

***Оценка по времени***

Таким образом, сложность алгоритма зависит от количества ненулевых элементов в векторе и матрице (не считая заполнение нулями массив storage). То есть, если количество ненулевых элементов в матрице = n1, а в векторе n2, то сложность составит O(n1 + n2), что при n1 << n\*m и n2 << n будет в разы быстрее O(n\*m).

***Оценка по памяти***

Поскольку в основе структуры разреженнной матрицы и разреженного вектора лежит СД вектор, которая динамически расширяется при необходимости, то при сжатом способе хранения размер памяти под вектор и под матрицу зависят от количества ненулевых элементов. Пусть количество ненулевых элементов в матрице n1, количество ненулевых в векторе n2, размер элементов и размер индексов = k, тогда:

* при стандартном способе хранении общее количество памяти под вектор и матрицу = (n\*m\*k) + (n\*k) = n\*k(m + 1).
* при сжатом способе хранения общее количество памяти под вектор и под матрицу = (n1 \* k + n1 \* k+ m \* k) + (n2 \* k + n2 \* k) = k(2\*n1 + 2\*n2 + m). При этом дополнительно необходим массив, размер которого: n\*k. Суммарно: k(n + m + 2\*n1+2\*n2).

Очевидно, что при n1 << n\*m и n2 << n памяти при сжатом способе хранения требуется меньше, чем при стандартном.

***Итог***

Из приведённых выше рассуждений очевидно, что при маленьком проценте заполнения матрицы (при большом количестве нулевых элементов) выгоднее и памяти и по времени использовать сжатый способ хранения. При достаточно заполненной матрице при сжатом способе хранения возникают лишние затраты по времени и памяти, поэтому разумно использовать стандартный способ хранения.

***Проверка выводов экспериментально***

Возьмём матрицу размером 1000\*1000 и вектор из 1000 элементов.

Посмотрим, какой алгоритм выгоднее использовать при умножении вектора на матрицу по памяти и по времени при разном проценте заполнения объектов (первый столбец) и проверим наш теоретический вывод экспериментально.

В таблице время указано в миллисекундах (во 2 и 3 столбце), память в байтах (в 4 и 5 столбцах).

**x--------------x------------x------------x-------------x-------------x**

**| percent | ordinary | sparse | size\_or | size\_sp |**

**x--------------x------------x------------x-------------x-------------x**

**| 1| 11081.00| 146.00| 8008000| 176168|**

**x--------------x------------x------------x-------------x-------------x**

**| 6| 9776.00| 740.00| 8008000| 976968|**

**x--------------x------------x------------x-------------x-------------x**

**| 11| 9557.00| 1274.00| 8008000| 1777768|**

**x--------------x------------x------------x-------------x-------------x**

**| 16| 8960.00| 1988.00| 8008000| 2578568|**

**x--------------x------------x------------x-------------x-------------x**

**| 21| 9719.00| 3187.00| 8008000| 3379368|**

**x--------------x------------x------------x-------------x-------------x**

**| 26| 8737.00| 3510.00| 8008000| 4180168|**

**x--------------x------------x------------x-------------x-------------x**

**| 31| 8966.00| 4214.00| 8008000| 4980968|**

**x--------------x------------x------------x-------------x-------------x**

**| 36| 9049.00| 5504.00| 8008000| 5781768|**

**x--------------x------------x------------x-------------x-------------x**

**| 41| 8667.00| 6481.00| 8008000| 6582568|**

**x--------------x------------x------------x-------------x-------------x**

**| 46| 8559.00| 7046.00| 8008000| 7383368|**

**x--------------x------------x------------x-------------x-------------x**

**| 51| 9043.00| 8675.00| 8008000| 8184168|**

**x--------------x------------x------------x-------------x-------------x**

**| 56| 9209.00| 9482.00| 8008000| 8984968|**

**x--------------x------------x------------x-------------x-------------x**

**| 61| 9924.00| 11965.00| 8008000| 9785768|**

**x--------------x------------x------------x-------------x-------------x**

**| 66| 9428.00| 14474.00| 8008000| 10586568|**

**x--------------x------------x------------x-------------x-------------x**

**| 71| 9947.00| 15043.00| 8008000| 11387368|**

**x--------------x------------x------------x-------------x-------------x**

**| 76| 9572.00| 16573.00| 8008000| 12188168|**

**x--------------x------------x------------x-------------x-------------x**

**| 81| 10231.00| 18652.00| 8008000| 12988968|**

**x--------------x------------x------------x-------------x-------------x**

**| 86| 9999.00| 20573.00| 8008000| 13789768|**

**x--------------x------------x------------x-------------x-------------x**

**| 91| 9401.00| 21023.00| 8008000| 14590568|**

**x--------------x------------x------------x-------------x-------------x**

**| 96| 9723.00| 23958.00| 8008000| 15391368|**

**x--------------x------------x------------x-------------x-------------x**

Из таблицы видно, что наши выводы подтвердились. При небольшом проценте заполнения сжатый способ хранения дает очень хорошие результаты по памяти и по времени. При этом при 51% заполнения сжатый способ хранения уже проигрывает по памяти, а при 56% проигрывает как по памяти, так и по времени. При больших процентах сжатый способ начинает показывать результаты хуже и хуже. Так, при 96% накладные расходы по памяти и времени оцениваются в 2 раза.

**Тестирование**

1. Позитивные тесты

|  |  |  |
| --- | --- | --- |
| Входные данные | Описание теста | Результат |
| 0 0 4 10  6 0 0 0  0 0 3 0  0 4 0 8  0 8 0 0 | Проверка умножения при стандартном хранении | 48 0 0 0 |
| data:  4 10 6 3 4 8  columns:  2 3 0 2 1 3  pointers:  0 2 3 4 6  data:  8  columns:  2 | Проверка умножения при сжатом хранении | data:  48  columns:  1 |
| 5 4 2 2 6  4 3 3 9 3  8 10 7 9 9  5 6 9 5 2  7 4 2 7 10  5 0 0 0 0 | Проверка умножения при стандартном хранении | 25 20 10 10 30 |
| data:  5 4 2 2 6 4 3 3 9 3 8 10 7 9 9 5 6 9 5 2 7 4 2 7 10  columns:  0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4  pointers:  0 5 10 15 20 25  data:  5  columns:  1 | Проверка умножения при сжатом хранении | data:  25 20 10 10 30  columns:  1 2 3 4 5 |

1. **Выводы по проделанной работе**

В ходе работы был проведён сравнительный анализ подходов к хранению и обработке разреженных матриц и векторов (на примере операции умножения), в результате которого установилось, что при маленьком проценте заполнения объектов (при большом количестве нулевых элементов) выгоднее по памяти и по времени использовать сжатый способ хранения. При достаточно заполненных объектах при сжатом способе хранения возникают лишние затраты по времени и памяти, поэтому разумно использовать стандартный способ хранения.

1. **Ответы на вопросы**

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

знаете?

Разреженная матрица – матрица, содержащая преимущественно нули. Хранить можно стандартно (все элементы) и с помощью следующей схемы:

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

См. пункт “Оценка по памяти” в описании алгоритмов.

1. Каков принцип обработки разреженной матрицы?

См. пункт “Сжатый способ хранения” в описании алгоритмов.

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

См. пункт “Итог” в описании алгоритмов.