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

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 «Обработка разреженных матриц»

Студент Ланкин Дмитрий Леонидович

Группа ИУ7 – 34Б

Принял(-а) Барышникова Марина Юрьевна

2022 г.

Оглавление

Описание условия задачи

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 и при различных размерах матриц.

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.

Техническое задание

Входные данные

Матрица (разреженная или стандартная), размера **M*N**. В случае разреженной матрицы — векторы с ненулевыми значениями матрицы, номерами столбцов и строк.

Каждая разреженная матрица содержит:

— массив A, хранящий ненулевые элементы матрицы;

— массив JA, хранящий номера столбцов соответственных элементов;

— массив IA, хранящий индексы элементов, с которых начинается каждая новая строка матрицы;

— количество строк матрицы;

— количество столбцов матрицы;

— количество ненулевых элементов матрицы.

Каждая стандартная матрица содержит:

— двумерный массив, хранящий элементы матрицы;

— количество строк матрицы;

— количество столбцов матрицы.

Правила ввода матриц:

— в случае самостоятельного заполнения матрицы необходимо вводить элементы последовательно в формате:

ЭЛЕМЕНТ НОМЕР_СТРОКИ НОМЕР_СТОЛБЦА;

— количество строк и столбцов не должно превышать 1000 и быть меньше 0;

— Процент заполнения находится в диапазоне от 1 до 100;

—

Выходные данные

Вектор-строка, являющаяся результатом умножения исходного вектора-строки и матрицы.

Описание задачи

Программа получает от пользователя номер необходимой ему команды и обрабатывает данные в зависимости от задачи пользователя.

Список команд

1. Смоделировать операцию умножения вектора-строки и матрицы, хранящихся в форме разреженных матриц.
2. Произвести операцию умножения, применяя стандартный алгоритм работы с матрицами.
3. Сравнить время выполнения операций и объём памяти при использовании этих 2-х алгоритмов при различном заполнении матриц.

0. Завершить работу программы.

Способ обращения к программе

Программа собирается с помощью утилиты «make» и вызывается пользователем из консоли.

Описание возможных аварийных ситуаций и ошибок пользователя

1. Невозможно выделить память для динамического типа данных.

   Сообщение об ошибке: «Невозможно выделить память.»
2. Некорректный ввод.

   Сообщение об ошибке: «Некорректный ввод.»
3. Указана несуществующая команда.

   Сообщение об ошибке: «Такой команды не существует!»
4. Указан некорректный метод заполнения матрицы.

Сообщение об ошибке: «Введите корректные данные!»

5. Невозможно умножить вектор-строку на матрицу.

   Сообщение об ошибке: «Ошибка умножения.»

Внутренняя структура данных

Необходимые для составления матрицы типы данных составляют следующие структуры:

```
 1  typedef struct
 2  {
 3      int *vector_a;
 4      int *vector_ja;
 5      int *list_ia;
 6      unsigned int rows;
 7      unsigned int cols;
 8      unsigned int non_zero_nums;
 9  } sparse_matrix;
10
11  typedef struct
12  {
13      unsigned int rows;
14      unsigned int cols;
15      int **matrix;
16  } std_matrix;
```

Листинг 1. Описание структуры разреженной и стандартной матриц.

— vector_a - массив элементов типа int для хранения ненулевых элементов;

— vector_ja — массив элементов типа int для хранения номеров столбцов соответственных элементов;

— list_ia — массив элементов типа int для хранения индексов чисел, с которых начинается каждая новая строка матрицы;

— rows — беззнаковое число, хранящее количество строк матрицы;

— cols — беззнаковое число, хранящее количество столбцов матрицы;

— non_zero_nums — беззнаковое число, хранящее количество ненулевых элементов;

— matrix — двумерный массив типа int, хранящий числа стандартной матрицы.

Особенности реализации алгоритма

1. Программа получает от пользователя необходимую команду по её номеру.
2. Пользователь выбирает необходимый размер матрицы и способ её заполнения.
3. В случае команд 1, 2 программа выполняет умножение вектора-строки на матрицу. В случае команды 3 осуществляет сравнительный замерный эксперимент, демонстрирующий время выполнения и количество затраченной памяти.

Описание функций

- **`size_t sparse_matrix_fill(sparse_matrix *matr);`**

  Описание: функция осуществляет заполнение разреженной матрицы путем самостоятельного ввода чисел пользователем.

  Входные данные: экземпляр структуры sparse_matrix, переданный по указателю.

  Выходные данные: заполненный экземпляр данной структуры, код возврата.

- **`size_t sparse_rndm_fill(sparse_matrix *matr, int percentage);`**

  Описание: функция осуществляет псевдослучайное заполнение разреженной матрицы с учетом процента её разреженности.

  Входные данные: экземпляр структуры sparse_matrix, переданный по указателю, и процент заполнения матрицы.

  Выходные данные: заполненный экземпляр данной структуры, код возврата.

- **`size_t sparse_matrix_handler(sparse_matrix *matr, unsigned int *list_len, size_t input_method);`**

  Описание: функция осуществляет полноценную обработку разреженной матрицы, в т.ч. и её заполнение в зависимости от метода, выбранного пользователем.

  Входные данные: экземпляр структуры sparse_matrix, переданный по указателю, длина массива IA и метод заполнения матрицы.

  Выходные данные: код возврата и подготовленная к умножению матрица.

- **`int **alloc_matrix(unsigned int rows, unsigned int cols);`**

  Описание: функция выделяет память под стандартную матрицу с помощью массива указателей и массива, хранящего числа.

  Входные данные: количество строк и столбцов матрицы.

  Выходные данные: указатель на выделенный участок памяти.

- **`void free_matrix(int **ptrs);`**

  Описание: функция освобождает выделенный под стандартную матрицу участок памяти.

  Входные данные: указатель на массив указателей.

- **`size_t std_matrix_fill(int **matrix, unsigned int rows, unsigned int cols, size_t type);`**

  Описание: функция реализует заполнение стандартной матрицы.

  Входные данные: двумерная матрица, количество строк и столбцов.

Выходные данные: код возврата, заполненная стандартная матрица.

– **`size_t random_fill_std_matrix(std_matrix *matrix, const int percentage);`**

Описание: функция заполняет стандартную матрицу в псевдослучайном порядке случайными числами.

Входные данные: экземпляр структуры std_matrix, переданный по указателю, процент заполнения матрицы.

Выходные данные: код возврата.

– **`size_t std_matrix_handler(std_matrix *matrix);`**

Описание: функция осуществляет обработку стандартной двумерной матрицы.

Входные данные: экземпляр структуры std_matrix, переданный по указателю.

Выходные данные: код возврата, подготовленная к умножению матрица.

– **`size_t sparse_mult(sparse_matrix sparse_matr, sparse_matrix sparse_row, sparse_matrix *row_res, unsigned int cols);`**

Описание: функция осуществляет умножение разреженных вектора-строки и матрицы.

Входные данные: разреженная матрица, разреженный вектор-строка, итоговый разреженный вектор-строка, переданный по указателю, количество столбцов.

Выходные данные: код возврата, вектор-строка — результат умножения.

– **`size_t std_matrix_mult(std_matrix matr, std_matrix row, std_matrix *res_row);`**

Описание: функция осуществляет умножение вектора-строки и стандартной матрицы.

Входные данные: стандартная матрица, вектор-строка и вектор-строка — результат умножения.

Выходные данные: результат умножения.

НАБОР ТЕСТОВЫХ СЛУЧАЕВ

№	Ситуация	Входные данные	Результат
1	Введены некорректные данные.	ay	Некорректный ввод.
2	Отрицательное количество строк матрицы.	-3 3	Некорректный ввод.
3	Отрицательное количество столбцов	3 -3	Некорректный

	матрицы.		ввод.
4	Недопустимый символ в записи количества строк.	Fd 3	Некорректный ввод.
5	Ввод несуществующей команды.	5	Такой команды не существует!
6	Недопустимый символ в записи количества столбцов.	4 fd	Некорректный ввод.
7	Неверный метод заполнения (число больше 2 или меньше 1).	3	Некорректный ввод.
8	Буквенное выражение в методе заполнения.	fdsf	Некорректный ввод.
9	Недопустимые символы при заполнении матрицы.	23 fd 3	Некорректный ввод.
10	Неверный процент заполнения матрицы.	101	Некорректный ввод.
11	Невозможность выделить целое число, большее 0, в зависимости от процента зполнения.	20 (при максимальном количестве ненулевых элементов 3)	Неверный ввод.
12	Случайное заполнение матриц и их умножение (1x3 * 3x3).	ROW 0 -14 0 MATRIX -12 19 -65 0 -87 0 -93 0 0	0 1218 0
13	Случайное заполнение матриц и их умножение (1x6 * 6x6).	ROW 0 0 0 0 26 51 MATRIX 0 -63 9 -19 0 -21 78 44 0 78 -46 72 0 0 14 0 0 0 0 0 -41 20 45 86 0 -5 0 0 0 0 -22 0 5 0 0 0	-408 1584 -2069 684 1060 3635
14	Случайное заполнение матриц и их умножение (1x3 * 3x4).	ROW 20 97 -4 MATRIX -7 55 0 0 -85 6 -81 -60 0 -64 0 0	-8117 1938 -7857
15	Случайное заполнение матриц и их	-	5652 0 75 1947

	умножение (1x10 * 10x15).		4443 1155 4098 1254 2214 2184 130513 0 0 0 0

Сравнение скорости работы и объема данных

Сравнение производилось при различных размерах матриц и различной их заполненности. Для матрицы 10x10 проводилось 100 000 запусков и полученное время делилось на 100 000. Для матрицы 100x100 проводилось 1 000 запусков и полученное время делилось на 1 000. Для матрицы 1000x1000 проводилось 100 запусков и полученное время делилось на 100.

Размер матрицы	Заполненность	Разреженная матрица, сек	Стандартная матрица, сек
10x10	25%	0.00072	0.00112
	50%	0.00093	0.00096
	75%	0.00106	0.00088
	100%	0.00105	0.00083
100x100	5%	0.00131	0.05113
	25%	0.00433	0.05122
	50%	0.01403	0.05113
	75%	0.02752	0.0517
	100%	0.04743	0.0513
1000x1000	5%	0.02	5.33
	25%	0.31	5.04
	50%	1.2	5.85
	55%	1.92	5.73
	60%	2.64	5.79
	65%	3.86	5.71
	70%	5.62	5.56
	75%	6.93	5.63
	100%	9.02	5.92

Таблица 1.Сравнение скорости работы алгоритмов умножения для разреженной и стандартной матриц.

Размер матрицы	Заполненность	Разреженная матрица, байт	Стандартная матрицы, байт
10x10	25%	284	504
	50%	548	
	75%	772	

	100%	996	
100x100	5%	4 252	40 824
	25%	21 036	
	50%	41 236	
	75%	61 436	
	100%	81 636	
1000x1000	5%	407 772	4 008 024
	25%	2 010 036	
	50%	4 012 036	
	55%	4 412 436	
	60%	4 812 836	
	65%	5 213 220	
	70%	5 613 620	
	75%	6 014 036	
	100%	8 014 636	

Таблица 2.Сравнение объема данных для алгоритмов умножения для разреженной и стандартной матриц.

Размер матрицы	Заполненность	Преимущество по скорости выполнения операции умножения для разреженной матрицы	Затраты памяти при использовании алгоритма умножения разреженных матриц по сравнению со стандартным алгоритмом, байт	Затраты памяти при использовании алгоритма умножения разреженных матриц по сравнению со стандартным алгоритмом, %
10x10	25%	55%	-220	-43,6
	50%	3,2%	40	8
	75%	-17%	268	52,6
	100%	-21%	492	98

100x100	5%	38 раз	-36 572	-89,6
	25%	11 раз	-19 788	-48,4
	50%	3,6 раза	412	1
	75%	1,88 раза	20 612	50
	100%	8%	40 812	100
1000x1000	5%	Более 265 раз	-3 600 252	-90
	25%	15 раз	-1 997 988	-50
	50%	Почти 4 раза	4 012	0,1
	55%	Почти 3 раза	404 412	10
	60%	Почти 2,2 раза	804 812	20
	65%	1,5 раза	1 205 196	30
	70%	-1%	1 605 596	40
	75%	-23%	2 006 012	50
	100%	-52%	4 006 612	100

Таблица 3.Численное сравнение алгоритмов умножения.

Выводы

Проделав данную лабораторную работу, я понял, что в ситуациях, когда матрица достаточно сильно разрежена, необходимо использовать алгоритмы работы с разреженными матрицами. Следует отметить, что при разреженности, большей 30% в случае матрицы размером 1000x1000 (т. е. матрица состоит не менее, чем на 30% из нулевых элементов), для умножения имеет смысл использовать алгоритм умножения разреженных матриц. При этом, при увеличении процента разреженности, наблюдается тенденция уменьшения необходимых затрат памяти. В случае матрицы 100x100, алгоритм умножения для разреженных матриц эффективнее стандартного даже в случае нулевой её разреженноси. В ином случае, программисту выгоднее и проще использовать стандартный алгоритм умножения матриц.

Контрольные вопросы

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

Разреженной матрицей называется матрица, состоящая из сравнительно небольшого количества ненулевых элементов. По большей части такая матрица состоит из нулей.

Схемы хранения:

- связанная схема;
- кольцевая связанная;
- двунаправленные стеки и очереди;
- диагональная схема;
- строчная или столбцовая схемы.

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

   Для хранения обычной матрицы необходимо выделить участок памяти, равный произведению количества её элементов и размера типа элементов.

   Количество памяти, выделяемое под хранение разреженной матрицы, зависит от схемы её хранения. В данном случае — сумма удвоенного произведения ненулевых элементов на размер типа данных этих элементов (массив ненулевых элементов и массив с номерами столбцов) и произведение количества строк + 1 на размер элемента.

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

   В связи с тем, что главной особенностью разреженной матрицы является полное отсутствие нулевых элементов, программисту необходимо работать исключительно с ненулевыми элементами, однако формат представления не матрица, а некоторый массив.

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

   Эффективнее для матриц использовать стандартные алгоритмы в том случае, когда в матрице находится небольшое количество нулевых элементов.