|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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

ДИСЦИПЛИНА Типы и структуры данных

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 «ОБРАБОТКА РАЗРЕЖЕННЫХ МАТРИЦ».**

Студент Иванов Павел Александрович

Группа ИУ7 – 35Б

Вариант №1

Проверил

**Оглавление**

[Цель работы 2](#_Toc85035297)

[Описание условия задачи 2](#_Toc85035298)

[Описание технического задания 4](#_Toc85035299)

[Описание структур данных 6](#_Toc85035300)

[Описание алгоритма 7](#_Toc85035301)

[Набор тестов 7](#_Toc85035302)

[Ответы на контрольные вопросы 9](#_Toc85035303)

[Ответы на контрольные вопросы 11](#_Toc85035304)

[Вывод 12](#_Toc85035305)

# **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 при различных размерах матриц.

# **Описание условия задачи**

Разработать программу умножения или сложения разреженных матриц. Предусмотреть возможность ввода данных как с клавиатуры, так и использования 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 x 1000). Сравнить эффективность (по памяти и по времени выполнения) стандартных алгоритмов обработки матриц с алгоритмами обработки разреженных матриц при различной степени разреженности матриц и различной размерности матриц.

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 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.

Все логически завершенные фрагменты алгоритма (ввод, вывод, обработка и т.п.) необходимо оформить как подпрограммы.

При разработке интерфейса программы следует предусмотреть:

• указание формата и диапазона вводимых данных,

• указание операции, производимой программой,

• наличие пояснений при выводе результата,

• указание формата выводимых данных

• возможность заполнения матриц вручную (даже при большой размерности, например, 1000 x 1000).

При тестировании программы необходимо:

- проверить правильность ввода

- проконтролировать правильность вывода данных (т.е. их соответствие требуемому формату);

- проверить правильность выполнения операций;

- обеспечить вывод сообщений при отсутствии входных данных («пустой ввод»);

- обеспечить вывод сообщений при нулевых результатах или вывод нулевого результата при ненулевом входе;

- обеспечить возможность ввода данных и вывода результата как при малых матрицах, так и при больших (например, 1000 x 1000).

- сравнить время выполнения стандартного алгоритма обработки матриц и алгоритма обработки разреженных матриц при различной заполненности матриц (от 1 элемента до того количества нулей (в %), при котором становится неэффективно использование алгоритма сокращенного умножения).

- сравнить объем требуемой памяти для реализации стандартного алгоритма обработки матриц и алгоритма обработки разреженных матриц при различном проценте заполнения матриц и при различном их размере.

Следует также протестировать программу при полной загрузке системы, то есть при полном заполнении матриц. Программа должна адекватно реагировать на неверный ввод, пустой ввод и выход за границы матрицы или вектора. Необходимо тщательно следить за освобождением динамической памяти (если она используется) при окончании программы.

# **Описание технического задания**

Входные данные:

*Число:*

Пользователь выбирает пункт меню в пределах допустимых значений. Также потребуется указывать размеры матрицы в *целых* числах. Размер может быть натуральным числом. При вводе матрицы с клавиатуры придется:

1. Указывать число ненулевых элементов и вводить каждый, указывая индексы (строка, столбец) — целые числа и значение — *действительное* число;
2. Вводить всю матрицу указанного до этого размера, причем каждый элемент — *действительное* число.

*Текстовый файл:*

Матрица может быть загружена из файла. Для этого следует указать имя файла в консоли при выборе источника загрузки данных. Перед загрузкой следует верно указать размерность матрицы. Файл не может заканчиваться пустой строкой.

Выходные данные:

Матрица (результат сложения исходных матриц) в её полном представлении или в виде трёх объектов (два вектора и список). В случае требования результатов измерения эффективности, выходные данные — числа, характеризующие время и объем памяти.

Действие программы:

Сложение матриц в разных представлениях: в естественном и сокращенном.

*Функции программы:*

1. Ввод матрицы
   1. Ввод матрицы из файла
   2. Ввод матрицы с клавиатуры
   3. Ввод матрицы через указание ненулевых элементов
2. Сложение введённых матриц
   1. Сложение обычным способом
   2. Сложение матриц в форме разреженной матрицы
   3. Вывод в формате разреженной матрицы
   4. Вывод в стандартном формате
3. Сравнение времени и памяти (вывод результатов измерений)

Обращение к программе:

Запускается файл app.exe. Если его нет, из консоли следует вызывать make run.

Аварийные ситуации:

* Не существует указанный файл;

Результат: сообщение «No file!»

* Введён несуществующий пункт меню;

Результат: сообщение «Wrong input!»

* В файле недостаточно элементов / слишком много элементов / есть не только числа;

Результат: сообщение «Wrong input!»

* С клавиатуры введены не только вещественные числа;

Результат: сообщение «Wrong input!»

* Введённые матрицы имеют разные размеры;

Результат: сообщение «Matrixes can’t be of different size!»

* Введен неверный размер матрицы;

Результат: сообщение «Wrong input!»

* Попытка ввода более двух матриц;

Результат: сообщение «Matrixes are already input!»

* Попытка измерений / проведения операций, когда матрицы не введены;

Результат: сообщение «No matrixes yet!»

# **Описание структур данных**

Для хранения обычной матрицы размера m x n используется массив указателей на строки матрицы. Преимущества этого подхода – универсальность использования в функциях. Память под полную матрицу выделяется динамически.

Для хранения матрицы в разреженном виде (CSR = Compressed Sparse Row) используется следующая структура данных:

typedef struct

{

long long nz;

long double \*a;

long long \*ja;

struct ptr\_list ia;

} csr\_matrix;

Данная структура имеет следующие поля: целое число nz – число ненулевых элементов (соответственно, длина вектора a и ja). Тип long long использован, поскольку по условию размер матриц может быть любой. Соответственно, уместно добавить поддержку любого размера матриц в программу.

Указатель на long double (тип выбран такой, поскольку ограничения на входные данные не заданы) a – указатель на вектор ненулевых значений матрицы. Соответствующие им столбцы хранятся в векторе ja. Номер столбца – целое число типа long long int. Этот тип обуславливается, опять же, любым размером исходной матрицы.

Память под вектора a и ja выделяется динамически.

Номера первых ненулевых элементов в k-ой строке хранятся в k-ом элементе списка ptr\_list. Его описание следующее:

typedef struct ptr\_list

{

long long i;

struct ptr\_list \*next;

} ptr\_list;

i - номер первого ненулевого элемента строке, \*next – указатель на следующую ячейку связного списка. Память под список выделяется динамиечески.

# **Описание алгоритма**

1. Ввод номера команды;
2. Ввод матриц. Если введено больше двух матриц, сообщение об ошибке. Иначе ввод из файла, или вручную, или поэлементно ненулевых элементов.
3. Сложение матриц. Для возможности проведения обоих способов, матрицы хранятся в обоих представлениях. Простой алгоритм сложения проходит по всем столбцам и строкам и записывает сумму соответствующих элементов в выходную матрицу. Алгоритм сложения упрощённых разреженных матриц следующий:
   * Создается результирующая матрица, выделяется память;
   * Идём по i-ой строке: ставим указатели на начало строки (его номер в i-ой ячейке списка ptr\_list) и сравниваем столбцы. Если номера столбцов совпадают, складываем элементы и двигаем оба указателя. Если нет, то записываем элемент с меньшим номером столбца и двигаем его указатель. Этот цикл идёт до тех пор, пока строки обеих матриц не закончатся;
   * Переходим на следующую строку и так до конца обеих разреженных матриц.

# **Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовый случай | Пользовательский ввод | Результат |
| 1 | Неверный пункт меню | -2 | Wrong input! |
| 2 | Неверный пункт меню | a | Wrong input! |
| 3 | Неверный ввод числа строк | 3a | Wrong input! The matrix was not created! You must try again! |
| 4 | Неверный ввод числа столбцов | -10 | Wrong input! The matrix was not created! You must try again! |
| 5 | Неверный выбор способа создания меню | S | Wrong input! The matrix was not created! You must try again! |
| 6 | Файл не существует | a.txt (не существует) | No file! The matrix was not created! You must try again! |
| 7 | Неверный ввод матрицы с клавиатуры | 1 2 a  3 4 5 | The matrix was not created! You must try again! |
| 8 | Файл содержит ошибки | a.txt (существует, в файле записано 1 2 a 3) | The matrix was not created! You must try again! |
| 9 | Неверное число ненулевых чисел для ввода | Размер: 3 x 3, введено 10 | Wrong input! |
| 10 | Неверное число ненулевых чисел для ввода | Размер: 3 x 3, введено a | Wrong input! |
| 11 | Матрицы имеют разные размеры | Размер первой матрицы: 2 x 3, Размер второй матрицы: 3 x 2 | Wrong input! Can’t add matrices! |
| 12 | Проверка сравнение эффективности | 3 | Выведена таблица с процентом заполнения и соответствующими ими данными по времени и памяти |
| 13 | Обычный тест на сложение | Две матрицы 2 x 2  и | Матрица |

# **Ответы на контрольные вопросы**

Оценка эффективно­­сти проводится автоматически с помощью пункта №3 меню. В измерение времени сложения не входит время создания новой матрицы, только её заполнения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Заполнение | Время сложения полной матрицы | Время сложения в CSR формате | Память, занимаемая полной матрицей | Память, занимаемая матрицей в формате CSR |
| Размер 10 х 10 | | | | |
| 20% | 1 | 5 | 1680 | 1208 |
| 40% | 0 | 6 | 1680 | 1880 |
| 60% | 1 | 9 | 1680 | 2328 |
| 80% | 1 | 6 | 1680 | 2648 |
| 100% | 1 | 6 | 1680 | 3000 |
| Размер 50 х 50 | | | | |
| 20% | 21 | 16 | 40400 | 2104 |
| 40% | 21 | 16 | 40400 | 3352 |
| 60% | 21 | 20 | 40400 | 4600 |
| 80% | 21 | 17 | 40400 | 5656 |
| 100% | 22 | 17 | 40400 | 6776 |
| Размер 250 х 250 | | | | |
| 20% | 658 | 63 | 1002000 | 5304 |
| 40% | 491 | 62 | 1002000 | 6584 |
| 60% | 513 | 72 | 1002000 | 7832 |
| 80% | 488 | 70 | 1002000 | 9112 |
| 100% | 517 | 75 | 1002000 | 10360 |
| Размер 1250 х 1250 | | | | |
| 20% | 7997 | 194 | 25010000 | 21304 |
| 40% | 9153 | 204 | 25010000 | 22552 |
| 60% | 8045 | 219 | 25010000 | 23864 |
| 80% | 8122 | 223 | 25010000 | 25080 |
| 100% | 9990 | 271 | 25010000 | 26360 |
| Размер 6250 х 6250 | | | | |
| 20% | 166829 | 796 | 625050000 | 101272 |
| 40% | 183346 | 696 | 625050000 | 102520 |
| 60% | 194024 | 813 | 625050000 | 103832 |
| 80% | 170402 | 694 | 625050000 | 105016 |
| 100% | 177189 | 694 | 625050000 | 106392 |

Исходя из результатов, можно отметить некоторые закономерности. Во-первых, на небольших размерах матриц (10 х 10 элементов) лучше использовать матрицу в обычном представлении. Так затраты по памяти будут или теми же самыми, или меньшими — а по времени гарантированно меньше (при любом проценте заполнения матрицы). В остальных же случаях всегда выгоднее использовать CSR формат. Связано это в том числе с выбором алгоритма сложения. Для обычного матричного сложения асимптотическая сложность — О(n2). Для сложения матриц в формате CSR асимптотическая сложность выбранного мной алгоритма — O(n). Этим и можно обосновать получение такого результата.

# **Ответы на контрольные вопросы**

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

Простыми словами, разреженная матрица — матрица, в которой сравнительно небольшое число ненулевых элементов. Строго выражаясь, матрица порядка n называется разреженной, если её число ненулевых элементов можно записать как n1+g, где g < 1. Существует много схем хранения разреженных матриц. Конечно, можно хранить их и как обычные матрицы, но тогда будут очень велики затраты по памяти. Чтобы их избежать, можно хранить разреженную матрицу линейным списком, диагональной схемой, схемой Кнута, схемой Чанга и Густавсона («разреженный строчный формат»).

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

Допустим, есть матрица порядка n. Тогда, если хранить матрицу целиком, под неё требуется выделение n2 \* (размер элемента матрицы) байт.

Если в этой же матрице nz ненулевых элементов, то, если мы храним разреженную матрицу в виде трёх массивов AN (массив элементов), JA (массив столбцов) и IA (связный список указателей на позиции AN и JA, с которых начинается описание очередной строки), то размер такой структуры будет 2 \* nz \* (размер элемента матрицы) + (количество ненулевых строк) \* ((размер целого) + (размер указателя на элемент матрицы)).

В обычном случае матрица занимает цельный кусок памяти, а в случае второго способа хранения — это два вектора и связный список, которые хранятся в памяти не одним куском.

1. *Каков принцип обработки разреженной матрицы?*

При работе с разреженной матрицей мы должны работать в зависимости от выбранного способа хранения. Главное, что объединяет эти способы — работа только с ненулевыми элементами.

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

Когда разреженная матрица имеет относительно большой процент заполнения ненулевыми элементами, то стоит задуматься о переходе на стандартный алгоритм обработки.

# **Вывод**

В ходе данной работы я познакомился с разными способами хранения матриц, понял, что такое разреженная матрица, и как с ними работать. После написания данной работы мне стало очевидно, что при малом числе ненулевых элементов в матрице стоит использовать особую структуру для хранения и обработки разреженной матрицы. Такой подход сильно экономит память и время на больших объемах данных (порядок матрицы больше 50). При маленьких же порядках матриц следует использовать обычную схему хранения и выполнять операции в привычном нам виде.

Таким образом, при больших объёмах данных матрицы наиболее удобно и эффективно хранить в специальных структурах.