|  |  |
| --- | --- |
| **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

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 «Обработка разреженных матриц» (ВАР. 6)**

Студент Смирнов Иван Владимирович

Группа ИУ7 – 32Б

Проверил Барышникова Марина Юрьевна

*2023 г.*

Оглавление

[Описание условия задачи 3](#_Toc148786931)

[Описание технического задания 4](#_Toc148786932)

[Описание внутренних структур данных 5](#_Toc148786933)

[Описание алгоритмов 6](#_Toc148786934)

[Сравнение времени работы и ресурсных затрат 7](#_Toc148786935)

[Тесты 8](#_Toc148786936)

[Ответы на контрольные вопросы 11](#_Toc148786937)

[Вывод 12](#_Toc148786939)

# Описание условия задачи

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов:

- вектор A содержит значения ненулевых элементов;

- вектор IA содержит номера строк для элементов вектора A;

- вектор JA, в элементе Nk которого находится номер компонент

в A и IA, с которых начинается описание столбца Nk матрицы A.

1. Смоделировать операцию умножения вектора-строки, хранящегося в форме вектора A и вектора, содержащего номера столбцов этих элементов, и матрицы, хранящейся в указанной форме, с получением результата в форме хранения вектора-строки.

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.

Описание технического задания

1. **Исходные данные и результат**

**Входные данные:**

Пользователь выбирает, какие именно вектор и матрицу необходимо умножить друг на друга (1 – разреженный алгоритм, 2 – обычный алгоритм). В зависимости от выбора, далее вектор и матрица вводятся следующим образом: если выбрали разреженный алгоритм, то указывается кол-во ненулевых элементов, а далее сами ненулевые элементы, вводятся их координаты и само значение; если выбрали обычный алгоритм, то указывается размер вектора-строки и матрицы и вводятся все значения (по строкам).

**Выходные данные:**

На экран выводятся изначально-введенные вектор и матрица, а затем результат их умножения. Формат вывода совпадает с форматом ввода.

1. **Задачи, реализуемые программой**

**Программа требует от пользователя выбор способа представления вектора-строки и матрицы, а также алгоритм умножения. Далее пользователь вводит вектор-строку и матрицу в зависимости от формата. Программа выводит введенные структуры, а затем с выбранным ранее алгоритмом вычисляет результат умножения вектора-строки на матрицу.**

**Способ обращения к программе**

**Программа запускается через терминал.**

1. **Возможные аварийные ситуации и ошибки пользователя**
2. **Пользователь ввел несуществующий номер способа представления (или ввел не цифры)**

**На выходе сообщение: «**Неправильный ввод способа!**»**

1. **Пользователь вводит вместо размера (или кол-ва ненулевых элементов) вектора (матрицы) строку или отрицательные числа.**

**На выходе сообщение: «**Число введено неверно!»

1. **Пользователь попытался записать два элемента на одно и то же место**

**На выходе сообщение: «Ошибка! Несколько элементов в одном!»**

1. **Пользователь вводит нулевой элемент вместо ненулевого в разреженном представлении**

**На выходе сообщение: «Нельзя вводить нулевой элемент!»**

1. **Пользователь ввел вектор-строку и матрицу таким образом, что количество столбцов вектора не равно количеству строк матрицы.**

**На выходе сообщение: «**Такое умножение невозможно! (кол-во столбцов вектора-строки не совпадает с кол-вом строк матрицы)»

# **Описание внутренних структур данных**

Вектор-строка и матрица хранятся в зависимости от выбора пользователя (либо в обычном виде, либо в разреженном):

|  |
| --- |
| typedef struct  {  size\_t rsize;  size\_t csize;  int \*mtr;  } Matrix;  typedef struct  {  size\_t size;  size\_t cnt;  int \*vtr;  } Vector;  typedef struct  {  size\_t cnt;  int \*A;  int \*IA;  int \*JA;  size\_t rsize;  size\_t csize;  } SparseMatrix;  typedef struct  {  int \*A;  int \*IA;  size\_t size;  } SparseVector; |

Matrix – обычный способ хранения матрицы, где rsize – кол-во строк, csize – кол-во столбцов, mtr – значения элементов матрицы (хранятся по строкам).

Vector – обычный способ хранения вектора-строки, где size – количество элементов вектора, cnt – количество ненулевых элементов, а vtr – значения элементов вектора.

SparseMatrix – способ хранения матрицы в разреженном виде, где cnt – количество ненулевых элементов, A – значения ненулевых элементов, IA – номера строк соответствующих элементов из А, JA – индексы элементов из A, с которых начинается очередной столбец, rsize – сколько по факту строк в матрице (вычисляется отдельно), csize – сколько по факту столбцов в матрице (вычисляется отдельно).

SparceVector – способ хранения вектора-столбца в разряженном виде, где size – количество ненулевых элементов, A – значения ненулевых элементов, IA – номера столбцов соответствующих элементов из А.

# Описание алгоритмов

* 1. После запуска программы пользователю необходимо выбрать способ работы (работа с обычными матрицами или с разреженными)
  2. Далее пользователь вводит вектор-столбец (в случае обычного способа: размер и все элементы; в случае разреженного: кол-во ненулевых и пары (значение, столбец))
  3. После пользователь вводит матрицу (аналогично вектору)
  4. Выполняется умножение вектора и матрицы (либо обычным алгоритмом, либо разреженным)
  5. На экран выводятся изначальные вектор и матрица, а затем результат произведения, представленный в виде вектора
  6. При желании можно провести эксперимент и вывести его результаты на экран

# Сравнение времени работы и ресурсных затрат

В программе реализована функция, которая по переданному ей размеру создает вектор-строку и матрицу этих размеров и заполняет случайными значениями (которые для одного и того же размера всегда одинаковые). Рассчитывается выделяемая память под вектор и матрицу в обычном представлении и в разреженном. Далее проводится замер времени выполнения умножения вектора на матрицу. Для одного размера выполняется 100 замеров и в качестве результата считается их среднее. Функция выводит только тот результат, когда разреженный алгоритм работает быстрее обычного. Для этого существует дополнительный параметр – разреженность (в данном случае процент заполнения нулями). Функция производит вычисления начиная с 0% разреженности и каждый раз увеличивает разреженность на 1%, пока не найдет тот случай, когда разреженный алгоритм работает быстрее обычного. Вся информация о времени, ресурсах, размерах и проценте разреженности выводится на экран.

|  |
| --- |
| Замерный эксперимент для матрицы 10x10: Sparsity: 51%  Всего замеров: 100  Время выполнения:  1. Обычный алгоритм - 0.470000мкс.  2. Разреженный алгоритм - 0.460000мкс.  Затраты по памяти:  1. Обычный алгоритм - 864Б  2. Разреженный алгоритм - 952Б  Замерный эксперимент для матрицы 50x50:  Sparsity: 67%  Всего замеров: 100  Время выполнения:  1. Обычный алгоритм - 7.700000мкс.  2. Разреженный алгоритм - 7.550000мкс.  Затраты по памяти:  1. Обычный алгоритм - 20224Б  2. Разреженный алгоритм - 20632Б  Замерный эксперимент для матрицы 100x100:  Sparsity: 74%  Всего замеров: 100  Время выполнения:  1. Обычный алгоритм - 28.890000мкс.  2. Разреженный алгоритм - 27.890000мкс.  Затраты по памяти:  1. Обычный алгоритм - 80424Б  2. Разреженный алгоритм - 81232Б  Замерный эксперимент для матрицы 200x200:  Sparsity: 78%  Всего замеров: 100  Время выполнения:  1. Обычный алгоритм - 120.980000мкс.  2. Разреженный алгоритм - 120.440000мкс.  Затраты по памяти:  1. Обычный алгоритм - 320824Б  2. Разреженный алгоритм - 322432Б |

Во всех случаях представление матрицы и вектора в разреженном виде занимает больше памяти, нежели в обычном виде. По замерам 10х10, 50х50, 100х100, 200х200 видно, что разность в разреженности между текущим и предыдущим размерам и все меньше и меньше и примерно стремится к 80%.

Вывод: На матрицах, содержащих большинство нулевых элементов, эффективнее использовать алгоритм разреженных матриц.

# Тесты

В следующих тестах символ «>» обозначает ввод пользователем пункта меню.

1. **Позитивные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| pos\_01 | >1  2  1 0  2 2  4  0 0 1  2 2 1  1 0 1  1 2 1 | (0, 0) 1  (0, 2) 2 | Умеет ли программа умножать разреженный вектор на разреженную матрицу (с пустым столбцом) |
| pos\_02 | >2  3  1 0 2  3 3  1 0 0  1 0 1  0 0 1 | 1 0 2 | Умеет ли программа выполнять обычное умножение |
| pos\_03 | >1  1  2 0  2  0 0 1  0 5 1 | (0, 0) 2  (0, 5) 2 | Умеет ли программа выполнять умножение, если в матрице пустые столбцы идут подряд |
| pos\_04 | >1  2  1 0  1 3  2  3 0 2  3 3 3 | (0, 0) 2  (0, 3) 3 | Умеет ли программа выполнять умножение, если в векторе несколько нулевых элементов идут подряд |
| pos\_05 | >1  2  -1 0  1 2  3  0 0 1  2 0 1  2 1 5 | (0, 1) 5 | В ходе умножения получились нулевые элементы, программа должна их не запоминать |

1. **Негативные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| neg\_01 |  | Сообщение: «Неправильный ввод способа!» | Пустой ввод |
| neg\_02 | >-1 | Сообщение: «Неправильный ввод способа!» | Правильность ввода способа |
| neg\_03 | >100 | Сообщение: «Неправильный ввод способа!» | Правильность ввода способа |
| neg\_04 | >2  (пустой ввод в любом поле) | Сообщение: «Ошибка при вводе (ничего не введено).» | Пустой ввод |
| neg\_05 | >1  2  1 0  2 0 | Сообщение: «Ошибка! Несколько элементов в одном! | Попытка записать 2 элемента на одно место |
| neg\_06 | >1  -1 | Сообщение: «Число введено неправильно» | Правильность ввода численных полей для количества ненулевых элементов |
| neg\_07 | >2  -1 | Сообщение:  «Число введено неправильно.» | Правильность ввода численных полей для количества ненулевых элементов |
| neg\_08 | >1  1  1 100  1  0 0 1 | Сообщение: «Такое умножение невозможно! (кол-во столбцов вектора-строки не совпадает с кол-вом строк матрицы)» | Правильность ввода размеров вектора и матрицы |
| neg\_09 | >1  1  0 4 | Сообщение: «Нельзя вводить нулевой элемент!» | Попытка записать нулевой элемент в разреженное представление |

# Ответы на контрольные вопросы

1. **Что такое разреженная матрица? Какие схемы хранения таких матриц вы знаете?**

Разреженная матрица - это матрица, в которой большинство элементов равны нулю. Она отличается от обычной (плотной) матрицы, в которой большинство элементов ненулевые. Существуют различные схемы хранения разреженных матриц, включая:

1. Схема координат: каждый ненулевой элемент хранится с указанием его координаты в матрице.
2. Схема связанных списков: каждая строка матрицы представляется в виде связанного списка, содержащего только ненулевые элементы и их столбцы.
3. Схема RLE (Run-Length Encoding): последовательности нулей в строках или столбцах заменяются на специальные коды, указывающие количество нулей.
4. **Каким образом и сколько памяти выделяется под хранение разреженной матрицы и обычной?**

В моем случае, в обычной матрице память занимают все её элементы, а также значения кол-во строк + столбцов, а в разреженной – все ненулевые элементы, их строки, номера столбцов компонент, кол-во ненулевых элементов и реальные значения кол-ва строк/столбцов.

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

Принцип обработки разреженной матрицы заключается в оптимизации операций над нулевыми элементами. Поскольку большинство элементов разреженной матрицы равны нулю, можно применять специальные алгоритмы и структуры данных, которые учитывают эту особенность. Например, при умножении разреженных матриц можно избегать операций с нулевыми элементами и выполнять только необходимые вычисления.

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

# Стандартные алгоритмы обработки матриц эффективнее применять в случае плотных матриц, то есть матриц, в которых большинство элементов ненулевые, или в случае маленьких матриц. Это связано с тем, что стандартные алгоритмы обработки матриц предназначены для работы с полными данными и не учитывают особенности разреженных матриц.

# Вывод

В рамках задания были реализованы разные представления вектора и матрицы, а также алгоритмы умножения с ними. В ходе эксперимента выяснилось, что в среднем разреженный алгоритм работает быстрее обычного при 80% заполненности нулями.