|  |  |
| --- | --- |
| **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»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

**«ОБРАБОТКА РАЗРЕЖЕННЫХ ДАННЫХ»**

**по курсу «Типы и структуры данных»**

Студент: Чепиго Дарья Станиславовна

Группа: ИУ7-34Б

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Чепиго Д.С

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Барышникова М.Ю.

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Условие 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;

-связный список JA, в элементе Nk которого находится номер компонентов A и IA, с которых начинается описание столбца 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*

Для корректной работы программы нужно ввести или сгенерировать:

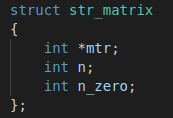
1. Вектор-строку

2.Матрицу

Каждый из этих двух параметров хранится как в обычном виде, так и в разреженной форме.

Вид хранения при обычном виде:

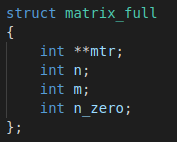
1. Вектор-строка

int \*mtr; - массив элементов вектора-строки

int n; - количество элементов вектора-строки

int n\_zero; - количество ненулевых элементов вектора-строки

2. Матрица



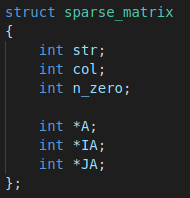
int \*\*mtr; - матрица элеменотв

int n; - количество строк

int m; - количество столбцов

int n\_zero; - количество ненулевых элементов

Структура для хранения в разреженном виде:

 int str; - количество строк в матрице

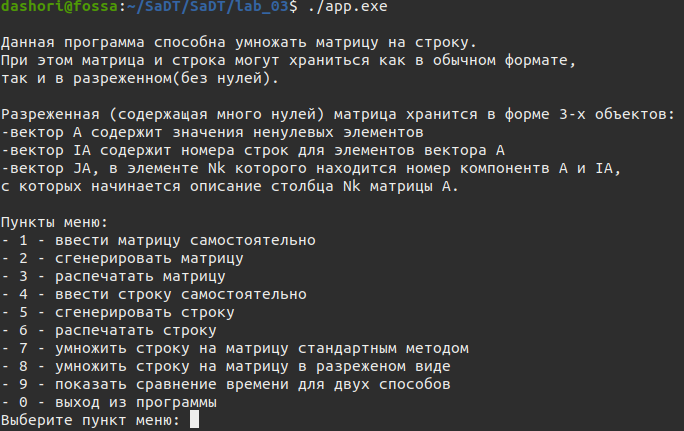
int col; - количество столбцов в матрице

int n\_zero; - количество ненулевых элементов

int \*A; - массив ненулевых значений

int \*IA; - массив индексов строк

int \*JA; - массив с начальными индексами по столбцам

*Описание меню*

*Сравнение эффективности*

*Время — в тактах*

*Память - в байтах*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Размерность | | % заполненности | | Разреженный формат | | Обычный формат | |
| Строка | Матрица | Строка | Матрица | время | память | время | память |
| 5 | 5x5 | 15 | 15 | 5 | 100 | 8 | 160 |
| 5 | 5x5 | 50 | 50 | 5 | 172 | 8 | 160 |
| 5 | 5x5 | 100 | 100 | 6 | 284 | 8 | 160 |
|  |  |  |  |  |  |  |  |
| 10 | 10x20 | 15 | 15 | 7 | 380 | 11 | 920 |
| 10 | 10x20 | 50 | 50 | 10 | 964 | 9 | 920 |
| 10 | 10x20 | 100 | 100 | 15 | 1804 | 11 | 920 |
|  |  |  |  |  |  |  |  |
| 50 | 50x100 | 15 | 15 | 36 | 6668 | 73 | 20600 |
| 50 | 50х100 | 50 | 50 | 138 | 20804 | 84 | 20600 |
| 50 | 50х100 | 100 | 100 | 177 | 41004 | 97 | 20600 |
|  |  |  |  |  |  |  |  |
| 500 | 500х1000 | 15 | 15 | 2009 | 606604 | 8783 | 2006000 |
| 500 | 500x1000 | 50 | 50 | 13599 | 2008004 | 8714 | 2006000 |
| 500 | 500x1000 | 100 | 100 | 19825 | 4006404 | 8511 | 2006000 |
|  |  |  |  |  |  |  |  |
| 5000 | 5000х5000 | 15 | 15 | 25514 | 30046004 | 231603 | 100060000 |
| 5000 | 5000х5000 | 50 | 50 | 176790 | 104060004 | 228692 | 100060000 |
| 5000 | 5000х5000 | 100 | 100 | 223046 | 200080004 | 232144 | 100060000 |

Выводы из таблицы измерений:

Мы можем заметить, что алгоритм умножения в разреженном виде эффективнее по памяти и по времени при маленькой заполненности матрицы (в моем случае заполненность 15%). При 15% заполненности он примерно в 3 раза эффективнее по памяти и в 4 раза эффективнее по времени(при 5000х5000 элементах быстрее в 9 раз).

При матрице, заполненной наполовину память при двух видах хранения практически равна. Что касается времени, тот на маленьких размерностях время примерно равное, но при размерностях больше 50х100 время при стандартном виде хранения быстрее на ~ 60%.

При полностью заполненной матрице для разреженного вида нужно в 2 раза больше памяти, чем для обычного вида. Но при этом, разреженный вид не выигрывает в скорости, и в среднем медленнее в 2 раза. Но, интересный момент, что на максимальной размерности и заполненности матрицы алгоритмы сравниваются по скорости, но не по памяти.

Вывод: при маленькой заполненности матрицы выгоднее использовать хранение в разреженном виде. При средней заполненности матрицы по памяти виды хранения примерно равны, а вот по скорости на небольших размерностях лучше использовать обычный вид хранения. При полной заполненности матрицы лучше всего использовать обычный вид хранения, он выгоднее и по памяти и по скорости.

**Тестирование**

*Позитивные тесты.*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Действия и выходные данные | Результат |
| 1 | Ключ = 0 | Информация о завершении программы | Код возврата - 0 |
| 2 | Ключ = 1  Ввод валидной матрицы  вручную | Ввод количества строк, столбцов и ненулевых элементов матрицы. Ввод самих ненулевых элементов матрицы. | Ожидание следующего ключа |
| 3 | Ключ = 2  Генерирование рандомной матрицы | Ввод количества строк, столбцов и ненулевых элементов матрицы. Генерирование матрицы | Ожидание следующего ключа |
| 4 | Ключ = 3  выбор 1  матрицы существует | На экран выводится матрица в стандартном виде. | Ожидание следующего ключа |
| 5 | Ключ = 3  выбор 2  матрицы существует | На экран выводится матрица в разреженном виде. | Ожидание следующего ключа |
| 6 | Ключ = 4 | Ввод количества сто | Ожидание следующего ключа |
| 7 | Ключ = 6 | Вывод отсортированной таблицы ключей (сортировка пузырьком) | Ожидание следующего ключа |
| 8 | Ключ = 7 | Вывод отсортированной таблицы ключей (быстрая сортировка) | Ожидание следующего ключа |
| 9 | Ключ = 8 | Вывод отсортированной полной таблицы (сортировка пузырьком) | Ожидание следующего ключа |
| 10 | Ключ = 9 | Вывод отсортированной полной таблицы(быстрая сортировка) | Ожидание следующего ключа |
| 11 | Ключ = 10 | Вывод отсортированной таблицы с помощью таблицы ключей (сортировка пузырьком) | Ожидание следующего ключа |
| 12 | Ключ = 11 | Вывод отсортированной таблицы с помощью таблицы ключей(быстрая сортировка) | Ожидание следующего ключа |
| 13 | Ключ = 12 | Вывод информации о времени для 4х видов сортировок и объема занимаемой памяти. | Ожидание следующего ключа |
| 14 | Ключ = 11  Ключ = 1 | В начале программа выведет отсортированную таблицу, а далее – несортированную исходную таблицу. | Ожидание следующего ключа |
| 15 | Ключ = 5  Ключ = 6  Ключ = 5 | В начале программа выводит несортированный массив ключей. После дважды выводит сортированный массив ключей. | Ожидание следующего ключа |

*Негативные тесты*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Результат |
| 1 | Ключ = 2  Неверный ввод структуры.  Italy Rome Colisis -123 | Сообщение где именно пользователь ввёл невалидный результат.  “Error number of house. Please try again” | Код возврата = 10 |
| 2 | Ключ = 1  Файл не найден | Сообщение об ошибке, что файл не найден | Код возврата = 2 |
| 3 | Ключ = 2  Неверный ввод параметров квартиры  Italy Rome Colisis 123 12 -1234 | Сообщение где именно пользователь ввёл невалидный результат.  "Error square of flat. Please try again.” | Код возврата = 11 |
| 4 | Ключ = 2, но в массиве уже есть 200 элементов | Сообщение, что массив полностью заполнен | Ожидание следующего ключа |
| *5* | *Ключ = 3, но массив пустой* | *Сообщение, что массив пустой и в нем нечего удалять* | Ожидание следующего ключа |

**Контрольные вопросы**