

Лабораторная работа №9

по дисциплине «Процедурное программирование на С»

Двумерный динамический массив.

Обработка матриц

Кострицкий А. С., Ломовской И. В.

Цель работы

1. Научиться обрабатывать матрицы, хранимые в виде динамических массивов указателей на другие динамические массивы.

Общее задание

Принять с клавиатуры

$$m \in \mathbb{N}, n \in \mathbb{N}, A \in M_{[m \times n]}(\mathbb{Z}),$$

$$p \in \mathbb{N}, q \in \mathbb{N}, B \in M_{[p \times q]}(\mathbb{Z}).$$

Удаляя строки или столбцы, в соответствии с вариантом привести матрицы A и B к квадратному виду A_1 и B_1 :

$$A_1 \in M_{[k \times k]}(\mathbb{Z}), \quad \text{where } k = \min(m, n),$$

$$B_1 \in M_{[s \times s]}(\mathbb{Z}), \quad \text{where } s = \min(p, q).$$

Обратите внимание: если у матрицы больше строк, чем столбцов, то удалять нужно только строки, иначе — только столбцы.

Добавляя в конец сначала строки, а потом столбцы, в соответствии с вариантом привести матрицы A_1 и B_1 к одному размеру A_2 и B_2 :

$$A_2, B_2 \in M_{[z \times z]}(\mathbb{Z}), \quad \text{where } z = \max(k, s).$$

Приняв с клавиатуры целые неотрицательные ρ и γ , вычислить и вывести на экран результат выражения

$$A_2^\rho B_2^\gamma = \underbrace{A_2 \cdot A_2 \cdot \dots \cdot A_2}_{\rho \text{ множителей}} \cdot \underbrace{B_2 \cdot B_2 \cdot \dots \cdot B_2}_{\gamma \text{ множителей}}.$$

Примечания

1. Считать любую квадратную матрицу в нулевой степени единичной.
2. Принять, что под вводом и выводом матриц подразумеваются «классические», насколько тут уместно это слово, построчные ввод и вывод оных.
3. Принять, что под перемножением матриц подразумевается «классическое», насколько тут уместно это слово, перемножение «строка на столбец».
4. Использовать при решении целые знаковые числа фиксированного стандартом объёма в 8 байт. Переполнение не контролировать.
5. Для каждой структуры данных в первую очередь реализовать набор подпрограмм группы CДИО — подпрограммы создания, удаления, ввода и вывода.
6. Вынести подпрограммы для каждой структуры данных в отдельный модуль.
7. Следовать правилу Тараса Бульбы для памяти: «Если в подпрограмме есть запрос динамической памяти, то либо в ней же должно осуществляться освобождение памяти, либо в имени подпрограммы должно быть *указание* для программиста на наличие запроса памяти внутри подпрограммы.» В качестве *указаний*, например, можно использовать слова и словосочетания: «allocate», «create», «set length», «new» и другие.

Варианты удаления

Удалять строки или столбцы с *максимальным* (в вариантах 1, 2, 3, 4) или *минимальным* (в вариантах 5, 6, 7, 8) элементом в матрице. Если обнаружено несколько, считать целевым *максимум/минимум*, который был бы встречен *первым* (в вариантах 1, 3, 5, 7) или *последним* (в вариантах 2, 4, 6, 8) при обходе *по строкам* (в вариантах 1, 2, 5, 6) или *по столбцам* (в вариантах 3, 4, 7, 8).

Варианты добавления

Добавлять всегда сначала новые строки, потом — столбцы.

Добавлять в новые строки округлённые к нижнему целому *средние арифметические* (в вариантах 1, 2, 3, 4) или *средние геометрические модулей* (в вариантах 5, 6, 7, 8) элементов столбцов.

Добавлять в новые столбцы *максимумы* (в вариантах 1, 3, 5, 7) или *минимумы* (в вариантах 2, 4, 6, 8) по строкам.

Пример

Вариант №1: «Удалять строки или столбцы с *максимальным* элементом в матрице. Если обнаружено несколько, считать целевым *максимум*, который был бы встречен *первым* при обходе *по строкам*. Добавлять всегда сначала новые строки, потом — столбцы. Добавлять в новые строки округлённые к нижнему целому *средние арифметические* элементов столбцов. Добавлять в новые столбцы *максимумы* по строкам.»

В рамку обведены целевые максимумы, красным выделены удаляемые строки или столбцы.

$$\text{In: } n = 2, m = 3, A = \begin{pmatrix} 4 & \boxed{7} & 1 \\ 3 & \textcolor{red}{2} & 7 \end{pmatrix}, p = 5, q = 3, B = \begin{pmatrix} \textcolor{red}{0} & \textcolor{red}{2} & \boxed{\textcolor{red}{9}} \\ 1 & 2 & 2 \\ 3 & 3 & 3 \\ \boxed{\textcolor{red}{8}} & \textcolor{red}{8} & \textcolor{red}{2} \\ 1 & 8 & 3 \end{pmatrix}, \rho = 2, \gamma = 3;$$

$$k = 2, A_1 = \begin{pmatrix} 4 & 1 \\ 3 & 7 \end{pmatrix}, s = 3, B_1 = \begin{pmatrix} 1 & 2 & 2 \\ 3 & 3 & 3 \\ 1 & 8 & 3 \end{pmatrix};$$

$$z = 3, A_2 = \begin{pmatrix} 4 & 1 & \max(4, 1) \\ 3 & 7 & \max(3, 7) \\ [3.5] & [4.0] & \max([3.5], [4.0]) \end{pmatrix} = \begin{pmatrix} 4 & 1 & 4 \\ 3 & 7 & 7 \\ 3 & 4 & 4 \end{pmatrix}, B_2 = \begin{pmatrix} 1 & 2 & 2 \\ 3 & 3 & 3 \\ 1 & 8 & 3 \end{pmatrix};$$

$$\text{Out: } A_2^\rho B_2^\gamma = A_2^2 B_2^3 = A_2 \cdot A_2 \cdot B_2 \cdot B_2 \cdot B_2 = \begin{pmatrix} 15464 & 34369 & 22134 \\ 36567 & 81282 & 52327 \\ 22680 & 50421 & 32461 \end{pmatrix}.$$

Взаимодействие с системой тестирования

1. Решение задачи оформляется студентом в виде многофайлового проекта. Для сборки проекта используется программа make, сценарий сборки помещается под версионный контроль. В сценарии должны присутствовать цель `app.exe` для сборки основной программы и цель `test.exe` — для сборки модульных тестов.

2. Исходный код лабораторной работы размещается студентом в ветви `lab_LL`, а решение каждой из задач — в отдельной папке с названием вида `lab_LL_CC_PP`, где `LL` — номер лабораторной, `CC` — вариант студента, `PP` — номер задачи.

Пример: решения восьми задач седьмого варианта пятой лабораторной размещаются в папках `lab_05_07_01`, `lab_05_07_02`, `lab_05_07_03`, ..., `lab_05_07_08`.

3. Исходный код должен соответствовать оглашённым в начале семестра правилам оформления.
4. Если для решения задачи студентом создаётся отдельный проект в IDE, разрешается поместить под версионный контроль файлы проекта, добавив перед этим необходимые маски в список игнорирования. Старайтесь добавлять маски общего вида. Для каждого проекта должны быть созданы, как минимум, два варианта сборки: `Debug` — с отладочной информацией, и `Release` — без отладочной информации. Крайне рекомендуем использовать IDE Qt Creator.
5. Для каждой программы ещё до реализации студентом заготавливаются и помещаются под версионный контроль функциональные тесты, демонстрирующие её работоспособность. Входные данные следует располагать в файлах вида `in_TT.txt`, выходные — в файлах вида `out_TT.txt`, где `TT` — номер тестового случая.

Под версионный контроль также помещается файл вида `FuncTestsDesc.md` с описанием в свободной форме содержимого каждого из тестов. Вёрстка файла на языке Markdown обязательной при этом не является, достаточно обычного текста.

Разрешается помещать под версионный контроль сценарии автоматического прогона функциональных тестов.

Если Вы используете при автоматическом прогоне функциональных тестов сравнение строк, не забудьте проверить используемые кодировки. Помните, что UTF-8 и UTF-8(BOM) — две разные кодировки.

Пример: функциональные тесты для задачи с двенадцатью классами эквивалентности должны размещаться в файлах `in_01.txt`, `in_02.txt`, ..., `in_12.txt`, `out_01.txt`, `out_02.txt`, ..., `out_12.txt`. В файле `FuncTestsDesc.md` при этом может содержаться следующая информация:

```
# Тесты для лабораторной работы №X
## Входные данные
int a, int b, int c
## Выходные данные
int d, int e
- in_01 -- негативный -- вместо числа a вводится символ
- in_02 -- негативный -- вместо числа b вводится символ
- in_03 -- негативный -- недостаточно аргументов вводятся с консоли
- in_04 -- позитивный -- обычный тест
- in_05 -- позитивный -- вводятся три одинаковых числа
...
```

6. Для каждой подпрограммы должны быть подготовлены модульные тесты, которые демонстрируют её работоспособность.
7. Все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены к моменту выхода из программы. Для контроля можно использовать, например, программы `valgrind` или `Dr. Memory`.

8. Успешность ввода должна контролироваться. При первом неверном вводе программа должна возвращать код ошибки. Обратите внимание, что даже в этом случае все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены.

9. Вывод Вашей программы может содержать текстовые сообщения и числа. Тестовая система анализирует только числа в потоке вывода, поэтому они могут быть использованы только для вывода результатов — использовать числа в информационных сообщениях запрещено.

Пример: сообщение «**Input point 1:**» будет неверно воспринято тестовой системой, а сообщения «**Input point A:**» или «**Input first point:**» — правильно.

10. Если не указано обратное, числа двойной точности следует выводить, округляя до шестого знака после запятой.

Пример: сообщение «**Input point 1:**» будет неверно воспринято тестовой системой, а сообщения «**Input point A:**» или «**Input first point:**» — правильно.

11. *Только для ЛРН^{№9}.* Порядок входных данных: $n, m, A, p, q, B, \rho, \gamma$.

12. *Только для ЛРН^{№9}.* Порядок выходных данных: $A_2^\rho B_2^\gamma$.