Матрицы. Двумерные списки

Продолжим тему работы со списками и поговорим подробнее о двумерных списках. Как мы уже говорили на прошлом занятии, двумерный список – это список, каждый элемент которого тоже является списком.

1 Вывод двумерного списка на экран

Для вывода двумерного списка на экран можно использовать вложенные циклы, которые перебирают индексы элементов списка:

```
arr = [[1, 2], [3, 4, 5], [6]]

for i in range(len(arr)):
    for j in range(len(arr[i])):
        print(arr[i][j], end =' ')
    print()
```

```
1 2
3 4 5
6
```

Внешний цикл перебирает индексы большого списка от `0` до `len(arr) - 1`, где `len(arr)` – количество элементов большого списка. Внутренний цикл перебирает индексы вложенных списков от `0` до длины вложенного списка минус 1.

Еще один вариант – перебирать не индексы, а сами элементы списка:

```
for line in arr:
    for el in line:
        print(el, end = ' ')
    print()
```

В этом случае внешний цикл перебирает элементы-списки, а внутренний цикл – элементы вложенных списков. Вызов `print()` на последней строке используется для перевода курсора на новую строку после вывода очередного вложенного списка (так как в предыдущем вызове `print` был указан аргумент `end = ' '`).

Вместо внутреннего цикла можно сделать вывод с использованием `*`, что сделает код очень лаконичным:

```
for line in arr:
    print(*line)
```

2 Матрицы

Рассмотрим особый вид двумерных списков – **матрицы**. Матрицы – это двумерные списки размером `n` на `m`, то есть они состоят из `n` элементов (часто их называют строками), где каждая строка – список из `m` элементов.

Для обращения к каждому элементу матрицы используется двойной индекс `a[1][2]`: первый указывает на номер строки, второй – на номер элемента в строке. По аналогии с электронными таблицами второй индекс иногда называют номером столбца.

Строки матрицы нумеруются сверху вниз, а столбцы слева направо:

00	**01**	**02**	**03**	**04**
10	**11**	**12**	**13**	**14**
20	**21**	**22**	**23**	**24**

Нумерация ячеек матрицы.
Красный – номер строки, черный – номер столбца

Обозначим за `i` счетчик строк, за `j` – счетчик элементов внутри строки. Мы будем использовать это обозначение во всех задачах на матрицы, чтобы не запутаться в обозначениях индексов.

2.1 Квадратные матрицы

Особый интерес для обработки представляют квадратные матрицы – матрицы, в которых размерности `n` и `m` совпадают. В квадратных матрицах выделяют интересные с точки зрения решения задач объекты – **диагонали**.

Главная диагональ – это набор элементов матрицы, взятый по диагонали от левого верхнего угла в направлении правого нижнего.

00	**01**	**02**
10	**11**	**12**
20	**21**	**22**

Главная диагональ

Для всех элементов лежащих на главной диагонали индексы `i` и `j` совпадают.

Для того, чтобы пройти по всем элементам главной диагонали достаточно организовать цикл по одному из параметров:

```
for i in range(n):
    print(arr[i][i])
```

Сделаем еще 2 важных вывода: для всех элементов, лежащих ниже (левее) главной диагонали, `i` $>$ `j`, тогда как для всех элементов, лежащих выше (правее) главной диагонали, `i` $<$ `j`:

00	**01**	**02**
10	**11**	**12**
20	**21**	**22**

(a) Элементы ниже главной диагонали

00	**01**	**02**
10	**11**	**12**
20	**21**	**22**

(b) Элементы выше главной диагонали

Это факт можно учесть, например, при переборе элементов, расположенных выше главной диагонали, и начинать внутренний цикл с `i + 1`.

Кроме главной диагонали, в задачах могут фигурировать диагонали, параллельные главным. Каждая из них начинается с элемента нулевого столбца или нулевой строки, а заканчивается элементом `n-1` строки или `n-1` столбца. Индексы каждого следующего элемента диагонали возрастают по `i` и по `j` на единицу:

00	**01**	**02**	**03**
10	**11**	**12**	**13**
20	**21**	**22**	**23**
30	**31**	**32**	**33**

Диагонали, параллельные главной

Побочная диагональ – диагональ, проведённая из правого верхнего угла в левый нижний.

00	**01**	**02**
10	**11**	**12**
20	**21**	**22**

Побочная диагональ

Индексы строки побочной диагонали меняются от `0` до `n - 1`, а индексы столбца – наоборот, от `n - 1` до `0`. Неизменным остается их сумма: равенство `i + j = n - 1` выполнено для любого элемента побочной диагонали. По этой формуле удобно проверять принадлежность элемента побочной диагонали матрицы.

Побочная диагональ так же, как и главная, разбивает матрицу на две симметричные части. Для всех элементов ниже побочной диагонали `i + j` $>$ `n - 1`, а для всех элементов выше: `i + j` $<$ `n - 1`:

00	**01**	**02**
10	**11**	**12**
20	**21**	**22**

(c) Элементы ниже побочной диагонали

00	**01**	**02**
10	**11**	**12**
20	**21**	**22**

(d) Элементы выше побочной диагонали

3 Симметричные отражения матриц

В некоторых задачах нужно проверить квадратную матрицу на симметричность или отразить матрицу относительно одной из осей симметрии. Для решения таких задач, нужно понять, какие пары элементов симметричны симметричны для заданной оси, и вывести общую формулу.

Допустим нас интересует главная диагональ как ось симметрии. В этом случае симметричные элементы будут иметь индексы вида `[i][j]` и `[j][i]`. Например, `a[1][3]` и `a[3][1]` – элементы матрицы `a`, симметричные главной диагонали.

Вертикальная и горизонтальные оси симметрии так же разбивают матрицу на две части. Для элемента с индексами `[i][j]`, симметричный относительно вертикали будет иметь индексы `[i][n-j-1]`, а симметричный относительно горизонтали – `[n-i-1][j]`.

Чтобы отразить матрица относительно одной из этих осей, достаточно просто поменять местами все пары симметричных элементов.

3.1 Поворот квадратной матрицы

Симметричные отражения элементов матриц можно применять для решения задач на поворот квадратной матрицы. Рассмотрим следующую задачу:

Задача

Задан квадратный массив A из N строк и N столбцов. Требуется произвести «поворот по часовой стрелке» массива таким образом, чтобы элементы 1 строки стали элементами N столбца, второй строки – элементами $N-1$ столбца и так далее.

Эту задачу можно решить, не используя дополнительную память, применив ранее рассмотренные симметричные отражения.

Сначала, отразим матрицу относительно вертикальной оси симметрии:

```
for i in range(n):
    for j in range(n//2):
        a[i][j], a[i][n-j-1] = a[i][n-j-1], a[i][j]
```

В представленном коде, мы проходим по всем элементам левой половины матрицы, и меняем их местеми с симметричными элементами из правой половины.

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

$\longrightarrow$

3	**2**	**1**
6	**5**	**4**
9	**8**	**7**

Первое отражение
(вертикальная симметрия)

Заметим, что элементы 1 и 9 встали на свои места (для них отражение произвело тот же эффект, что и поворот на 90 градусов), а остальные элементы находятся в позициях, симметричных нужным относительно побочной диагонали. Второе отражение относительно побочной диагонали поставит все элементы в нужные позиции:

```
for i in range(n):
    for j in range(n - j - 1):
        a[i][j], a[n - i - 1][n - j - 1] = a[n - i - 1][n - j - 1], a[i][j]
```

3	**2**	**1**
6	**5**	**4**
9	**8**	**7**

$\longrightarrow$

7	**4**	**1**
8	**5**	**2**
9	**6**	**3**

Второе отражение
(симметрия относительно побочной диагонали)

Как видим, при помощи комбинации двух отражений мы смогли повернуть матрицу на 90 градусов. Похожие рассуждения можно использовать для решения задачи о повороте матрицы на 180 или 270 градусов.

4 Задачи на заполнение двумерных списков

В некоторых задачах нужно заполнить двумерный список `n` на `m` по определенному правилу. Например, рассмотрим следующую задачу:

Задача

Даны два числа, n и m. Заполните массив $n \times m$ элементов по принципу “шахматной доски”.

Входные данные	*Выходные данные*
3 4	0 1 0 1 1 0 1 0 0 1 0 1

Для получения общего решения нужно понять закономерность расположения элементов матрицы и их индексов и описать её в виде формул. Выпишем индексы, в которых нужно расположить нули, и индексы, в которых нужно расположить единицы:

$$\textit{нули: } (0,0),\ (0,2),\ (1,1),\ (1,3),\ (2,0),\ (2,2)$$

$$\textit{единицы: } (0,1),\ (0,3),\ (1,0),\ (1,2),\ (2,1),\ (2,3)$$

Заметим, что для всех позиций, в которых стоит ноль, сумма индексов будет четной, а для всех позиций, где стоит единица – нечетной. Значит, остаток от деления суммы индексов на 2 определяет, какое число поставить в матрицу в эту позицию.

Для заполнения такой матрицы можно использовать генератор. Эта однострочная конструкция будет итерироваться по элементам строк с помощью индекса `i`, а по элементам столбцов – `j`:

```
arr = [[(i + j) % 2 for j in range(m)] for i in range(n)]
```

Написать генератор для заполнения матрицы можно и для более сложных задач. Рассмотрим такой пример:

Задача

Даны два числа, n и m. Заполните массив $n \times m$ элементов "змейкой".

Входные данные	*Выходные данные*
4 5	1 2 3 4 5 10 9 8 7 6 11 12 13 14 15 20 19 18 17 16

В этой задаче мы видим, что для заполнения четных и нечетных строк матрицы нужно использовать разные формулы:

для четных строк: `a[i][j] = m*i + j + 1`

для нечетных строк: `a[i][j] = m*i + m - j`

Для записи такого условия можно использовать тернарный оператор, и выбирать, какую формулу использовать, в зависимости от четности строки:

```
a[i][j] = (m*i + j + 1) if (i % 2 == 0) else (m*i + m - j)
```

Тот же результат можно получить, если включать или выключать часть формулы в зависимости от четности строки. Для этого можно использовать остаток от деления на 2: если нам нужно включить часть формулы, умножаем её на 1, если выключить, умножаем на ноль:

```
a[i][j] = m*i + (1 - i % 2)*(j + 1) + (i % 2)*(m - j)
```

Если `i % 2 == 0`, то коэффициент перед `(j + 1)` будет 1 (часть формулы "включится"), а перед `(m - j)` коэффициент будет 0 (часть "выключится"). Если же `i % 2 == 1`, то все будет наоборот – `(j + 1)` "выключится", а `(m - j)` – "включится".

Но не для всех задач решение можно уместить в конструкцию генератора. Рассмотрим, например, задачу о заполнении квадратной матрицы числами по спирали:

Задача

Дано число n. Заполните массив $n \times n$ элементов по спирали.

Входные данные	*Выходные данные*
4	1 2 3 4 12 13 14 5 11 16 15 6 10 9 8 7

Посмотрим на уже заполненную числами матрицу и попробуем выявить закономерность. Первая строка заполнена числами от `1` до `n`. Дальнейшее заполнение идет так: движение вниз, движение влево, движение вверх, и движение вправо:

1	**2**	**3**	**4**	**5**
16	**17**	**18**	**19**	**6**
15	**24**	**25**	**20**	**7**
14	**23**	**22**	**21**	**8**
13	**12**	**11**	**10**	**9**

Заполнение матрицы по спирали
(разными цветами отмечены разные отрезки пути)

Для каждого из таких движений можно описать приращения `di` и `dj` для текущих индексов `i` и `j`. Например, для движения вниз приращение по `i` равно единице, а по `j` – нулю. Заметим, что порядок обхода будет сохранятся: вниз, влево, вверх, вправо, изменится только количество шагов. Каждый второй поворот сокращает длину пути на один.

Разработав математическую модель решения этой задачи, можно приступать к написанию кода. Создадим заготовку под наш ответ - матрицу `n` × `n`, заполненную

нулями:

```
a = [[0] * n for i in range(n)]
```

Затем, заполним нулевую строку числами от одного до `n`:

```
a[0] = list(range(1, n + 1))
```

Сохраним изменения координат для каждого движения в списке кортежей `changes` (такой прием мы уже использовали при моделировании движения шахматного коня):

```
changes = [(1, 0), (0, -1), (-1, 0), (0, 1)]
```

Переменная `track_len` будет хранить длину текущего отрезка пути. Изначально она равна `n-1`:

```
track_len = n - 1
```

Для того, чтобы заполнять ячейки подряд идущими числами введем счетчик `cnt`, который будет хранить текущее число. Так как мы уже заполнили верхнюю строку, то старт нашего обхода начнется с последнего элемента нулевой строки и значения счетчика `cnt = n`. Переменные `i` и `j` будут хранить наше текущее положение:

```
cnt = n
i, j = 0, n - 1
```

Итак, теперь мы можем перейти к самому заполнению, которое представляет из семя конструкцию из трех вложенных циклов:

```
while cnt < n**2:
    for di, dj in changes:
        for k in range(track_len):
            cnt += 1
            i += di
            j += dj
            a[i][j] = cnt
        if di == 0:
            track_len -= 1
```

Разберем этот код поподробнее:

1. Внешний цикл `while` определяет, нужно ли продолжать построение спирали.

   Для нашей таблицы `n` × `n` последним значением счетчика `cnt` будет `n` в квадрате. Пока наш счетчик не достиг этого значения, будем продолжать заполнять матрицу.

2. Второй цикл выполняет один оборот из 4-х движений (вниз, влево, вверх, и вправо). Направление движения задается парой смещений из списка `changes`.

   Важно не забыть, что каждые 2 шага этого цикла `track_len` должен уменьшаться на 1, так как сокращаются длины отрезков, которые мы должны проходить.

3. Внутренний цикл выполняет шаги одного движения. Именно здесь мы увеличиваем счетчик `cnt` на 1, меняем координаты и записываем в элемент с новыми координатами значение счетчика.

Запустите код и убедитесь в том, что полученное заполнение матрицы полностью соответствует поставленной задаче.

Теперь, когда вы освоили заполнение матрицы змейкой и по спирали, вы можете переходить к самостоятельному решению задач в практической части курса.