BÀI TẬP THỰC HÀNH
MÔN TRÍ TUỆ NHÂN TẠO
LAB 4: TÌM KIẾM KHÔNG GIAN TRẠNG THÁI

Numpy là một trong những thư viện quan trọng nhất của Python dành cho tính toán số. Numpy hỗ trợ cho Python để làm việc với *mảng nhiều-chiều* (multi-dimensional array) và ma trận và thao tác nhanh trên những mảng như vậy.

ndarray của NumPy

- NumPy bao gồm hai phần quan trọng: *ndarray* and *Ufuncs* (các hàm phổ quát - universal function).
- *ndarray* là đối tượng mảng nhiều-chiều, là kiểu cấu trúc dữ liệu quan trọng cho nhiều tác vụ của NumPy.
- Các *hàm phổ quát* là những hàm làm việc trên các đối tượng ndarray theo kiểu xử lý từng phần tử một.
- Mảng hay ma trận là một trong những cách biểu diễn dữ liệu căn bản. Thông thường một mảng sẽ gồm những phần tử cùng kiểu dữ liệu và có thể có nhiều chiều. *ndarray* là sự tổng quát hóa loại dữ liệu như vậy.

Thí dụ về cách tạo mảng

- >>> import numpy as np # import the numpy library
- >>> arr = np.array([1,3,4,5,6]) # tạo ra một mảng một chiều
- >>> arr
- array([1, 3, 4, 5, 6])
- >>> arr.shape
- (5,)
- >>> arr.dtype
- dtype('int32')
- Thuộc tính *shape* của một đối tượng mảng sẽ cho biết về kích thước của mảng.
- Thuộc tính *dtype* của một đối tượng mảng sẽ cho biết về kiểu dữ liệu của các phần tử của mảng.
- **Lưu ý**: tất cả mọi phần tử của mảng phải cùng một kiểu dữ liệu.

```
>>> arr = np.array([[1,2,3],[2,4,6],[8,8,8]])
>>> arr.shape
(3, 3)
>>> arr
array([[1, 2, 3],
       [2, 4, 6],
       [8, 8, 8]])
```

Hàm *np.zeros*: tạo ra một ma trận với kích thước cho trước mà toàn chứa giá trị zero.

```
>>> arr = np.zeros((2,4))
>>> arr
array([[ 0., 0., 0., 0.],
```

[0., 0., 0., 0.]])

- Hàm *np.ones*: tạo ra một ma trận với kích thước cho trước mà toàn chứa giá trị 1.

```
>>> arr = np.ones((2,4))
>>> arr
array([[ 1., 1., 1., 1.],
       [ 1., 1., 1., 1.]])
```

Hàm np.identity tạo ra ma trận đơn vị với một kích thước được cho:

```
>>> arr = np.identity(3)
>>> arr
array([[ 1., 0., 0.],
      [ 0., 1., 0.],
       [ 0., 0., 1.]])
```

- Ta cũng có thể tạo ra một ma trận với kích thước cho trước mà chứa những trị ngẫu nhiên. Điều này được thực hiện bằng cách dùng hàm *randn* trong gói *numpy.random* :

```
>>> arr = np.random.randn(3,4)
>>> arr
array([[ 0.0102692 , -0.13489664, 1.03821719, -0.28564286],
       [-1.12651838, 1.41684764, 1.11657566, -0.1909584 ],
       [ 2.20532043, 0.14813109, 0.73521382, 1.1270668 ]])
```

Hàm *np.arrange*() tạo ra một mảng chứa những giá trị được xác định, một phiên bản của hàm *range*() dành cho các đối tượng dữ liệu array.

Thí dụ,

```
>>> a = np.arrange(5)
>>> a
array([0, 1, 2, 3, 4]
>>> arr = np.arrange(3,7,2)    # step size = 2
>>> arr
array([3, 5])
>>> b = np.arrange(12).reshape(3,4)        # a matrix with size 3 × 4
>>> b
array([[0, 1, 2, 3],
       [4, 5, 6, 7],
       [8, 9, 10, 11])
```

- Numpy cung cấp một số cách để truy đạt đến phần tử của mảng.

Đánh chỉ số căn bản

```
>>> arr = np.array([[1,2,3],[2,4,6],[8,8,8]])
>>> arr[0]
array([1, 2, 3])
>>> arr = np.arrange(12).reshape(2,2,3)     # hai ma trận có kích thước 2×3
>>> arr
array([[[ 0, 1, 2],
        [ 3, 4, 5]],
```

```
        [[ 6, 7, 8],
         [ 9, 10, 11]]])
>>> arr[0]
array([[0, 1, 2],
       [3, 4, 5]])
```

Đánh chỉ số bằng số nguyên (integer array indexing)

```
>>>: arr = np.arrange(9).reshape(3,3)
>>>arr
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> arr[[0,1,2],[1,0,0]]
array([1, 3, 6])
```

Trong thí dụ này, chúng ta đã cung cấp một mảng trong đó phần thứ nhất xác định vị trí *hàng* mà ta muốn truy đạt và phần thứ hai xác định vị trí *cột* mà ta muốn truy đạt.

Đánh chỉ số bằng điều kiện:

- Cách này được dùng khi ta muốn truy đạt đến dữ liệu dựa vào một điều kiện nào đó. Trong thí dụ sau đây, to dùng một mảng chứa tên của một số thành phố và một mảng thứ hai chứa dữ liệu về các thành phố đó.

```
>>> cities = np.array(["delhi","bangalore","mumbai","chennai","bhopal"])
>>> city_data = np.random.randn(5,3)
>>>city_data
array([[ 1.78780089, -0.25099029, -0.26002244],
       [ 1.41016167, -0.43878679, 0.4912639 ],
       [-0.32176723, -0.01912549, -1.22891881],
       [-0.93371835, -0.03604015, -0.37319556],
       [ 1.48625779, 0.62758167, 0.77321756]])
>>> city_data[cities =="delhi"]
array([[ 1.78780089, -0.25099029, -0.26002244]])
>>> city_data[city_data >0]
array([ 1.78780089, 1.41016167, 0.4912639 , 1.48625779, 0.62758167, 0.77321756])
```

Hàm reshape

Thí dụ:

```
>>> arr = np.arrange(15).reshape(3,5)
>>> arr
array([[ 0, 1, 2, 3, 4],
       [ 5, 6, 7, 8, 9],
       [10, 11, 12, 13, 14]])
>>> arr + 5
array([[ 5, 6, 7, 8, 9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> arr * 2
```

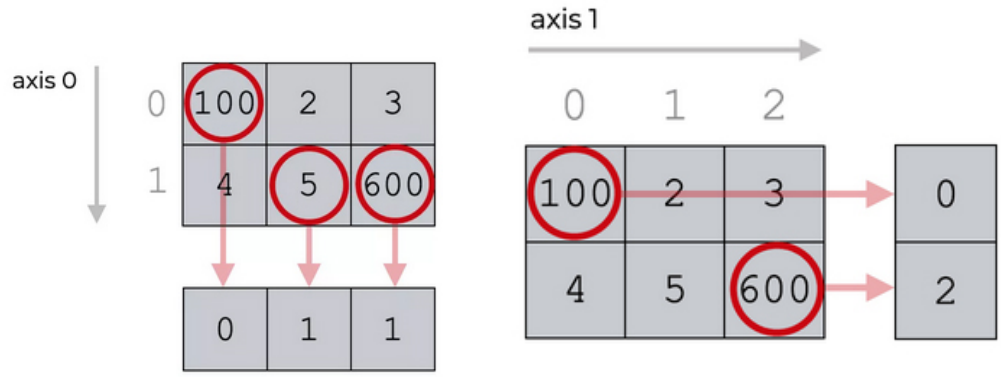
```
array([[ 0, 2, 4, 6, 8],
       [10, 12, 14, 16, 18],
      [20, 22, 24, 26, 28]])
```
Ta không cần dùng vòng **for** để thực hiện hai công tác trên
```
>>> arr1 = np.arrange(15).reshape(5,3)
>>>arr1
array([[ 0, 1, 2],
       [ 3, 4, 5],
       [ 6, 7, 8],
       [ 9, 10, 11],
       [12, 13, 14]])
>>> arr2 = np.arrange(5).reshape(5,1)
array([[0],
       [1],
       [2],
       [3],
       [4]])
>>> arr2 + arr1
array([[ 0, 1, 2],
       [ 4, 5, 6],
       [ 8, 9, 10],
       [12, 13, 14],
       [16, 17, 18]])
```
Hàm concatenate: được dùng để kết (join) hai hay nhiều mảng có cùng shape dọc theo một trục (axis) cho trước. Cú pháp của hàm: concatenate((a_1, a_2, ...), axis)
```
>>>a=np.array([[1,2],[3,4]])
>>>b = np.array([[5,6],[7,8]])
>>>print np.concatenate((a,b))            # kết theo axis 0 (theo hàng)
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
>>>print np.concatenate((a,b), axis =1)       # kết theo axis 1 (theo cột)
[[1 2 5 6]
 [3 4 7 8]]
```

Hàm *amin* trả về giá trị nhỏ nhất trong một array được cho dọc theo một trục (axis) nào đó. Khi không nêu tên trục, có nghĩa là array được phẳng hóa trước khi tính min hay max.
```
>>>a = np.array([[3,7,5],[8,4,3],[2,4,9]])
>>>print np.amin(a,1)    # axis  =1
[3 3 2]
>>>print np.amin(a,0)       # axis  = 0
[2 4 3]
```

```
>>>print np.amax(a)
9
>>> a = np.array([[0, 2], [3, -1], [3, 5]], float)
>>> a.mean(axis=0)
array([ 2., 2.])
>>> a.mean(axis=1)
array([ 1., 1., 4.])
```

- Hàm *argmax* trả về chỉ số của phần tử lớn trị lớn nhất trong một array được cho dọc theo một trục nào đó. Khi không nêu tên trục, có nghĩa là array được phẳng hóa trước khi xác định argmax.

```
>>>my_2d_ar = np.array([[100,2,3],[4,5,600]])
>>> np.argmax(my_2d_ar, axis = 0)
array([0, 1, 1])
>>> np.argmax(my_2d_ar, axis = 1)
array([0, 2])
```



Hàm add

```
>>> a = np.arrange(4).reshape(2,2)
>>> a
array([[0, 1],
       [2, 3]])
>>> b = np.arrange(3, 7).reshape(2,2)
>>>b
array([[3, 4],
       [5, 6]])
>>> c= np.add(a, b)          # tương đương với c = a + b    cộng ma trận
>>> c
array([[3, 5],
       [7, 9]])
```

Hàm *multiply* (nhân từng phần tử với nhau)

```
>>>a = np.array([[0, 1],[2,3])
>>>b = np.array([[3, 4],[5,6])
>>> d = np.multiply(a, b)
```

```
>>> d
array([[0, 4],
       [10, 18]])
```
Tích ma trận (nhân matrận)
```
>>> A = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> B = np.array([[9,8,7],[6,5,4],[1,2,3]])
>>> A.dot(B)                    # tương đương với np.dot(A, B)
array([[ 24, 24, 24],
       [ 72, 69, 66],
       [120, 114, 108]])
```
Chuyển vị ma trận:
- Hàm T

```
>>> A = np.arrange(15).reshape(3,5)
>>> A
array([[ 0, 1, 2, 3, 4],
       [ 5, 6, 7, 8, 9],
       [10,11,12,13,14])
>>> A.T
array([[ 0, 5, 10],
       [ 1, 6, 11],
       [ 2, 7, 12],
       [ 3, 8, 13],
       [ 4, 9, 14]])
```
Giải hệ phương trình:

Cho một hệ phương trình như sau:

$$7x + 5y - 3z = 16$$
$$3x - 5y + 2z = -8$$
$$5x + 3y - 7z = 0$$

Hệ phương trình có thể được biểu diễn bằng hai ma trận: một ma trận hệ số (gọi tên là a trong thí dụ này) và một vector hằng số (gọi tên là b).
```
>>> a = np.array([[7,5,-3], [3,-5,2],[5,3,-7]])
>>> b = np.array([16,-8,0])
>>> x = np.linalg.solve(a, b)
>>> x
array([ 1., 3., 2.])
```
- Ta có thể kiểm tra lời giải có đúng không, bằng cách dùng hàm *np.allclose*.

```
>>> np.allclose(np.dot(a, x), b)
True
```
Nghịch đảo ma trận vuông

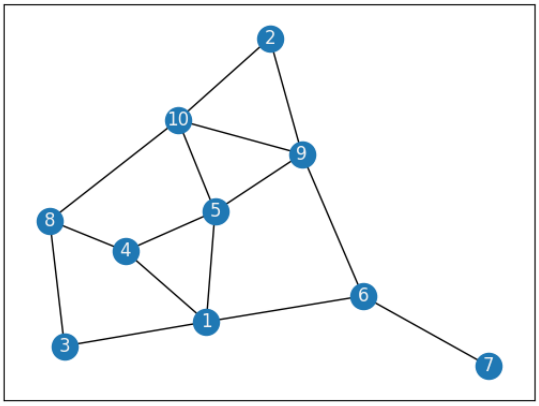
np.linalg.inv(a)

Tính định thức (determinant) của một ma trận vuông

np.linalg.det(a)

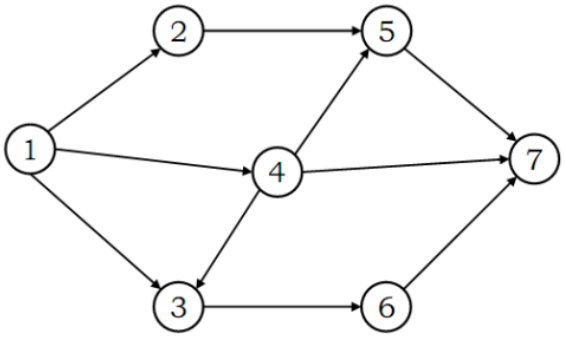
BÀI TẬP

Cho đồ thị trong hình bên dưới, thực hiện cho bài tập từ 1 đến 4:



1. Sử dụng ma trận kề để biểu diễn đồ thị.
2. Sử dụng danh sách cạnh để biểu diễn đồ thị.
3. Hiện thực lại giải thuật BFS – Duyệt đồ thị từ đỉnh 5
4. Hiện thực lại giải thuật DFS – Duyệt đồ thị từ đỉnh 7

Cho đồ thị trong hình bên dưới, thực hiện cho bài tập từ 5 đến 8:



5. Sử dụng ma trận kề để biểu diễn đồ thị.
6. Sử dụng danh sách cạnh để biểu diễn đồ thị.
7. Hiện thực lại giải thuật BFS – Duyệt đồ thị từ đỉnh 1
8. Hiện thực lại giải thuật DFS – Duyệt đồ thị từ đỉnh 4

—HẾT—