Defining the shape of the grid.

In [1]:

```
N = 5
```

In [2]:

```
import numpy as np
```

Sample array.

In [3]:

```
Array1 = np.arange(0,25).reshape(N,N)
Array1
```

Out[3]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [19]:

```
Array1 = np.array([[0,82,2,6,7],
                   [4,3,1,5,21],
                   [6,4,20,2,8],
                   [6,6,64,1,8],
                   [1,65,1,6,4]])
```

Creating permutations of all the paths possible.

- Logic used: for any path in a N X N grid, we can reach from the top left corner to the bottom right corner in (N-1) + (N-1) steps.
- There will be (N - 1) downs and (N - 1) ups to traverse the possible paths.

In [20]:

```
from itertools import permutations
```

In [21]:

```
moves = ['down']*(N-1) + ['right']*(N-1)
moves
```

Out[21]:

```
['down', 'down', 'down', 'down', 'right', 'right', 'right', 'right']
```

In [22]:

```
permut = list(set(permutations(moves)))
```

In [23]:

```
# Converting the (list of tuples) to (list of lists).  {Can skip.}
permut = [list(ii) for ii in permut]
```

In [24]:

```
# Top 10 permutations.
permut[:10]
```

Out[24]:

```
[['right', 'right', 'down', 'right', 'down', 'down', 'right', 'down'],
 ['right', 'down', 'right', 'down', 'right', 'right', 'down', 'down'],
 ['down', 'right', 'down', 'down', 'down', 'right', 'right', 'right'],
 ['down', 'right', 'right', 'right', 'right', 'down', 'down', 'down'],
 ['right', 'right', 'right', 'down', 'down', 'down', 'down', 'right'],
 ['right', 'right', 'down', 'down', 'right', 'right', 'down', 'down'],
 ['right', 'down', 'right', 'right', 'down', 'down', 'down', 'right'],
 ['right', 'right', 'right', 'right', 'down', 'down', 'down', 'down'],
 ['down', 'down', 'right', 'down', 'down', 'right', 'right', 'right'],
 ['right', 'down', 'down', 'down', 'right', 'right', 'down', 'right']]
```

Defining the right and down movements using functions.

In [25]:

```
def right(arr = Array1, row = 0, col = 0):
    '''When you move right the column number increases by one.'''
    if col<N:
        return (arr[row][col + 1], row, col+1)
    # Returns a tuple of array element after moving right and its row and column indices.
    else:
        print('Wall on the right.')
```

In [26]:

```
def down(arr = Array1, row = 0, col = 0):
    '''When you move right the row number increases by one.'''
    if row<N:
        return (arr[row+1][col], row+1, col)
    # Returns a tuple of array element after moving down and its row and column indices.
    else:
        print('Wall down.')
```

Keeping track of the possible paths traversed.

Initializing the array.

In [27]:

```
traverse = np.zeros(shape = np.array(permut).shape)
traverse[:5]
```

Out[27]:

```
array([[0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.]])
```

Filling the traverse list with all the paths taken from top left to the bottom right.

In [28]:

```
for jj in range(len(permut)):
    r,c = 0,0
    for ii in range(len(permut[jj])):
        if permut[jj][ii] == 'down':
            traverse[jj][ii],r,c = down(Array1, r, c)

        elif permut[jj][ii] == 'right':
            traverse[jj][ii],r,c = right(Array1, r , c)
```

In [29]:

```
traverse[:5]
```

Out[29]:

```
array([[82.,  2.,  1.,  5.,  2.,  1.,  8.,  4.],
       [82.,  3.,  1., 20.,  2.,  8.,  8.,  4.],
       [ 4.,  3.,  4.,  6., 65.,  1.,  6.,  4.],
       [ 4.,  3.,  1.,  5., 21.,  8.,  8.,  4.],
       [82.,  2.,  6.,  5.,  2.,  1.,  6.,  4.]])
```

Keeping track of the scores of all the paths as we traverse the array with that path from top left to the bottom right.

Fucntion that returns a score track (list) for a particular path.

In [30]:

```
def score_tracker_for_path(path_list):
    current_score = 0
    score_list = [0]
    for ii in range(len(path_list)):
        current_score = np.floor(current_score/2) + path_list[ii]
        score_list.append(current_score)

    return score_list
```

Applying it to all possible paths.

In [31]:

```
grand_score_tracker = []
for entry in traverse:
    grand_score_tracker.append(score_tracker_for_path(entry))
```

In [32]:

```
grand_score_tracker[:5]
```

Out[32]:

```
[[0, 82.0, 43.0, 22.0, 16.0, 10.0, 6.0, 11.0, 9.0],
 [0, 82.0, 44.0, 23.0, 31.0, 17.0, 16.0, 16.0, 12.0],
 [0, 4.0, 5.0, 6.0, 9.0, 69.0, 35.0, 23.0, 15.0],
 [0, 4.0, 5.0, 3.0, 6.0, 24.0, 20.0, 18.0, 13.0],
 [0, 82.0, 43.0, 27.0, 18.0, 11.0, 6.0, 9.0, 8.0]]
```

Creating a dictionary for the answer.

In [33]:

```
answer = dict()

# Keeping a list of the scores from all the paths.
top_scores = [scoretrack[-1] for scoretrack in grand_score_tracker]   # Scores from all pat

#np.where(np.array(top_scores) == min(top_scores))

# Storing the index for the path that gives minimum score.
index_for_optimum_score = top_scores.index(min(top_scores))

# Summarizing all info into a dictionary.
answer['Optimum Path Pattern'] = permut[index_for_optimum_score]
answer['Optimum Path'] = traverse[index_for_optimum_score]
answer['Score Tracker'] = grand_score_tracker[index_for_optimum_score]
answer['Optimum Score'] = min(top_scores)
```

Displaying the answer.

In [34]:

```
for key,item in answer.items():
    print(str(key) + ' = ' + str(item))
    print('---------------')
```

```
Optimum Path Pattern = ['down', 'right', 'right', 'right', 'down', 'down',
'down', 'right']
---------------
Optimum Path = [4. 3. 1. 5. 2. 1. 6. 4.]
---------------
Score Tracker = [0, 4.0, 5.0, 3.0, 6.0, 5.0, 3.0, 7.0, 7.0]
---------------
Optimum Score = 7.0
---------------
```

P. S. There will be other optimum paths as well. You can find their indices using #np.where(np.array(top_scores) == min(top_scores)).

The End.