SUMMARY

QUESTION-1: Write a generic function which will find all 4, 8 & m-paths between two points (x1, y1) & (x2, y2) in an image using python.

Language/Libraries Used: → Python
→ NumPy
→ CV2 (OpenCV)

Description of Variables Used:

- **I:** An Image as 2D matrix:

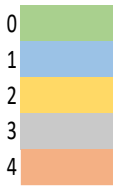
1	0	3	2	4
4	3	4	0	2
2	2	1	3	0
2	4	0	2	3
3	2	4	1	0

- **x1, y1, x2, y2**: Coordinates of start point P(x1, y1) & end point Q(x2, y2). For an example we have taken P(3, 0) & Q(1, 4) for the testing purpose.

- **V**: A set of values of Intensities which are considered Foreground. For an example we have taken V = {4,2} for the testing purpose.

- **path_type:** A variable which can takes values as 4, 8 or 10 for 4-path, 8-path and m-path respectively. We have tested the code on all 3 types of paths.

- Input Image Matrix **(I)**



1	0	3	2	4
4	3	4	0	2
2	2	1	3	0
2	4	0	2	3
3	2	4	1	0

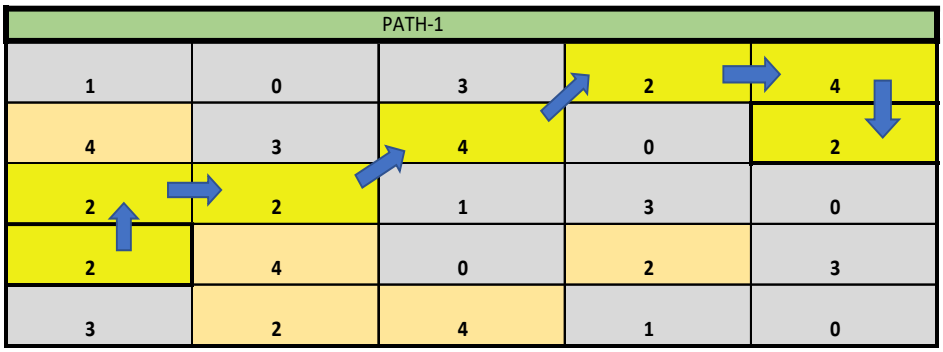
- Input Image Matrix with V = {4, 2} as foreground.

1	0	3	2	4
4	3	4	0	2
2	2	1	3	0
2	4	0	2	3
3	2	4	1	0

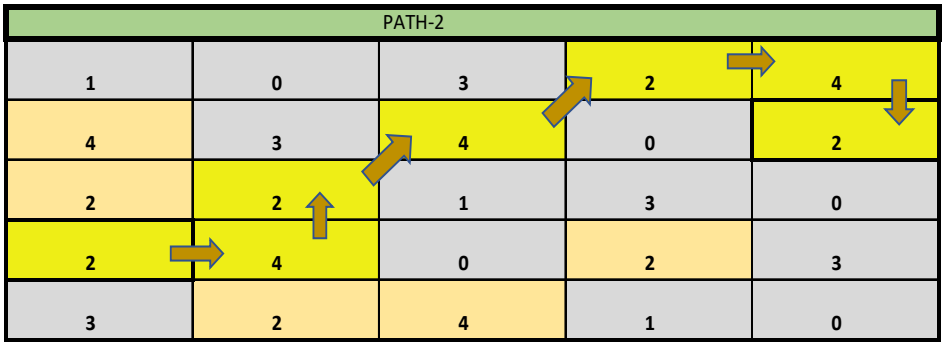
OUTPUT ANALYSIS

For the Path Type: **10**

```
Path Type: 10

    Length of Path #1: 6
    Path #1: (3,0):2 -> (2,0):2 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (0,4):4 -> (1,4):2

    Length of Path #2: 6
    Path #2: (3,0):2 -> (3,1):4 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (0,4):4 -> (1,4):2

    Total 2 10-path exists.

    Length of Shortest Path: 6
    Shortest Path #1: (3,0):2 -> (2,0):2 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (0,4):4 -> (1,4):2
    Shortest Path #2: (3,0):2 -> (3,1):4 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (0,4):4 -> (1,4):2
```



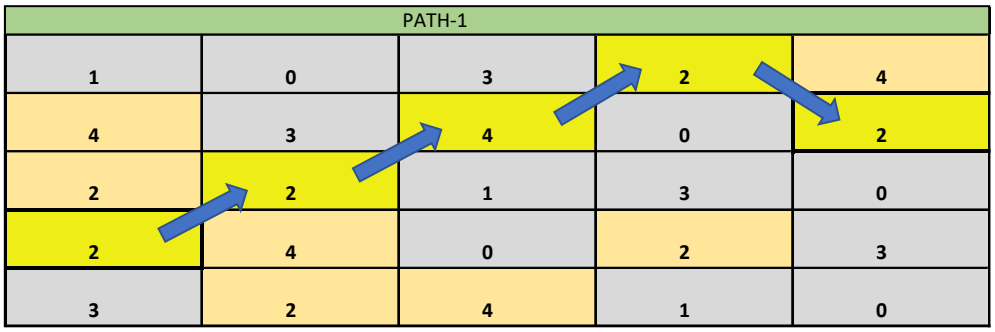
PATH-1		
Step No.	From	To
1	(3,0)	(2,0)
2	(2,0)	(2,1)
3	(2,1)	(1,2)
4	(1,2)	(0,3)
5	(0,3)	(0,4)
6	(0,4)	(1,4)



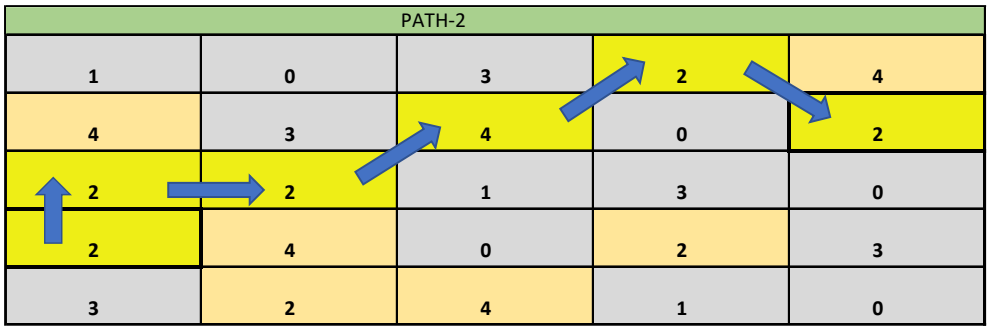
PATH-2		
Step No.	From	To
1	(3,0)	(3,1)
2	(3,1)	(2,1)
3	(2,1)	(1,2)
4	(1,2)	(0,3)
5	(0,3)	(0,4)
6	(0,4)	(1,4)

For the Path Type: **8**

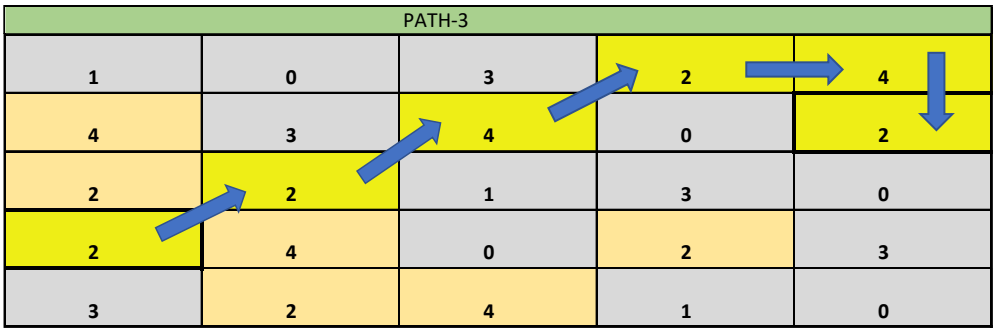
```
Path Type: 8

    Length of Path #1: 4
    Path #1: (3,0):2 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (1,4):2

    Length of Path #2: 5
    Path #2: (3,0):2 -> (2,0):2 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (1,4):2

    Length of Path #3: 5
    Path #3: (3,0):2 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (0,4):4 -> (1,4):2

    Total 3 8-path exists.

    Length of Shortest Path: 4
    Shortest Path #1: (3,0):2 -> (2,1):2 -> (1,2):4 -> (0,3):2 -> (1,4):2
```



PATH-1		
Step No.	From	To
1	(3,0)	(2,1)
2	(2,1)	(1,2)
3	(1,2)	(0,3)
4	(0,3)	(1,4)



PATH-2		
Step No.	From	To
1	(3,0)	(2,0)
2	(2,0)	(2,1)
3	(2,1)	(1,2)
4	(1,2)	(0,3)
5	(0,3)	(1,4)



PATH-3		
Step No.	From	To
1	(3,0)	(2,1)
2	(2,1)	(1,2)
3	(1,2)	(0,3)
4	(0,3)	(0,4)
5	(0,4)	(1,4)

For the Path Type: **4**

- No path found.

```
Path Type: 4

No Path Found.
```

RESULTS

- The total number of **m-path between P & Q is 2**.
- M-Path_1 is (3,0) => (2,0) => (2,1) => (1,2) => (0,3) => (0,4) => (1,4) and has length of path as 6.
- M-Path_2 is (3,0) => (3,1) => (2,1) => (1,2) => (0,3) => (0,4) => (1,4) and has length of path as 6.
- Since we have two m-paths of same length, both m-path_1 and m-path_2 are the shortest m-paths between P & Q
- The total number of **8-path between P & Q is 3.**
- 8-Path_1 is (3,0) => (2,1) => (1,2) => (0,3) => (1,4) and has length of path as 4.
- 8-Path_2 is (3,0) => (2,0) => (2,1) => (1,2) => (0,3) => (1,4) and has length of path as 5.
- 8-Path_3 is (3,0) => (2,1) => (1,2) => (0,3) => (0,4) => (1,4) and has length of path as 5.
- The shortest 8-path is the '8-Path_1' and its length is 4.
- The **4-path between P & Q does not exist** for the given image matrix as both possible 4-paths across (2,1) and (4,2) does not have any 4-adjacent pixel next to them.

QUESTION 2: Digital Image Creation (non-overlapping rectangles) using Python.

Language/Libraries Used: → Python
→ NumPy
→ CV2 (OpenCV)

Description of Variables Used:

M x N -- Is the image size.

Border -- Black Border of thickness (Border).

n -- Number of non-overlapping rectangles.

[w1, w2] -- Width uniformly distributed in range [w1, w2] for rectangles.

Alpha -- Rectangles height to width ratio.

Orientation -- Landscape (1) or Portrait (2).

Vf and Vb -- Foreground and background intensity (optional).

M_total, N_total -- Final dimension after adding border.

```
M_total = M + 2 * (border)
N_total = N + 2 * (border)
```

corner_i, corner_j -- Corner pixel point of the rectangles to be drawn.

Discussion Related to Function Used in the Code:

```
def valid_rectangle(image_bg, M_total, N_total, height, width,
border, Vb, rect_orientation, corner_i, corner_j):
```

→ Above function is used in our code to check if the new rectangle to be drawn has valid and non-overlapping corners with any other existing rectangle.

→ In this function, code for both landscape and portrait orientation has written.

→ This function is recursively called to find out non-overlapping space for the new rectangle.

```
def create_rectangles(M, N, border, n, w1, w2, alpha,
orientation, Vf=[0], Vb=[255]):
```

→ This function is used in our code to draw valid rectangles and to create final image using *cv2.imwrite()* OpenCV function.

→ Vf and Vb values are passed in function argument itself in case if Vf(black) and Vb(white) values are not passed by the user.

→ In case of exception *RecursionError* i.e., if the condition of non-overlapping rectangles is not satisfied, we increase the size of the image to (2*M) x (2*N) and redraw the rectangles.

```
def main():
```

→ In main() function all the value of parameter is passed and under this function only **create_rectangles** function is called.

OUTPUT ANALYSIS

CASE-I: All the parameters are passed along with Vf = [0, 128] and Vb = [129, 255]

INPUT

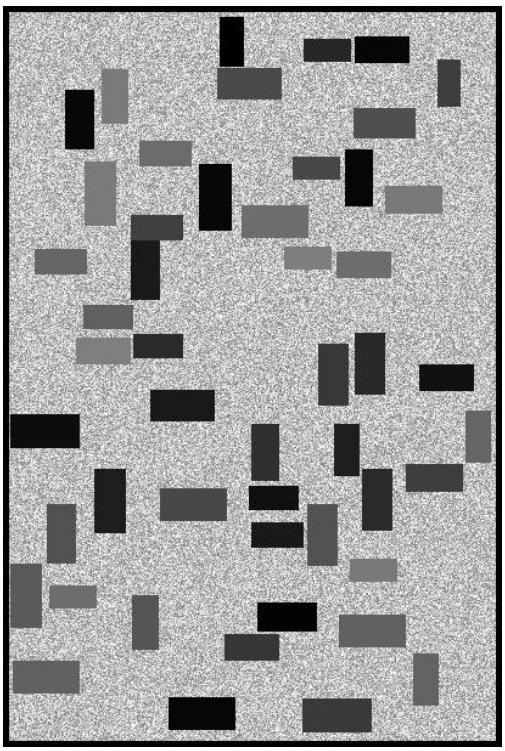
```
M = 300                  # no of rows
N = 200                  # no of columns
border = 5               # size of the border
n = 50                   # number of rectangles to fit
w1 = 20                  # lower bound for the width of rectangle
w2 = 30                  # upper bound for the width of rectangle
alpha = 2                # fixed [height, width] ratio of
rectangle
orientation = [1, 2]     # uniformly distributed orientation of
rectangle [portrait, landscape]
Vf = [0, 128]            # foreground colors distributed
uniformly, if not provided then default = 0
Vb = [129, 255]          # background colors distributed
uniformly, if not provided then default = 255
```

Function Call:

```
create_rectangles(M, N, border, n, w1, w2, alpha, orientation,
Vf, Vb)
```

RESULT CASE-I:

Final shape of the image: (610, 410)



- From this output, we can see all the rectangles are non-overlapping.
- Rectangles colour intensity is uniformly distributed over range Vf given, also same for the Vb.

CASE-II: All the parameters are same as previous case except Vf and Vb values not being passed as they are optional.

INPUT

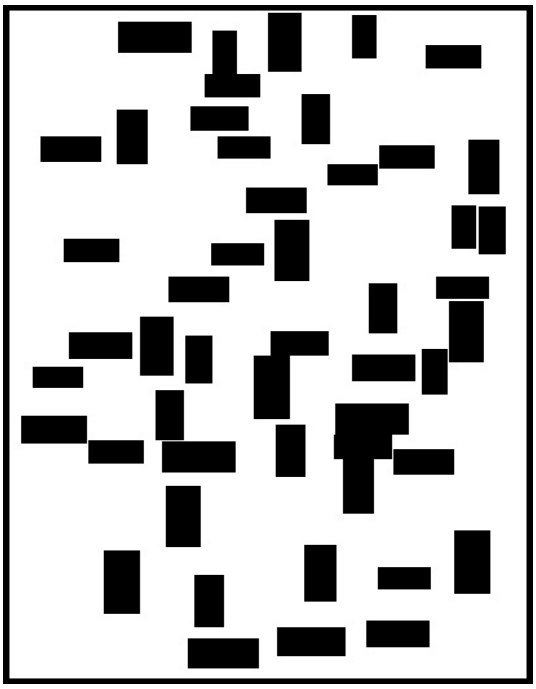
```
M = 300                    # no of rows
N = 200                    # no of columns
border = 5                 # size of the border
n = 50                     # number of rectangles to fit
w1 = 20                    # lower bound for the width of rectangle
w2 = 30                    # upper bound for the width of rectangle
alpha = 2                  # fixed [height, width] ratio of
rectangle
orientation = [1, 2]       # uniformly distributed orientation of
rectangle [portrait, landscape]
```

Function Call:

```
create_rectangles(M, N, border, n, w1, w2, alpha, orientation)
```

RESULT CASE-II:

Final shape of the image: (610, 410)



- From this image, it is clear when Vf and Vb values are not provided we get non-overlapping rectangles with Vf=0 and Vb=255 along with black border.