Digital Image Processing
Assignment 1
Group No.4

Abinash Acharya (11840050)
Ashutosh Garg (11840250)
Thummala Milind Kesar (11841160)

September 2021

1 Q1 Digital Paths

The program files corresponding to Q1 are q1.py and q1_test.py.
q1.py contains the main function paths_info()
q1_test.py is a test script which calls this function with the example inputs provided in Q1.

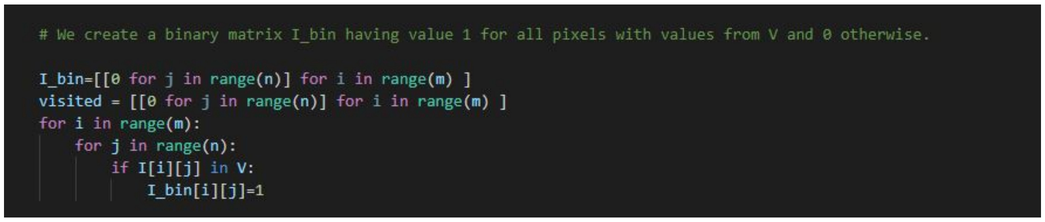
Note :- The top-left corner of the 2-D image is considered as the origin to index the pixels. Also, zero-indexing has been used here.

The description of the main function used is as follows :
paths_info(I,x1,y1,x2,y2,V,path_type)
The various parameters are defined below :

(i) (x_1,y_1) :- These are respectively the x-coordinate (row-number of pixel) and y-coordinate (column-number of pixel) of source point.
(ii) (x_2,y_2) :- These are respectively the x-coordinate (row-number of pixel) and y-coordinate (column-number of pixel) of destination point.
(iii) I :- This is a 2-D matrix specifying the intensity values of the image.
(iv) V :- List of pixel values forming the foreground.
(v) Path_type :- This specifies type of path to be found. It can be 4 for

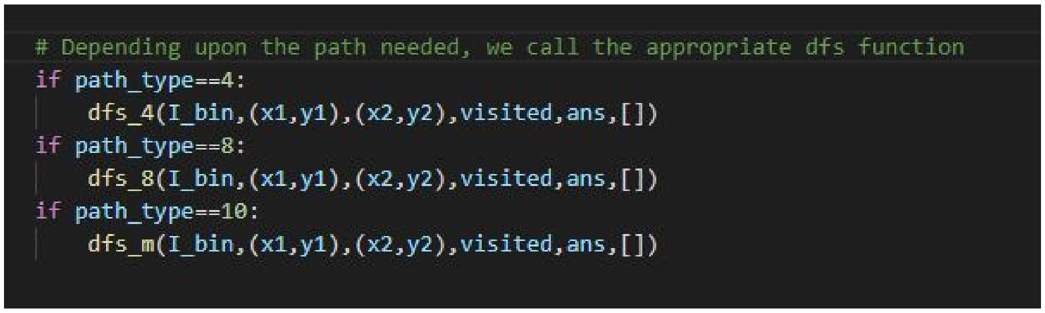
4-path, 8 for 8-path and 10 for m-path.

The steps involved in the process are :
(i) Binarizing the matrix using the list V.



```
# We create a binary matrix I_bin having value 1 for all pixels with values from V and 0 otherwise.

I_bin=[[0 for j in range(n)] for i in range(m) ]
visited = [[0 for j in range(n)] for i in range(m) ]
for i in range(m):
    for j in range(n):
        if I[i][j] in V:
            I_bin[i][j]=1
```

(ii) Now, depending on the path type we call the appropriate custom DFS (Depth First Search) function.

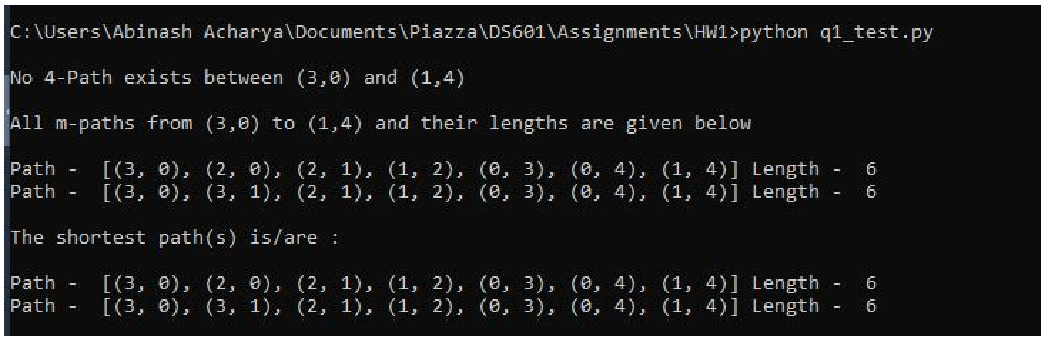


```
# Depending upon the path needed, we call the appropriate dfs function
if path_type==4:
    dfs_4(I_bin,(x1,y1),(x2,y2),visited,ans,[])
if path_type==8:
    dfs_8(I_bin,(x1,y1),(x2,y2),visited,ans,[])
if path_type==10:
    dfs_m(I_bin,(x1,y1),(x2,y2),visited,ans,[])
```

(iii) There are 3 DFS functions, namely, dfs_4, dfs_8 and dfs_m, for 4-paths, 8-paths and m-paths respectively.
All of these functions recursively call themselves on adjacent pixels until the destination pixel is reached and the paths obtained (as lists of tuples i.e. coordinates) are appended to the variable 'ans'.
(iv) The function paths_info returns a 3-tuple containing ans (List of paths), Lengths (their respective lengths) and shortest_paths(List of shortest length paths).
(v) Sample image and its output.

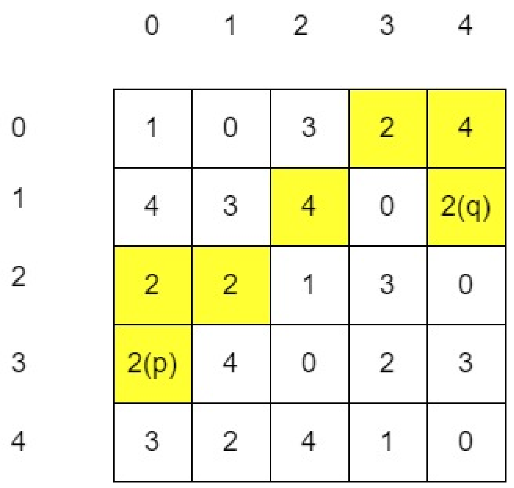
	0	1	2	3	4
0	1	0	3	2	4
1	4	3	4	0	2(q)
2	2	2	1	3	0
3	2(p)	4	0	2	3
4	3	2	4	1	0

V={4,2}
The output returned is as follows :

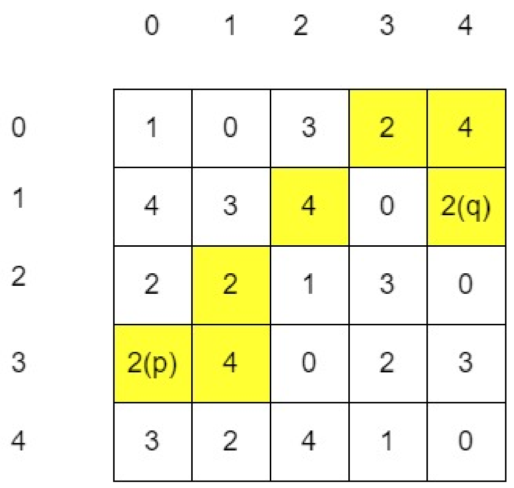


```
C:\Users\Abinash Acharya\Documents\Piazza\DS601\Assignments\HW1>python q1_test.py

No 4-Path exists between (3,0) and (1,4)

All m-paths from (3,0) to (1,4) and their lengths are given below

Path -  [(3, 0), (2, 0), (2, 1), (1, 2), (0, 3), (0, 4), (1, 4)] Length -  6
Path -  [(3, 0), (3, 1), (2, 1), (1, 2), (0, 3), (0, 4), (1, 4)] Length -  6

The shortest path(s) is/are :

Path -  [(3, 0), (2, 0), (2, 1), (1, 2), (0, 3), (0, 4), (1, 4)] Length -  6
Path -  [(3, 0), (3, 1), (2, 1), (1, 2), (0, 3), (0, 4), (1, 4)] Length -  6
```

As we can see, there are no 4-paths between p(3,0) and q(1,4)
There are 2 m-paths between p(3,0) and q(1,4).
Both are of same length i.e. 6.
So, both of them are listed as shortest m-paths.
These 2 m-paths have been shown in the figure below



	0	1	2	3	4
0	1	0	3	**2**	**4**
1	4	3	**4**	0	**2(q)**
2	**2**	**2**	1	3	0
3	**2(p)**	4	0	2	3
4	3	2	4	1	0



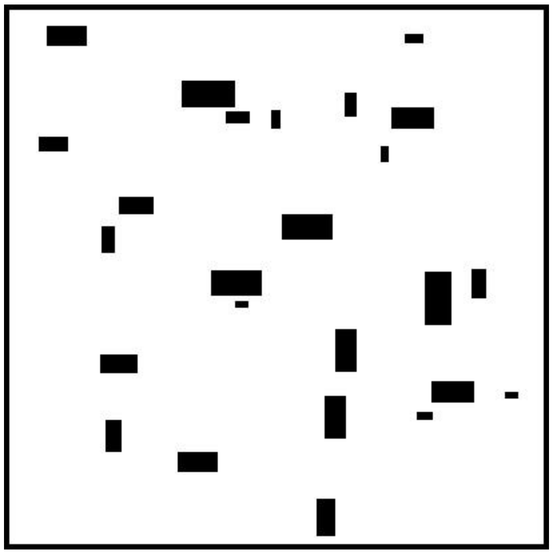
	0	1	2	3	4
0	1	0	3	**2**	**4**
1	4	3	**4**	0	**2(q)**
2	2	**2**	1	3	0
3	**2(p)**	**4**	0	2	3
4	3	2	4	1	0

2 Q2 Rectangle Creation

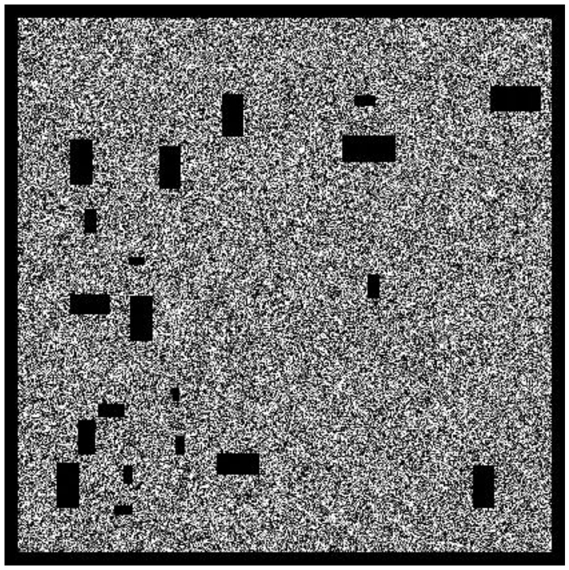
The python file corresponding to this question is Q2.py. The user can specify the different parameters to generate different digital images as explained in the ReadME file.
The description of the main function is as follows:
create_rectangles(M, N, border, n, w1, w2, alpha, orientation, vf, vb). The various parameters are defined below:

- (M,N): Specify the size of the image, i.e., the image is of size MxN.
- Border: It specifies the thickness of the black border around the image.
- n: The number of non-overlapping rectangles.
- (w1,w2): Specify the uniform distribution of widths of rectangles.
- alpha: The ratio of height to width of the rectangles.
- orientation: The orientation of the image (Two possible orientations).
- vf: An array specifying the uniform distribution of all intensity values of foreground. Default = [0] (black).
- vb: An array specifying the uniform distribution of all intensity values of background. Default = [255] (white).

To handle cases where we are not able to find n non overlapping rectangles in the specified size MxN after $2 * n$ iterations, we increase the size of image to 2Mx2N as suggested in the question.
Some examples of outputs of our function for different parameters specified by user are as follows:
M=400, N=400, border=4, n=25, alpha=2, w1=5, w2=20, vf=[0],vb =[255]



M=400, N=400, border=10, n=20, alpha=2, w1=5, w2=20, vf=[0, 1], vb=[129, 10, 255].



We observe the above image because vf is randomly selected from [0,1] and vb from [129,10,255]. Another observation is that the border thickness has increased as we have specified it to be 10.