Digital Imaging Systems

Project 01 – Option 0

Name- Manthan Sunil Talegaonkar

Unity Id- mstalega

Q1.

Code: Also provided in Zip file. Better to open it in a Jupyter Notebook or Google Colab.

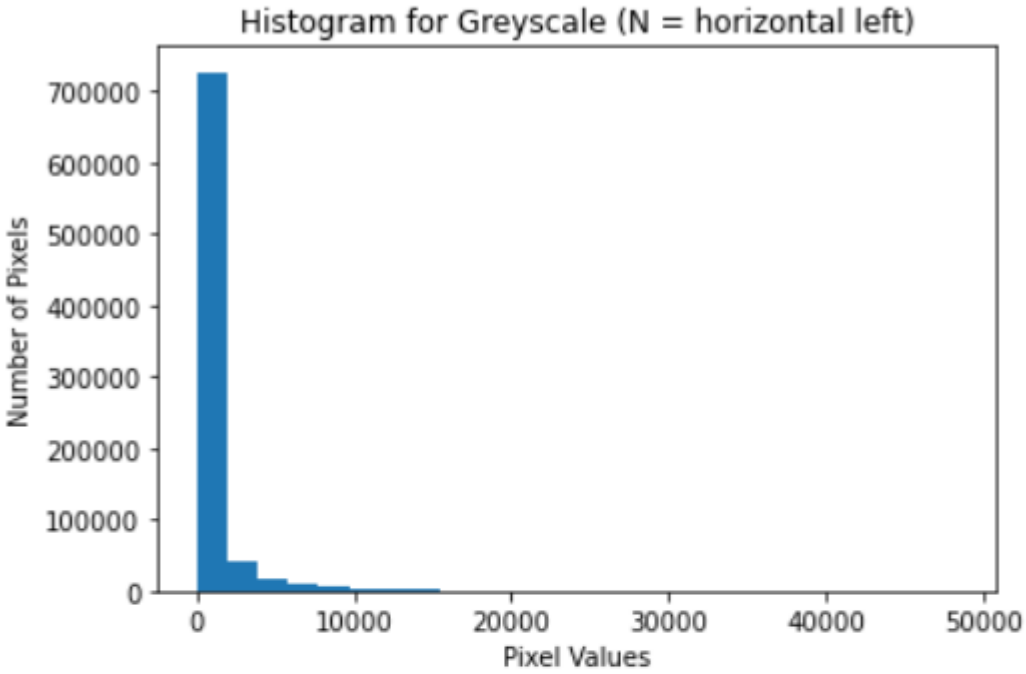
```
from matplotlib.colors import hsv_to_rgb
from numpy.core.memmap import dtype
import cv2
from matplotlib import pyplot as plt
import numpy as np
import time as t
im_bgr = cv2.imread(r"D:\ECE558\ECE558-HW01\ECE558-HW01\wolves.png",1) #
im_grey = cv2.imread(r"D:\ECE558\ECE558-HW01\ECE558-HW01\wolves.png",0)

ihsv = cv2.cvtColor(im_bgr, cv2.COLOR_BGR2HSV) #hsv
ilab = cv2.cvtColor(im_bgr, cv2.COLOR_BGR2Lab) #lab

def histogram(img,bins,title):
    plt.hist(img.ravel(),bins);
    plt.title(title)
    plt.xlabel('Pixel Values')
    plt.ylabel('Number of Pixels')
    plt.show()


# In[21]:


im_grey = cv2.imread(r"D:\ECE558\ECE558-HW01\ECE558-HW01\wolves.png",0)
imgrey = np.array(im_grey,dtype='int32')
def neighbor(mat,q):
    height,width= im_grey.shape
    z = np.zeros((height,width),dtype='int32')
    for i in range(height):
        for j in range(width):
            if i+q[0]<height and j+q[1]<width and i+q[0]>0 and j+q[1]>0:
                z[i,j] += (mat[i,j]-mat[i+q[0],j+q[1]])**2
    return z


# In[23]:


#Intensity
t0 = t.time()
g1 = neighbor(imgrey,[0,-1])
t1 =t.time()
```

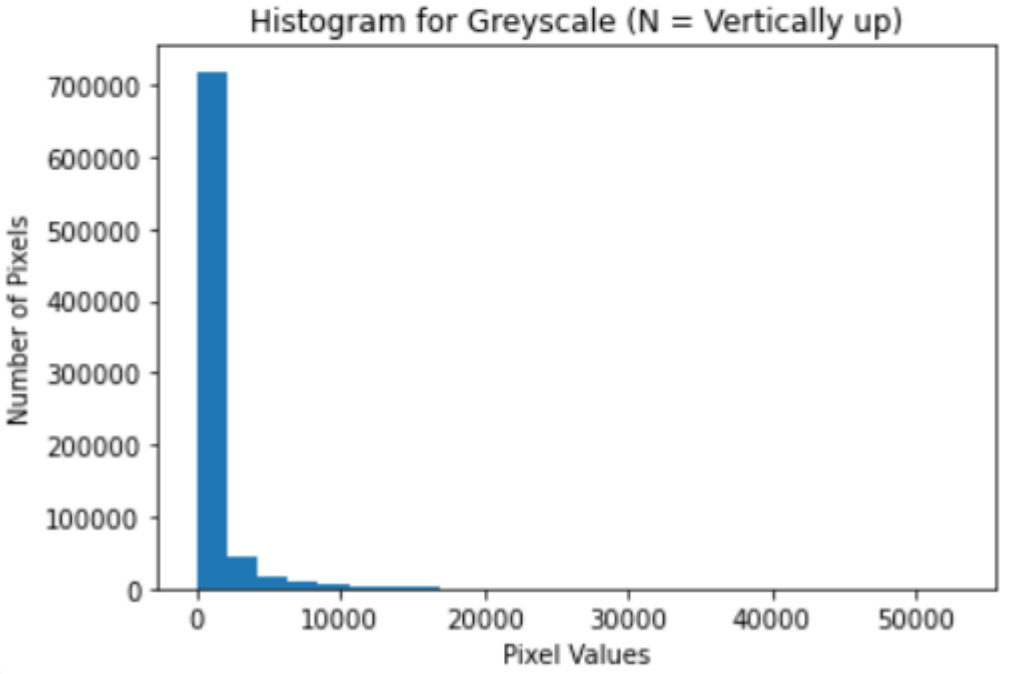
```
g2 = neighbor(imgrey,[-1,0])

# calculate histogram
print(f'Time taken by the function is {t1-t0} Sec\n')

histogram(g1,25,'Histogram for Greyscale (N = horizontal left)')
histogram(g2,25,'Histogram for Greyscale (N = Vertically up)')


# In[16]:



def neighbor_3d(mat,q):
  height,width,depth= mat.shape
  img = np.array(mat,dtype='int32')
  u = np.zeros((height,width),dtype='int32')
  for i in range(height):
    for j in range(width):
      for k in range(depth):
        if i+q[0]<height and j+q[1]<width and i+q[0]>=0 and j+q[1]>=0:
          u[i,j] += np.square(img[i,j,k]-img[i+q[0],j+q[1],k])
  return u


# In[17]:


t0 = t.time()
bgr1 = neighbor_3d(im_bgr,[0,1])
t1 = t.time()
bgr2 = neighbor_3d(im_bgr,[1,0])

hsv1 = neighbor_3d(ihsv,[-1,-1])
hsv2 = neighbor_3d(ihsv,[-1,1])

lab1 = neighbor_3d(ilab,[1,-1])
lab2 = neighbor_3d(ilab,[1,1])


# calculate histogram

print(f'Time taken by the function is {t1-t0} Sec')

histogram(bgr1,25,'Histogram for BGR (N = horizontal right)')
histogram(bgr2,25,'Histogram for BGR (N = Vertically down)')

histogram(hsv1,25,'Histogram for HSV (N = Diagonal left up)')
histogram(hsv2,25,'Histogram for HSV (N = Diagonal right up)')

histogram(lab1,25,'Histogram for Lab (N = Diagonal left down)')
histogram(lab2,25,'Histogram for Lab (N = Diagonal right down)')
```

Results-

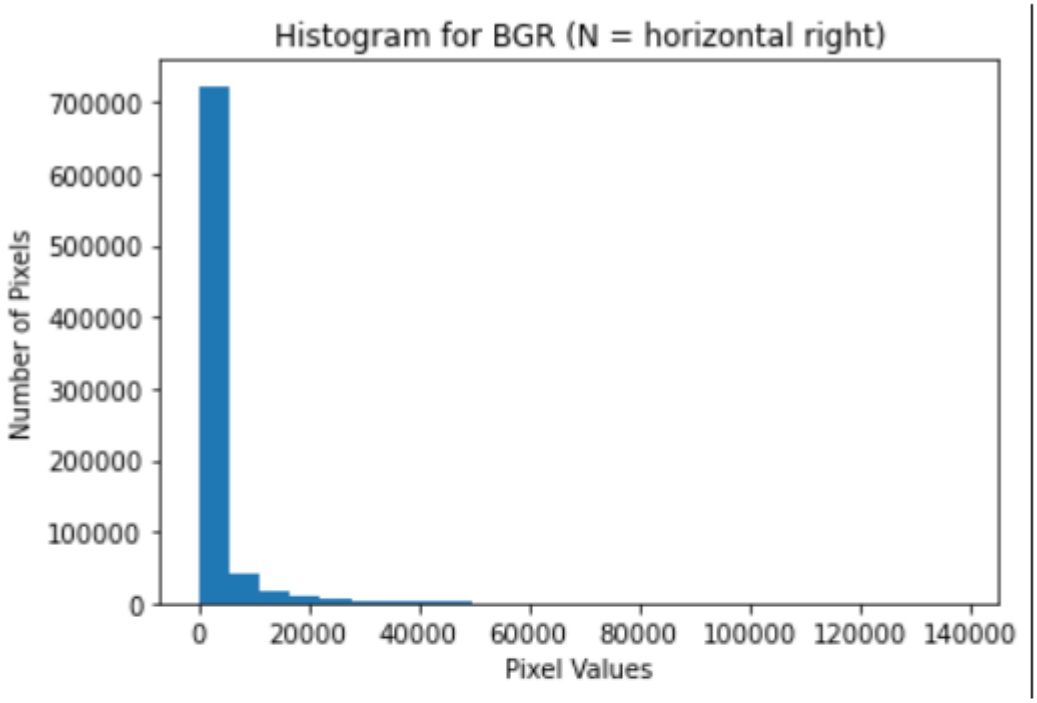
Run Time is also mentioned for particular function.

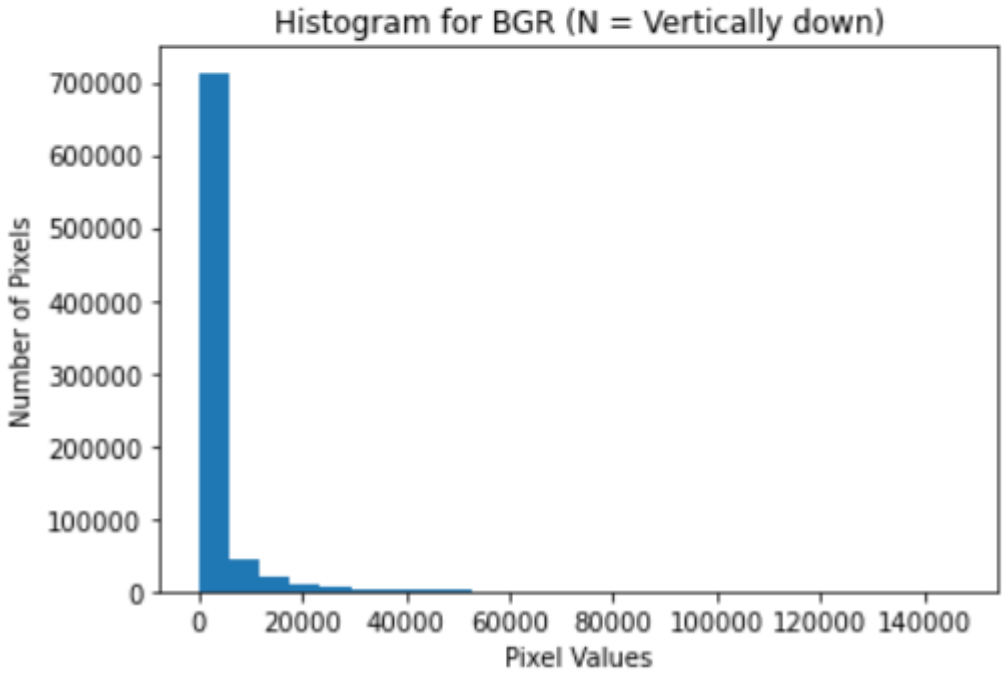
```
Time taken by the function is 1.121391773223877 Sec
```

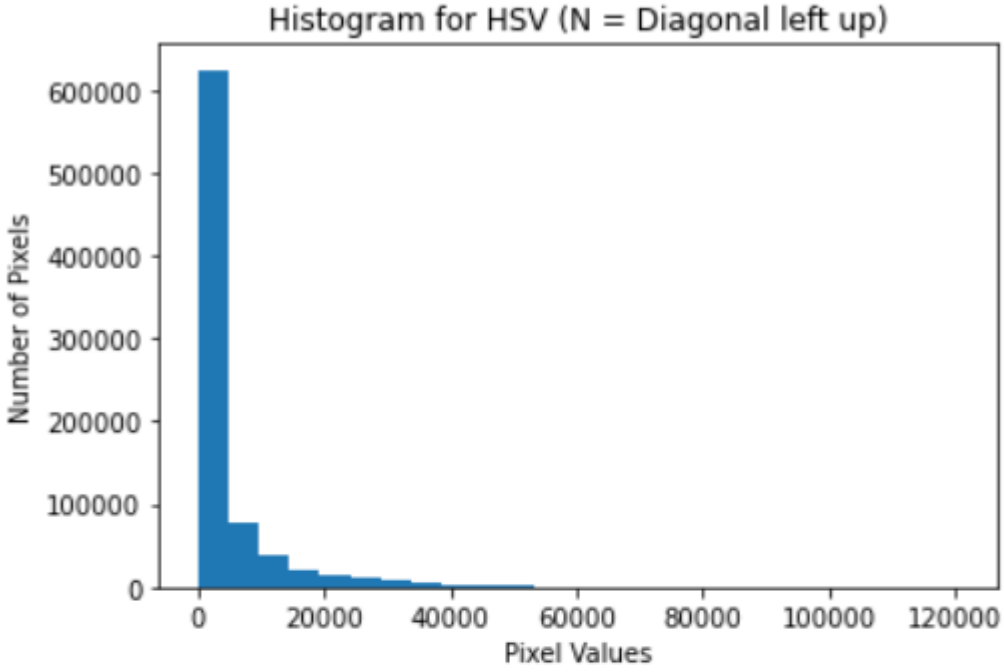


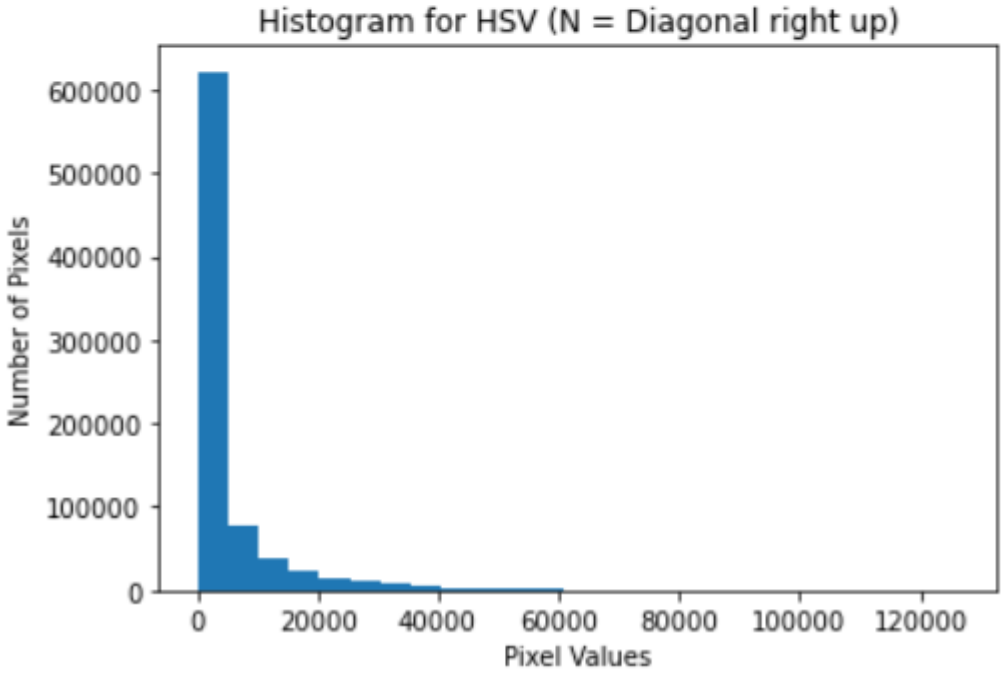


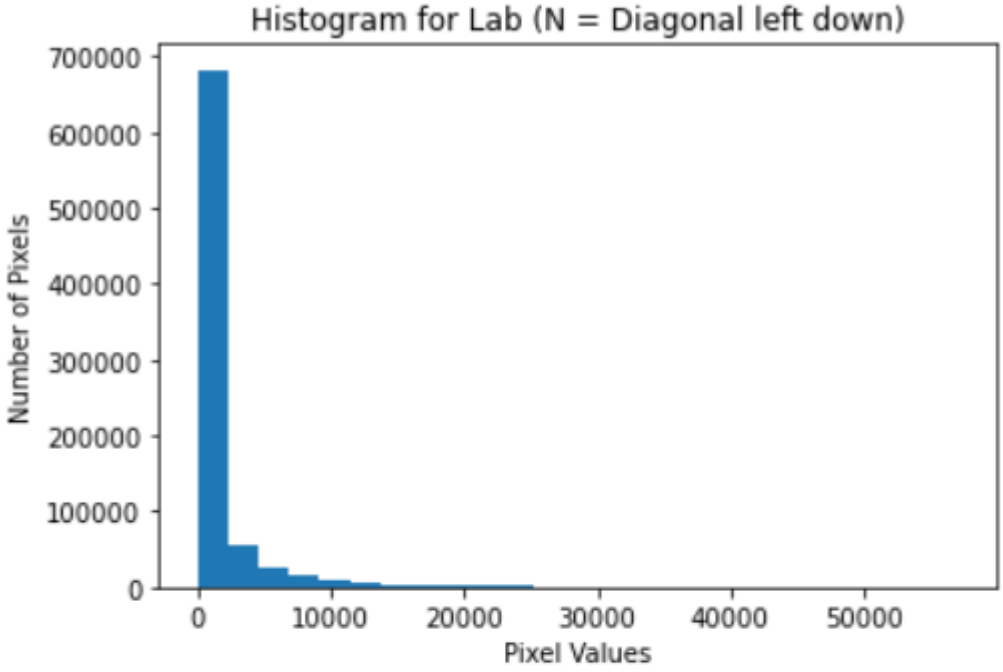
```
Time taken by the function is 6.551021099090576 Sec
```

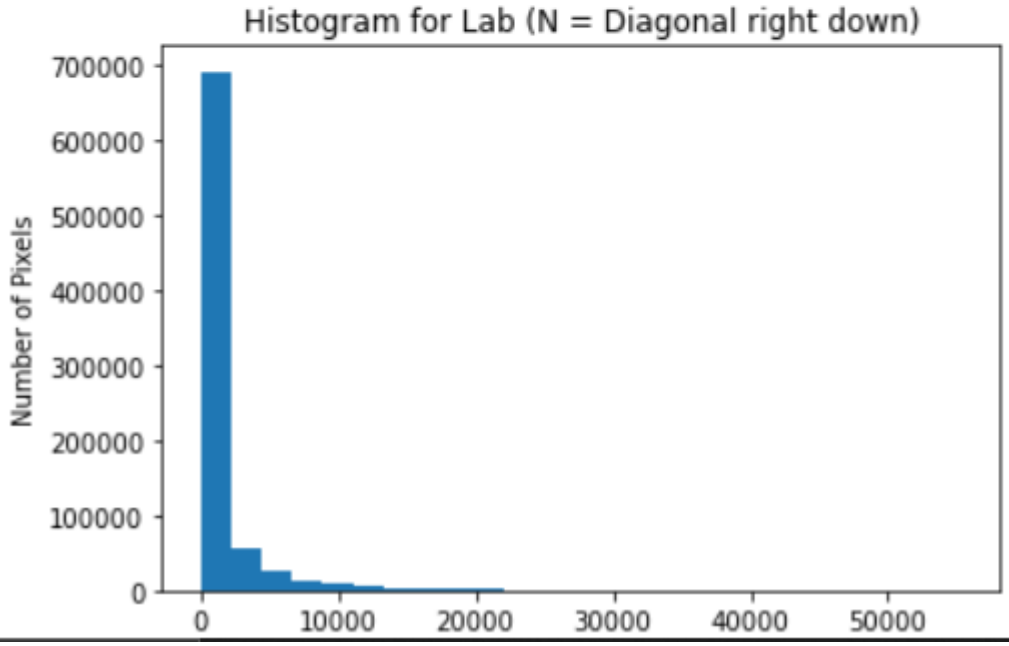





Histogram for HSV (N = Diagonal left up)
Number of Pixels
600000
500000
400000
300000
200000
100000
0
0
20000
40000
60000
80000
100000
120000
Pixel Values


Histogram for HSV (N = Diagonal right up)
Number of Pixels
600000
500000
400000
300000
200000
100000
0
0
20000
40000
60000
80000
100000
120000
Pixel Values


Histogram for Lab (N = Diagonal left down)
Number of Pixels
Pixel Values
0
100000
200000
300000
400000
500000
600000
700000
0
10000
20000
30000
40000
50000

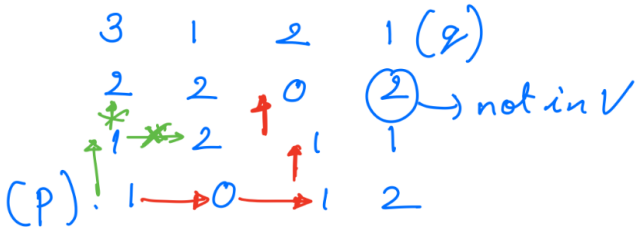

Histogram for Lab (N = Diagonal right down)
Number of Pixels
0
100000
200000
300000
400000
500000
600000
700000
0
10000
20000
30000
40000
50000

Q2.

Q.

a). $V = \{0, 1\}$

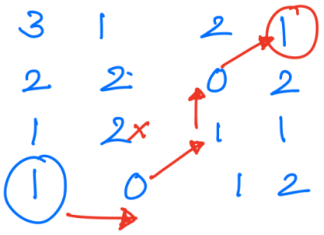
4-path



$N_4(p) = \{0, 1\}$ Both lie in V

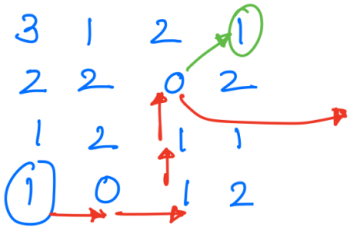
Not possible

8-path.



path length = 4

m - path



$N_4(p) = \{2, 2, 2\}$

$N_4(q) = \{2, 2\}$

$N_4(p) \cap N_4(q) = \{2, 2\}$

$\Downarrow$

Not lying in V

$\Leftarrow$

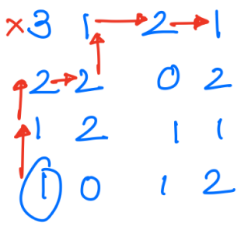
m-path is possible

length = 5

b.

b). $V = \{1, 2\}$.

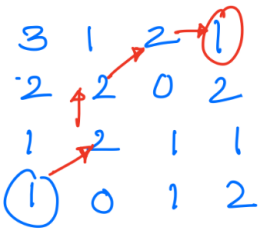
4-path



possible

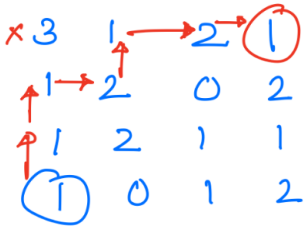
Length = 6

8-path



Length = 4

m-path



Length = 6

c.

Considered total of 5 Examples and included in the code as well.

```
# BFS algorithm for 4-,8- neighbor
import collections
import numpy as np
import cv2

def bfs(img,V,p,q,direc,parent):
  # p = start pixel location (x,y)
  # q = end pixel location (x,y)
  # img is the  given matrix
  # V is the points which can be visited
  # direc defines the 4, 8, m path for example [(0,1),(0,-1),(1,0),(-1,0)]
or [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
  # using que to know which neighbor to visit next

  if p[0]<0 or p[1]>=len(img[0]) or q[0]<0 or q[1]>=len(img[0]):
    return False


  if img[p[0],p[1]] not in V or img[q[0],q[1]]  not in V:
    return False

  que = collections.deque([(p[0],p[1],0)])

  for r in range(len(img)):
    for c in range(len(img[0])):
      parent[r,c]= tuple([r,c])

  #Visited
  visited = np.zeros((len(img),len(img[0])),dtype = bool)

  visited[(p[0],p[1])]= True

  while que:

    x,y,leng = que.popleft()

    if (x,y) == (q[0],q[1]):
      return (leng,x,y)

    for e in direc:
      x_new = x + e[0]
      y_new = y + e[1]

      if 0<=x_new<len(img) and 0<=y_new<len(img[0]) and img[x_new,y_new] in
V and visited[x_new,y_new] == False:

        que.append((x_new,y_new,leng+1))
        parent[x_new,y_new]= (x,y)

        #Mark Visited
        visited[x_new,y_new]= True

  return False


# In[20]:
```

```
# General Path for 4-,8-,m-
def path(parent,cord):
  x =cord[0]
  y =cord[1]

  path_list =[]

  while parent[x,y] != (x,y):
    path_list.append((x,y))
    (x,y) = parent[x,y]
  path_list.append((x,y))
  return list(reversed(path_list))


# In[21]:


# For m-Neighbor condition

def check(x_new,y_new,img,visited,V): # Checking boundary and if the pixel
is already visited
  if 0<=x_new<len(img) and 0<=y_new<len(img[0]) and img[x_new,y_new] in V
and visited[x_new,y_new] == False:
    return True
  return False

def m_adj_check(x_diag,x,y_diag,y,img,V): # checking condition for m-
adjacency
  if x_diag == x - 1 and y_diag == y - 1 and ((img[x-1,y] in V) or
(img[x,y-1] in V)):
    return False
  if x_diag == x + 1 and y_diag == y - 1 and ((img[x+1,y] in V) or
(img[x,y-1] in V)):
    return False
  if x_diag == x - 1 and y_diag == y + 1 and ((img[x,y+1] in V) or (img[x-
1,y] in V)):
    return False
  if x_diag == x + 1 and y_diag == y + 1 and ((img[x,y+1] in V) or
(img[x+1,y] in V)):
    return False
  return True

def bfs_m(img,V,p,q,direc,parent,diag): # BFS algorithm for m-neighbor
  # p = start pixel location (x,y)
  # q = end pixel location (x,y)
  # img is the  given matrix
  # V is the points which can be visited
  # direc defines the 4, 8, m path for example [(0,1),(0,-1),(1,0),(-1,0)]


  if img[p[0],p[1]] not in V or img[q[0],q[1]]  not in V:
    return False

  que = collections.deque([(p[0],p[1],0)])

  for r in range(len(img)):
    for c in range(len(img[0])):
      parent[r,c]= tuple([r,c])

  #Visited array
```

```
  visited = np.zeros((len(img),len(img[0])),dtype = bool)

  visited[(p[0],p[1])]= True

  while que:

    x,y,leng = que.popleft()

    if (x,y) == (q[0],q[1]):
      return (leng,x,y)

    for e in diag:
      x_diag = x + e[0]
      y_diag = y + e[1]

      if not check(x_diag,y_diag,img,visited,V):
        continue
      if not m_adj_check(x_diag,x,y_diag,y,img,V):
        continue

      que.append((x_diag,y_diag,leng+1))
      parent[x_diag,y_diag]= (x,y)

      visited[x_diag,y_diag]= True

    for e in direc:
      x_new = x + e[0]
      y_new = y + e[1]

      if check(x_new,y_new,img,visited,V):

        que.append((x_new,y_new,leng+1))

        parent[x_new,y_new]= (x,y)

        visited[x_new,y_new]= True

  return False
##########################################################################
##########################################################
##########################################################################
##########################################################
# In[25]:



#Example 1  The given Matrix
A = np.array(([3,1,2,1],[2,2,0,2],[1,2,1,1],[1,0,1,2]))
print(A)
p = (3,0)

q = (0,3)

V = [0,1]
print(f'V={V}')
direc1 = [(0,1),(0,-1),(1,0),(-1,0)]
direc2 = [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
#(1,1),(1,-1),(-1,1),(-1,-1)
parent = np.empty((len(A),len(A[0])),dtype = object)

len_cord1  = bfs(A,V,p,q,direc1,parent)
```

```
len_cord2 = bfs(A,V,p,q,direc2,parent)
#for 4-neighbor
if len_cord1:
  print(f'Shortest Length of {len(direc1)} neighbor is {len_cord1[0]}')
  final_path = path(parent,len_cord1[1:])
  print(f'Path of {len(direc1)} neighbor is {final_path}')

else:
  print(f"For {len(direc1)}-neighbor: No Path Found")
# for 8- neighbor
if len_cord2:
  print(f'Shortest Length of {len(direc2)} neighbor is {len_cord2[0]}')
  final_path = path(parent,len_cord2[1:])
  print(f'Path of {len(direc2)} neighbor is {final_path}')

else:
  print(f"For {len(direc2)}- neighbor No Path Found")

direc = [(0,1),(0,-1),(1,0),(-1,0)]
diag =[(1,1),(1,-1),(-1,1),(-1,-1)]
parent = np.empty((len(A),len(A[0])),dtype = object)

len_cord  = bfs_m(A,V,p,q,direc,parent,diag)
# for m-neighbor
if len_cord:
  print(f'Shortest Length of m-neighbor is {len_cord[0]}')
  final_path = path(parent,len_cord[1:])
  print(f'Path of m-neighbor is {final_path}')

else:
  print("For m-neighbor:No Path Found")


# In[26]:


#Example 2 for V=[1,2]
A = np.array(([3,1,2,1],[2,2,0,2],[1,2,1,1],[1,0,1,2]))
print(A)
p = (3,0)

q = (0,3)

V = [1,2]
print(f'V={V}')

direc1 = [(0,1),(0,-1),(1,0),(-1,0)]
direc2 = [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
#(1,1),(1,-1),(-1,1),(-1,-1)
parent1 = np.empty((len(A),len(A[0])),dtype = object)
parent2 = np.empty((len(A),len(A[0])),dtype = object)
len_cord1  = bfs(A,V,p,q,direc1,parent1)
len_cord2 = bfs(A,V,p,q,direc2,parent2)
#for 4-neighbor
if len_cord1:
  print(f'Shortest Length of {len(direc1)} neighbor is {len_cord1[0]}')
  final_path = path(parent1,len_cord1[1:])
  print(f'Path of {len(direc1)} neighbor is {final_path}')

else:
  print(f"For {len(direc1)}-neighbor: No Path Found")
```

```
# for 8- neighbor
if len_cord2:
  print(f'Shortest Length of {len(direc2)} neighbor is {len_cord2[0]}')
  final_path = path(parent2,len_cord2[1:])
  print(f'Path of {len(direc2)} neighbor is {final_path}')

else:
  print(f"For {len(direc2)}- neighbor: No Path Found")

direc = [(0,1),(0,-1),(1,0),(-1,0)]
diag =[(1,1),(1,-1),(-1,1),(-1,-1)]
parent = np.empty((len(A),len(A[0])),dtype = object)

len_cord  = bfs_m(A,V,p,q,direc,parent,diag)
# for m-neighbor
if len_cord:
  print(f'Shortest Length of m-neighbor is {len_cord[0]}')
  final_path = path(parent,len_cord[1:])
  print(f'Path of m-neighbor is {final_path}')

else:
  print("For m-neighbor:No Path Found")


# In[27]:


# Example 3
imgrey = cv2.imread(r'D:\ECE558\ECE558-HW01\ECE558-HW01\wolves.png',0)
A = imgrey[300:305,400:405]
print(A)
p = (4,0)

q = (0,4)

V = [34, 43, 46,53,80,118]
print(f'V={V}')

direc1 = [(0,1),(0,-1),(1,0),(-1,0)]
direc2 = [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
#(1,1),(1,-1),(-1,1),(-1,-1)
parent1 = np.empty((len(A),len(A[0])),dtype = object)
parent2 = np.empty((len(A),len(A[0])),dtype = object)
len_cord1  = bfs(A,V,p,q,direc1,parent1)
len_cord2 = bfs(A,V,p,q,direc2,parent2)
#for 4-neighbor
if len_cord1:
  print(f'Shortest Length of {len(direc1)} neighbor is {len_cord1[0]}')
  final_path = path(parent1,len_cord1[1:])
  print(f'Path of {len(direc1)} neighbor is {final_path}')

else:
  print(f"For {len(direc1)}-neighbor: No Path Found")
# for 8- neighbor
if len_cord2:
  print(f'Shortest Length of {len(direc2)} neighbor is {len_cord2[0]}')
  final_path = path(parent2,len_cord2[1:])
  print(f'Path of {len(direc2)} neighbor is {final_path}')

else:
  print(f"For {len(direc2)}- neighbor No Path Found")
```

```
direc = [(0,1),(0,-1),(1,0),(-1,0)]
diag =[(1,1),(1,-1),(-1,1),(-1,-1)]
parent = np.empty((len(A),len(A[0])),dtype = object)

len_cord  = bfs_m(A,V,p,q,direc,parent,diag)
# for m-neighbor
if len_cord:
  print(f'Shortest Length of m-neighbor is {len_cord[0]}')
  final_path = path(parent,len_cord[1:])
  print(f'Path of m-neighbor is {final_path}')

else:
  print("For m-neighbor:No Path Found")


# In[28]:


#Example 4 taking different matrix 2
imgrey = cv2.imread(r'D:\ECE558\ECE558-HW01\ECE558-HW01\wolves.png',0)
A = imgrey[4:9,5:10]
print(A)
p = (4,0)

q = (0,4)

V = [0]
print(f'V={V}')

direc1 = [(0,1),(0,-1),(1,0),(-1,0)]
direc2 = [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
#(1,1),(1,-1),(-1,1),(-1,-1)
parent1 = np.empty((len(A),len(A[0])),dtype = object)
parent2 = np.empty((len(A),len(A[0])),dtype = object)

len_cord1 = bfs(A,V,p,q,direc1,parent1)
len_cord2 = bfs(A,V,p,q,direc2,parent2)
#for 4-neighbor
if len_cord1:
  print(f'Shortest Length of {len(direc1)} neighbor is {len_cord1[0]}')
  final_path = path(parent1,len_cord1[1:])
  print(f'Path of {len(direc1)} neighbor is {final_path}')

else:
  print(f"For {len(direc1)}-neighbor: No Path Found")
# for 8- neighbor
if len_cord2:
  print(f'Shortest Length of {len(direc2)} neighbor is {len_cord2[0]}')
  final_path = path(parent2,len_cord2[1:])
  print(f'Path of {len(direc2)} neighbor is {final_path}')

else:
  print(f"For {len(direc2)}- neighbor No Path Found")

direc = [(0,1),(0,-1),(1,0),(-1,0)]
diag =[(1,1),(1,-1),(-1,1),(-1,-1)]
parent = np.empty((len(A),len(A[0])),dtype = object)

len_cord  = bfs_m(A,V,p,q,direc,parent,diag)
# for m-neighbor
```

```
if len_cord:
  print(f'Shortest Length of m-neighbor is {len_cord[0]}')
  final_path = path(parent,len_cord[1:])
  print(f'Path of m-neighbor is {final_path}')

else:
  print("For m-neighbor:No Path Found")


# In[29]:


#Example 5 Invalid source V=[(3,4)]
A = np.array(([3,1,2,1],[2,2,0,2],[1,2,1,1],[1,0,1,2]))
print(A)
p = (3,0)

q = (0,3)

V = [3,4]
print(f'V={V}')

direc1 = [(0,1),(0,-1),(1,0),(-1,0)]
direc2 = [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
#(1,1),(1,-1),(-1,1),(-1,-1)
parent = np.empty((len(A),len(A[0])),dtype = object)

len_cord1  = bfs(A,V,p,q,direc1,parent)
len_cord2 = bfs(A,V,p,q,direc2,parent)
#for 4-neighbor
if len_cord1:
  print(f'Shortest Length of {len(direc1)} neighbor is {len_cord1[0]}')
  final_path = path(parent,len_cord1[1:])
  print(f'Path of {len(direc1)} neighbor is {final_path}')

else:
  print(f"For {len(direc1)}-neighbor: No Path Found")
# for 8- neighbor
if len_cord2:
  print(f'Shortest Length of {len(direc2)} neighbor is {len_cord2[0]}')
  final_path = path(parent,len_cord2[1:])
  print(f'Path of {len(direc2)} neighbor is {final_path}')

else:
  print(f"For {len(direc2)}- neighbor: No Path Found")

direc = [(0,1),(0,-1),(1,0),(-1,0)]
diag =[(1,1),(1,-1),(-1,1),(-1,-1)]
parent = np.empty((len(A),len(A[0])),dtype = object)

len_cord  = bfs_m(A,V,p,q,direc,parent,diag)
# for m-neighbor
if len_cord:
  print(f'Shortest Length of m-neighbor is {len_cord[0]}')
  final_path = path(parent,len_cord[1:])
  print(f'Path of m-neighbor is {final_path}')

else:
  print("For m-neighbor:No Path Found")
```

Results-

5 Different Examples considered

```
Runtime is 0.000997304916381836 for bfs().
Runtime is 0.0009975433349609375 for bfs_m()
```

Example 1-

```
[[3 1 2 1]
 [2 2 0 2]
 [1 2 1 1]
 [1 0 1 2]]
V=[0, 1]
For 4-neighbor: No Path Found
Shortest Length of 8 neighbor is 4
Path of 8 neighbor is [(3, 0), (3, 1), (2, 2), (1, 2), (0, 3)]
Shortest Length of m-neighbor is 5
Path of m-neighbor is [(3, 0), (3, 1), (3, 2), (2, 2), (1, 2), (0, 3)]
```

Example 2 –

```
[[3 1 2 1]
 [2 2 0 2]
 [1 2 1 1]
 [1 0 1 2]]
V=[1, 2]
Shortest Length of 4 neighbor is 6
Path of 4 neighbor is [(3, 0), (2, 0), (2, 1), (2, 2), (2, 3), (1, 3), (0, 3)]
Shortest Length of 8 neighbor is 4
Path of 8 neighbor is [(3, 0), (2, 0), (1, 1), (0, 2), (0, 3)]
Shortest Length of m-neighbor is 6
Path of m-neighbor is [(3, 0), (2, 0), (2, 1), (2, 2), (2, 3), (1, 3), (0, 3)]
```

Example 3 –

```
[[ 34  70  61 103 118]
 [ 68  46  66  80  90]
 [ 56  43  53  46  61]
 [ 87  53  43  59  28]
 [ 43  39  46  43  34]]
V=[34, 43, 46, 53, 80, 118]
For 4-neighbor: No Path Found
Shortest Length of 8 neighbor is 4
Path of 8 neighbor is [(4, 0), (3, 1), (2, 2), (1, 3), (0, 4)]
Shortest Length of m-neighbor is 6
Path of m-neighbor is [(4, 0), (3, 1), (3, 2), (2, 2), (2, 3), (1, 3), (0, 4)]
```

Example 4 –

```
[[ 0  9  0  0  0]
 [ 0  0  0  0  0]
 [ 0  0  0  0 21]
 [21  0  0  0 34]
 [ 0  0  0  0 21]]
V=[0]
Shortest Length of 4 neighbor is 8
Path of 4 neighbor is [(4, 0), (4, 1), (4, 2), (4, 3), (3, 3), (2, 3), (1, 3), (1, 4), (0, 4)]
Shortest Length of 8 neighbor is 4
Path of 8 neighbor is [(4, 0), (3, 1), (2, 2), (1, 3), (0, 4)]
Shortest Length of m-neighbor is 8
Path of m-neighbor is [(4, 0), (4, 1), (4, 2), (4, 3), (3, 3), (2, 3), (1, 3), (1, 4), (0, 4)]
```

Example 5 –

```
[[3 1 2 1]
 [2 2 0 2]
 [1 2 1 1]
 [1 0 1 2]]
V=[3, 4]
For 4-neighbor: No Path Found
For 8- neighbor: No Path Found
For m-neighbor:No Path Found
```