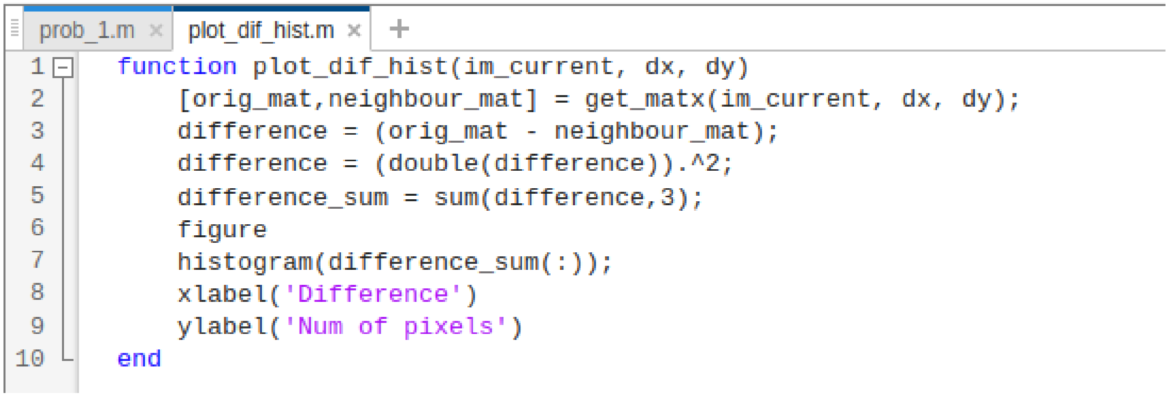
Option 0

Problem 1:

Function for handling neighbors, calculating difference and plotting histogram:

plot_dif_hist.m



```
function plot_dif_hist(im_current, dx, dy)
    [orig_mat,neighbour_mat] = get_matx(im_current, dx, dy);
    difference = (orig_mat - neighbour_mat);
    difference = (double(difference)).^2;
    difference_sum = sum(difference,3);
    figure
    histogram(difference_sum(:));
    xlabel('Difference')
    ylabel('Num of pixels')
end
```

Inner fucntion: ***get_matx.m*** outputs two arrays of same size.

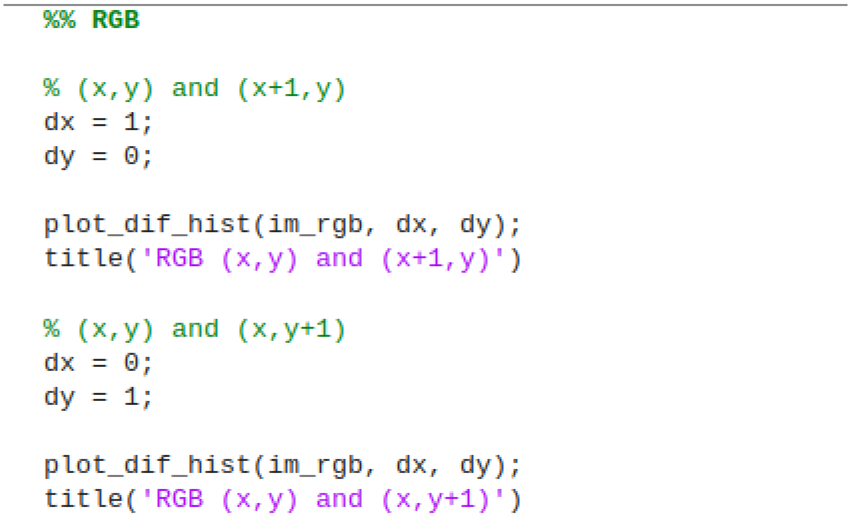
Usage:

Neighbors select

dx = 1, dy = 0 to select (x,y) and (x+1,y)

dx = 0, dy = 1 to select (x,y) and (x,y+1)

Example of use:



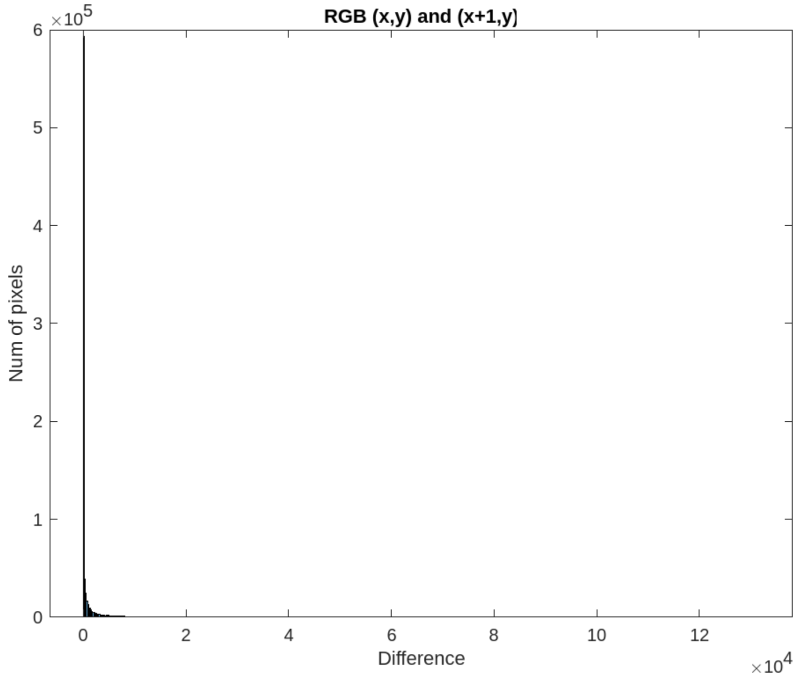
```
%% RGB

% (x,y) and (x+1,y)
dx = 1;
dy = 0;

plot_dif_hist(im_rgb, dx, dy);
title('RGB (x,y) and (x+1,y)')

% (x,y) and (x,y+1)
dx = 0;
dy = 1;

plot_dif_hist(im_rgb, dx, dy);
title('RGB (x,y) and (x,y+1)')
```

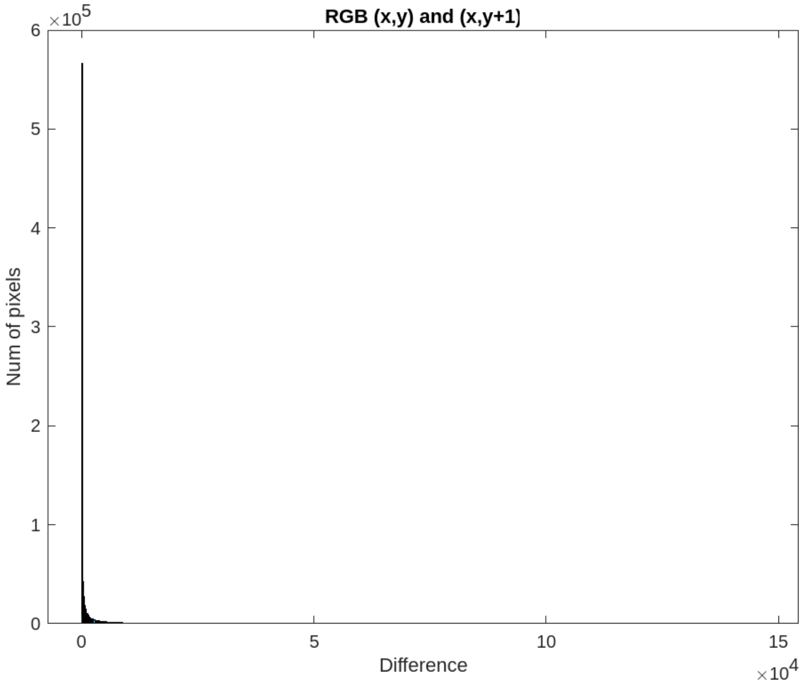
Testing all required cases with ***prob_1.m*** with run time measurement:

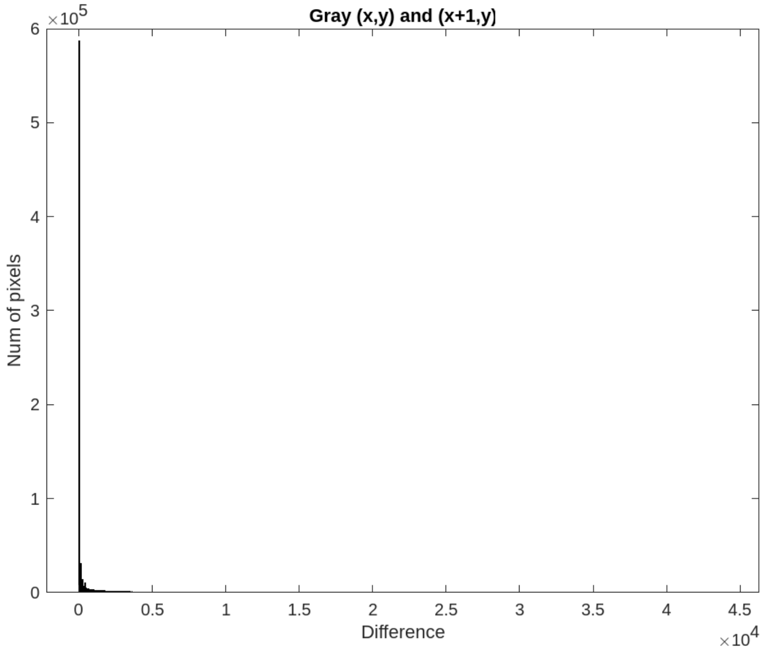
8 test cases:

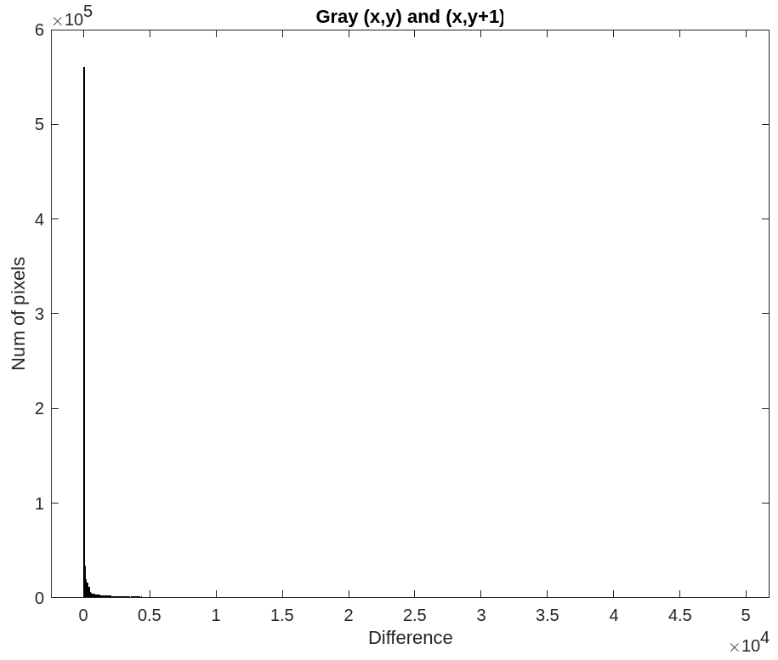
Case	Neighbors	Image type
1	(x,y) and (x+1,y)	RGB
2	(x,y) and (x,y+1)	RGB
3	(x,y) and (x+1,y)	Intensity (gray)
4	(x,y) and (x,y+1)	Intensity (gray)
5	(x,y) and (x+1,y)	HSV
6	(x,y) and (x,y+1)	HSV
7	(x,y) and (x+1,y)	LAB
8	(x,y) and (x,y+1)	LAB
Total runtime of the code : **1.27 seconds** (includes reading images, calculation and plotting)		

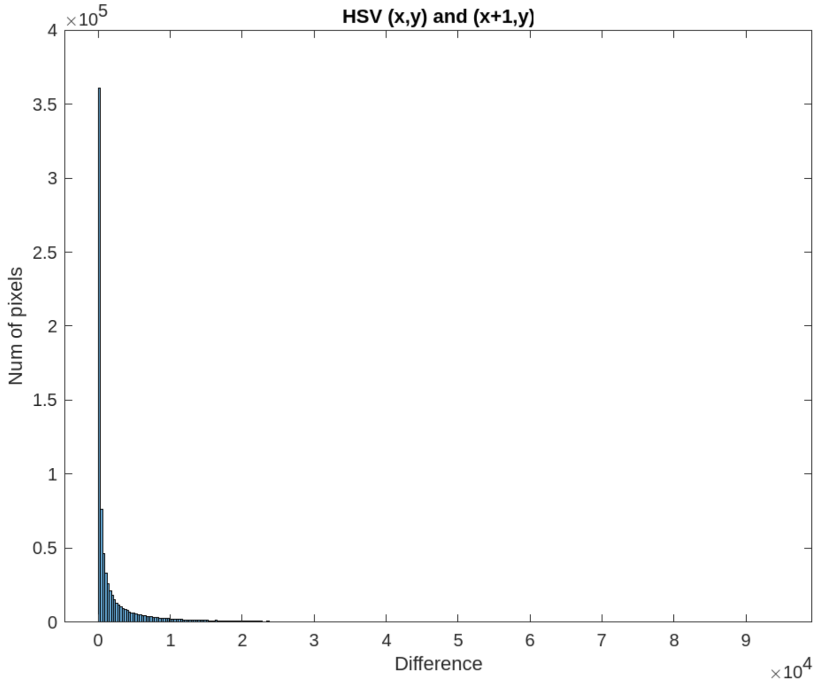
<u>Results:</u>

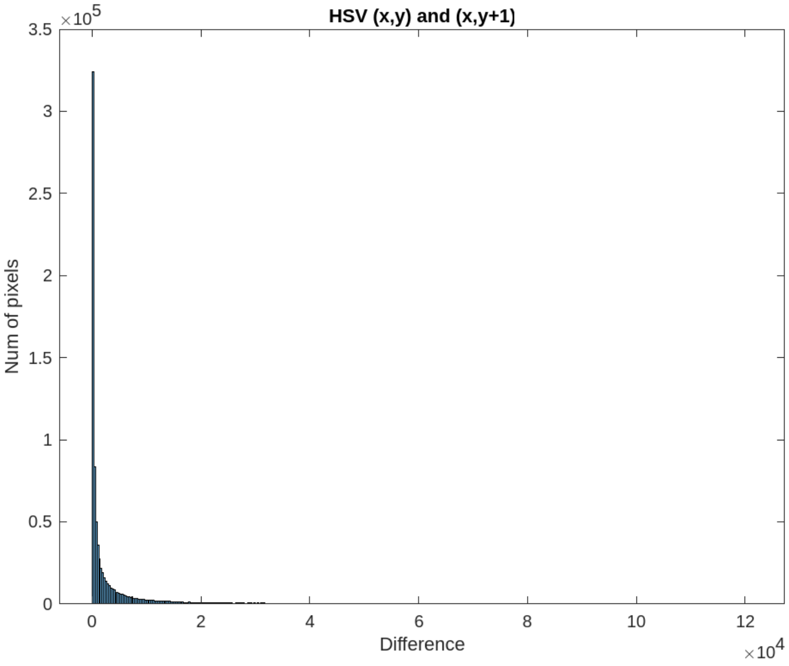


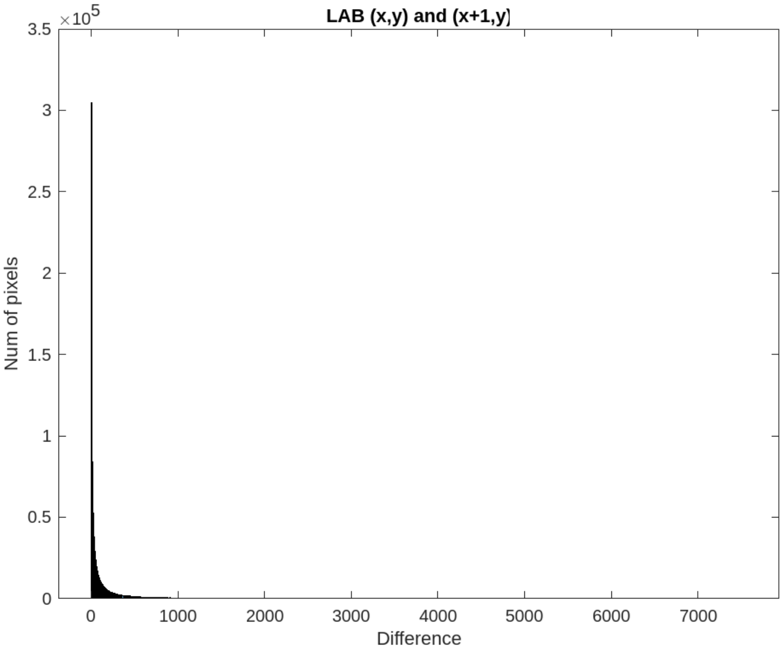

RGB (x,y) and (x,y+1)
×10^5
6
5
4
3
2
1
0
Num of pixels
0
5
10
15
Difference
×10^4


Gray (x,y) and (x+1,y)
×10^5
6
5
4
3
2
1
0
Num of pixels
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Difference
×10^4

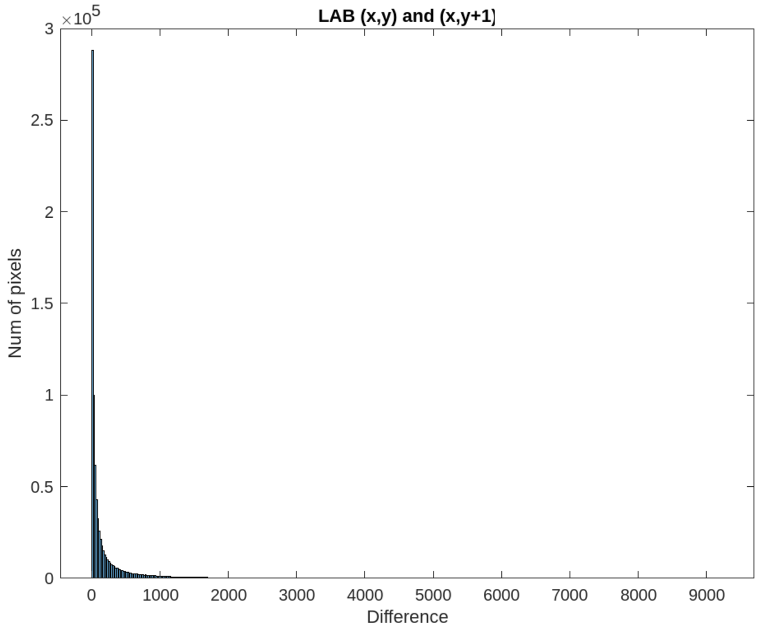

Gray (x,y) and (x,y+1)
×10^5
6
5
4
3
2
1
0
Num of pixels
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Difference
×10^4


HSV (x,y) and (x+1,y)
×10^5
4
3.5
3
2.5
2
1.5
1
0.5
0
Num of pixels
0
1
2
3
4
5
6
7
8
9
Difference
×10^4


HSV (x,y) and (x,y+1)
×10^5
3.5
3
2.5
2
1.5
1
0.5
0
Num of pixels
0
2
4
6
8
10
12
Difference
×10^4

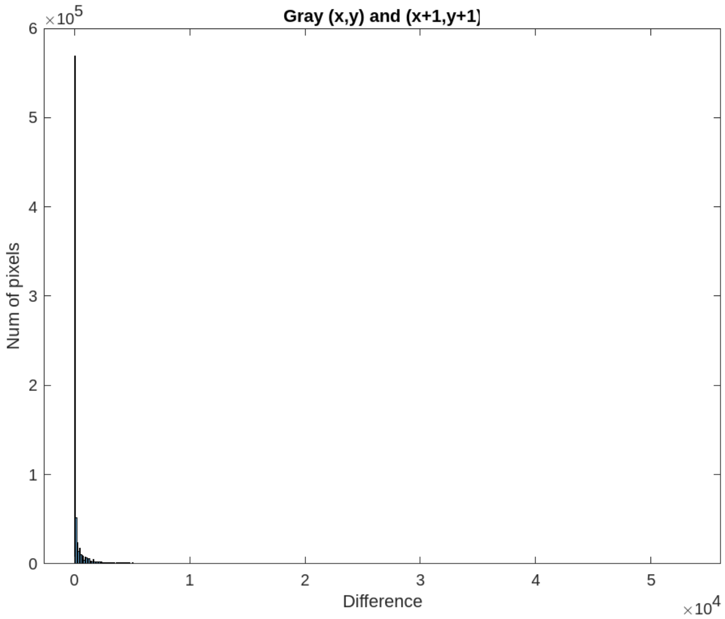

LAB (x,y) and (x+1,y]
×10^5
3.5
3
2.5
2
1.5
1
0.5
0
Num of pixels
0
1000
2000
3000
4000
5000
6000
7000
Difference

Results:

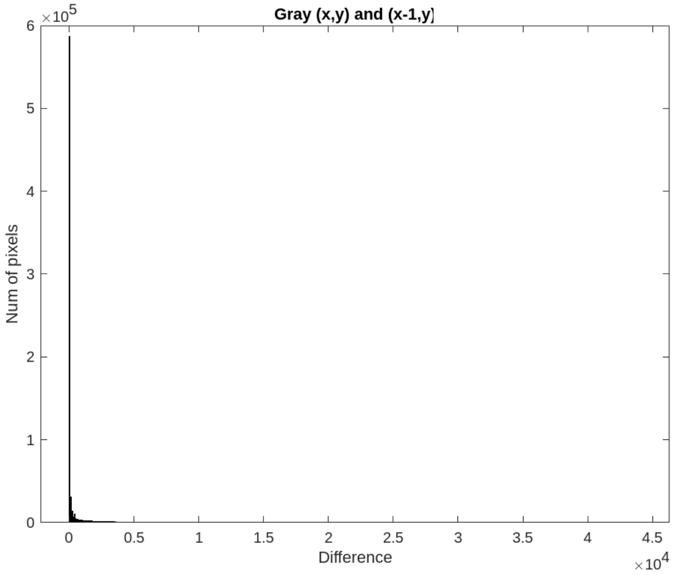


Extra cases:

Grayscale image with neighbors (x,y) and (x+1, y+1)



Grayscale image with neighbors (x,y) and (x-1, y)

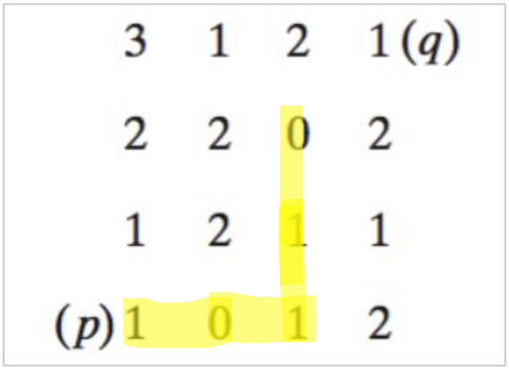


Problem 2:

(a)

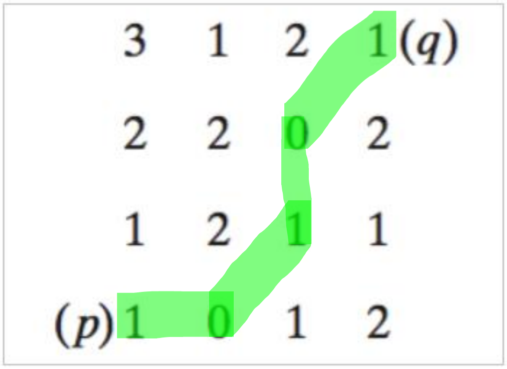
V = { 0, 1 }

4- path: Path doesn’t exist. The q point has no neighbor that satisfies the intensity requirement of V.



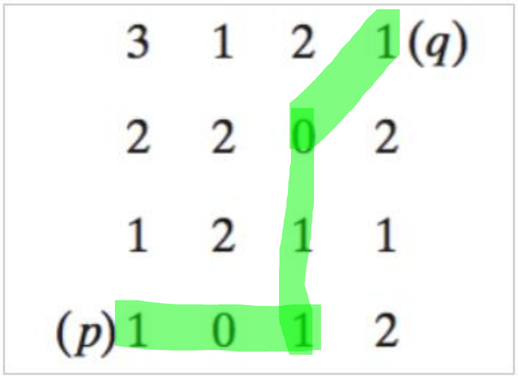
	3	1	2	1 (*q*)
	2	2	0	2
	1	2	1	1
(*p*)	1	0	1	2

8- path: The length of shortest path is 4



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

m- path: The length of shortest path is 5

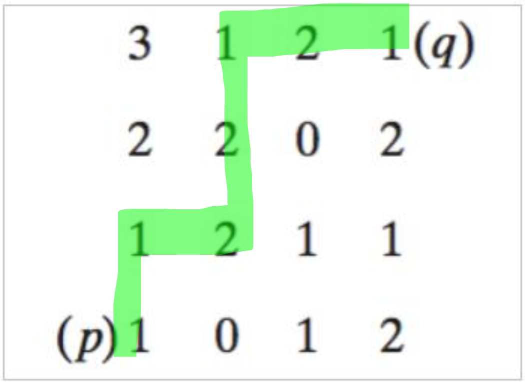


	3	1	2	1	(*q*)
	2	2	0	2	
	1	2	1	1	
(*p*)	1	0	1	2	

(b)

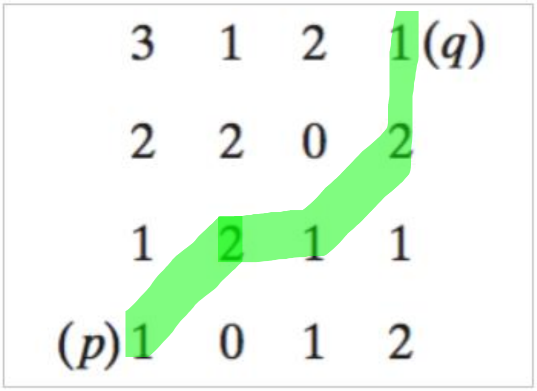
V = { 1, 2 }

4- path: The length of the shortest path is 6 (there are multiple paths of length 6, only one is shown)



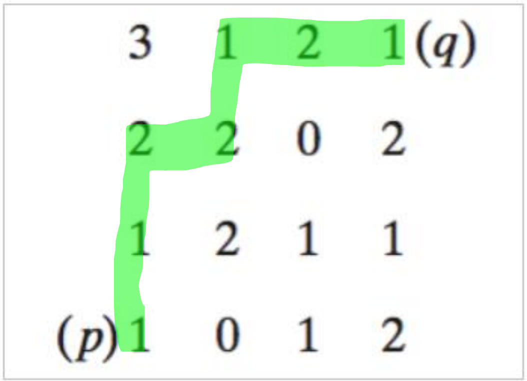
	3	1	2	1	(*q*)
	2	2	0	2	
	1	2	1	1	
(*p*)	1	0	1	2	

8- path: The length of the shortest path is 4 (there are multiple paths of length 4, only one is shown)



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

m- path: The length of the shortest path is 6 (there are multiple paths of length 6, only one is shown)



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

(C) Matlab code:

I have wriiten 1 function file and 1 custom matlab class to use in prob_2.m

my_path.m : It is a class which takes the image, intensity range (v), starting point, end point and path type (4/8/m) as input and creates an object. This initializes a cost map for all the points in the image grid of value positive infinity. Then we can use a function of this object ***change_value*** to set the cost value of point p (one of the two points) to zero. ***change_value*** function takes care of iteratively changing the cost value for all the points. It handles the path type, considers the intensity values and update cost maps automatically. Finally, we call ***get_path*** function to get the cost value (distance) of any point within the image grid.

If the cost is not infinite, then we can say that there is a path between the two points. Otherwise, there is no path. The class takes care of storing the connection data within the points and we can use those data to draw path between the two points if path exists.

Draw_lines.m: It is the function to draw path on an image given the image and the x and y coordinates of the path.

Implementation of the class:

```
classdef my_path

%MY_PATH Summary of this class goes here

% Detailed explanation goes here

properties

points

points_new

img

W

H

p
```

```
q
track_x
track_y
dist
v
map % m-path connection map (constant)
path % m or 4 or 8
end
methods
function obj = my_path(img,p,q,v,path)
obj.img = img;
obj.H = size(img,1);
obj.W = size(img,2);
obj.points = p;
obj.points_new = zeros(0,2);
obj.p = p;
obj.q = q;
%obj.queue = p;
obj.track_x = zeros(size(img));
obj.track_y = zeros(size(img));
obj.dist = Inf(size(img));
obj.v = v;
obj.path = path;
obj.map = [[3,4];[1,4];[2,3];[1,2]];
if ~(obj.is_valid(p(2),p(1)) && obj.is_valid(q(2),q(1)))
toc
error("invalid points (p/q) given")
end
end
function val = is_valid(obj,x,y)
val = x>0 && y>0 && x<=obj.W && y<=obj.H;
```

```matlab
end
function neighbors = get_neighbors(obj,x,y)
if obj.path == "4"
neighbors = [x+1,y; x,y+1; x-1,y; x,y-1];
elseif obj.path == "8" || obj.path == "m"
neighbors = [x+1,y; x,y+1; x-1,y; x,y-1;
x-1,y-1; x+1, y-1; x-1, y+1; x+1, y+1];
else
toc
error("only 4/8/m paths are supported")
end
end
function obj = make_connection(obj,y,x,y1,x1)
if obj.dist(y,x)+1 < obj.dist(y1,x1)
obj.track_x(y1,x1) = x;
obj.track_y(y1,x1) = y;
obj = obj.change_value(x1,y1,obj.dist(y,x)+1);
end
end
function obj = process_adjacents(obj, x,y)
ngbrs = obj.get_neighbors(x,y);
%draw_line(1, ngbrs);
if obj.path ~= "m"
for i=1:length(ngbrs)
y1 = ngbrs(i,2);
x1 = ngbrs(i,1);
if obj.is_valid(x1, y1) && any(obj.img(y1,x1)==obj.v)
obj = obj.make_connection(y,x,y1,x1); % dfs
end
end
else
```

```
flag = false(1,4);
for i=1:4 % right bottom left top
y1 = ngbrs(i,2);
x1 = ngbrs(i,1);
if obj.is_valid(x1, y1) && any(obj.img(y1,x1)==obj.v)
obj = obj.make_connection(y,x,y1,x1);
flag(i) = true;
end
end
% 5 6 7 8
% (left-top, right-top, left-bottom, right-bottom)
%map = [[3,4];[1,4];[2,3];[1,2]];
for i=1:4
y1 = ngbrs(i+4,2);
x1 = ngbrs(i+4,1);
if ~any(flag(obj.map(i,:))) && obj.is_valid(x1, y1) &&...
any(obj.img(y1,x1)==obj.v)
obj = obj.make_connection(y,x,y1,x1);
end
end
end
end
function obj = change_value(obj,x,y,val)
obj.dist(y,x) = val;
%disp(obj.dist)
obj = obj.process_adjacents(x,y);
end
function path_xy = get_path(obj, q)
path_xy = zeros(0,2);
if obj.dist(q(1),q(2)) < inf
path_xy = [path_xy; q(2), q(1)];
```

```
current = q;

while current(1) > 0 && current(2) > 0

path_xy = [path_xy; obj.track_x(current(1),current(2)),...

obj.track_y(current(1),current(2))];

current = flip(path_xy(end,:));

end

path_xy = path_xy(1:end-1,:);

end

end

end

end
```

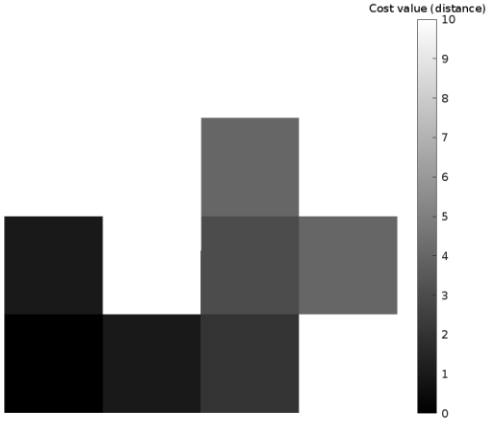
Result:

Case	v	path	p,q	Shortest path	Runtime
1	[0,1]	4	(4,1), (1,4)	No path exists	0.38 seconds
2		8		4	
3		m		5	
4	[1,2]	4		6	0.41 seconds
5		8		4	
6		m		6	

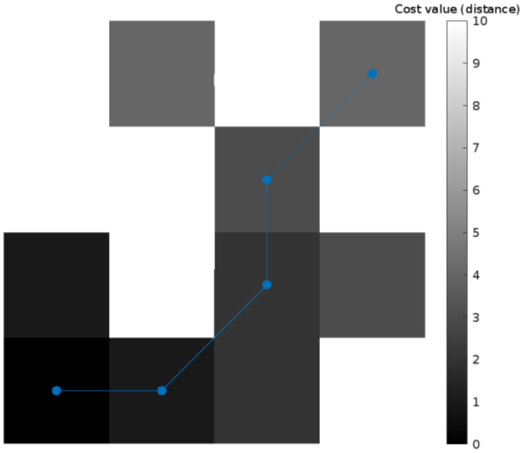
P and Q coordinates here follows matlab style indexing: (row, colum)

Cost map (distance):

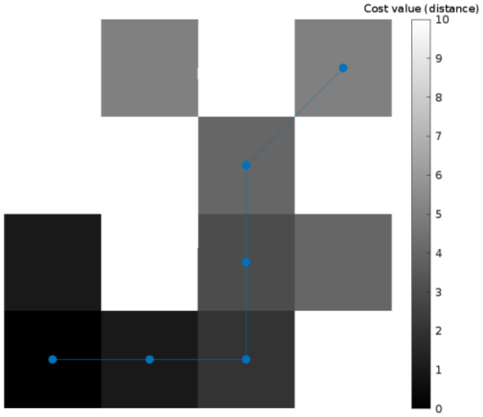
Case 1: V = [0,1], 4-path



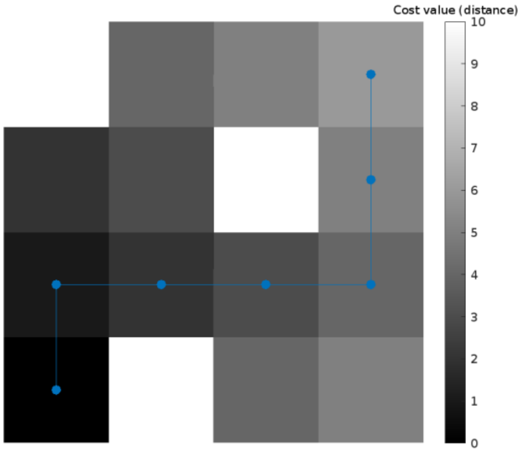
Case 2: V = [0,1], 8-path



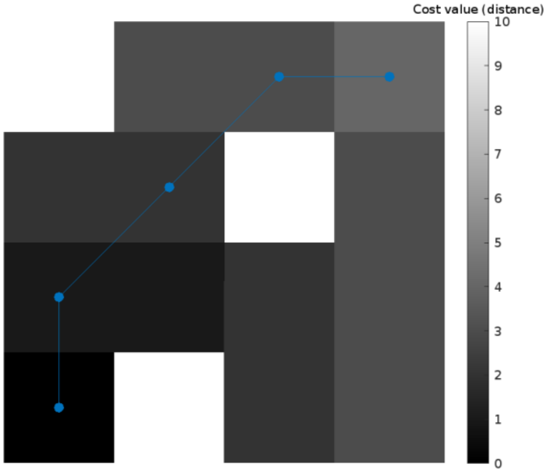
Case 3: V = [0,1], m-path



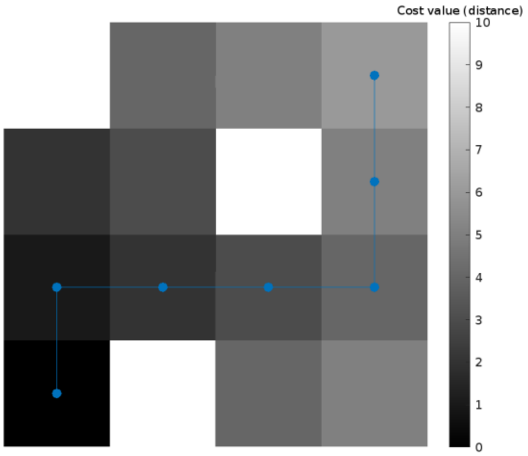
Case 4: V = [1,2], 4-path



Case 5: V = [1,2], 8-path



Case 6: V = [1,2], m-path



Extra cases:

Case	v	path	p,q	Shortest path	Run time
7	[0,1]	4	(4,1), (3,2)	No path exists	0.45 seconds
8		8		No path exists	
9		m		No path exists	
10	[0,1]	4	(4,1), (2,3)	4	0.64 seconds
11		8		3	
12		m		4	
13	[0,1]	4	(8,1), (2,3)	Invalid p/q	0.00 seconds
14		8		Invalid p/q	
15		m		Invalid p/q	
16	[1,2]	4	(4,1), (3,2)	2	0.62 seconds
17		8		1	
18		m		2	
19	[1,2]	4	(4,1), (2,3)	No path exists	0.44 seconds
20		8		No path exists	
21		m		No path exists	
22	[1,2]	4	(8,1), (2,3)	Invalid p/q	0.00 seconds
23		8		Invalid p/q	
24		m		Invalid p/q	

Comment on 7,8 and 9: Intensity of (3,2) is 2 which is not within v. And so p and q can't have a path between them irrespective of path type.

Matlab output:

```
4- path
 Inf Inf Inf Inf
 Inf Inf 4 Inf
 1 Inf 3 4
 0 1 2 Inf

no path exists

8- path
 Inf 4 Inf 4
 Inf Inf 3 Inf
 1 Inf 2 3
 0 1 2 Inf



no path exists

m- path
 Inf 5 Inf 5
 Inf Inf 4 Inf
 1 Inf 3 4
 0 1 2 Inf



no path exists

Elapsed time is 0.449454 seconds.
```

<u>Comment on 10,11 and 12:</u> 4-path and m-path has equal distance

```
4- path
 Inf Inf Inf Inf
 Inf Inf 4 Inf
 1 Inf 3 4
 0 1 2 Inf



minimum length = 4

8- path
 Inf 4 Inf 4
 Inf Inf 3 Inf
 1 Inf 2 3
 0 1 2 Inf



minimum length = 3

m- path
 Inf 5 Inf 5
 Inf Inf 4 Inf
 1 Inf 3 4
 0 1 2 Inf


minimum length = 4

Elapsed time is 0.638872 seconds.
```

Comment on 13,14 and 15: (8,1) doesn’t exist on image grid

```
Error using my_path

invalid points (p/q) given
```

Comment on 16,17 and 18: 4-path and m-path has equal distance

```
4- path
 Inf 4 5 6
 2 3 Inf 5
 1 2 3 4
 0 Inf 4 5


minimum length = 2

8- path
 Inf 3 3 4
 2 2 Inf 3
 1 1 2 3
 0 Inf 2 3


minimum length = 1

m- path
 Inf 4 5 6
 2 3 Inf 5
 1 2 3 4
 0 Inf 4 5


minimum length = 2

Elapsed time is 0.619639 seconds.
```

Comment on 19,20 and 21: Intensity of (2,3) is 0 which is not within v. And so p and q can’t have a path between them irrespective of path type.

```
4- path
 Inf 4 5 6
 2 3 Inf 5
 1 2 3 4
 0 Inf 4 5



no path exists

8- path
 Inf 3 3 4
 2 2 Inf 3
 1 1 2 3
 0 Inf 2 3

no path exists

m- path


Inf 4 5 6
 2 3 Inf 5
 1 2 3 4
 0 Inf 4 5

no path exists

Elapsed time is 0.435609 seconds.
```

Comment on 22,23 and 24: (8,1) doesn’t exist on image grid

```
Error using my_path

invalid points (p/q) given
```

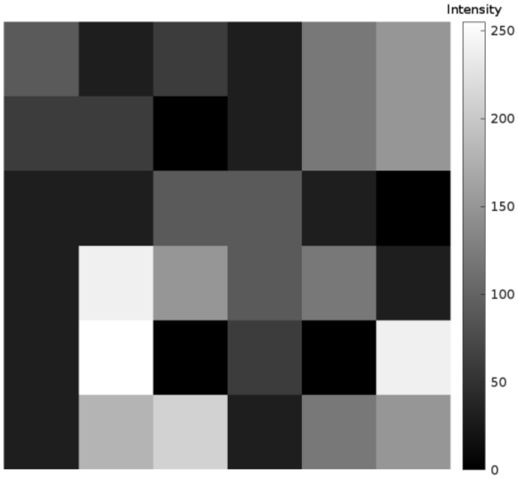
One final special case:

Changing the image:

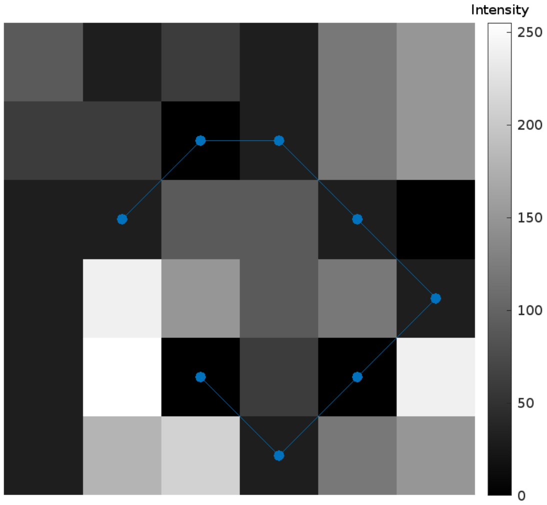
```
mat = uint8([ ...
3 1 2 1 4 5;
2 2 0 1 4 5;
1 1 3 3 1 0;
1 8 5 3 4 1;
1 9 0 2 0 8;
1 6 7 1 4 5]);
V = [0,1]
p = [3,2];
q = [5,3];
```

Image intensity image:

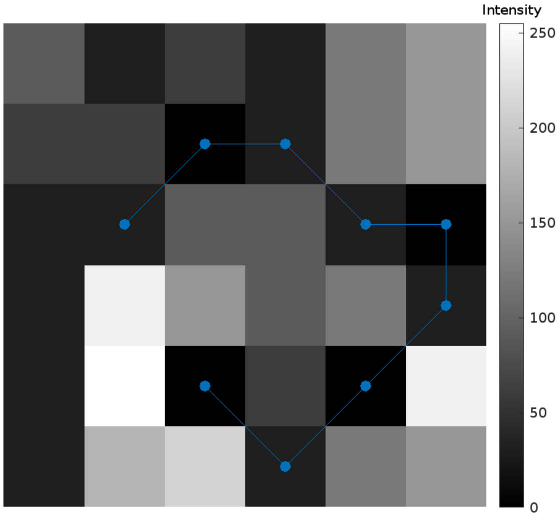
4-path result: no path



8-path result: Minimum path distance is 7



m-path result: Minimum path distance is 8


Intensity