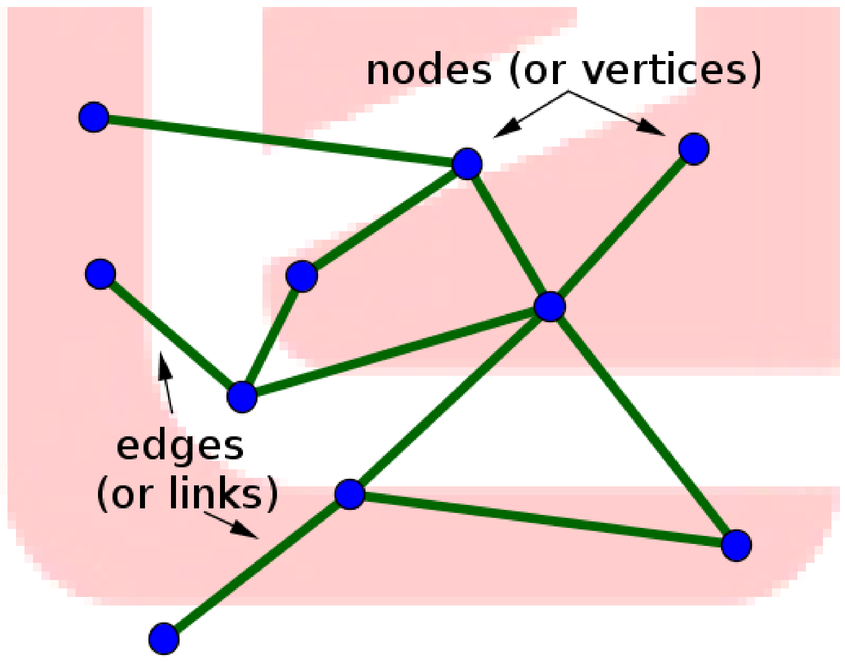
Task-1 – Practice

Section – 2

Mazes in general are represented as undirected graphs so as to depict all traversable neighbors. In this section, you are supposed to find out how to represent a full maze in form of an **undirected graph** in python.

An undirected graph is a set of objects (called vertices or nodes) that are connected together, where all the edges are bidirectional. An undirected graph is sometimes called an undirected network. In contrast, a graph where the edges point towards specified directions is called a directed graph.

When drawing an undirected graph, the edges are typically drawn as lines between pairs of nodes, as illustrated in the following figure:



You can read about graphs in python from **www.python.org**

1. Open the **Task1_Practice folder**.
2. In the **Section-2 folder**, open the ***section2.py*** file.
3. The ***section2.py*** script contains the **readImage(), findNeighbours()** and **colourCell()** functions. You can copy the function definitions for these functions from the ***section1.py*** script you modified in the last section.
4. Follow the instructions to modify functions in the ***section2.py*** file as given below:

MAIN

```
if __name__ == '__main__':
    filePath = 'maze00.jpg'## Specify the filepath of image here
    img = main(filePath)
    cv2.imshow('canvas', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

This section of code calls the main() function with a filepath which you can specify. You can specify filepath as “maze00.jpg” to “maze09.jpg”. Apart from the filepath please do not change any other code in this section.

MAIN FUNCTION

```
def main(filePath, flag = 0):
    img = readImage(filePath)
    breadth = len(img)/20          ## Breadthwise number of cells
    length = len(img[0])/20        ## Lengthwise number of cells
    if length == 10:
       initial_point = (0,0)
       final_point = (9,9)
    else:
       initial_point = (0,0)
       final_point = (19,19)
    graph = buildGraph(    )       ## Build graph from maze image.
    shortestPath = findPath(    )  ## Find shortest path.
    print shortestPath             ## Print shortest path to verify
    string = str(shortestPath) + "\n"
    for i in shortestPath:        ## Loop to paint the solution path.
        img = colourCell(img, i[0], i[1], 200)
    if __name__ == '__main__':  ## Return value for main() function.
       return img
    else:
        if flag == 0:
            return string
        else:
            return graph
```

This is the main() function which is being called in the previous section. The functions called in this main function() will be explained in detail. You are expected to understand what this snippet of code does, on your own. You can refer online resources to learn the python syntax such as given here: https://www.tutorialspoint.com/python/

You need to provide arguments in the function calls for buildGraph() and findPath() functions. Other than that you are not allowed to change this section of code.

The following section will explain the functions used in the ***section2.py*** script in detail:

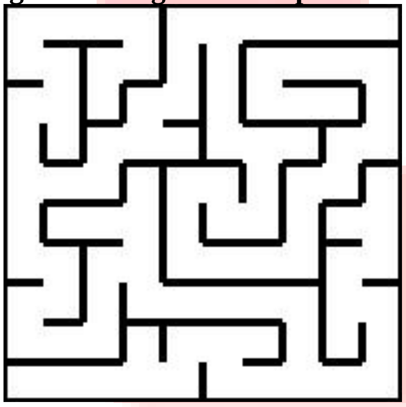
FUNCTIONS

```
## Function that accepts some arguments from user and returns
## the graph of the maze image.
def buildGraph():## You can pass your own arguments in this space.
    graph = {}
    ##############      Add your Code here   #################


    #########################################################
    return graph
```

The buildGraph( ) function accepts some input arguments and returns the maze in the form of an undirected graph. Teams are encouraged to read the theory behind undirected graphs from books and online sources.

Suppose we are using the following maze image as test input :



The output graph for this function will be similar to this :-
{ (0,0) : [(1,0) , (0,1)] , (0,1) : [(0,0) , (0,2)], (0,2) : [(0,1) , (0,3)], (0,3) : [(0,2) , (1,3)], (0,4) : [(1,4) , (0,5)] ………….. }

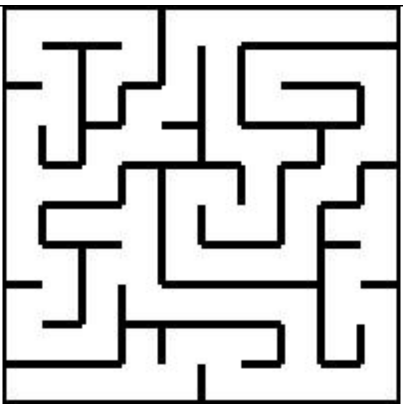
FUNCTIONS

```
## Finds shortest path between two coordinates in the maze.
## Returns a set of coordinates from initial point to final
## point.
def findPath(): ## You can pass your own arguments in this space.
    #############  Add your Code here   #################


    #####################################################
    return shortest
```

The findPath() accepts some input parameters such as graph of a maze and initial and final points and returns the shortest path between those two points.

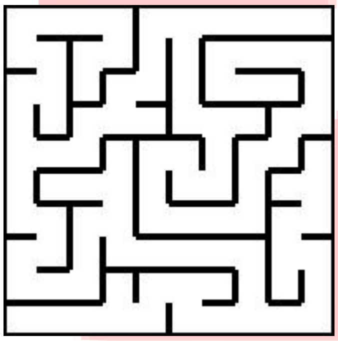
Suppose we are using the following maze image as test input :



Suppose the initial coordinates and final coordinates have been specified as (0,0) and (4,3) for this test image.
This function will yield the output as a set of coordinates that need to be traversed from initial to final points which is:
[(0, 0), (1, 0), (1, 1), (2, 1), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (6, 1), (7, 1), (7, 0), (8, 0), (8, 1), (8, 2), (7, 2), (6, 2), (6, 3), (5, 3), (4, 3)]

EXPECTED OUTPUT OF PROGRAM

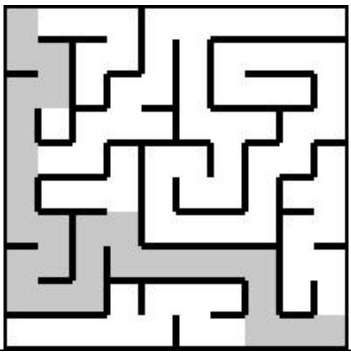
For the *section2.py* script, if we provide the filepath of the following image in the space indicated:



The output for this program should be:

```
>>>
[(0, 0), (1, 0), (1, 1), (2, 1),
(2, 0), (3, 0), (4, 0), (5, 0),
(6, 0), (6, 1), (7, 1), (7, 0),
(8, 0), (8, 1), (8, 2), (7, 2),
(6, 2), (6, 3), (7, 3), (7, 4),
(7, 5), (7, 6), (7, 7), (8, 7),
(9, 7), (9, 8), (9, 9)]
```

The output image should be:

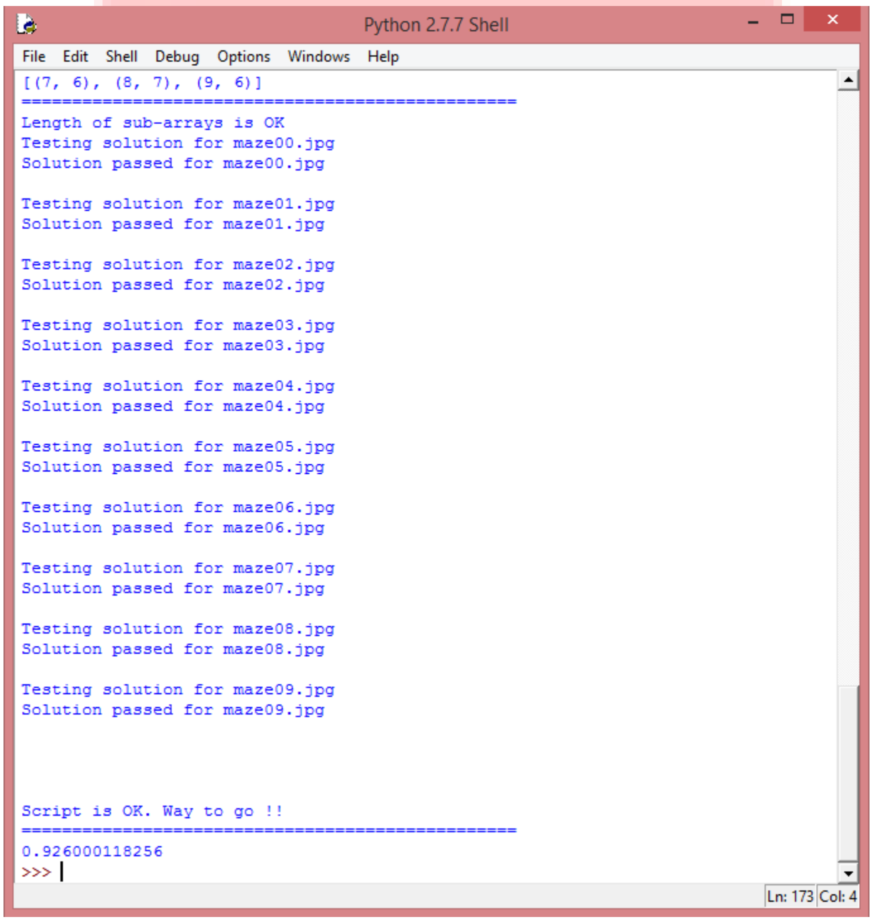


Testing the Solution

In the **Section-2 folder,** in addition to maze test images and ***section2.py*** there are three more files, namely the ***section2.txt*** file, ***hash.txt file*** and the ***TestSuite_2.py.***

You are **not allowed** to make any changes to these three files. Anybody found to have tampered with these files will be disqualified.

After you are done modifying the code in the ***section2.py*** file, open the ***TestSuite_2.py*** file and run it in the python shell. The output of **TestSuite_2** script should resemble the following screenshot:



```
[(7, 6), (8, 7), (9, 6)]
==================================================
Length of sub-arrays is OK
Testing solution for maze00.jpg
Solution passed for maze00.jpg

Testing solution for maze01.jpg
Solution passed for maze01.jpg

Testing solution for maze02.jpg
Solution passed for maze02.jpg

Testing solution for maze03.jpg
Solution passed for maze03.jpg

Testing solution for maze04.jpg
Solution passed for maze04.jpg

Testing solution for maze05.jpg
Solution passed for maze05.jpg

Testing solution for maze06.jpg
Solution passed for maze06.jpg

Testing solution for maze07.jpg
Solution passed for maze07.jpg

Testing solution for maze08.jpg
Solution passed for maze08.jpg

Testing solution for maze09.jpg
Solution passed for maze09.jpg



Script is OK. Way to go !!
==================================================
0.926000118256
>>> 
```

If it runs successfully without any errors then your solution is correct and it passed all 10 test cases provided to you.

Please note that files submitted by you will be run through similar test cases.

Now you can proceed to Section-3 of Task 1.

Way to go!!