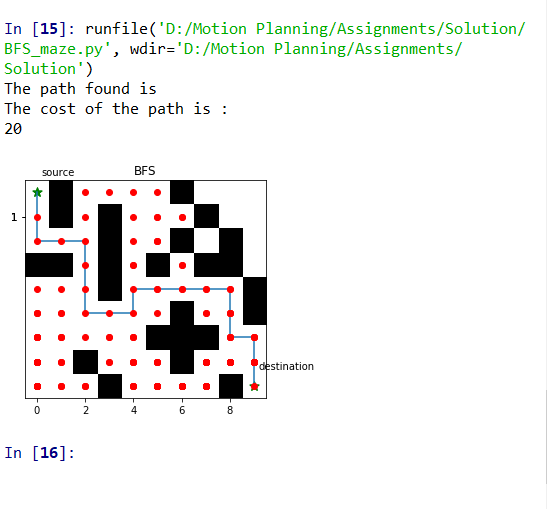
**Report**

1. BFS

* This algorithm is implemented in the form of queue. That is FIFO
* A binary maze is taken wherein 1’s depict free space and 0’s depict the obstacle
* Next source and destination are defined
* Neighbors are found using the 4-connected way.
* First we have taken the source and added it to the open list. Neighbors of the source are found.
* Next we iterate through the neighbors of the source and check whether the each of the neighbor is valid or not
* Valid neighbors are added to the open list
* Validity is checked on the basis of if the neighbor’s x,y co-ordinates are greater than zero, are within the limits of the maze and whether the node is not in the visited list. It is also checked that the node is not in the obstacle region that is it is not zero.
* Then the current node is added to the closed list. In this way the closed list is updated as and when nodes are generated
* Initially we start from source , and start expanding the frontier which is our 4-connected neighbor
* We start exploring the maze, nodes on the same level are all explored first. Then we move on to the next level that is neighbors of the nodes of the previous level. The process is stopped when we reach the destination. Hence we explore all the nodes of a particular level until the destination is reached.
* BFS method consumes lot of computational space, takes more time
* The path found by BFS is not optimal as well.

Reference : <https://www.geeksforgeeks.org/shortest-path-in-a-binary-maze/>

Output:

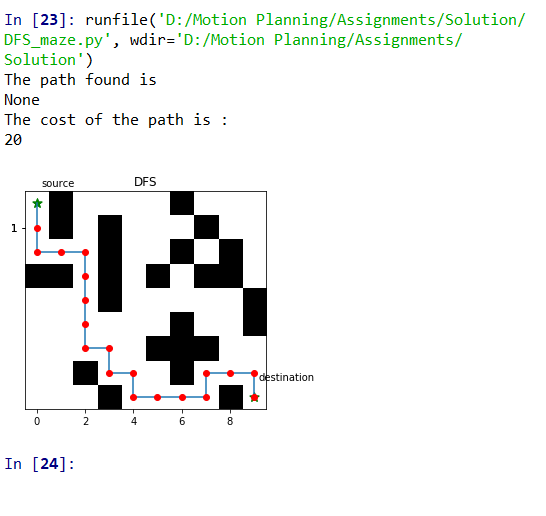


1. DFS

* This algorithm is implemented in the form of stack that is LIFO
* This algorithm starts from source, finds neighbors of the source
* From the list of neighbors, it picks the last stored neighbor and explores its neighbors. This process is continued until we find the goal.
* A binary maze is taken wherein 1’s depict free space and 0’s depict the obstacle
* Source is taken and neighbors of the source are found and stored in the open list.
* Last element of the open list is taken out, the neighbors of this element are found.
* Neighbors are added to the open list only if they are valid.
* Validity is checked on the basis of if the neighbor’s x,y co-ordinates are greater than zero, are within the limits of the maze and whether the node is not in the visited list. It is also checked that the node is not in the obstacle region that is it is not zero.
* If this node is not the destination, then further the neighbors of this node are found. Valid neighbors are added to open list and the node under examination is added to closed list.
* This process is continued until the destination is found.
* The time taken by DFS is less than that of BFS in our case whereas it depends on problems in general

Reference: #https://www.geeksforgeeks.org/shortest-path-in-a-binary-maze/

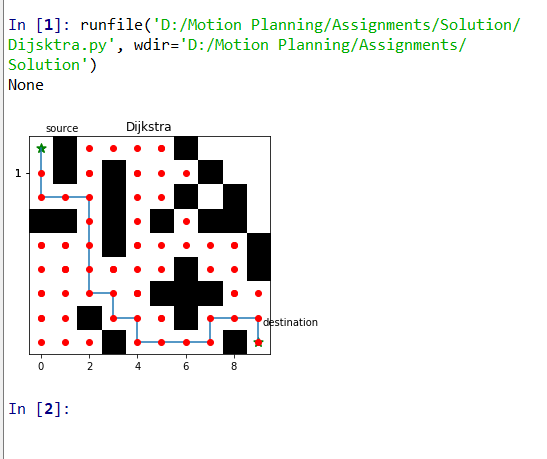
OUTPUT:



1. Dijkstra

* This algorithm finds the path between source and destination based on cost of the path which is path calculated with respect to source.
* SO it basically checks from which intermediate node we get less path cost from the source
* We have taken the Manhattan distance as the path cost
* The distance between two points measured along axes at right angles. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is
* Distance=|x1 - x2| + |y1 - y2|.
* A binary maze has been taken. Source and Destination are defined
* Source is taken. Its neighboring nodes are found based on 4-connected neighbors.
* Validity is checked on the basis of if the neighbor’s x,y co-ordinates are greater than zero, are within the limits of the maze and whether the node is not in the visited list. It is also checked that the node is not in the obstacle region that is it is not zero.
* The path cost of valid neighbors that is Manhattan distance from the source is calculated and stored in the dictionary with maintains the node and its cost. This dictionary acts as open list.
* The source is added to the visited list.
* A dictionary with parent, child and cost is also maintained.
* We sort the dictionary which maintains the open nodes in descending order and take out the node with minimum cost.
* This node is now our current node. Its neighbors are calculated and valid neighbors with cost are added to open list.
* But here if the a neighbor already in open has the less cost from origin, we update the cost of that neighbor and also the parent of that neighbor as the node through which less cost is found.
* Then the current node is added to the explored list.
* This process is continued until all the elements of open list are explored
* Here we explore those nodes as well which are not required.
* Once open list is empty, we backtrace from destination to node to find the path. Like parent of destination and so on.
* After finding the path, we plot the path.
* Red dots are the nodes which were explored as the algorithm is progressed.
* We see that the explored region is more than A\*
* Reference: <https://www.youtube.com/watch?v=pVfj6mxhdMw&t=1s>
* This algorithm consumes less space than BFS and DFS and finds the optimal path.

Output:



1. A\*

* This algorithm is used to find the optimal path from source to destination
* So the node is selected based on the minimum total cost. Total cost is calculated as

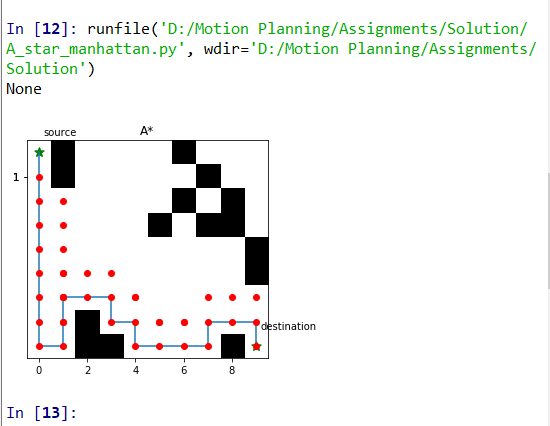
Total cost=g(s)+h(s)

* G(s) in our case is the sum of the cost of that node from its parent plus cost of parent from the source
* H(s) which is the heuristic, in our case it is calculated as the distance of the node from the destination.
* Distance is calculated as Manhattan distance in our case.
* Initially, we take the source node and find its 4-connected neighbors.
* Valid neighbors are added to the open list along with their total cost.
* Validity is checked on the basis of if the neighbor’s x,y co-ordinates are greater than zero, are within the limits of the maze and whether the node is not in the visited list. It is also checked that the node is not in the obstacle region that is it is not zero.
* We have maintained the visited list as dictionary with the node and its cost.
* When the valid neighbors are added, we sort the dictionary in descending order and the node with minimum cost is selected.
* The source is added to the visited list as well to the list which will be used to store the optimal path
* Then the neighbors of current node are found. Heuristic and path cost of the valid neighbors are found and these neighbors are added to the open list.
* Note that if the path cost of a node which is already present in the open list has more path cost than the one which is calculated with respect to the current node, we update the cost in the list otherwise it is kept as it is.
* So one by one, the nodes are explored based on the minimum total cost.
* As soon as the destination is reached, we terminate the process unlikely as in Dijkstra and return the path found.
* The path found is printed along with the explored nodes plotted by red-dots.

Reference :

# <https://www.youtube.com/watch?v=eSOJ3ARN5FM&t=476s>

Output:



1. Weighted A\*

* Heuristic is an important factor in A\* which basically pulls the path finding process towards the goal.
* Here we calculate the total cost as below

Total cost = g(s)+epsilon\*h(s) epsilon > 1

* Note that we have multiplied with epsilon here. Epsilon weights the heuristic in such a way that it guides the algorithm more towards the goal.
* For our case, A\* and Weighted A\* give the same path as they give the optimal path.
* A\* explores more than the Weighted A\*
* G(s) in our case is the sum of the cost of that node from its parent plus cost of parent from the source
* H(s) which is the heuristic, in our case it is calculated as the distance of the node from the destination.
* Distance is calculated as Manhattan distance in our case.
* Initially, we take the source node and find its 4-connected neighbors.
* Valid neighbors are added to the open list along with their total cost.
* Validity is checked on the basis of if the neighbor’s x,y co-ordinates are greater than zero, are within the limits of the maze and whether the node is not in the visited list. It is also checked that the node is not in the obstacle region that is it is not zero.
* We have maintained the visited list as dictionary with the node and its cost.
* When the valid neighbors are added, we sort the dictionary in descending order and the node with minimum cost is selected.
* The source is added to the visited list as well to the list which will be used to store the optimal path
* Then the neighbors of current node are found. Heuristic and path cost of the valid neighbors are found and these neighbors are added to the open list.
* Note that if the path cost of a node which is already present in the open list has more path cost than the one which is calculated with respect to the current node, we update the cost in the list otherwise it is kept as it is.
* So one by one, the nodes are explored based on the minimum total cost.
* As soon as the destination is reached, we terminate the process unlikely as in Dijkstra and return the path found.
* The path found is printed along with the explored nodes plotted by red-dots.

Reference:

<https://www.youtube.com/watch?v=eSOJ3ARN5FM&t=476s>

