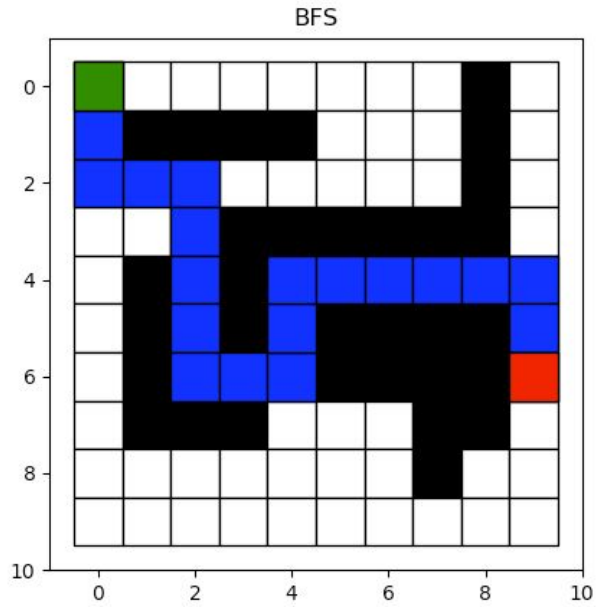
Basic Search Algorithms Report: Jash Mehta

Breadth first search:

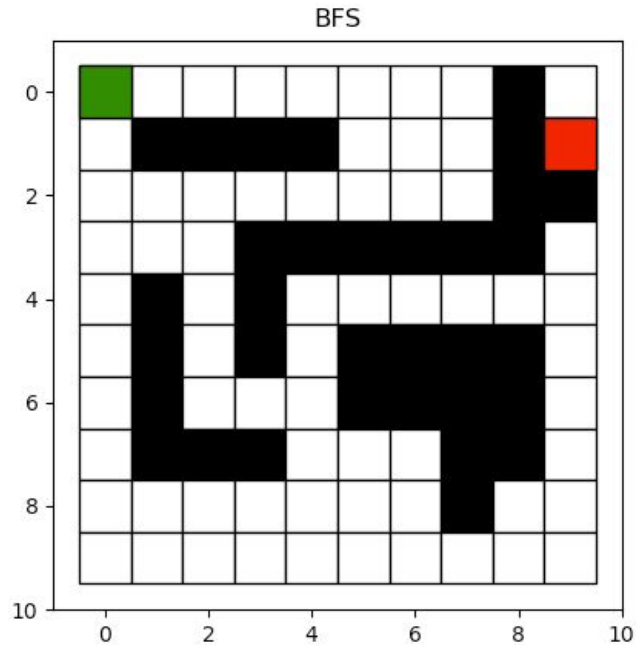
Pseudo code

```
#Initialise start parameters
Row, Col, visited, prev=...
#initialise queue to store values of nodes
rq,cq=....
Assign value “B” to goal point =...
#If condition to check if start is already at goal
If start==goal:
        Stop the search

While elements still left in queue
        Loop over the nodes in the queue to find new neighbors (avoiding obstacles, out of bound conditions and already visited nodes)
                Add new neighbors to queue
                Mark current node as visited
                Save value of parent node in prev
                Check if goal “B” is reached
                Exit loop
        Exit loop

#backtrack the path to get to start
at=coordinates of goal
While at not equal to zero:
        Add at value to path
        Update at with parent node to till it becomes zero
#This path is the path from goal to start so we reverse the order
Reverse path
```

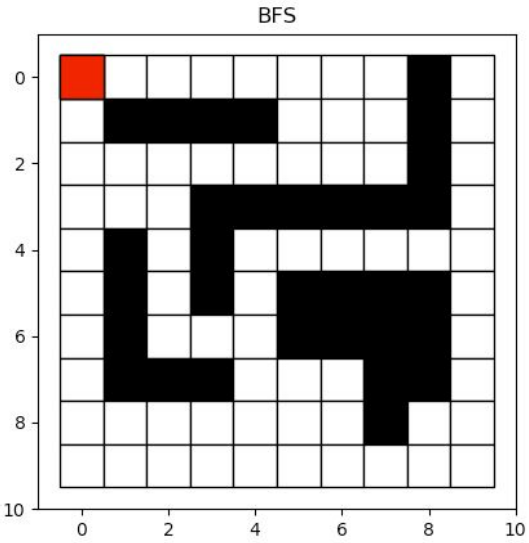
Test Cases for BFS




```
It takes 64 steps to find a path using BFS
```




```
No path found
No path found
No path found
No path found
```



When start = goal

```
It takes 0 steps to find a path using BFS
It takes 0 steps to find a path using DFS
It takes 0 steps to find a path using Dijkstra
It takes 0 steps to find a path using A*
```

Working of code:

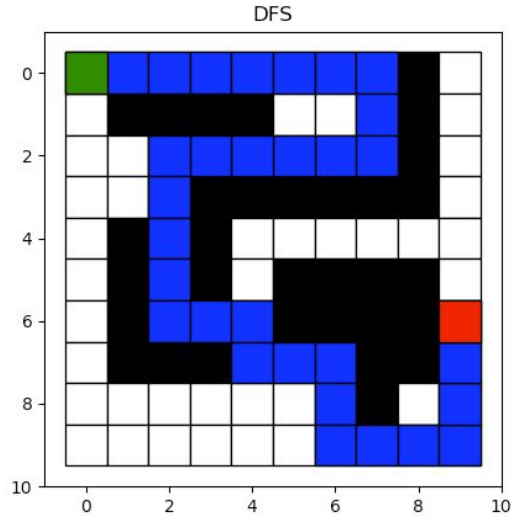
We initialize all the necessary variables and create and queue which stores the nodes that are still left to explore. We add the start nodes to this queue. We run a while loop over all nodes in the queue to find their neighbors. Boundary conditions are tested and if all are satisfied, we add the new neighbors to the queue in FIFO order and save their parent node in a prev queue. We make that node as visited and increase the number of steps. After all the nodes are explored in the queue we backtrack and find the path from the goal to start and then reverse the order.

Depth First Search:

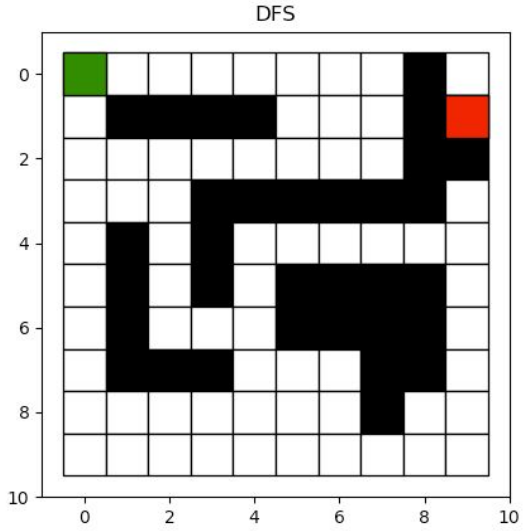
Pseudo code

```
#Initialise start parameters
Row, Col, visited, prev=...
#initialise stack to store values of nodes
rq,cq=....
Assign value “B” to goal point =...
#If condition to check if start is already at goal
If start==goal:
        Stop the search

While elements still left in queue
        Create temp queue to store neighbors
        Loop over the nodes in the queue to find new neighbors (avoiding obstacles, out of bound conditions and already visited nodes)
                Add new neighbors to temp queue
                Mark current node as visited
                Save value of parent node in prev
                Stack =temp queue + stack
                Check if goal “B” is reached
                Exit loop
        Exit loop

#backtrack the path to get to start
at=coordinates of goal
While at not equal to zero:
        Add at value to path
        Update at with parent node to till it becomes zero
#This path is the path from goal to start so we reverse the order
Reverse path
```

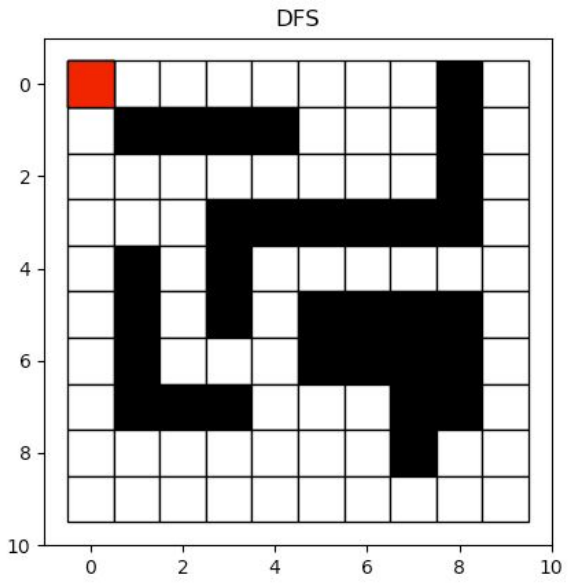
Test cases for DFS




```
It takes 33 steps to find a path using DFS
```



No path found




```
It takes 0 steps to find a path using DFS
```

Working of code:

We initialize all the necessary variables and create and queue which stores the nodes that are still left to explore. We add the start nodes to this stack. We run a while loop over all nodes in the stack to find their neighbors. Boundary conditions are tested and if all are satisfied, we add the new neighbors to a temporary stack and save their parent node in a prev queue. We then add this temporary stack to the original stack thus making it a LIFO order. We make that node as visited and increase the number of steps. After all the nodes are explored in the queue we backtrack and find the path from the goal to start and then reverse the order.

Dijkstra:

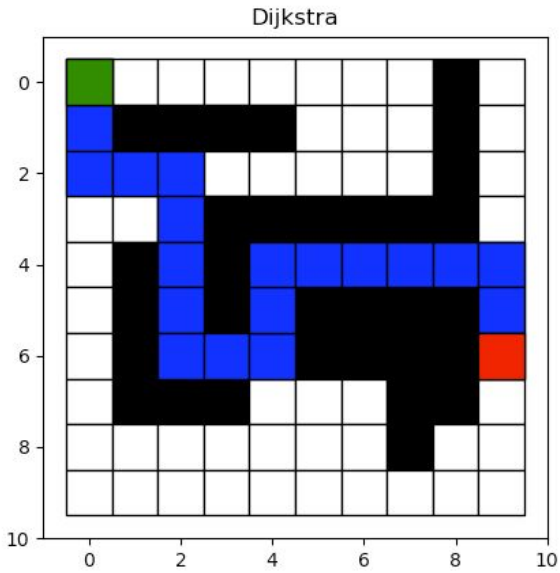
Pseudo code

#Initialise start parameters
Row, Col, visited, prev=...
#initialise cost as infinity of all nodes
cost=inf
#initialise queue to store values of nodes and its corresponding cost
queue=....
Assign value “B” to goal point =...
#If condition to check if start is already at goal
If start==goal:
    Stop the search

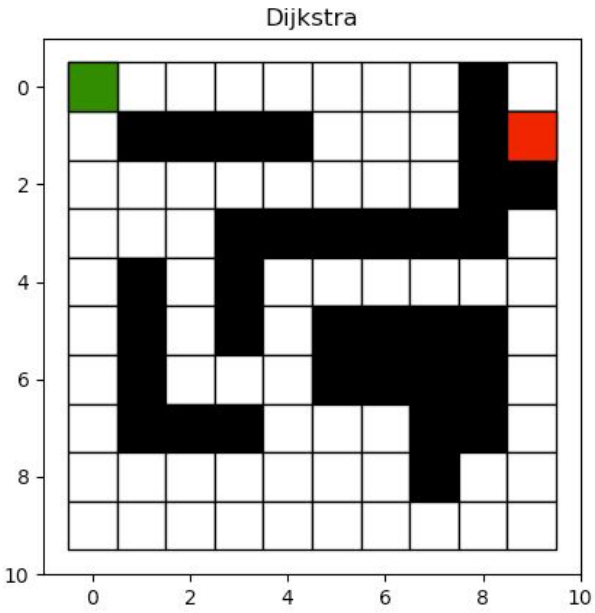
While elements still left in queue
    If Loop over the nodes in the queue to find new neighbors (avoiding obstacles, out of bound conditions and already visited nodes)
        Calculate cost to neighbor node= distance in x + distance in y travelled from start
        Add new neighbors along with corresponding cost to queue
        Mark current node as visited
        Save value of parent node in prev
        Check if goal “B” is reached
        Exit loop
    Sort queue based on cost and run while loop again
    Exit loop

#backtrack the path to get to start
at=coordinates of goal
While at not equal to zero:
    Add at value to path
    Update at with parent node to till it becomes zero
#This path is the path from goal to start so we reverse the order
Reverse path

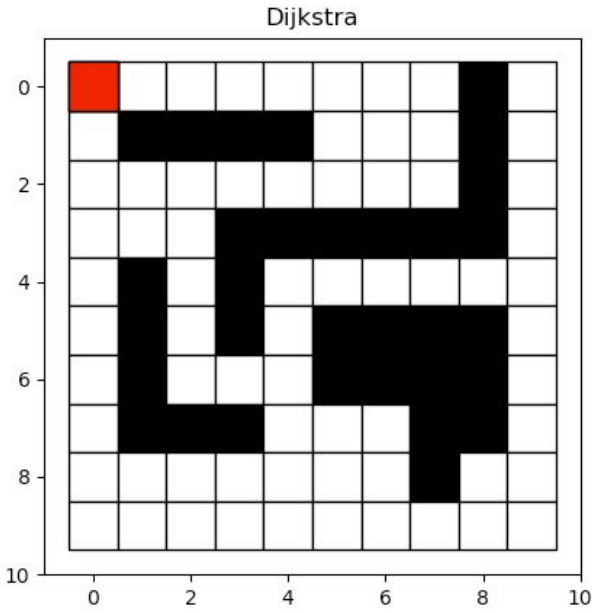
Test cases for Dijkstra




```
It takes 61 steps to find a path using Dijkstra
```



No path found



Start = goal

Working of code:

We initialize all the necessary variables and cost to all nodes as infinity and create a queue which stores the cost and the corresponding nodes that are still left to explore.. We add the start nodes to this queue with cost 0. We run a while loop over all nodes in the queue to find their neighbors. Boundary conditions are tested and if all are satisfied, we add the new neighbors along with the cost (manhattan distance from start) to the queue in FIFO order and save their parent node in a prev queue. We make that node as visited and increase the number of steps. We then sort the queue based on the cost and explore neighbors again. After all the nodes are explored in the queue we backtrack and find the path from the goal to start and then reverse the order.

A* algorithm:

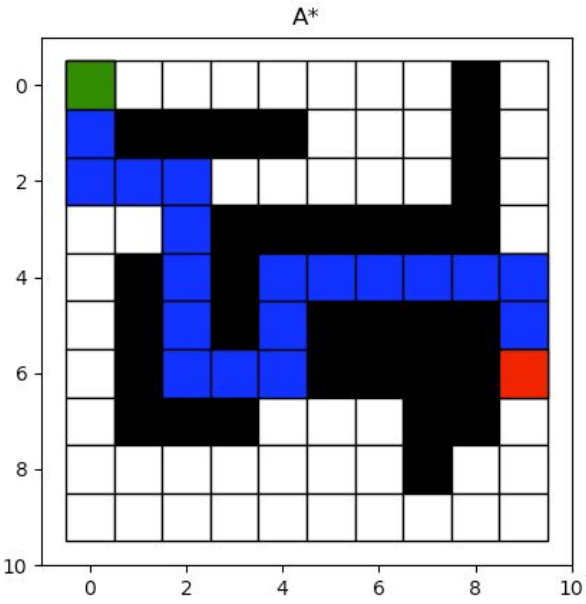
Pseudo code

#Initialise start parameters
Row, Col, visited, prev=...
#initialise cost as infinity of all nodes
cost=inf
#initialise queue to store values of nodes and its corresponding cost
queue=....
Assign value “B” to goal point =...
#If condition to check if start is already at goal
If start==goal:
    Stop the search

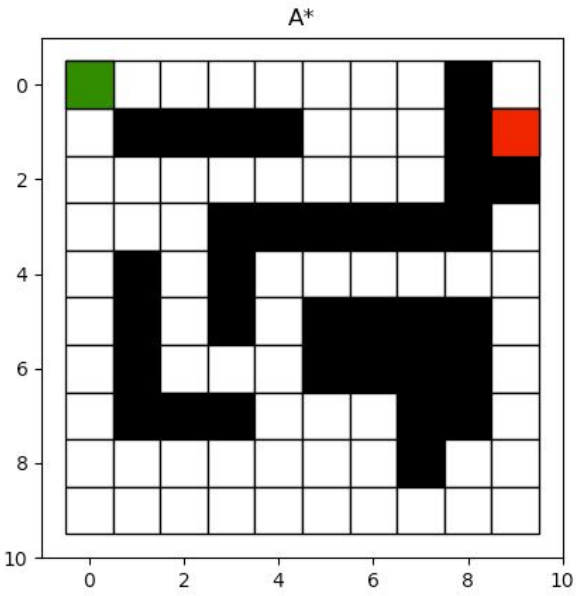
While elements still left in queue
    If Loop over the nodes in the queue to find new neighbors (avoiding obstacles, out of bound conditions and already visited nodes)
        cost g(x) to neighbor node= distance in x + distance in y travelled from start
        Heuristic h(x) = distance in x + distance in y from goal
        f(x)= g(x)+h(x)
        Add new neighbors along with corresponding cost f(x) to queue
        Mark current node as visited
        Save value of parent node in prev
        Check if goal “B” is reached
        Exit loop
    Sort queue based on cost and run while loop again
    Exit loop

#backtrack the path to get to start
at=coordinates of goal
While at not equal to zero:
    Add at value to path
    Update at with parent node to till it becomes zero
#This path is the path from goal to start so we reverse the order
Reverse path

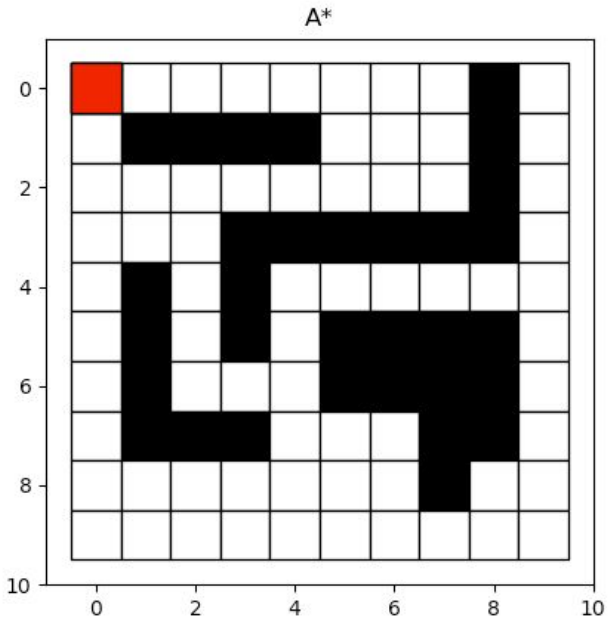
Test cases for A*




```
It takes 44 steps to find a path using A*
```



No path found



Start = goal

Working of code:

We initialize all the necessary variables and cost to all nodes as infinity and heur and create a queue which stores the cost and the corresponding nodes that are still left to explore.. We add the start nodes to this queue with cost 0. We run a while loop over all nodes in the queue to find their neighbors. Boundary conditions are tested and if all are satisfied, we add the new neighbors along with the cost (manhattan distance from start) and heuristic (manhattan distance from goal to node) to the queue in FIFO order and save their parent node in a prev queue. We make that node as visited and increase the number of steps. We then sort the queue based on the cost (g(x)+h(x)) and explore neighbors again. After all the nodes are explored in the queue we backtrack and find the path from the goal to start and then reverse the order.

References:

1. https://towardsdatascience.com/graph-theory-bfs-shortest-path-problem-on-a-grid-1437d5cb4023
2. https://www.youtube.com/watch?v=KiCBXu4P-2Y&t=485s BFS algorithm
3. https://www.youtube.com/watch?v=7fujbpJ0LB4 DFS algorithm
4. https://www.youtube.com/watch?v=GazC3A4OQTE&t=435s Dijkstra algorithm
5. https://www.youtube.com/watch?v=ySN5Wnu88nE A* algorithm