Aadesh Varude
avarude@wpi.edu

Q1 . BFS Explained

- Start BFS traversal from the first cell, i.e. given to us as stsrt, and enqueue the index of this cell into the queue.

```
 queue=[]
queue.append(start)
```

- Initialize a boolean matrix to mark the visited cells of the matrix. Mark the cell as visited while traversing through the path.Initial a parent grid of the same size as the map in order to keep track of the path.

```
parent=np.negative(np.ones((len(grid),len(grid[0])),dtype=object))
 visited=np.zeros((len(grid),len(grid[0])))
```

- Iterate in the while loop till your queue is not empty.
  - In the while loop checks for the nodes if visited or not mark visited if not and explore from that node.
  - Implement a for loop for the four directions as mentioned in the assignment.
  - Within the for loop check for validity condition of traversal and obstacles-free nodes and visited nodes and append in the queue accordingly.

```
        r=[[0, 1], [1, 0], [0, -1], [-1, 0]] # r is the direction of exploration
    for i in r:
                next_x=x+i[0]
                next_y=y+i[1]
                if(if_valid(next_x,next_y,len(grid),len(grid[0]))):
                    if(grid[next_x][next_y]==0 and visited[next_x][next_y]==0):
                        queue.append([next_x,next_y])
                        if(parent[next_x][next_y]==-1):
                            parent[next_x][next_y]=(x,y)
```

- If you find the goal then backtrack the path using the parent matrix.

```
   def path_finder(start,goal,parent):
 path=[]
 x=goal[0]
 y=goal[1]
 path.append(goal)
 # print("goal pose",goal)
 while parent[x][y]!=(start[0],start[1]):
     # print("values appending in path",[parent[x][y][0],parent[x][y][1]])
     path.append([parent[x][y][0],parent[x][y][1]])
     # print(parent)
     x1=parent[x][y][0]
     y=parent[x][y][1]
     x=x1

 path.append(start)
 path.reverse()
 return path
```

2 . DFS Explained

- Start DFS traversal from the first cell, i.e. given to us as a start, and put it on the top of the stack.

```
 stack=[]
stack.append(start)
```

- Initialize a boolean matrix to mark the visited cells of the matrix. Mark the cell as visited while traversing through the path.Initial a parent grid of the same size as the map in order to keep track of the path.

```
parent=np.negative(np.ones((len(grid),len(grid[0])),dtype=object))
 visited=np.zeros((len(grid),len(grid[0])))
```

- Iterate in the while loop till your queue is not empty.
  - In the while loop checks for the nodes if visited or not mark visited if not and explore from that node.
  - Implement a for loop for the four directions as mentioned in the assignment.
  - Within the for loop check for validity condition of traversal and obstacles-free nodes and visited nodes and append in the queue accordingly.

```
        r=[[-1, 0],[0, -1],[1, 0],[0, 1]]# r is the direction of exploration it is reversed then the
            given priority as the stack pops from the top.
          for i in r:
              next_x=x+i[0]
              next_y=y+i[1]
              if(if_valid(next_x,next_y,len(grid),len(grid[0]))):
                  if(grid[next_x][next_y]==0 and visited[next_x][next_y]==0):
                      stack.append([next_x,next_y])
                      parent[next_x][next_y]=(x,y)
```

- If you find the goal then backtrack the path using the parent matrix. As previously mentioned in the BFS code same function is used here too.

Similarities between DFS and BFS

- There are a lot of similarities with respect to the code in DFS and BFS algorithms.
- For path backtracking I have incorporated a function that takes the goal, destination and parent matrix and returns the path array
- Also the conditions to be valid x and y are the same for both the implementation hence encoded within a function.

Difference between DFS and BFS

- The main difference between DFS and BFS is that in DFS we prioritized the deepest node in the frontier. In our case, we go to the right node and then explore the right node of the previous node so on and so forth. Whereas, in BFS we search for the neighbouring nodes and explore all the valid neighbours of the node and then traverse along.
- The pseudocode for BFS is remarkably close to DFS, the only difference is that the frontier is a queue instead of a stack.
- The data structure used in BFS is queue and DFS is stack.

Results

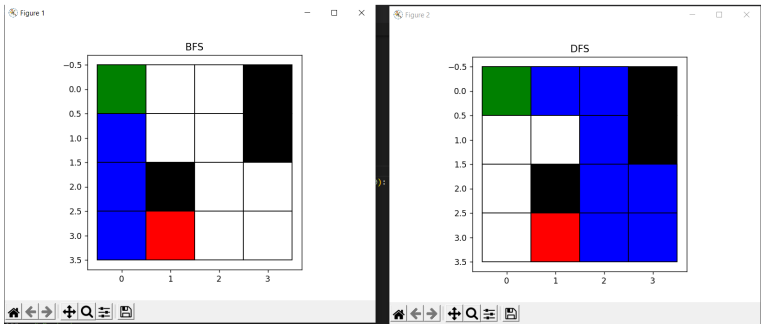

Figure 1: Images of BFS DFS on the given testmap csv file.

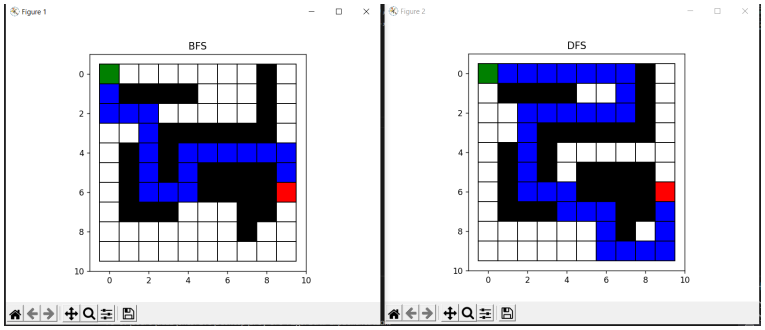

Figure 2: Images of BFS DFS on the given map csv file.

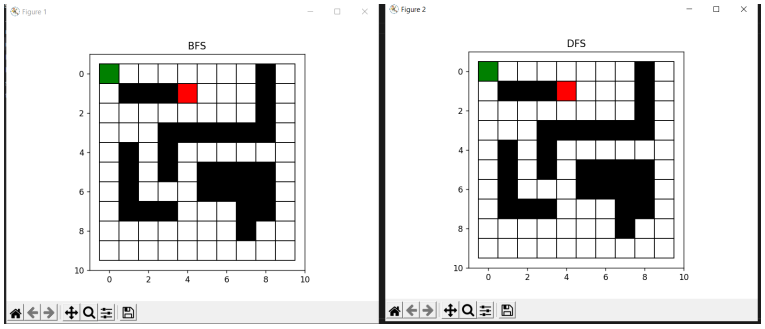

Figure 3: Images of BFS DFS on the given map when the obstacle is the goal.

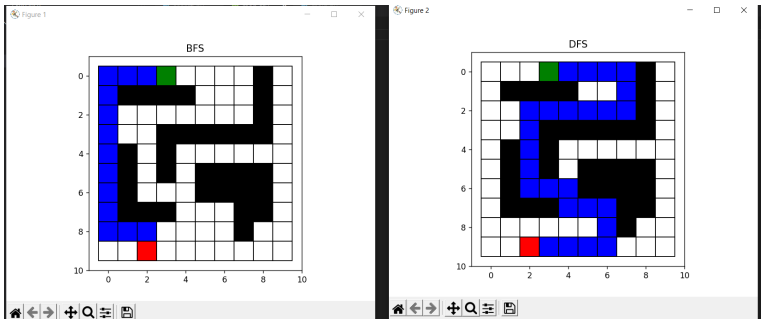

Figure 4: Images of BFS DFS on the given map with different start and goal point.

References

1. https://medium.com/nerd-for-tech/dfs-bfs-introduction-26a65fca2344
2. https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/
3. https://www.tutorialspoint.com/difference-between-bfs-and-dfs