

## Maze (Grid Form):

```

S   .   .
#  #   .
.   .   E

```

S = Start (0,0)

E = End (2,2)

. = Free path

# = Wall (blocked)

## Graph Representation (Adjacency List):

```

graph = {
  (0,0): [(0,1)],      # S can go right
  (0,1): [(0,0), (0,2)], # middle top
  (0,2): [(0,1), (1,2)], # top right
  (1,2): [(0,2), (2,2)], # connects to goal
  (2,0): [(2,1)],      # bottom left
  (2,1): [(2,0), (2,2)], # bottom middle
  (2,2): [(1,2), (2,1)] # Goal (E)
}

```

## BFS Tree Format:

```

BFS Tree Format:(0,0) <-- Start (S)
├── (0,1)
│   ├── (0,0) [already visited]
│   └── (0,2)
│       ├── (0,1) [already visited]
│       └── (1,2)

```

```

└── (0,2) [already visited]
└── (2,2) <-- Goal (E)

```

here's also another branch (from bottom side of the maze):

```

(2,0)
└── (2,1)
    ├── (2,0) [already visited]
    └── (2,2) <-- Goal (E)

```

## BFS Traversal (Shortest Path Search):

```
from collections import deque
```

```
def bfs_shortest_path(graph, start, goal):
    visited = set()
    queue = deque([(start, [start])]) # store (node, path)

    while queue:
        node, path = queue.popleft()
        if node == goal:
            return path # shortest path found
        if node not in visited:
            visited.add(node)
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append((neighbor, path + [neighbor]))
    return None
```

```
# Graph definition (maze as adjacency list)
```

```
graph = {
    (0,0): [(0,1)],
    (0,1): [(0,0), (0,2)],
    (0,2): [(0,1), (1,2)],
    (1,2): [(0,2), (2,2)],
    (2,0): [(2,1)],
    (2,1): [(2,0), (2,2)],
    (2,2): [(1,2), (2,1)]
}
```

```
# Run BFS from Start to Goal
```

```
start, goal = (0,0), (2,2)
```

```
path = bfs_shortest_path(graph, start, goal)
```

```
print("Shortest Path from S to E:", path)
```

## Output:

Shortest Path from S to E: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2)]

## screenshot of output:

```

1 def find_start_end(maze):
2     start = None
3     end = None
4     for r in range(len(maze)):
5         for c in range(len(maze[0])):
6             if maze[r][c] == 'S':
7                 start = (r, c)
8             elif maze[r][c] == 'E':
9                 end = (r, c)
10    return start, end
11
12 def is_valid(r, c, rows, cols, maze, visited):
13     return (0 <= r < rows and 0 <= c < cols and
14           maze[r][c] != '#' and (r, c) not in visited)
15
16 directions = [(0,1), (0,-1), (1,0), (-1,0)]
17
18 def dfs(maze):
19     rows, cols = len(maze), len(maze[0])
20     start, end = find_start_end(maze)
21     if not start or not end:
22         return None
23
24     stack = [(start, [start])]
25     visited = set([start])
26
27     while stack:
28         (r, c), path = stack.pop()
29         if (r, c) == end:
30             return path
31         for dr, dc in directions:
32             nr, nc = r + dr, c + dc
33             if is_valid(nr, nc, rows, cols, maze, visited):
34                 visited.add((nr, nc))
35                 stack.append(((nr, nc), path + [(nr, nc)]))
36
37     return None

```

```

1 from collections import deque
2
3 def find_start_end(maze):
4     start = None
5     end = None
6     for r in range(len(maze)):
7         for c in range(len(maze[0])):
8             if maze[r][c] == 'S':
9                 start = (r, c)
10            elif maze[r][c] == 'E':
11                end = (r, c)
12    return start, end
13
14 def is_valid(r, c, rows, cols, maze, visited):
15     return (0 <= r < rows and 0 <= c < cols and
16           maze[r][c] != '#' and (r, c) not in visited)
17
18 directions = [(0,1), (0,-1), (1,0), (-1,0)]
19
20 def bfs(maze):
21     rows, cols = len(maze), len(maze[0])
22     start, end = find_start_end(maze)
23     if not start or not end:
24         return None
25
26     queue = deque([(start, [start])])
27     visited = set([start])
28
29     while queue:
30         (r, c), path = queue.popleft()
31         if (r, c) == end:
32             return path
33         for dr, dc in directions:
34             nr, nc = r + dr, c + dc
35             if is_valid(nr, nc, rows, cols, maze, visited):
36                 visited.add((nr, nc))
37                 queue.append(((nr, nc), path + [(nr, nc)]))
38
39     return None

```

