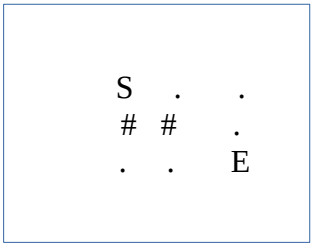
Maze (Grid Form):



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: