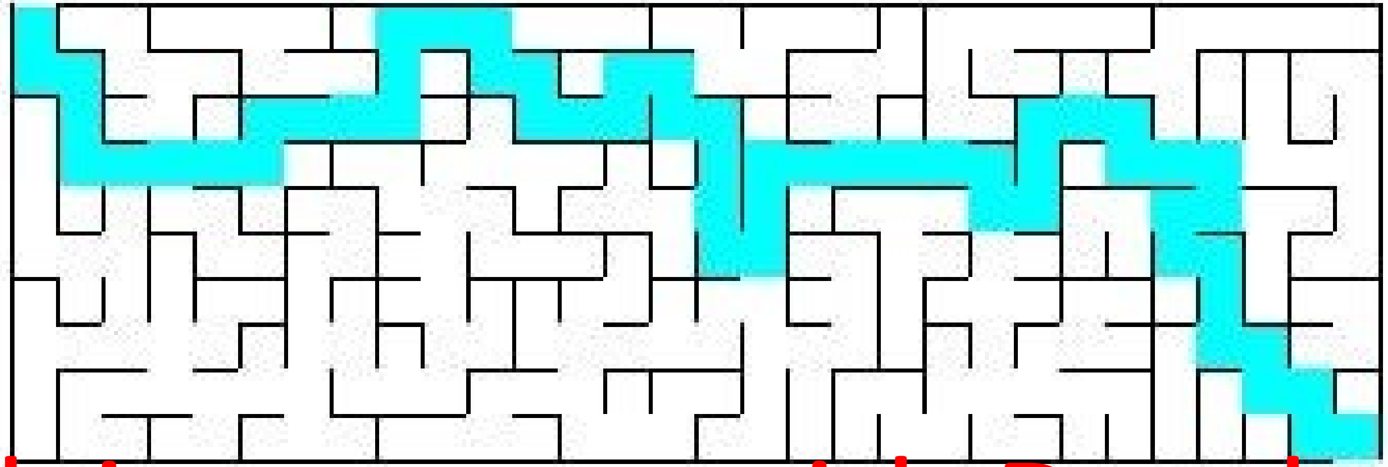


Solving mazes with Depth-First Search

Lab Assignment 6

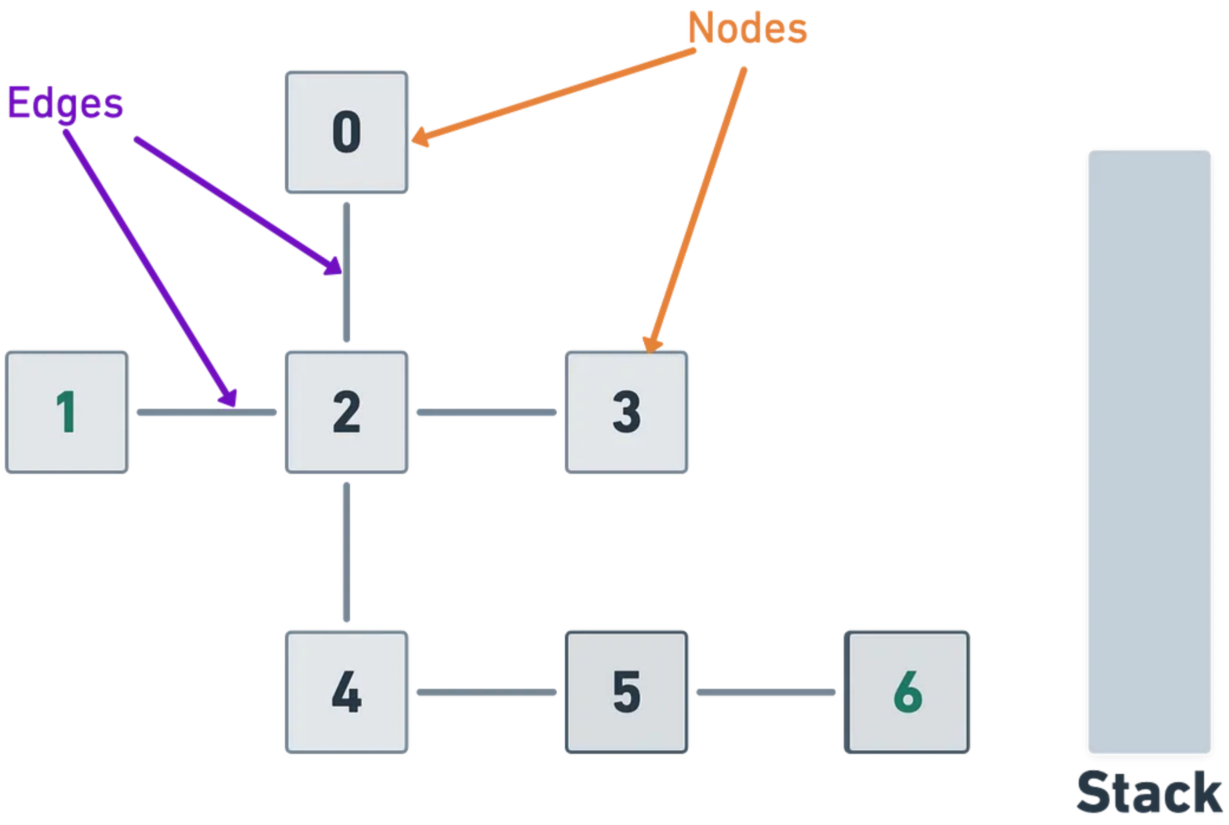
Dr. Subhadip Pramanik

Assistant Professor

School of Computer Engineering

KIIT Deemed to be University

Graph



Step 1

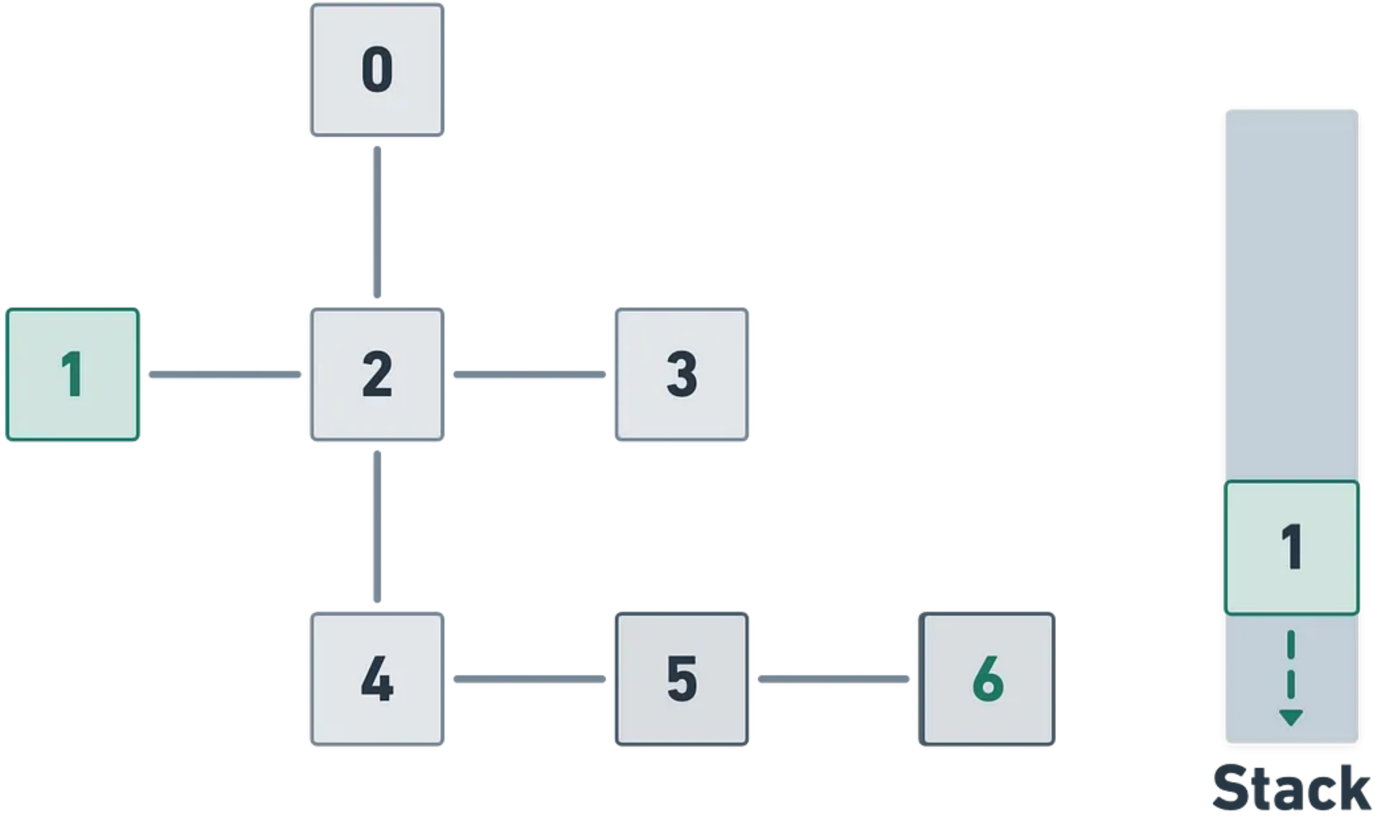


Figure 3 — Green indicates the node we’re actively evaluating.

Step 2

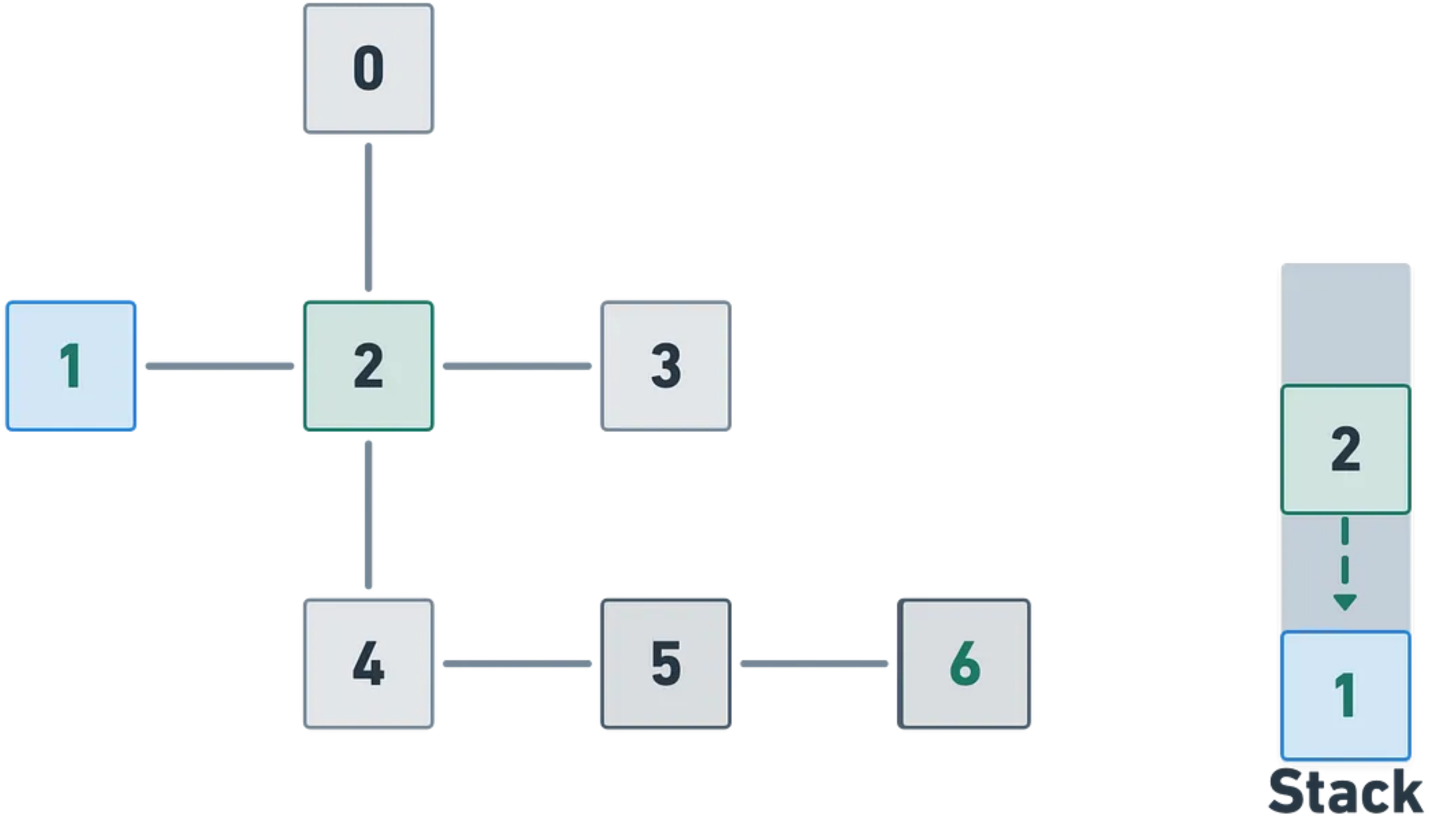
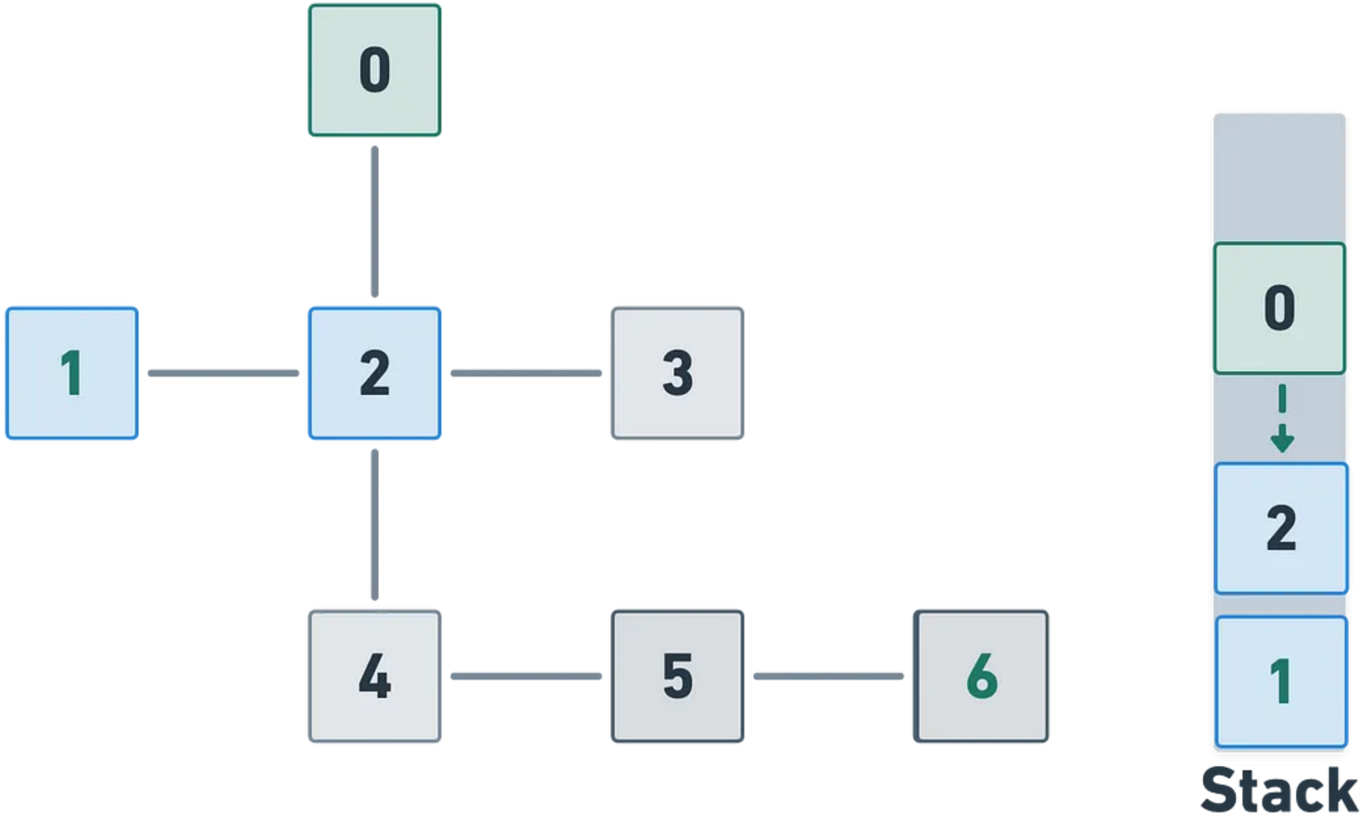
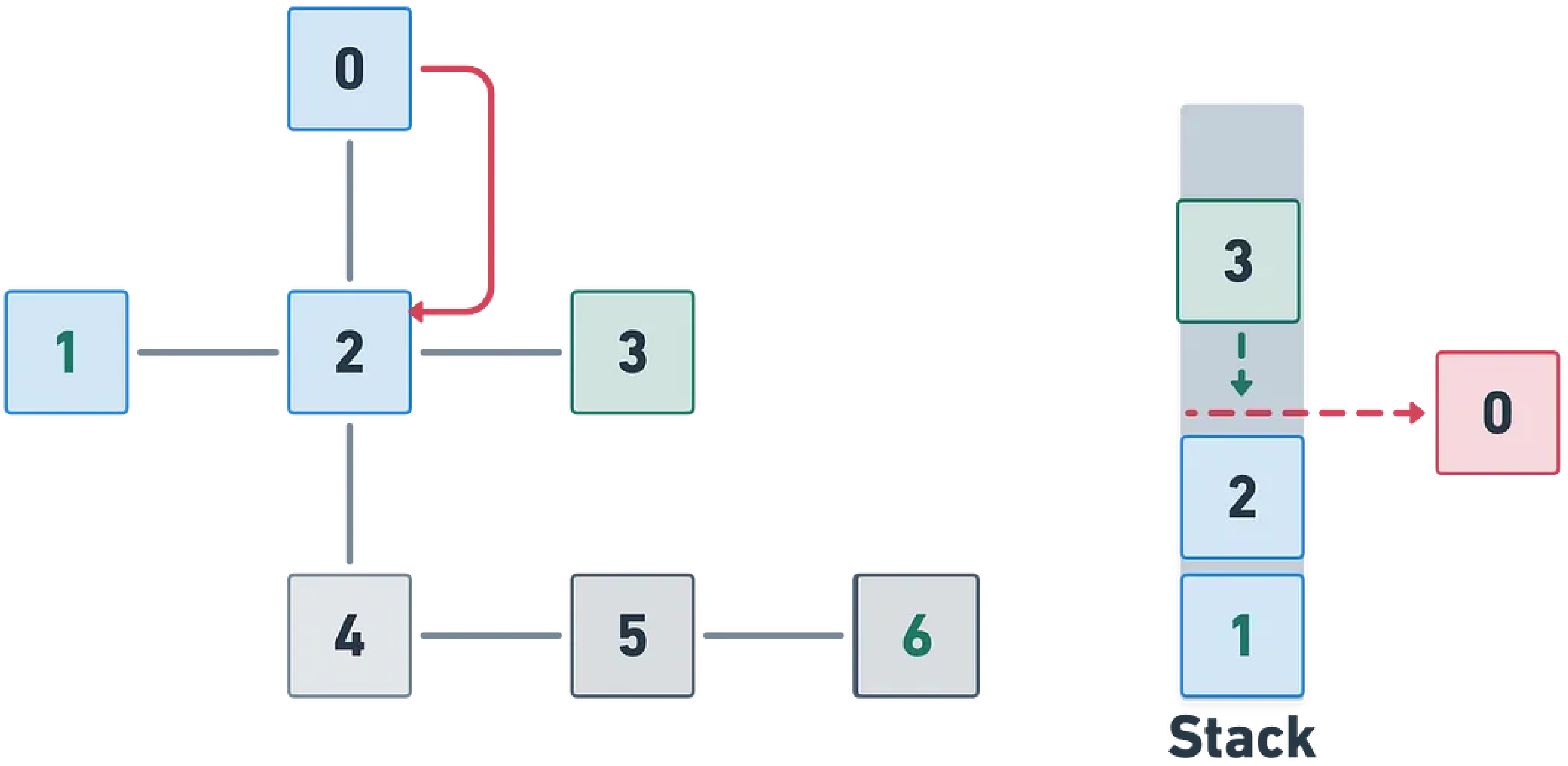


Figure 3 — Blue indicates an already evaluated(visited) node.

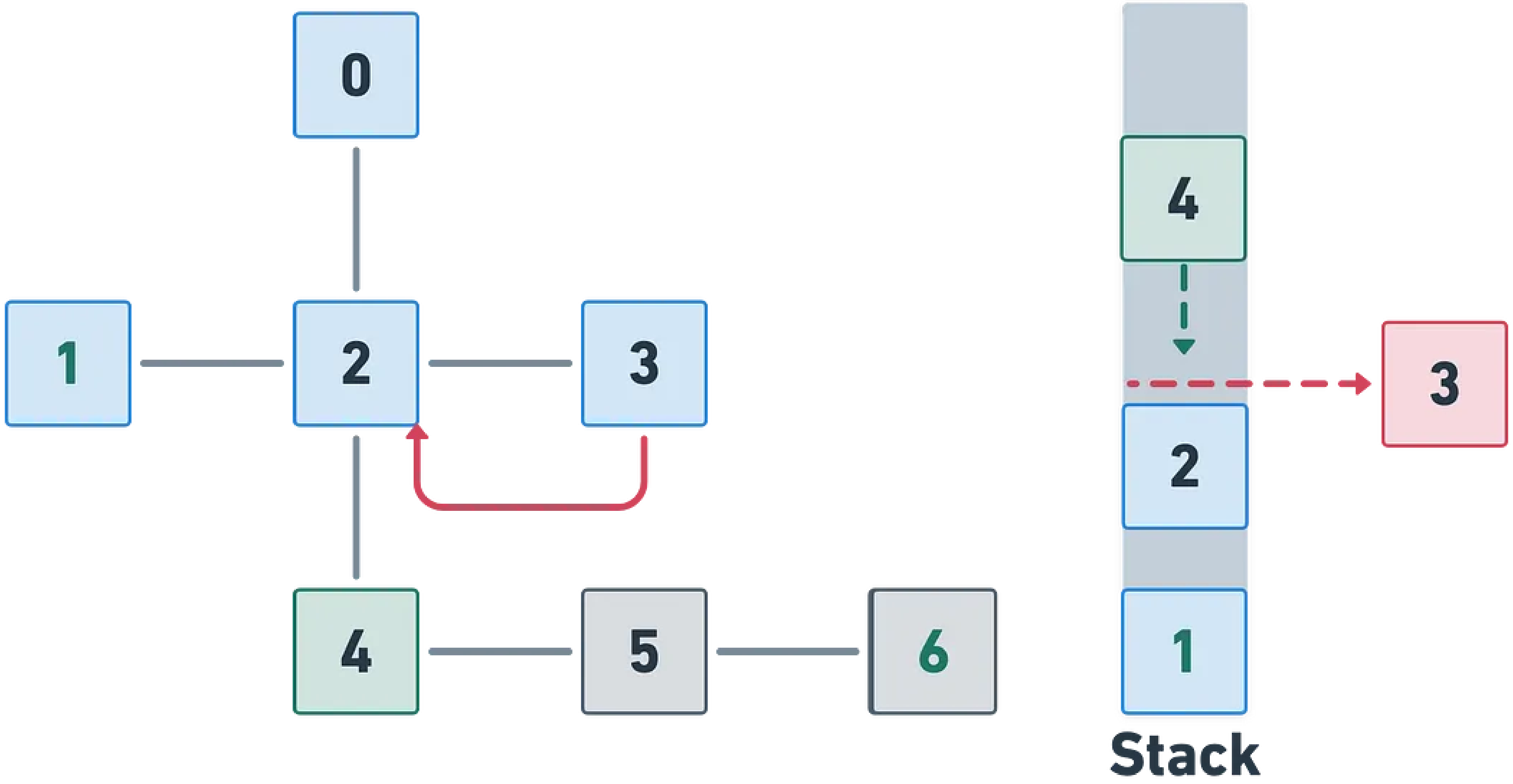
Step 3



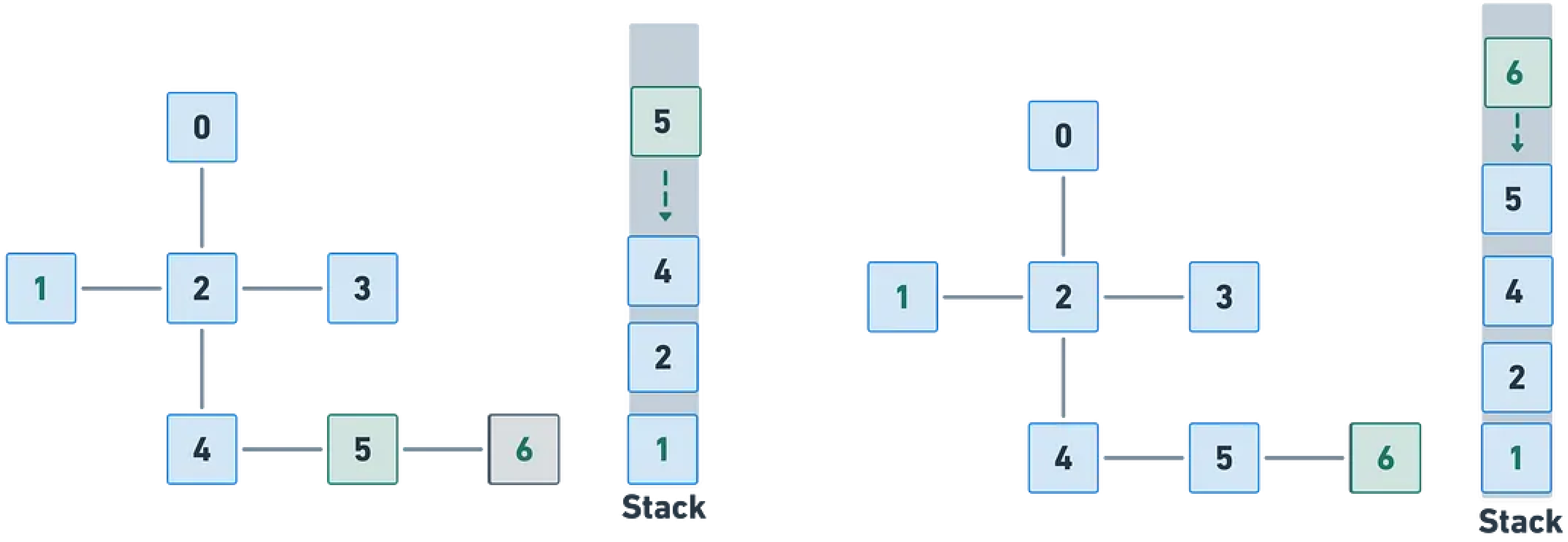
Step 4



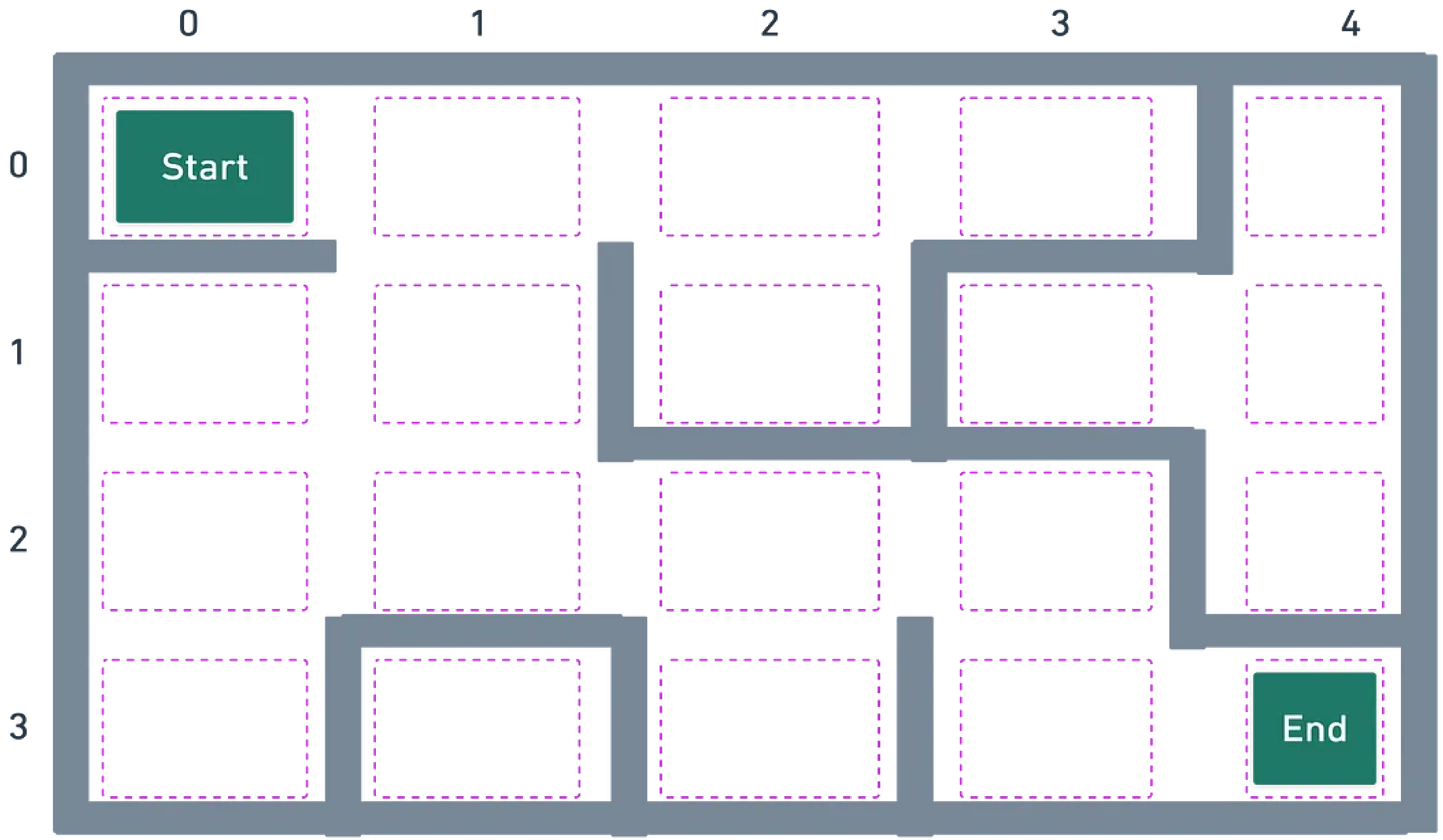
Step 5



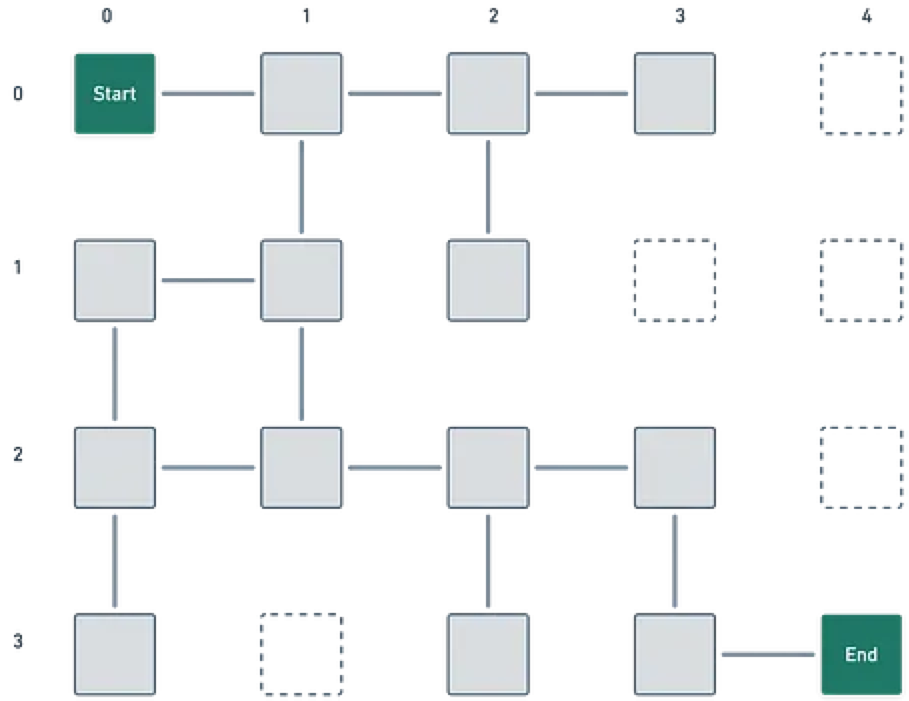
Step 6 and Step 7

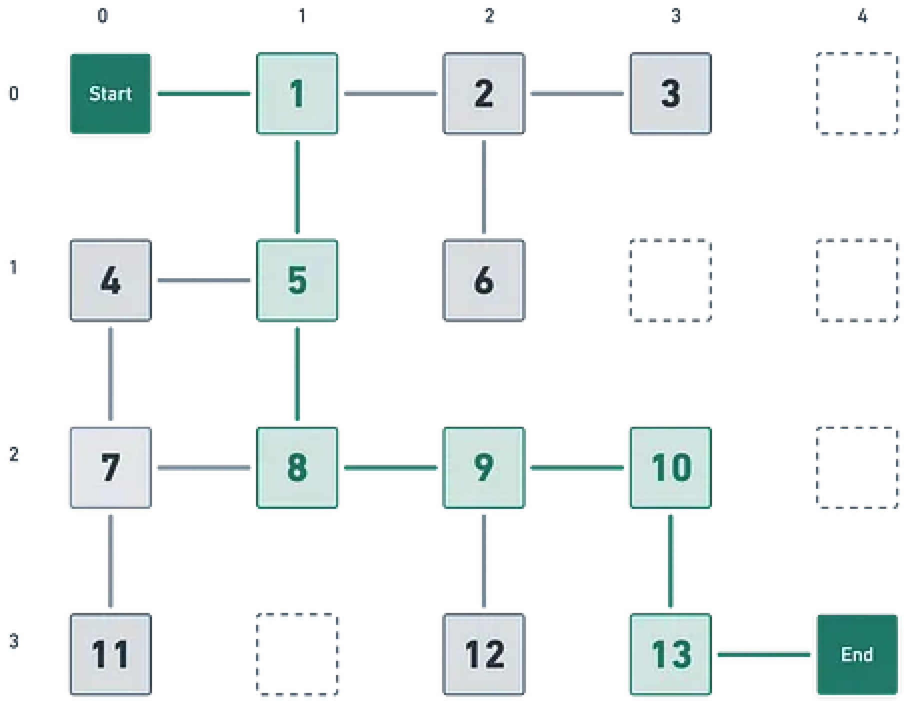


Maze

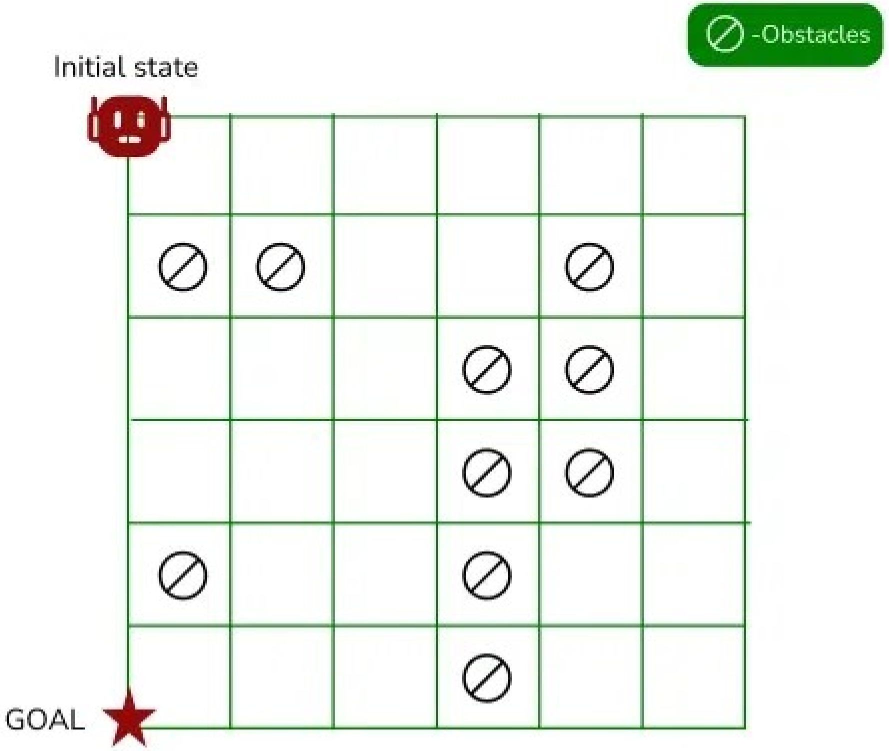


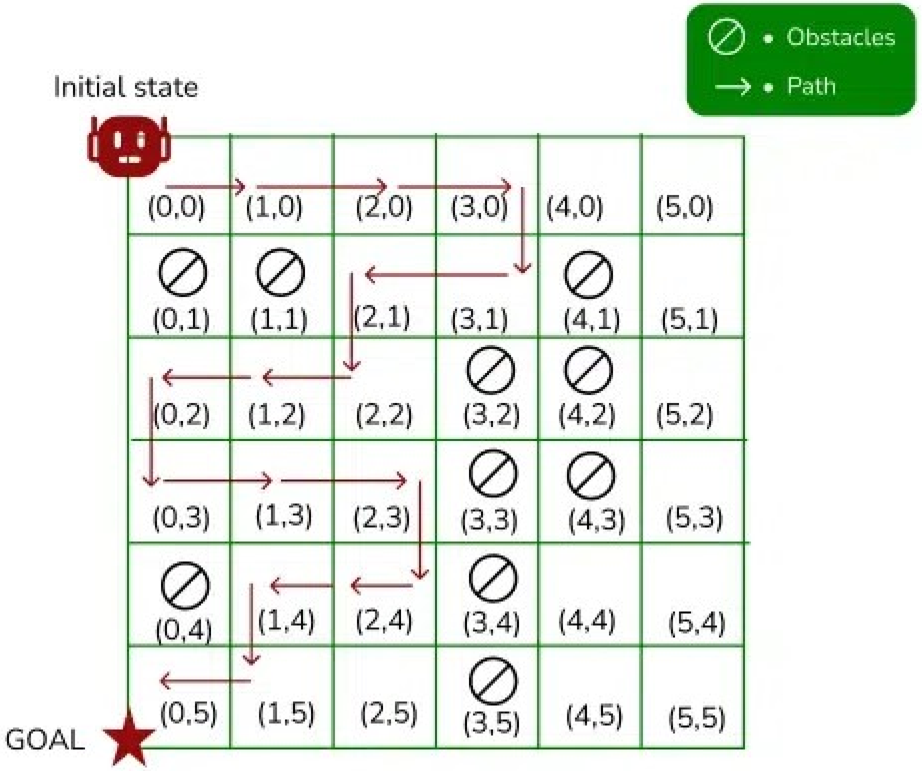
Abstract the image





Example





The robot is positioned at the initial state (0,0) and aims to reach the goal state (0,5).

# Maze dimensions and obstacles

```
maze_size = 6
obstacles = [(0,1),(1,1),(3,2),(3,3),(3,4),(3,5),(0,4),(4,1),(4,2),(4,3)]
start = (0,0)
goal = (0,5)
```

Step 2: Define a is_valid function

The ***is_valid*** function checks whether a given position of (x,y) is valid such that it inspects that it's within the bounds of the maze and not obstructed by any obstacles.

```
def is_valid(x,y):
  return 0 <= x < maze_size and 0 <= y < maze_size and (x,y) not in obstacles
```

Step 3: Define dfs function (Depth-first search):

```
def dfs (current, visited, path):
  x, y = current
  if current == goal:
    path.append(current)
    return True
  visited.add(current)
  moves = [(x-1,y), (x+1, y), (x, y-1), (x, y+1)]
  for move in moves:
    if is_valid(*move) and move not in visited:
      if dfs(move, visited, path):
        path.append(current)
        return True
  return False
```

Step 4: Call DFS function to find the path

```
visited = set()
path = []
if dfs(start, visited, path):
  path.reverse()
  print("Path found:")
  for position in path:
    print(position)
else:
  print("No path found!")
```