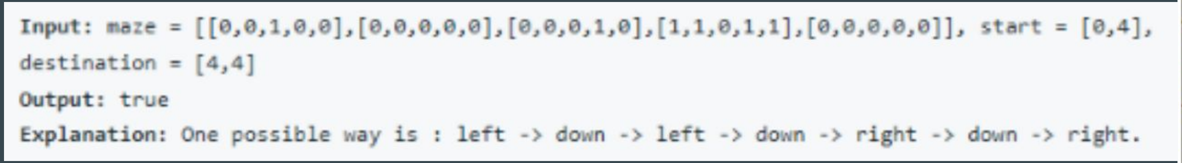
Comparison of DFS and BFS for '490. The Maze'

•••

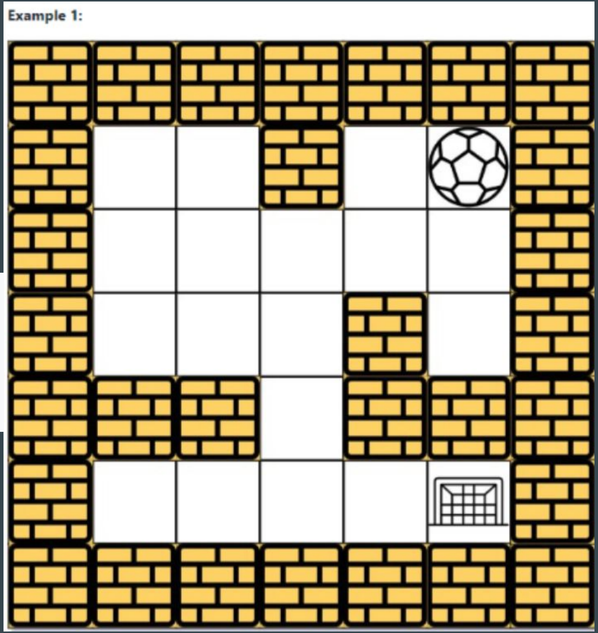
Dan Xu

Problem Statement: LeetCode 490 - The Maze

LeetCode 490 presents us with a maze represented as a 2D array. The maze consists of walls and open passages. The task is to determine if there exists a path from the start to the destination within the maze.



```
Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], start = [0,4],
destination = [4,4]
Output: true
Explanation: One possible way is : left -> down -> left -> down -> right -> down -> right.
```

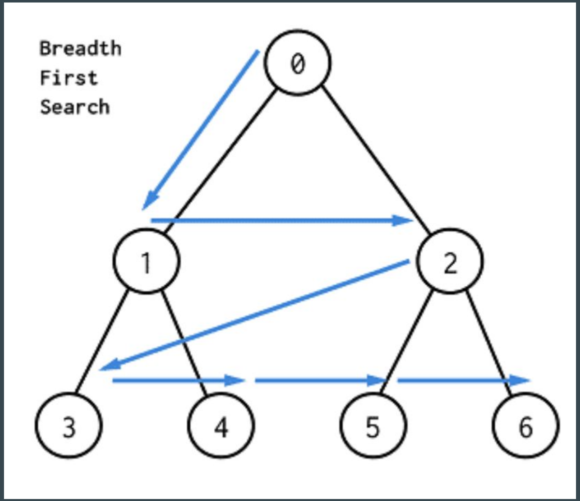


BFS (Breadth-First Search) Approach

BFS is an algorithm that explores the neighbor nodes first before moving to the next level of neighbors. It guarantees the shortest path in an unweighted graph.

To apply BFS to the maze problem, we start from the entrance and explore all possible paths level by level until we reach the destination.

BFS allows us to find the shortest path, but it may require more memory compared to DFS.

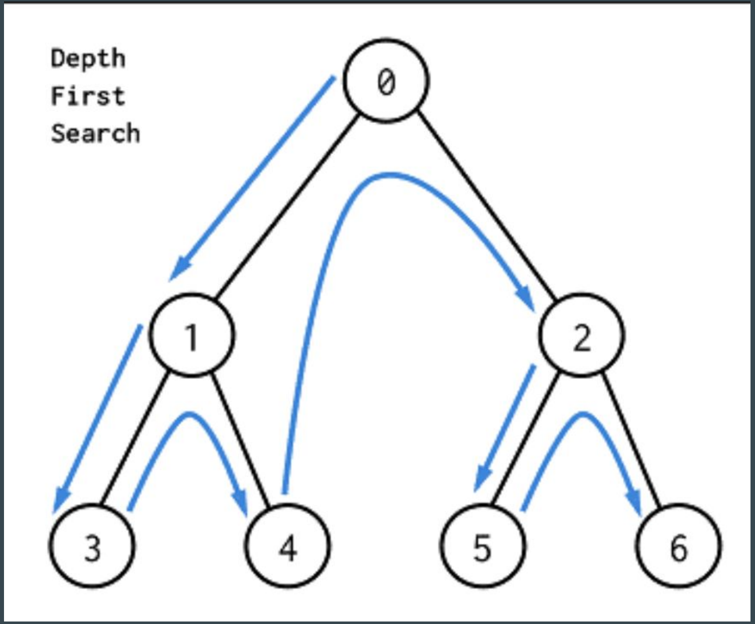


DFS (Depth-First Search) Approach

DFS is an algorithm that explores as far as possible along each branch before backtracking. It is implemented using a stack.

To apply DFS to the maze problem, we start from the entrance and explore a path until we either reach the destination or encounter a dead-end. Then, we backtrack and explore other paths.

DFS may not necessarily find the shortest path, but it can be more memory-efficient than BFS.



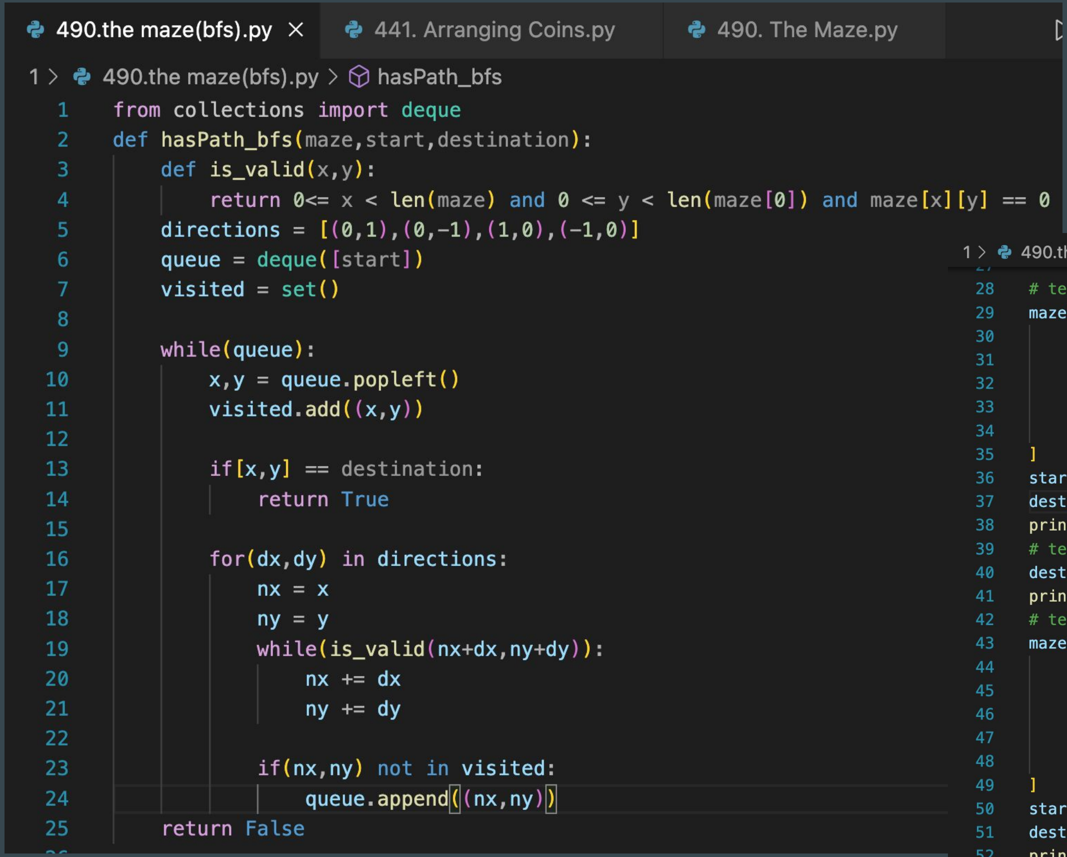
Comparison of BFS and DFS

Time Complexity: Both BFS and DFS have a time complexity of O(V + E), where V is the number of vertices and E is the number of edges. In our maze problem, the time complexity is O(N * M), where N and M are the dimensions of the maze.

Space Complexity: BFS generally requires more memory than DFS due to the need to store all nodes at each level in a queue. DFS uses a smaller memory footprint as it explores deeply before backtracking.

Performance: BFS guarantees the shortest path, but DFS may not always find the optimal solution. The choice between the two depends on the maze structure and desired outcome.

Implementation & Test



490.the maze(bfs).py × | 441. Arranging Coins.py | 490. The Maze.py

1 > 490.the maze(bfs).py > hasPath_bfs

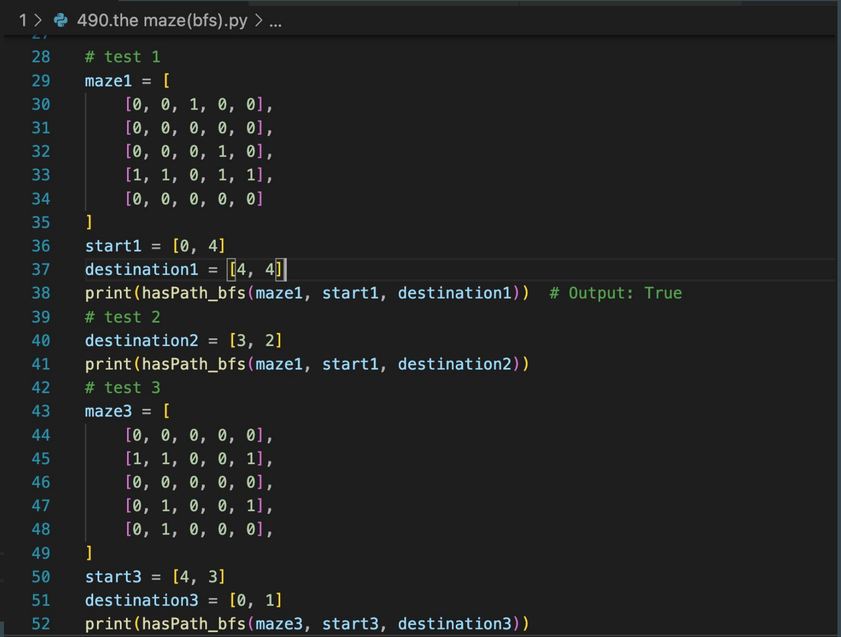
```python
from collections import deque
def hasPath_bfs(maze,start,destination):
    def is_valid(x,y):
        return 0<= x < len(maze) and 0 <= y < len(maze[0]) and maze[x][y] == 0
    directions = [(0,1),(0,-1),(1,0),(-1,0)]
    queue = deque([start])
    visited = set()

    while(queue):
        x,y = queue.popleft()
        visited.add((x,y))

        if[x,y] == destination:
            return True

        for(dx,dy) in directions:
            nx = x
            ny = y
            while(is_valid(nx+dx,ny+dy)):
                nx += dx
                ny += dy

            if(nx,ny) not in visited:
                queue.append((nx,ny))
    return False
```



1 > 490.the maze(bfs).py > ...

```python
# test 1
maze1 = [
    [0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0],
    [1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0]
]
start1 = [0, 4]
destination1 = [4, 4]
print(hasPath_bfs(maze1, start1, destination1))  # Output: True
# test 2
destination2 = [3, 2]
print(hasPath_bfs(maze1, start1, destination2))
# test 3
maze3 = [
    [0, 0, 0, 0, 0],
    [1, 1, 0, 0, 1],
    [0, 0, 0, 0, 0],
    [0, 1, 0, 0, 1],
    [0, 1, 0, 0, 0],
]
start3 = [4, 3]
destination3 = [0, 1]
print(hasPath_bfs(maze3, start3, destination3))
```

Enhancement Ideas

Although BFS and DFS are effective, there are ways to improve their performance further.

One enhancement idea is to implement bidirectional search algorithms that combine BFS and DFS for improved efficiency.

Another approach is to use heuristics such as A* algorithm to guide the search process intelligently.

Conclusion

Both BFS and DFS are valuable tools for solving maze-related problems.

BFS guarantees the shortest path, making it suitable for situations where finding the optimal solution is critical.

DFS, on the other hand, is memory-efficient and may be preferred in memory-constrained scenarios.

The choice between BFS and DFS depends on the specific requirements of the problem.