```python
import heapq
import matplotlib.pyplot as plt
import numpy as np

# Define the directions for movement (up, down, left, right)
DIRECTIONS = [(0, 1), (1, 0), (0, -1), (-1, 0)]

class Node:
    """Class to represent a node in the maze."""
    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0  # Cost from start to current node
        self.h = 0  # Heuristic cost to the goal
        self.f = 0  # Total cost (f = g + h)

    def __lt__(self, other):
        return self.f < other.f

def heuristic(a, b):
    """Calculate Manhattan distance as the heuristic."""
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def a_star(maze, start, end):
    """A* algorithm to find the shortest path in a maze."""
    open_list = []
    closed_set = set()

    start_node = Node(start)
    end_node = Node(end)
```

```
heapq.heappush(open_list, start_node)

while open_list:
    current_node = heapq.heappop(open_list)
    closed_set.add(current_node.position)

    # Check if we reached the goal
    if current_node.position == end_node.position:
        path = []
        while current_node:
            path.append(current_node.position)
            current_node = current_node.parent
        return path[::-1]  # Return reversed path

    # Explore neighbors
    for direction in DIRECTIONS:
        neighbor_pos = (current_node.position[0] + direction[0],
                        current_node.position[1] + direction[1])

        # Check if the neighbor is within bounds and not a wall
        if (0 <= neighbor_pos[0] < maze.shape[0] and
            0 <= neighbor_pos[1] < maze.shape[1] and
            maze[neighbor_pos] == 0 and
            neighbor_pos not in closed_set):

            neighbor_node = Node(neighbor_pos, current_node)
            neighbor_node.g = current_node.g + 1
            neighbor_node.h = heuristic(neighbor_pos, end_node.position)
            neighbor_node.f = neighbor_node.g + neighbor_node.h

            # Check if this path to the neighbor is better
```

```
            if not any(open_node.position == neighbor_pos and open_node.f <= neighbor_node.f for
open_node in open_list):
                heapq.heappush(open_list, neighbor_node)

    return None  # No path found

def plot_maze(maze, path=None):
    """Visualize the maze and the path."""
    plt.figure(figsize=(8, 8))
    plt.imshow(maze, cmap="binary")
    if path:
        path_x, path_y = zip(*path)
        plt.plot(path_y, path_x, color="red", linewidth=2)  # Path in red
    plt.title("Maze Navigation")
    plt.show()

if __name__ == "__main__":
    # Define a simple maze (0 = free space, 1 = wall)
    maze = np.array([
        [0, 1, 0, 0, 0],
        [0, 1, 0, 1, 0],
        [0, 0, 0, 1, 0],
        [0, 1, 1, 1, 0],
        [0, 0, 0, 0, 0]
    ])

    start = (0, 0)  # Starting position
    end = (4, 4)    # Goal position

    # Find the shortest path using A* algorithm
    path = a_star(maze, start, end)
```

```
if path:
    print("Path found:", path)
    plot_maze(maze, path)
else:
    print("No path found!")
    plot_maze(maze)
```