جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قسم الانظمة الطبية الذكية
Lab: (4)

Best-First Search (Python Code)

Subject: Artificial Intelligence
Class: Third
Lecturer: Dr. Maytham N. Meqdad

Best-First Search is a pathfinding algorithm that prioritizes nodes based on a heuristic function, often estimating the cost to reach the goal from the current node.This example assumes the nodes are represented in a graph and each node has a heuristic value associated with it, such as an estimated distance to the goal.

Python Code for Best-First Search

```
import heapq
class Graph:
    def __init__(self):
        self.edges = {}

    def add_edge(self, node1, node2, cost=1):
        if node1 not in self.edges:
            self.edges[node1] = []
        if node2 not in self.edges:
            self.edges[node2] = []
        self.edges[node1].append((node2, cost))
        self.edges[node2].append((node1, cost))

def best_first_search(graph, start, goal, heuristic):
    open_list = []
    heapq.heappush(open_list, (heuristic[start], start))
    came_from = {start: None}
    visited = set()

    while open_list:
        current_cost, current_node = heapq.heappop(open_list)

        if current_node in visited:
            continue
        visited.add(current_node)

        if current_node == goal:
            path = []
            while current_node is not None:
                path.append(current_node)
                current_node = came_from[current_node]
            return path[::-1]  # Return reversed path from start to goal

        for neighbor, cost in graph.edges.get(current_node, []):
            if neighbor not in visited:
                came_from[neighbor] = current_node
                heapq.heappush(open_list, (heuristic[neighbor], neighbor))

    return None  # If no path found

# Example usage:
# Define a graph and heuristics (heuristic values are estimations of distance to the goal)
graph = Graph()
```

```
graph.add_edge("A", "B", 1)
graph.add_edge("A", "C", 3)
graph.add_edge("B", "D", 2)
graph.add_edge("C", "D", 1)
graph.add_edge("B", "E", 5)
graph.add_edge("D", "E", 2)

# Heuristic values for each node to goal 'E'
heuristic = {
    "A": 5,
    "B": 3,
    "C": 4,
    "D": 2,
    "E": 0
}

# Perform Best-First Search from 'A' to 'E'
path = best_first_search(graph, "A", "E", heuristic)
print("Path from A to E:", path)
```

Explanation of Code

1. **Graph Representation**: The `Graph` class stores nodes and edges using an adjacency list.

2. **Heuristic Function**: `heuristic` is a dictionary that maps each node to an estimated cost to reach the goal.

3. **Best-First Search Function**: The algorithm uses a priority queue (min-heap) to prioritize nodes based on heuristic values, adding nodes with lower estimated costs first.

4. **Path Reconstruction**: Once the goal is reached, the path is reconstructed by backtracking from the goal to the start node using `came_from`.

5. **Output**: The path from the start node to the goal node is printed.

Example Output

Path from A to E: ['A', 'B', 'D', 'E']

This output represents the path found by the Best-First Search algorithm from node `A` to node `E` based on the given heuristic values.