IMPLEMENTATION OF DIFFERENT AI SEARCHING TECHNIQUES

Contents

Introduction:

Implementing different AI searching techniques involves coding algorithms to explore and traverse a search space to find a solution to a problem. Here, I'll provide simplified examples for some classic searching algorithms:

1. Breadth-First Search (BFS):

```python
from collections import deque


def bfs(graph, start, goal):
    queue = deque([(start, [start])])


    while queue:
        current, path = queue.popleft()
        if current == goal:
            return path
        for neighbor in graph[current]:
            if neighbor not in path:
                queue.append((neighbor, path + [neighbor]))


# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F', 'G'],
    'D': ['B'],
    'E': ['B', 'H'],
    'F': ['C'],
    'G': ['C'],
    'H': ['E']
}
```

```
start_node = 'A'
goal_node = 'G'

result_path = bfs(graph, start_node, goal_node)
print("BFS Result Path:", result_path)
```

2. Depth-First Search (DFS):

```python
def dfs(graph, current, goal, path=None):
    if path is None:
        path = [current]
    if current == goal:
        return path
    for neighbor in graph[current]:
        if neighbor not in path:
            result = dfs(graph, neighbor, goal, path + [neighbor])
            if result:
                return result

# Example usage:
result_path = dfs(graph, start_node, goal_node)
print("DFS Result Path:", result_path)
```

3. A* Search:

```python
import heapq


def heuristic(node, goal):
    # Define a heuristic function (e.g., Euclidean distance)
    return 0


def astar(graph, start, goal):
    priority_queue = [(0, start, [])]


    while priority_queue:
        cost, current, path = heapq.heappop(priority_queue)
        if current == goal:
            return path + [current]
        for neighbor in graph[current]:
            if neighbor not in path:
                heapq.heappush(priority_queue, (cost + heuristic(neighbor, goal), neighbor, path + [current]))


# Example usage:
result_path = astar(graph, start_node, goal_node)
print("A* Search Result Path:", result_path)
```