bfs

October 22, 2023

[ ]:
```
class Node:
    def __init__(self, state, parent, actions, totalcost):
        self.state = state
        self.parent = parent
        self.actions = actions
        self.totalcost = totalcost
```

[ ]:
```
romania = {
    "Arad": Node("Arad", None, ["Zerind", "Sibiu", "Timisoara"], None),
    "Zerind": Node("Zerind", None, ["Arad", "Oradea"], None),
    "Oradea": Node("Oradea", None, ["Zerind", "Sibiu"], None),
    "Sibiu": Node("Sibiu", None, ["Arad", "Oradea", "Fagaras", "Rimnicu␣
 ↪Vilcea"], None),
    "Timisoara": Node("Timisoara", None, ["Arad", "Lugoj"], None),
    "Lugoj": Node("Lugoj", None, ["Timisoara", "Mehadia"], None),
    "Mehadia": Node("Mehadia", None, ["Lugoj", "Drobeta"], None),
    "Drobeta": Node("Drobeta", None, ["Mehadia", "Craiova"], None),
    "Craiova": Node("Craiova", None, ["Drobeta", "Rimnicu Vilcea", "Pitesti"],␣
 ↪None),
    "Rimnicu Vilcea": Node("Rimnicu Vilcea", None, ["Sibiu", "Craiova",␣
 ↪"Pitesti"], None),
    "Fagaras": Node("Fagaras", None, ["Sibiu", "Bucharest"], None),
    "Pitesti": Node("Pitesti", None, ["Rimnicu Vilcea", "Craiova",␣
 ↪"Bucharest"], None),
    "Bucharest": Node("Bucharest", None, ["Fagaras", "Pitesti", "Giurgiu",␣
 ↪"Urziceni"], None),
    "Giurgiu": Node("Giurgiu", None, ["Bucharest"], None),
    "Urziceni": Node("Urziceni", None, ["Bucharest", "Hirsova", "Vaslui"],␣
 ↪None),
    "Hirsova": Node("Hirsova", None, ["Urziceni", "Eforie"], None),
    "Eforie": Node("Eforie", None, ["Hirsova"], None),
    "Vaslui": Node("Vaslui", None, ["Urziceni", "Iasi"], None),
    "Iasi": Node("Iasi", None, ["Vaslui", "Neamt"], None),
    "Neamt": Node("Neamt", None, ["Iasi"], None),
}
```

[ ]:
```python
def BFS(graph, start, goal):
    queue = []
    visited = []
    queue.append(start)
    visited.append(start)

    while queue:
        current = queue.pop(0)
        if current == goal:
            return actionSequence(graph, start, goal)  # Change to start here
        for neighbor in graph[current].actions:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.append(neighbor)
                graph[neighbor].parent = current


def actionSequence(graph, start, goal):
    solution = [goal]
    current = goal
    while current != start:
        currentParent = graph[current].parent
        solution.append(currentParent)
        current = currentParent
    solution.reverse()
    return solution
```

[ ]:
```python
def DFS(graph, start, goal):
    stack = []
    visited = []
    stack.append(start)
    visited.append(start)
    path = []

    while stack:
        current = stack.pop()
        path.append(current)

        if current == goal:
            return path

        for neighbor in graph[current].actions:
            if neighbor not in visited:
                stack.append(neighbor)
                visited.append(neighbor)
                graph[neighbor].parent = current
```

```
    return path
```

[ ]:
```
print(DFS(romania, "Arad", "Bucharest"))
```

```
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti',
'Bucharest']
```

[ ]:
```
print(BFS(romania, "Arad", "Bucharest"))
```

```
['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
```