lab-8

October 22, 2024

[2]:

```
graph = {
    'A': ['B', 'C'],
    'B': ['D'],
    'C': ['E'],
    'D': ['C', 'E'],
    'E': []
}

def find_path(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return path
    if start not in graph:
        return None

    for node in graph[start]:
        if node not in path:
            newpath = find_path(graph, node, end, path)
            if newpath:
                return newpath
    return None
print(find_path(graph, 'A', 'D'))
```

```
['A', 'B', 'D']
```

[4]:

```
# Directed Graph
directed_graph = {
    'A': ['B'],
    'B': ['C'],
    'C': ['D'],
    'D': ['E'],
    'E': [],
    'F': [],
    'G': []
}

# Un-directed Graph
undirected_graph = {
```

```python
    'A': ['B'],
    'B': ['A', 'C'],
    'C': ['B', 'D'],
    'D': ['C', 'E'],
    'E': ['D'],
    'F': [],
    'G': []
}

# Weighted Graph
weighted_graph = {
    'A': {'B': 1},
    'B': {'C': 2},
    'C': {'D': 3},
    'D': {'E': 4},
    'E': {},
    'F': {},
    'G': {}
}


def find_shortest_path(graph, start, end):
    queue = [(start, [start])]
    while queue:
        (node, path) = queue.pop(0)
        for neighbor in graph.get(node, {}):
            if neighbor == end:
                return path + [neighbor]
            else:
                queue.append((neighbor, path + [neighbor]))
    return None


def find_neighbors(graph, node):
    return graph.get(node, [])


def edge_exists(graph, node1, node2):
    return node2 in graph.get(node1, {})

# Test cases
print("Shortest Path (A to E):", find_shortest_path(weighted_graph, 'A', 'E'))
print("Neighbors of B:", find_neighbors(directed_graph, 'B'))
print("Edge A -> B exists?", edge_exists(weighted_graph, 'A', 'B'))
```

```
Shortest Path (A to E): ['A', 'B', 'C', 'D', 'E']
Neighbors of B: ['C']
```

```
Edge A -> B exists? True
```

[ ]:

```

```