Bahria University,
Karachi Campus

LAB EXPERIMENT NO.
03

LIST OF TASKS

TASK NO	OBJECTIVE
01	Implement BFS & DFS Algorithm in python on the given graph:
02	Implement the BFS and DFS Algorithm using recursion on the given graph with starting node = 1 and goal =6
03	Apply the UCS algorithm on a map given below. Find optimal cost from ARAD to BUCHAREST
04	Implement the Travelling Salesmen problem using uninformed searches on given Directed graph

Submitted On:

Date: 27/02/2024

Task No 01: Implement BFS & DFS Algorithm in python on the given graph:

Solution:

```
def bfs(graph, start):
    visited = set()
    queue = [start]
    bfs_traversal = []
    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            bfs_traversal.append(vertex)
            visited.add(vertex)
            queue.extend(graph[vertex])
    return bfs_traversal
def dfs(graph, start):
    visited = set()
    stack = [start]
    dfs_traversal = []
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            dfs_traversal.append(vertex)
            visited.add(vertex)
            stack.extend(reversed(graph[vertex]))
    return dfs_traversal
graph = {
    'A': ['B', 'C','D'],
    'B': ['E', 'F'],
    'C': ['F'],
    'D': [],
    'E': [],
    'F': []
}
start_node = 'A'
bfs_result = bfs(graph, start_node)
dfs_result = dfs(graph, start_node)
print("BFS Traversal:", bfs_result)
print("DFS Traversal:", dfs_result)
```

Output:

```
BFS Traversal: ['A', 'B', 'C', 'D', 'E', 'F']
DFS Traversal: ['A', 'B', 'E', 'F', 'C', 'D']
```

Task No 02: Implement the BFS and DFS Algorithm using recursion on the given graph with starting node = 1 and goal =6

Solution:

```
def bfs(graph, start, goal, queue=None, visited=None):
    if queue is None:
```

```
        queue = [start]
    if visited is None:
        visited = set()
    if not queue:
        return []
    vertex = queue.pop(0)
    visited.add(vertex)
    if vertex == goal:
        return [vertex]
    for neighbor in graph[vertex]:
        if neighbor not in visited and neighbor not in queue:
            queue.append(neighbor)
    return [vertex] + bfs(graph, start, goal, queue, visited)
def dfs(graph, current, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(current)
    if current == goal:
        return [current]
    for neighbor in graph[current]:
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [current] + path
    return []
graph = {
    1: [2, 3,4],
    2: [5,6],
    3: [6],
    4: [7,8],
    5: [9,10],
    6: [],
    7:[11,10],
    8:[],
    9:[],
    10:[],
    11:[],
}
start_node = 1
goal_node = 6
bfs_result = bfs(graph, start_node, goal_node)
dfs_result = dfs(graph, start_node, goal_node)
print("BFS Path:", bfs_result)
print("DFS Path:", dfs_result)
```

Output:

```
BFS Path: [1, 2, 3, 4, 5, 6]
DFS Path: [1, 2, 6]
```

Task No 03: Apply the UCS algorithm on a map given below. Find optimal cost from ARAD to BUCHAREST

Solution:

```
import heapq
graph = {
    'Arad': [('Zerind', 75), ('Sibiu', 140), ('Timisoara', 118)],
    'Zerind': [('Arad', 75), ('Oradea', 71)],
    'Oradea': [('Zerind', 71), ('Sibiu', 151)],
    'Sibiu': [('Arad', 140), ('Oradea', 151), ('Fagaras', 99), ('Rimnicu
Vilcea', 80)],
    'Timisoara': [('Arad', 118), ('Lugoj', 111)],
    'Lugoj': [('Timisoara', 111), ('Mehadia', 70)],
    'Mehadia': [('Lugoj', 70), ('Drobeta', 75)],
    'Drobeta': [('Mehadia', 75), ('Craiova', 120)],
    'Craiova': [('Drobeta', 120), ('Rimnicu Vilcea', 146), ('Pitesti',
138)],
    'Rimnicu Vilcea': [('Sibiu', 80), ('Craiova', 146), ('Pitesti', 97)],
    'Fagaras': [('Sibiu', 99), ('Bucharest', 211)],
    'Pitesti': [('Rimnicu Vilcea', 97), ('Craiova', 138), ('Bucharest',
101)],
    'Bucharest': [('Fagaras', 211), ('Pitesti', 101), ('Giurgiu', 90),
('Urziceni', 85)],
    'Giurgiu': [('Bucharest', 90)],
    'Urziceni': [('Bucharest', 85), ('Hirsova', 98), ('Vaslui', 142)],
    'Hirsova': [('Urziceni', 98), ('Eforie', 86)],
    'Eforie': [('Hirsova', 86)],
    'Vaslui': [('Urziceni', 142), ('Iasi', 92)],
    'Iasi': [('Vaslui', 92), ('Neamt', 87)],
    'Neamt': [('Iasi', 87)]
}
def ucs(graph, start, goal):
    frontier = [(0, start)]
    explored = {start: 0}
    while frontier:
        current_cost, current_node = heapq.heappop(frontier)
        if current_node == goal:
            return explored[current_node]
        for neighbor, cost in graph[current_node]:
            total_cost = current_cost + cost
            if neighbor not in explored or total_cost <
explored[neighbor]:
```

```
                explored[neighbor] = total_cost
                heapq.heappush(frontier, (total_cost, neighbor))
start_city = 'Arad'
goal_city = 'Bucharest'
optimal_cost = ucs(graph, start_city, goal_city)
print("Optimal cost from", start_city, "to", goal_city, ":", optimal_cost)
```

Output:

```
Optimal cost from Arad to Bucharest : 418
```

Task No 04: Implement the Travelling Salesmen problem using uninformed searches on given Directed graph

Solution:

```
graph = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
def dfs_tsp(graph, start, visited, path_length):
    if len(visited) == len(graph):
        return path_length + graph[start][0]
    min_cost = float('inf')
    for city in range(len(graph)):
        if city not in visited and city != start:
            visited.add(city)
            current_cost = dfs_tsp(graph, city, visited, path_length +
graph[start][city])
            min_cost = min(min_cost, current_cost)
            visited.remove(city)
    return min_cost
def solve_tsp_dfs(graph):
    start_city = 0
    visited = {start_city}
    path_length = 0
    optimal_cost = dfs_tsp(graph, start_city, visited, path_length)
    return optimal_cost
optimal_cost = solve_tsp_dfs(graph)
print("Optimal cost for TSP using DFS:", optimal_cost)
```

Output:

```
Optimal cost for TSP using DFS: 80
```