23rd February 2023

Artificial Intelligence

Assignment

# 1

Submitted by:

Syed Ali Raza

200901028

Submitted to:

Ma’am Reeda Saeed

**Code**

graph = {

"Arad": {"Zerind": 75, "Timisoara": 118, "Sibiu": 140},

"Oradea": {"Zerind": 71, "Sibiu": 151},

"Zerind": {"Oradea": 71, "Arad": 75},

"Timisoara": {"Arad": 118, "Lugoj": 111},

"Lugoj": {"Timisoara": 118, "Mehadia": 70},

"Mehadia": {"Lugoj": 70, "Dobreta": 75},

"Dobreta": {"Mehadia": 75, "Craiova": 120},

"Craiova": {"Rimnicu Vilcea": 146, "Pitesti": 138},

"Rimnicu Vilcea": {"Craiova": 146, "Pitesti": 97, "Sibiu": 80},

"Pitesti": {"Rimnicu Vilcea": 97, "Craiova": 138, "Bucharest": 101},

"Sibiu": {"Arad": 140, "Oradea": 151, "Fagarus": 99, "Rimnicu Vilcea": 80},

"Fagarus": {"Sibiu": 99, "Bucharest": 211},

"Bucharest": {"Pitesti": 97, "Urziceni": 85, "Giurgui": 90, "Fagarus": 211},

"Giurgui": {"Bucharest": 90},

"Urziceni": {"Hirsova": 98, "Bucharest": 85, "Vaslui": 142},

"Hirsova": {"Urziceni": 98, "Eforie": 86},

"Vaslui": {"Iasi": 92},

"Iasi": {"Vaslui": 92, "Neamt": 87},

"Neamt": {"Iasi": 87},

"Eforie": {"Hirsova": 86}

}

def depth\_first\_search(graph, start, goal, visited = None):

if visited is None:

visited = set()

visited.add(start)

if start == goal:

return [start], 0

shortest\_path = None

shortest\_distance = float("inf")

for neighbor in graph[start]:

if neighbor not in visited:

path, distance = depth\_first\_search(graph, neighbor, goal, visited)

if path:

if distance + graph[start][neighbor] < shortest\_distance:

shortest\_path = [start] + path

shortest\_distance = distance + graph[start][neighbor]

return shortest\_path, shortest\_distance

def breadth\_first\_search(graph, start, goal):

queue = [(start, [start], 0)]

shortest\_path = None

shortest\_distance = float("inf")

while queue:

node, path, distance = queue.pop(0)

if node == goal:

if distance < shortest\_distance:

shortest\_distance = distance

shortest\_path = path

for neighbor in graph[node]:

if neighbor not in path:

queue.append((neighbor, path + [neighbor], distance + graph[node][neighbor]))

return shortest\_path, shortest\_distance

source = "Arad"

goal = "Bucharest"

print("DFS", depth\_first\_search(graph, source, goal))

print("BFS", breadth\_first\_search(graph, source, goal))

if (DFS\_result > BFS\_result):

print("DFS gives the most optimised result")

else:

print("BFS gives the most optimised result")

**Output:**

****

**Github Repository:**