Name : Abdul Rehman Section :B

Course : AI&ES Roll No : CT-22052  
---------------------------------------------------------------------------------------------------------------------  
 **Lab # 4**

**Task 1:**graph1={

'A':['B','E','C'],

'B':['D','E'],

'C':[],

'D':[],

'E':[]

}

def dfs(graph, node, visited):

if node not in visited:

visited. append (node)

for n in graph[node]:

dfs(graph,n, visited)

return visited

visited = dfs(graph1, 'A', [])

print(visited)  
  
**Output:** **Task 2:**graph = {

'A': set(['B', 'E', 'C']),

'B': set(['D', 'E']),

'C': set([]),

'D': set([]),

'E': set([])

}

def find\_path(graph, start, end, path=None):

if path is None:

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', 'E'))  
  
**Output:** **Task 3:**graph = {

'A': set(['B', 'E', 'C']),

'B': set(['D', 'E']),

'C': set([]),

'D': set([]),

'E': set([])

}

def find\_shortest\_path(graph, start, end, path=None):

if path is None:

path = []

path = path + [start]

if start == end:

return path

if start not in graph:

return None

shortest = None

for node in graph[start]:

if node not in path:

newpath = find\_shortest\_path(graph, node, end, path)

if newpath:

if not shortest or len(newpath) < len(shortest):

shortest = newpath

return shortest

print(find\_shortest\_path(graph, 'A', 'E'))

def all\_paths\_dfs(graph, start, goal, visited=None, path=None):

if visited is None:

visited = set()

if path is None:

path = []

visited.add(start)

path = path + [start]

if start == goal:

yield path

else:

for next\_vertex in graph[start] - visited:

yield from all\_paths\_dfs(graph, next\_vertex, goal, visited, path)

visited.remove(start)

graph = {

'A': set(['B', 'C']),

'B': set(['A', 'C', 'D']),

'C': set(['A', 'B', 'D']),

'D': set(['B', 'C', 'E']),

'E': set(['D', 'F']),

'F': set(['E', 'G']),

'G': set(['F', 'H']),

'H': set(['G', 'I']),

'I': set(['H', 'J']),

'J': set(['I', 'K']),

'K': set(['J'])

}

print(list(all\_paths\_dfs(graph, 'A', 'D')))  
  
  
**Output:**

**Exercise:**  
**Question # 1:**

graph = {

'1': set(['2', '4','3']),

'2': set(['1','3','4']),

'3': set(['1','2','4']),

'4': set(['1', '2','3','5']),

'5': set(['4', '6', '7','8']),

'6': set(['5','7','8']),

'7': set(['5', '6', '8']),

'8': set(['5', '6', '7']),

}

def dfs(graph, node, visited):

if node not in visited:

visited. append (node)

for n in graph[node]:

dfs(graph,n, visited)

return visited

def find\_path(graph, start, end, path=None):

if path is None:

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

def find\_shortest\_path(graph, start, end, path=None):

if path is None:

path = []

path = path + [start]

if start == end:

return path

if start not in graph:

return None

shortest = None

for node in graph[start]:

if node not in path:

newpath = find\_shortest\_path(graph, node, end, path)

if newpath:

if not shortest or len(newpath) < len(shortest):

shortest = newpath

return shortest

def all\_paths\_dfs(graph, start, goal, visited=None, path=None):

if visited is None:

visited = set()

if path is None:

path = []

visited.add(start)

path = path + [start]

if start == goal:

yield path

else:

for next\_vertex in graph[start] - visited:

yield from all\_paths\_dfs(graph, next\_vertex, goal, visited, path)

visited.remove(start)

# Part a

visited\_task1 = dfs(graph, '1', [])

print("Part a")

print("Traversal path using DFS:", visited\_task1)

print('')

# Part b

single\_path\_task2 = find\_path(graph, '1', '6')

print("Part b")

print("Single path between 1 & 6:", single\_path\_task2)

print('')

# Part c

all\_paths\_task3 = list(all\_paths\_dfs(graph, '1', '6')) # Convert generator to list

print("Part c")

print("All paths between 1 & 6:", all\_paths\_task3)

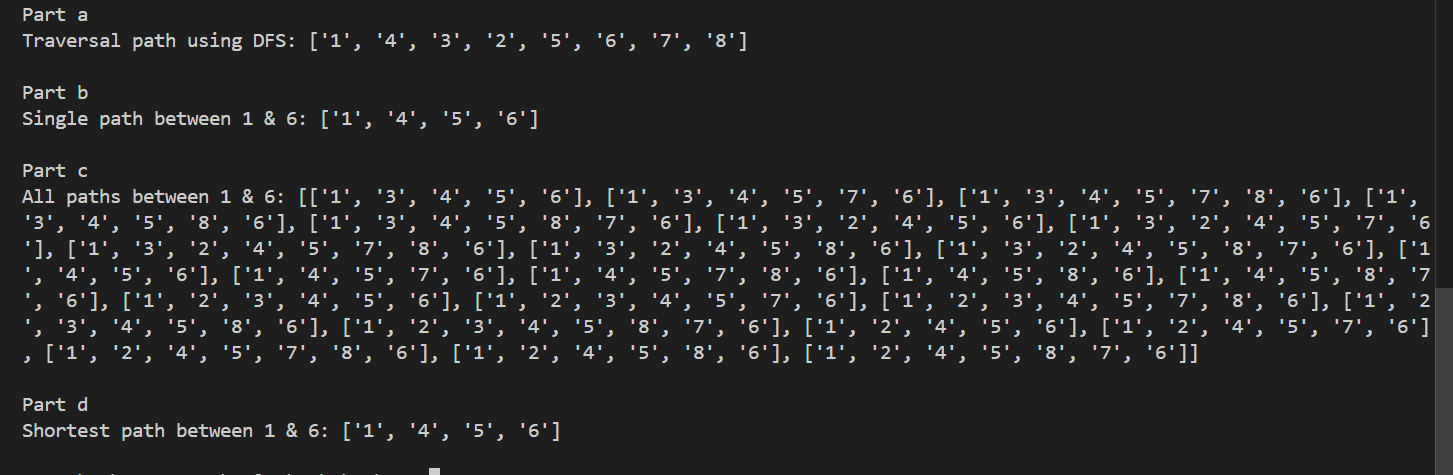
print('')

# Part d

shortest\_path\_task4 = find\_shortest\_path(graph, '1', '6')

print("Part d")

print("Shortest path between 1 & 6:", shortest\_path\_task4)

print('')  
  
  
**Output:** **Question # 2:**

graph\_2 = {

'A': set(['B', 'C','D']),

'B': set(['A','E']),

'C': set(['A','F']),

'D': set(['A', 'E','G']),

'E': set(['B', 'D', 'G']),

'F': set(['C','G']),

'G': set(['D', 'E', 'F'])

}

def dfs(graph, node, visited):

if node not in visited:

visited. append (node)

for n in graph[node]:

dfs(graph,n, visited)

return visited

def find\_path(graph, start, end, path=None):

if path is None:

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

def find\_shortest\_path(graph, start, end, path=None):

if path is None:

path = []

path = path + [start]

if start == end:

return path

if start not in graph:

return None

shortest = None

for node in graph[start]:

if node not in path:

newpath = find\_shortest\_path(graph, node, end, path)

if newpath:

if not shortest or len(newpath) < len(shortest):

shortest = newpath

return shortest

def all\_paths\_dfs(graph, start, goal, visited=None, path=None):

if visited is None:

visited = set()

if path is None:

path = []

visited.add(start)

path = path + [start]

if start == goal:

yield path

else:

for next\_vertex in graph[start] - visited:

yield from all\_paths\_dfs(graph, next\_vertex, goal, visited, path)

visited.remove(start)

# Part a

visited\_task1 = dfs(graph\_2, 'A', [])

print("Part a")

print("Traversal path using DFS:", visited\_task1)

print('')

# Part b

single\_path\_task2 = find\_path(graph\_2, 'A', 'G')

print("Part b")

print("Single path between A & G:", single\_path\_task2)

print('')

# Part c

all\_paths\_task3 = list(all\_paths\_dfs(graph\_2, 'A', 'G')) # Convert generator to list

print("Part c")

print("All paths between A & G:", all\_paths\_task3)

print('')

# Part d

shortest\_path\_task4 = find\_shortest\_path(graph\_2, 'A', 'G')

print("Part d")

print("Shortest path between A & G:", shortest\_path\_task4)

print('')  
  
  
**Output:**