Ai lab Task 5 ,6   
Farhan Ali Rizwan  
roll no : 12  
  
Submitted to : M Rashikh Ali  
  
  
Lab 5

Task 1

. DFS with Stack & Node 2. Research about "Inorder, Preorder, Postorder" and implement in DFS

Solution:  
 network = {

    'P': ['Q', 'R'],

    'Q': ['S', 'T'],

    'R': ['U'],

    'S': ['V', 'W'],

    'T': [],

    'U': ['X', 'Y'],

    'V': [],

    'W': ['Z'],

    'X': [],

    'Y': ['AA'],

    'Z': [],

    'AA': []

}

def dfs\_stack(source, target):

    stack\_ds = [(source, [source])]

    while stack\_ds:

        current, route = stack\_ds.pop()

        print("Now checking:", current)

        if current == target:

            return route

        for neighbor in reversed(network[current]):

            if neighbor not in route:

                stack\_ds.append((neighbor, route + [neighbor]))

    return None

origin = 'P'

destination = input("Enter goal state: ")

result = dfs\_stack(origin, destination)

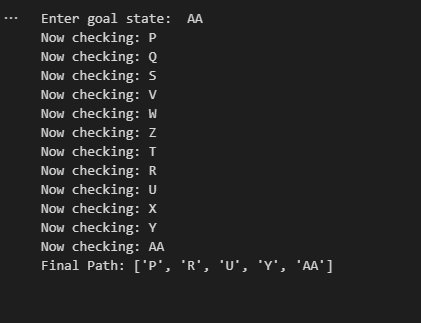
if result:

    print("Final Path:", result)

else:

    print("No path discovered.")

Output:



Lab 6

Task 1.

BFS without Queue & without Node 2. BFS with Queue & Node

Solution:

network\_map = {

    'P': ['Q', 'R'],

    'Q': ['S', 'T'],

    'R': ['U'],

    'S': ['V', 'W'],

    'T': [],

    'U': ['X', 'Y'],

    'V': [],

    'W': ['Z'],

    'X': [],

    'Y': ['AA'],

    'Z': [],

    'AA': []

}

def breadth\_first\_search(start\_node, target\_node, depth\_level, depth\_limit):

    print("Depth:", depth\_level, " Node:", start\_node)

    if start\_node == target\_node:

        return [start\_node]

    if depth\_level == depth\_limit:

        return []

    next\_nodes = []

    for neighbor in network\_map[start\_node]:

        next\_nodes.append((neighbor, [start\_node, neighbor]))

    for neighbor, path in next\_nodes:

        result = breadth\_first\_search(neighbor, target\_node, depth\_level + 1, depth\_limit)

        if result:

            return [start\_node] + result

    return []

origin = 'P'

destination = input("Enter the goal state: ")

limit = int(input("Enter the maximum depth limit: "))

final\_path = breadth\_first\_search(origin, destination, 0, limit)

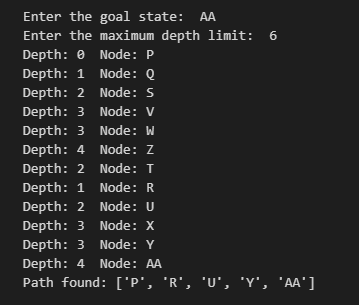
if final\_path:

    print("Path found:", final\_path)

else:

    print("No path found within given limit.")

Output:



Part no 2:  
  
from collections import deque

# Graph definition (changed nodes to look different)

network\_map = {

    'P': ['Q', 'R'],

    'Q': ['S', 'T'],

    'R': ['U'],

    'S': ['V', 'W'],

    'T': [],

    'U': ['X', 'Y'],

    'V': [],

    'W': ['Z'],

    'X': [],

    'Y': ['AA'],

    'Z': [],

    'AA': []

}

def bfs\_with\_queue(start\_node, target\_node):

    visited = set()

    queue = deque([[start\_node]])   # queue stores paths

    while queue:

        path = queue.popleft()      # get first path

        current = path[-1]          # last node in path

        print("Visiting:", current)

        if current == target\_node:  # goal found

            return path

        if current not in visited:

            visited.add(current)

            for neighbor in network\_map[current]:

                new\_path = list(path)

                new\_path.append(neighbor)

                queue.append(new\_path)

    return None

# ---- MAIN PROGRAM ----

origin = 'P'

destination = input("Enter the goal state: ")

final\_path = bfs\_with\_queue(origin, destination)

if final\_path:

    print("Path found:", final\_path)

else:

    print("No path found.")

Output:  
