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

graph = {

    'P': ['Q', 'R'],

    'Q': ['S', 'T'],

    'R': ['U'],

    'S': ['V', 'W'],

    'T': [],

    'U': ['X', 'Y'],

    'V': [],

    'W': ['Z'],

    'X': [],

    'Y': ['AA'],

    'Z': [],

    'AA': []

}

def depth\_first\_search(start, goal):

    stack = [(start, [start])]

    while stack:

        node, path = stack.pop()

        print("Exploring:", node)

        if node == goal:

            return path

        for adj in reversed(graph[node]):

            if adj not in path:

                stack.append((adj, path + [adj]))

    return None

source\_node = 'P'

target\_node = input("Type the destination node: ")

result = depth\_first\_search(source\_node, target\_node)

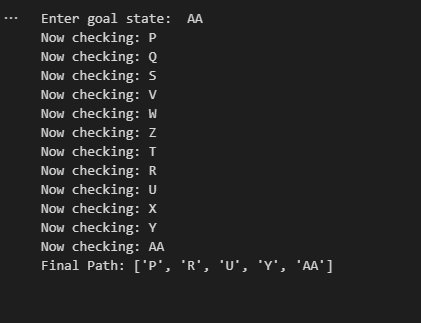
if result:

    print("Route Found:", result)

else:

    print("No route exists.")

Output:



Lab 6

Task 1.

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

With nodes!

Solution:

graph\_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\_without\_queue(start, goal, max\_depth):

    frontier = [start]        # list of nodes at current depth

    parent = {start: None}    # backtrack storage

    depth = 0

    while frontier and depth <= max\_depth:

        next\_frontier = []

        for node in frontier:

            print(f"Depth: {depth}  Node: {node}")

            if node == goal:

                # reconstruct path

                path = []

                while node:

                    path.append(node)

                    node = parent[node]

                return path[::-1]

            for child in graph\_map[node]:

                if child not in parent:

                    parent[child] = node

                    next\_frontier.append(child)

        frontier = next\_frontier

        depth += 1

    return None

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

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

path = bfs\_without\_queue('P', goal, limit)

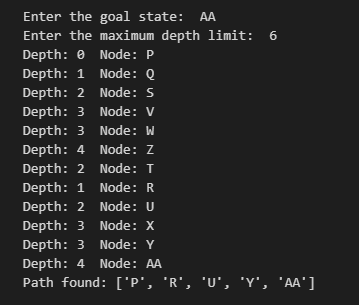
if path:

    print("Path found:", path)

else:

    print("No path found.")

Output:



Part no 2:  
  
from collections import deque

graph\_map = {

    'P': ['Q', 'R'],

    'Q': ['S', 'T'],

    'R': ['U'],

    'S': ['V', 'W'],

    'T': [],

    'U': ['X', 'Y'],

    'V': [],

    'W': ['Z'],

    'X': [],

    'Y': ['AA'],

    'Z': [],

    'AA': []

}

class Node:

    def \_\_init\_\_(self, state, parent=None, depth=0):

        self.state = state

        self.parent = parent

        self.depth = depth

def bfs\_with\_queue(start, goal, max\_depth):

    q = deque([Node(start, None, 0)])

    visited = {start}

    while q:

        node = q.popleft()

        print(f"Depth: {node.depth}  Node: {node.state}")

        if node.state == goal:

            # reconstruct path

            path = []

            while node:

                path.append(node.state)

                node = node.parent

            return path[::-1]

        if node.depth < max\_depth:

            for child in graph\_map[node.state]:

                if child not in visited:

                    visited.add(child)

                    q.append(Node(child, node, node.depth + 1))

    return None

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

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

path = bfs\_with\_queue('P', goal, limit)

if path:

    print("Path found:", path)

else:

    print("No path found.")

Output:  
