SSN College of Engineering, Kalavakkam

Department of Computer Science and Engineering

IV Semester – B.E (CSE)

R2024

UCS3461 - Foundations of Artificial Intelligence (TCP-Lab)

Academic Year: 2025-2026 Even **Batch: 2024-2028**

Exercise 1a: Uninformed Search Strategy – Depth First Search

Implement Depth First Search (DFS) for the Water Jug problem based on the following specifications. (CO2, K3, PIs: 1.4.1,2.1.3,3.1.1)

Representing the Search Problem

A search problem consists of the following:

- a start state
- a neighbors function: given a state, returns all possible successor states
- a goal state: a Boolean function that returns true if a state satisfies the goal condition

A state can be represented in any suitable form. In this problem, represent a state as a pair *(x, y)* where *x* is the amount of water in Jug A and *y* is the amount of water in Jug B.

Define a class ***SearchProblem*** with the following methods:

- start_state()
- is_goal(state)
- neighbors(state)

Edge Representation

The neighbors function should return a list of edges. An edge represents an action that transforms one state into another and consists of the following:

- from_state
- to_state
- an action label
- a non-negative cost (default cost = 1)

Implement a class Edge and define a suitable *__repr__()* method to display the edge.

Water Jug Problem Specification

Consider two water jugs:

- Jug A with capacity 4 litres
- Jug B with capacity 3 litres

Initial state: (0, 0)

Goal: Obtain exactly 2 litres of water in Jug A

At any state, the permissible actions are restricted to filling either Jug A or Jug B to their full capacities, emptying either jug completely, or pouring water from one jug to the other until the source jug is empty or the destination jug becomes full. These constraints ensure that only valid and physically feasible state transitions are generated.

Define a class **WaterJugProblem** that:

- stores jug capacities and the start state
- implements the methods defined in SearchProblem
- generates all valid successor states using the allowed actions
- includes a suitable *__repr__()* method to describe the problem

Path Representation

A search algorithm returns a path from the start state to a goal state. Represent a path using a recursive structure that can share subparts:

- a single state (path of length 0), or
- a path followed by an edge, where the from_state of the edge matches the end state of the path

Implement a class **Path** with the following methods:

- a method end() that returns the final state of the path
- a suitable __repr__() method to display the path

Implement the Depth First Search (DFS) algorithm that uses a stack for frontier management, avoids revisiting already explored states and returns a valid path from the start state to a goal state, if one exists.

Print the final solution path obtained using DFS and also the actions that made the transitions.

Additional Question:

Find all the possible solutions for the above given problem.
Implement Breadth First Search (BFS) strategy for the above given problem.