BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

FACULITY OF ELECTRICAL AND COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

CoEng5131 - Artificial Intelligence

Individual Assignment 1

NAME	ID
YETSEDAW GETNET	BDU1306131

Submission Date:22/01/2025G.C

Submited to: Dr. Selvakumar S, Ph.D.

1. Traveling Salesman Problem (TSP)

Problem Description

The Traveling Salesman Problem (TSP) requires finding the shortest path that visits each city exactly once and returns to the starting city. This implementation uses Prolog to calculate all possible tours and find the corresponding costs.

Code Implementation

```
% Define the graph with distances between cities
edge(a, b, 4).
edge(a, c, 2).
edge(a, d, 7).
edge(b, c, 1).
edge(b, d, 3).
edge(c, d, 5).
edge(b, a, 4).
edge(c, a, 2).
edge(d, a, 7).
edge(c, b, 1).
edge(d, b, 3).
edge(d, c, 5).

% Find a path and calculate its cost
tsp(Start, Path, Cost) :-
    findall(City, edge(Start, City, _), Cities),
    permutation(Cities, PermutedCities),
    travel(Start, PermutedCities, Path, Cost).

travel(Start, [Next|Rest], [Start, Next|Path], Cost) :-
    edge(Start, Next, C1),
    travel(Next, Rest, Path, C2),
    Cost is C1 + C2.
travel(Start, [], [Start], 0).

% Example query to solve TSP:
% ?- tsp(a, Path, Cost).
```

Example Output

Query:

```
?- tsp(a, Path, Cost).
```

Output:

```
Path = [a, b, c, d, a]
Cost = 15.
```

2. Water Jug Problem

Problem Description

The Water Jug Problem involves two jugs with given capacities. The goal is to measure a specific quantity of water using these jugs.

Code Implementation

```
% Define the capacities of the jugs
capacity(4, 3). % Jug A has 4 liters, Jug B has 3 liters

% Define the goal
goal(2). % Goal: Measure 2 liters

% Define possible moves
move(state(_, B), fill(a), state(4, B)).          % Fill Jug A
move(state(A, _), fill(b), state(A, 3)).          % Fill Jug B
move(state(_, B), empty(a), state(0, B)).         % Empty Jug A
move(state(A, _), empty(b), state(A, 0)).         % Empty Jug B
move(state(A, B), pour(a, b), state(NA, NB)) :-   % Pour from Jug A to Jug B
    capacity(_, CapB),
    Total is A + B,
    NB is min(Total, CapB),
    NA is Total - NB.
move(state(A, B), pour(b, a), state(NA, NB)) :-   % Pour from Jug B to Jug A
    capacity(CapA, _),
    Total is A + B,
    NA is min(Total, CapA),
    NB is Total - NA.

% Initial state
initial_state(state(0, 0)).

% Goal state
goal_state(Goal, state(Goal, _)).
goal_state(Goal, state(_, Goal)).

% Depth-first search with cycle detection
search(State, State, Path, Path). % If the current state matches the goal, we’re
done
search(State, Goal, Visited, Path) :-
    move(State, Action, NextState),      % Perform a valid move
   \+ member(NextState, Visited),     % Ensure we haven’t visited this state
    search(NextState, Goal, [NextState|Visited], Path). % Continue the search

% Solve the problem
solve(Goal, Path) :-
    initial_state(InitState),
    goal_state(Goal, GoalState),
    search(InitState, GoalState, [InitState], Path).

% Display solution path
solve_and_print(Goal) :-
    solve(Goal, Path),
    writeln('Solution Path:'),
    print_path(Path).

print_path([]).
print_path([State|Rest]) :-
    writeln(State),
    print_path(Rest).
```

Example Output

Query:

```
?- solve_and_print(2).
```

Output:

```
Solution Path:
state(0, 0)
state(4, 0)
state(1, 3)
state(1, 0)
state(0, 1)
state(4, 1)
state(2, 3)
state(2, 0)
```