AIM: Implementation of Logic programming using PROLOG-DFS for water jug problem.

THEORY:

This is the jug problem using simple depth-first search of a graph. The modified water-jug problem is as follows: Jug A holds 4 liters, and jug B holds 3 liters. There is a pump, which can be used to fill either Jug. How can you get exactly 2 liters of water into the 4-liter jug?

ASSUMPTIONS:

• We can fill a jug from the pump

• We can pour water out of the jug onto the ground

• We can pour water from one jug to another

• There are no other measuring devices available

• To solve the water jug problem, apart from problem statement we also need a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to state is made as described in corresponding right side and the resulting state is checked to see if it corresponds to a goal state. As long as it does not the cycle continues.

SOURCE CODE:

```
move(jugs(J1,J2), jugs(0,J2)) :- J1 > 0. % 3. empty large
move(jugs(J1,J2), jugs(J1,0)) :- J2 > 0. % 4. empty small
move(jugs(J1,J2), jugs(4,N)) :- % 5. fill large from small until large is full
J2 > 0,
J1 + J2 >= 4, % small must contain enough liquidto fill up large
N is J2 - (4 - J1).
move(jugs(J1,J2), jugs(N,3)) :- % 6. fill small from large until small is full
J1 > 0,
J1 + J2 >= 3, % large must contain enough liquid to fill up small
N is J1 - (3 - J2).
move(jugs(J1,J2), jugs(N,0)) :- % 7. empty small into large
J2 > 0,
J1 + J2 < 4, % all liquid in small must fit in large
N is J1 + J2.
move(jugs(J1,J2), jugs(0,N)) :- % 8. empty large into small
J1 > 0,
J1 + J2 < 3, % all liquid in large must fit in small
N is J1 + J2.
dfs_no_cycles_deep(State, Goal, Path) :-
dfs_deep_help([State], Goal, RevPath, 1), % start with maximum depth 1
reverse(RevPath, Path).
dfs_deep_help(PathSoFar, Goal, Path, DepthBound) :- % helper to try increasingly larger depths
dfs_bounded(PathSoFar, Goal, Path, DepthBound). % until a solution is found
dfs_deep_help(PathSoFar, Goal, Path, DepthBound) :-
NewDepth is DepthBound + 1, % add 1 to the maximum depth allowed
   NewDepth < 40, % uncomment to set an absolute limit on depth
dfs_deep_help(PathSoFar, Goal, Path, NewDepth). % try again with the increased maximum depth
dfs_bounded([Goal|PathSoFar], Goal, [Goal|PathSoFar], 1).
```

```
dfs_bounded([State|PathSoFar], Goal, Path, DepthBound) :-
DepthBound > 0, % check so maximum depth is not exceeded
move(State, NextState),not(member(NextState, [State|PathSoFar])), % check against cycles in our traversal
NewDepth is DepthBound - 1,
dfs_bounded([NextState,State|PathSoFar], Goal, Path, NewDepth).
% dfs_no_cycles_deep(jugs(0,0), jugs(2,0), Path).
% this works: 6 transitions from the initial state
% dfs_no_cycles_deep(jugs(0,0), jugs(3,0), Path).
% this works: 2 transitions from the initial state
% dfs_no_cycles_deep(jugs(0,0), jugs(2,2), Path).
% this has no solution
```

OUTPUT:

```
% c:/Users/NARENDER KESWANI/Documents/Prolog/water_jug_prob.pl compiled 0.00 sec, 13 clauses
?-
|    dfs_no_cycles_deep(jugs(0,0), jugs(2,0), Path).
Path = [jugs(0, 0), jugs(0, 3), jugs(3, 0), jugs(3, 3), jugs(4, 2), jugs(0, 2), jugs(2, 0)]
```

CONCLUSION:

From this practical, I have learned about Prolog and implantation of water jug problem.