Techno Main Salt Lake
EM-4/1, Sector V, Bidhannagar,
Kolkata, West Bengal 700091
Department: CSE (AI & ML)
Year: 2nd Semester: 4th

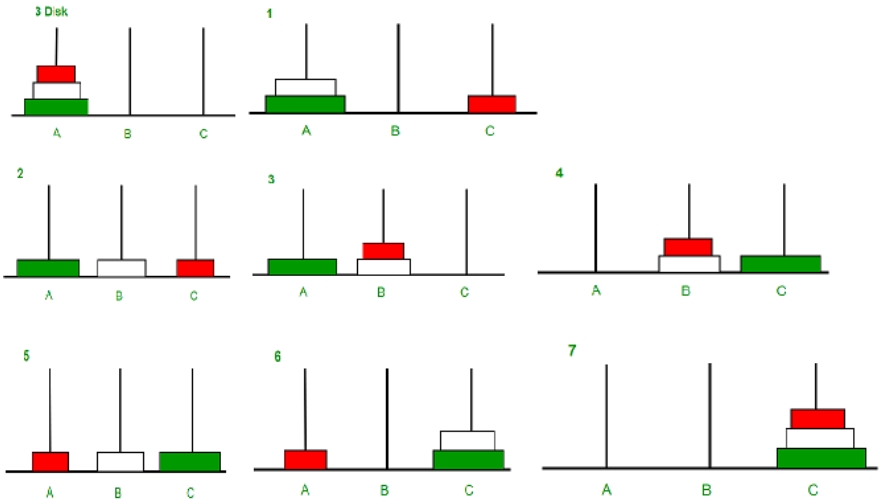
Artificial Intelligence Lab (PCCAIML491)

Assignment 4

1. Write a Prolog program to find a solution of Tower of Hanoi problem. Towers of Hanoi Problem is a famous puzzle to move N disks from the source peg/tower to the target peg/tower using the intermediate peg as an auxiliary holding peg. There are two conditions that are to be followed while solving this problem:

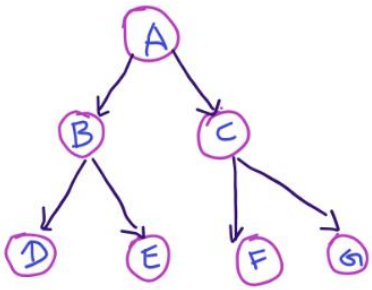
   a) A larger disk cannot be placed on a smaller disk.

   b) Only one disk can be moved at a time.



Sample Test Case:
?- move(3,source,destination,auxillary).
Move top disk from source to destination
Move top disk from source to auxillary
Move top disk from destination to auxillary
Move top disk from source to destination
Move top disk from auxillary to source
Move top disk from auxillary to destination
Move top disk from source to destination
True

2. Write a Prolog program to implement DFS on the following graph considering F and G as the goal nodes.



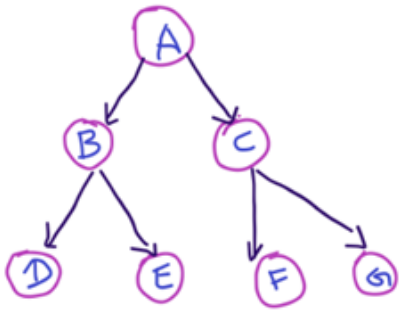
Solution:

```
%Facts
graph(a,b).
graph(a,c).
graph(b,d).
graph(b,e).
graph(c,f).
graph(c,g).

%defining facts for goals
goal(f).
goal(g).

%membership check
member(X,[X|_]).
member(X,[_|Tail]):-member(X,Tail).

%predicate to call from console
%Solution is a path (in reverse order)
%from Start to a goal.
solve(Node,Solution):-dfs([],Node,Solution).

%dfs predicate implementation
dfs(Path,Node,[Node|Path]):-goal(Node).

dfs(Path,Node,Sol):-
  graph(Node,Node1),
  not(member(Node1,Path)),
  dfs([Node|Path],Node1,Sol).
```

Sample Test Case:

```
[1]  ?- solve(a,Sol).
Sol = [f, c, a] ;
Sol = [g, c, a] ;
false.
```

3. Write a Prolog program to implement BFS on the following graph considering F and G as the goal nodes.



Techno Main Salt Lake
EM-4/1, Sector V, Bidhannagar,
Kolkata, West Bengal 700091
Department: CSE (AI & ML)
Year: 2nd Semester: 4th

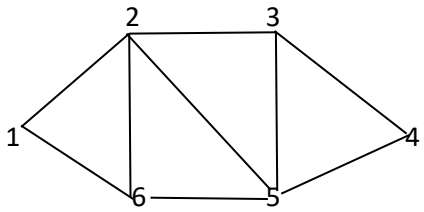
Solution:

```
%Facts
graph(a,b).
graph(a,c).
graph(b,d).
graph(b,e).
graph(c,f).
graph(c,g).

%defining facts for goals
goal(f).
goal(g).

%predicate to call from console
%Solution is a path (in reverse order)
%from Start to a goal.
solve1(Start,Solution):-
   bfs([[Start]],Solution).

%bfs predicate implementation
%Always first check if goal reached
bfs([[Node|Path]|_],[Node|Path]):-
   goal(Node).

% If not, then extend this path by all
% possible edges, put these new paths on the
% end of the queue (Paths1) to check, and do
% breadthfirst on this new collection of
% paths, Paths1:
bfs([Path|Paths],Solution):-
   extend(Path,NewPaths),conc(Paths,NewPaths,Paths1),bfs(Paths1,Solution).

% extend([N|Rest], NewPaths).
% Extend the path [N|Rest] using all edges
% through N that are not already on the path.
% This produces a list NewPaths.
%The built-in predicate bagof(+Template, +Goal, -Bag) is used to collect a list Bag of all the items
Template that satisfy some goal Goal.
%Exclamation point ! denotes Cut in Prolog, a special goal that always succeeds, and
%blocks backtracking for all branches above it that may have alternatives.
extend([Node|Path], NewPaths) :-
 bagof([NewNode, Node|Path],
       (graph(Node, NewNode),
        not(member(NewNode,[Node|Path]))),
       NewPaths),
 !.

%findall failed: no edge
extend(_,[]).
```

%concatenating paths
conc([],L2,L2).
conc([X|L1],L2,[X|L3]):-conc(L1,L2,L3).

Sample Test Case:
?- solve1(a,Solution).
Solution = [f, c, a] ;
Solution = [g, c, a] ;
false.

4. Write a Prolog program to find all possible paths from a given node to another for the following graph.



Solution:

%facts
edge(1,2).
edge(1,6).
edge(2,3).
edge(2,5).
edge(2,6).
edge(3,4).
edge(3,5).
edge(4,5).
edge(5,6).

%since no direction i.e. bidirectional given
connected(X,Y):-edge(X,Y);edge(Y,X).

%predicate needs to be called from console
%reverse inbuilt predicate
path(A,B,Path):-traverse(A,B,[A],Q),reverse(Q,Path).

traverse(A,B,P,[B|P]):-connected(A,B).

traverse(A,B,Visited,Path):-
  connected(A,C),
  C\==B,
  not(member(C,Visited)),
  traverse(C,B,[C|Visited],Path).

Sample Test Case:
?- path(1,4,Path).
Path = [1, 2, 3, 4] ;

Path = [1, 2, 3, 5, 4] ;
Path = [1, 2, 5, 4] ;
Path = [1, 2, 5, 3, 4] ;
Path = [1, 2, 6, 5, 4] ;
Path = [1, 2, 6, 5, 3, 4] ;
Path = [1, 6, 2, 3, 4] ;
Path = [1, 6, 2, 3, 5, 4] ;
Path = [1, 6, 2, 5, 4] ;
Path = [1, 6, 2, 5, 3, 4] ;
Path = [1, 6, 5, 4] ;
Path = [1, 6, 5, 2, 3, 4] ;
Path = [1, 6, 5, 3, 4] ;
false.

5. Write a Prolog program to count all possible paths from a given node to another for the graph given in question no. 4.