deerwalk
DWIT College

Lab Report 3

Submitted by:

Abhishek Kadariya

0501

2019

Submitted to:

Birodh Rijal

(Artificial Intelligence Lecturer)

1. Given the following directed graph:

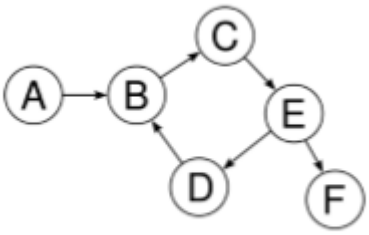

Figure 1: A directed graph with six nodes

Write a prolog program that would check if there is a path from any node X to any other node Y. Encode your database of facts like: edge('A','B'). , edge('E','F'). and so on.

Now wrie a rule for the predicate path(X,Y) that is true if there is a path from node represented by the variable X to the path represented by variable Y. You might have guessed that this requires the rule to be recursive. As you have to do while writing recursive function, think of the base case and the recursive step.

Answer:

edge(a,b).

edge(b,c).

edge(d,b).

edge(c,e).

edge(e,d).

edge(e,f).

path(X,Y):-edge(X,Y).

path(X,Y):-edge(X,Z),path(Z,Y).

```
?- consult("F:\\DWIT\\5. Fifth semester\\4. Artificial Intelligence\\lab\\lab 4\\graph.pl").
true.

?- path(a, b).
true .

?- path(a, c).
true .

?- path(d, f).
true .

?- path(f, e).
false.

?- path(f, a).
false.
```

2. Using the concept of recursion add a rule predecessor(X,Y) to the family tree in the third lab that is true if X is the predecessor of Y (i.e. X comes before Y in the parent relation).

Answer:

predecessor(X,Y):-parent(X,Y).

predecessor(X,Y):-parent(X,W),predecessor(W,Y).

```
?- consult("F:\\DWIT\\5. Fifth semester\\4. Artificial Intelligence\\lab\\lab 2\\lab.pl").
true.

?- predecessor(abraham,maggie).
true .
```

predecessor(bart,homer).

Flase.

3. Monkey and banana problem.

There is a monkey at the door into a room. In the middle of the room a banana is hanging from the ceiling. The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor. At the window of the room there is a box the monkey may use.

To solve the following problem in prolog the following program is used.

do(state(middle, onbox, middle, hasnot), grab, state(middle, onbox, middle, has)).

do(state(L, onfloor, L, Banana), climb, state(L, onbox, L, Banana)).

do(state(L1, onfloor, L1, Banana), push(L1, L2), state(L2, onfloor, L2, Banana)).

do(state(L1, onfloor, Box, Banana), walk(L1, L2), state(L2, onfloor, Box, Banana)).

canget(state(_, _, _, has)).

canget(State1) :- do(State1, Action, State2), canget(State2).

```
?-  canget(state(atwindow, onfloor, atwindow, has)).
true .

?- canget(state(atdoor, onfloor, atwindow, hasnot)).
true .

?-  canget(state(atwindow, onbox, atwindow, hasnot)).
false.
```

4. Towers of Hanoi

Answer:

The Tower of Hanoi is a mathematical game or puzzle. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

To implement this game in prolog following code can be used,

```
Make_move(1,X,Y,_) :- write('Move top disk from '), write(X), write(' to '), write(Y), nl.

Make_move(N,X,Y,Z) :- N>1, M is N-1, Make_move(M,X,Z,Y), Make_move(1,X,Y,_), Make_move(M,Z,Y,X).

hanoi(N) :- Make_move(N,'A','C','B').
```

```
?- hanoi(3).
Move top disk from A to C
Move top disk from A to B
Move top disk from C to B
Move top disk from A to C
Move top disk from B to A
Move top disk from B to C
Move top disk from A to C
true .
```