Yachay Tech University

Mathematical and Computational Logic

Prolog Lab 3: Graph Traversal

1. Introduction

Graphs are widely used in computer science to model networks, relationships, and problems. In Prolog, graphs can be expressed naturally using facts (edge/2) and explored using recursion and backtracking.

2. Representing Graphs

A graph can be represented with edge/2 facts:

```
edge(a, b).
edge(b, c).
edge(a, d).
edge(d, c).
```

3. Defining Paths

We define a path between two nodes recursively:

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

4. Handling Cycles

In graphs with cycles, naive recursion may loop forever. We prevent this by keeping a list of visited nodes:

```
path(X, Y) :- path(X, Y, []).
path(X, Y, _) :- edge(X, Y).
path(X, Y, Visited) :-
   edge(X, Z),
   \+ member(Z, Visited),
   path(Z, Y, [X|Visited]).
```

5. Collecting Paths

We can collect all possible paths using findall/3:

```
findall(P, path(a, c, P), Paths).
```

6. Example Run

Knowledge base:

```
edge(a, b).
edge(b, c).
edge(a, d).
edge(d, c).
```

Query:

```
?- path(a, c).
true.
```

Lab: Graph Traversal in Prolog

Part 1 – Basics

1. Define edges for this graph: a → b → c, a → d → c.
2. Test reachability using path/2.
3. Query: Is there a path from a to c? From b to a?

Part 2 – Cycles

1. Add a cycle: edge(c, a).
2. Run ?- path(a, c).
3. Observe infinite recursion.
4. Fix it using the visited list approach.

Part 3 – Listing All Paths

1. Use findall/3 to get all possible paths between a and c.
2. Print the result.

Part 4 – Student Extension

1. Create a graph representing a maze (rooms connected by doors).
2. Write rules to find a path from entrance to exit.
3. Extend the program to print the actual path (list of nodes).

Deliverables

- A .pl file with:
  - Graph representation
  - Path-finding with cycle handling

  - Queries demonstrating paths
- Written answers: screenshots of successful queries