	Smt. Indira Gandhi College of Engineering **Ghansoli – Navi Mumbai** **Computer Engineering Department** **Academic Year 2022-23 (Even Sem)**

Student Name: Aryan Mishra **Roll No:** 54 **Class:** TE **Sem:** VI

Course Name: Artificial Intelligence Lab

Course Code: CSL604

Experiment No. 07

Experiment Title: IMPLEMENTING FIRST ORDER LOGIC USING PROLOG : PARSING OF CONTEXT FREE GRAMMAR

Date of Performance	Date of Submission	Marks (10)					Sign / Remark
		A	B	C	D	E	
		2	3	2	2	1	
		Total Marks					

A: On Time Submission **B:** Understanding **C:** Analytical Skill

D: Critical Thinking **E:** Presentation

AV
SS

Date:

~~Date~~ Experiment NO. 07

Title: Implementation ~~[illegible]~~ [illegible] logic [illegible]: Prolog of [illegible] [illegible]

Aim: To implement first order logic using Prolog [illegible] of [illegible] [illegible]

Theory:

FOL

1) FOL also known as predicate logic
2) It expands on propositional logic, [illegible] [illegible] predicate

(2) In propositional logic, the atomic sentences [illegible] symbol [illegible] [illegible] letter

[illegible]

A Prolog program consists of a knowledge base [illegible] [illegible] [illegible] [illegible] conjunction predicate [illegible] first [illegible] [illegible]

For instance

$\forall a, b, c, d$ [illegible] $\wedge$ [illegible]

A [illegible] sentence in Prolog

[illegible] (A) :- [illegible] (A, B) [illegible] (B, C), [illegible] (C, A)

54
A

Predicates:

```
        edge(~~[illegible]~~Node1, Node2)
    route(Start, End, Path)

    ~~[illegible]~~ dfs(Start, End, Visited, [illegible] visited[])
    dfs(Start, End, visited, path)

    member(Element, List)
    reverse(List, Reversed)
```

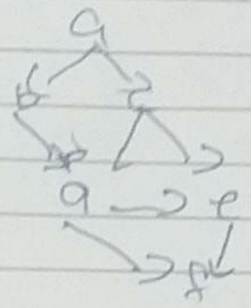
Clauses:

```
        edge(a,b)
        edge(a,c)
        edge(b,dd)

    edge(c,dd)
    edge(c,e)
    edge(c,f)
  route(Start, End, Path):
    dfs(Start, End [Start], Reversed Path),
    reverse(Reversed Path, Path)
  dfs(Start, End, Visited, [End|Visited]):
           edge(Start, End)
  dfs(Start, End, Visited, Path):
         edge(Start, Next)

     \+ member(Next, Visited)
```

search graph tree



?- route(a, f, path)
path = [a, b, d, f]

conclusion:
prolog can be used to find routes in a cycles. The program uses a DFS algorithm to find a path between two vertices in the graph. The program defines a set of rules and facts that define the edges of the graph and the logic for performing the search.

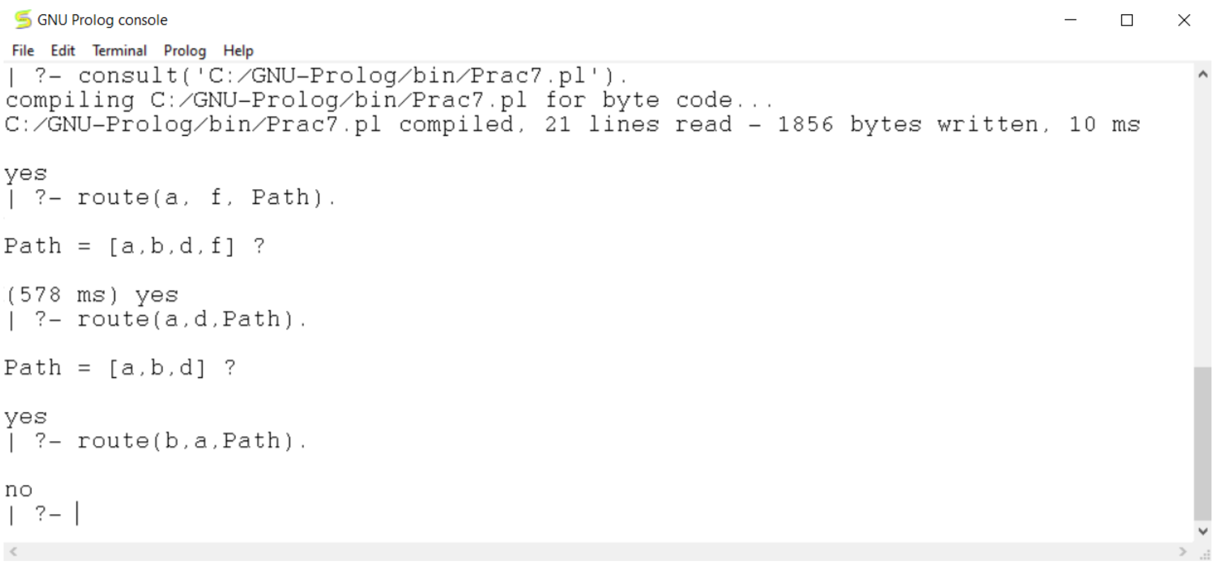
PROGRAM:

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

route(Start, End, Path) :-
    dfs(Start, End, [Start], ReversePath),
    reverse(ReversePath, Path).

dfs(Start, End, Visited, [End|Visited]) :-
    edge(Start, End).
dfs(Start, End, Visited, Path) :-
    edge(Start, Next),
    \+ member(Next, Visited),
    dfs(Next, End, [Next|Visited], Path).
```

OUTPUT:



```
| ?- consult('C:/GNU-Prolog/bin/Prac7.pl').
compiling C:/GNU-Prolog/bin/Prac7.pl for byte code...
C:/GNU-Prolog/bin/Prac7.pl compiled, 21 lines read - 1856 bytes written, 10 ms

yes
| ?- route(a, f, Path).

Path = [a,b,d,f] ?

(578 ms) yes
| ?- route(a,d,Path).

Path = [a,b,d] ?

yes
| ?- route(b,a,Path).

no
| ?- |
```