Shri Vile Parle Kelavani Mandal's

INSTITUTE OF TECHNOLOGY

DHULE (M.S.)

DEPARMENT OF COMPUTER ENGINEERING

Name: Pagariya Sakshi **Roll no**:17

Subject : Artificial Intelligence Lab **Subject Code:** BTCOL707

Class: Final Year Comp. Engg. **Expt. No. :** 03

Title : Solve any problem using depth first search.

Problem Staement :	Solve any problem using depth first search.
Software Required :	Prolog
Theory :	A basic search technique called Depth-First Search (DFS) is used in computer science and artificial intelligence to explore and navigate graphs and tree structures. It is frequently utilized in many different AI applications, such as knowledge representation, pathfinding, and problem solving involving state space exploration. Here's an illustration of a typical artificial intelligence application of DFS: AI Pathfinding: Assume you wish to determine the shortest route between a start point and a goal point in a maze or grid-based environment. In order to explore the grid, one can employ Depth-First Search, which involves recursively traveling as far down a path as feasible before turning around when there is no more way to proceed. DFS(node): if node is the goal: return "Goal found!"

```
    mark node as visited

    for each neighbor of node:

        if neighbor is not visited:

            result = DFS(neighbor)

            if result is "Goal found!":

                return "Goal found!"

    return "Goal not found"
```

In this case, DFS investigates the grid or maze by following a path as far as it can go before turning around to look at other possibilities. An indication of success is returned if the target is reached.

When you want to delve extensively into a search field before thinking about other options, DFS is especially helpful. It might not always provide the quickest path, though, and in some circumstances it might be prone to becoming caught in an endless cycle.

DFS can be used to navigate state spaces in more complicated AI applications, such as those involving planning, gaming, and decision-making. To further optimize the search process, it can be coupled with heuristics and other search algorithms.

```
% Define the edges of the graph.
edge(a, b).
edge(b, c).
edge(b, d).
edge(c, e).
edge(e, f).
edge(f, g).

% Define the DFS algorithm.
dfs(Start, End, Path) :-
    dfs(Start, End, [Start], Path).
dfs(Current, End, Visited, Path) :-
    Current == End,
    reverse(Visited, Path).

dfs(Current, End, Visited, Path) :-
    edge(Current, Next),
    not(member(Next, Visited)),
    dfs(Next, End, [Next | Visited], Path).

% Example usage:
% Find a path from 'a' to 'g'.
?- dfs(a, g, Path).
```

We define a simple directed graph with edge/2 predicates in this Prolog code. The DFS search is started with the dfs/3 predicate. It returns the path after receiving the start and end nodes.

The recursive DFS search is the dfs/4 predicate. It searches for a route to the End node beginning from the Current node. The path is constructed and the visited nodes are tracked in the Visited list. It obtains the path by reversing the visited nodes when it reaches the End node.

By passing the start and finish nodes to dfs/3, you may use this code to find a path between any two nodes in the given graph.

Conclusion: