1. Write a Program in Prolog to solve any problem using Depth First Search.

```
Answer: % Define your graph as facts describing the edges.
% Replace these with your actual graph.
edge(a, b).
edge(b, c).
edge(b, d).
edge(c, e).
edge(d, e).
edge(e, f).
% Define a predicate to check if two nodes are connected.
connected(X, Y) := edge(X, Y) ; edge(Y, X).
% Define the depth-first search algorithm with recursion.
dfs(Node, Goal, Path) :-
  dfs recursive(Node, Goal, [Node], ReversedPath), % Start with the path in reverse
order
  reverse(ReversedPath, Path). % Reverse the path to get it in the correct order
% Base case: We reached the goal node.
dfs recursive(Node, Node, Path, Path).
% Recursive case: Continue exploring the graph.
dfs recursive(Current, Goal, Path, Result):-
  connected(Current, Next),
  \+ member(Next, Path), \% Avoid cycles
  dfs recursive(Next, Goal, [Next | Path], Result).
% Predicate to initiate the DFS search and return the result.
dfs search(Start, Goal, Path) :-
  dfs(Start, Goal, Path).
% Example usage:
% To solve a problem, call dfs search(StartNode, GoalNode, Path).
% Replace StartNode and GoalNode with your problem's specific values.
Output:
```

```
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (comes with ABSOLUTELY ND WARRANTY. This is free software.
Pleaser un? - License. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

? consult('C:/Users/ashis/OneDrive/Documents/(NCER) final year/sem 7/college authority or teacher given thing and my respective work/AI/AI practical/3/d
fs.pl').

Ture.

? - dfs_search(a, f, Path).
Path = [a, b, c, e, f].

? - dfs_search(a, e, Path).
Path = [a, b, c, e].
```

2. Write a Program in Python to solve any problem using Depth First Search.

```
Answer:
class Graph:
  def __init__(self):
     self.graph = \{\}
  def add edge(self, node, neighbor):
     if node in self.graph:
       self.graph[node].append(neighbor)
     else:
       self.graph[node] = [neighbor]
  def dfs(self, start, goal):
     visited = set()
     path = self.dfs recursive(start, goal, visited, [start])
     return path
  def dfs recursive(self, node, goal, visited, path):
     if node == goal:
       return path
     if node not in visited:
       visited.add(node)
       for neighbor in self.graph.get(node, []):
          if neighbor not in visited:
            new path = path + [neighbor]
            result = self.dfs recursive(neighbor, goal, visited, new path)
            if result:
               return result
     return None
# Example usage:
if __name__ == "__main__":
  g = Graph()
```

```
g.add_edge('a', 'b')
g.add_edge('b', 'c')
g.add_edge('b', 'd')
g.add_edge('c', 'e')
g.add_edge('d', 'e')
g.add_edge('e', 'f')

start_node = 'a'
goal_node = 'f'

path = g.dfs(start_node, goal_node)
if path:
    print(f"Path from {start_node} to {goal_node}: {path}")
else:
    print(f"No path found from {start_node} to {goal_node}")
```

Output:

```
The fdt Shell Debug Options Window Help

| Python 3.10.4 (tags/v3.10.4;9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
| Type "help", "copyright", "credit-" or "license()" for more information.

| RESTART: C:\Users\ashis\OneDrive\Documents\(NCER)\ final year\sem 7\college authority or teacher given thing and my respective work\AI\AI practical\3\bf system
| Shortest path from a to f: ['a', 'b', 'c', 'e', 'f']

| Interest | Line | Lin
```