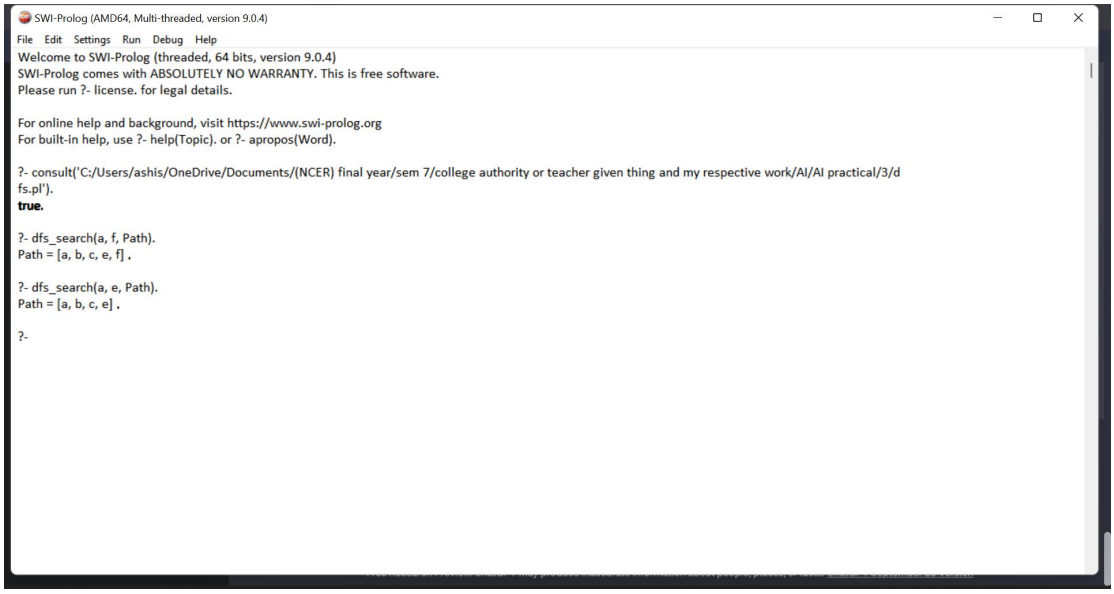
1. Write a Program in Prolog to solve any problem using Depth First Search.

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:



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: