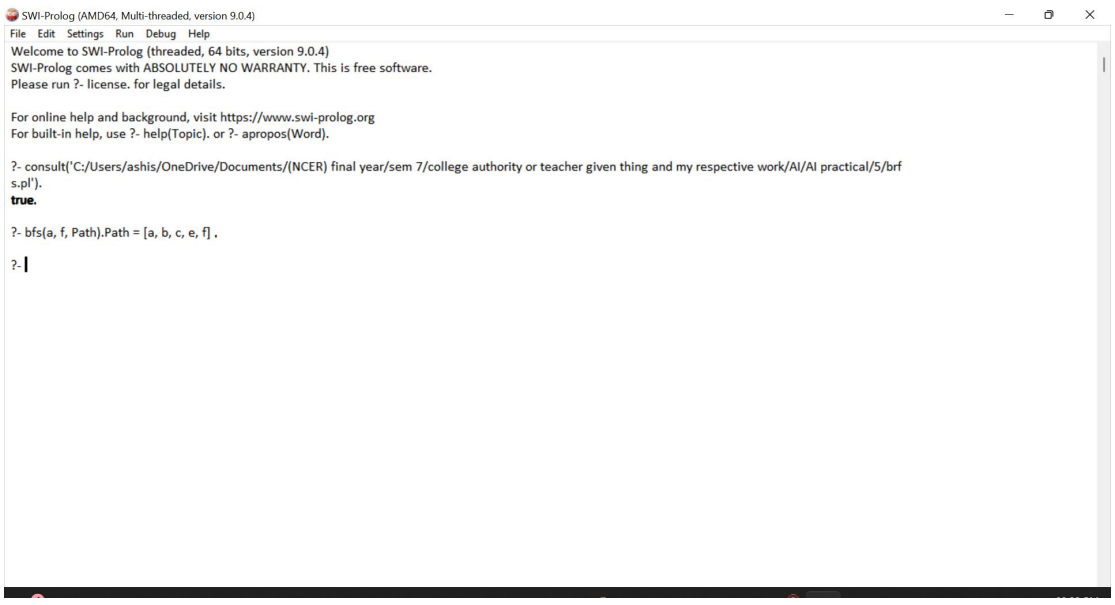
1. Write a Program in Prolog to solve any problem using Breadth First Search.

Answer:

```
% Define the graph with edges.
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 breadth-first search algorithm.
bfs(Start, Goal, Path) :-
    bfs_search([ [Start] ], Goal, RevPath),
    reverse(RevPath, Path).

bfs_search([ [Goal | Path] | _ ], Goal, [Goal | Path]).
bfs_search([ [Node | Path] | Rest ], Goal, Result) :-
    findall([Next, Node | Path], (connected(Node, Next), not(member(Next, Path))),
NewPaths),
    append(Rest, NewPaths, AllPaths),
    bfs_search(AllPaths, Goal, Result).

% Example usage:
%?- bfs(a, f, Path).
```

Output:



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

Answer:

```
from collections import deque

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 bfs(self, start, goal):
        visited = set()
        queue = deque()
        queue.append([start])

        while queue:
            path = queue.popleft()
            node = path[-1]

            if node == goal:
                return path

            if node not in visited:
                for neighbor in self.graph.get(node, []):
                    new_path = list(path)
                    new_path.append(neighbor)
                    queue.append(new_path)

                visited.add(node)

        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.bfs(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: