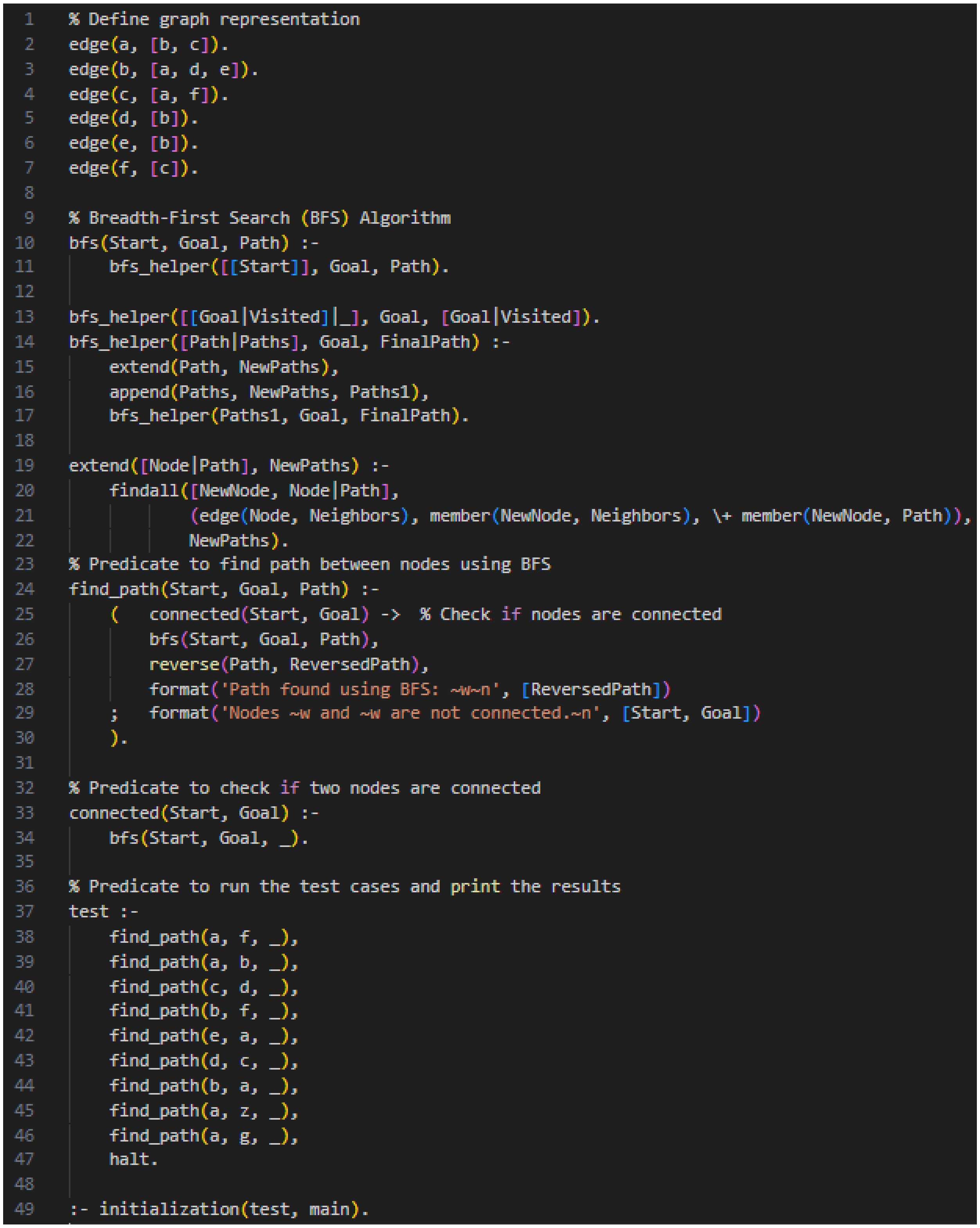
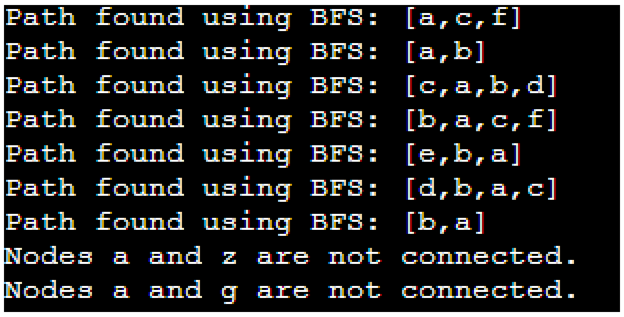
Project Sreecharan Vanam

Code:



```prolog
% Define graph representation
edge(a, [b, c]).
edge(b, [a, d, e]).
edge(c, [a, f]).
edge(d, [b]).
edge(e, [b]).
edge(f, [c]).

% Breadth-First Search (BFS) Algorithm
bfs(Start, Goal, Path) :-
    bfs_helper([[Start]], Goal, Path).

bfs_helper([[Goal|Visited]|_], Goal, [Goal|Visited]).
bfs_helper([Path|Paths], Goal, FinalPath) :-
    extend(Path, NewPaths),
    append(Paths, NewPaths, Paths1),
    bfs_helper(Paths1, Goal, FinalPath).

extend([Node|Path], NewPaths) :-
    findall([NewNode, Node|Path],
            (edge(Node, Neighbors), member(NewNode, Neighbors), \+ member(NewNode, Path)),
            NewPaths).
% Predicate to find path between nodes using BFS
find_path(Start, Goal, Path) :-
    (   connected(Start, Goal) ->  % Check if nodes are connected
        bfs(Start, Goal, Path),
        reverse(Path, ReversedPath),
        format('Path found using BFS: ~w~n', [ReversedPath])
    ;   format('Nodes ~w and ~w are not connected.~n', [Start, Goal])
    ).

% Predicate to check if two nodes are connected
connected(Start, Goal) :-
    bfs(Start, Goal, _).

% Predicate to run the test cases and print the results
test :-
    find_path(a, f, _),
    find_path(a, b, _),
    find_path(c, d, _),
    find_path(b, f, _),
    find_path(e, a, _),
    find_path(d, c, _),
    find_path(b, a, _),
    find_path(a, z, _),
    find_path(a, g, _),
    halt.

:- initialization(test, main).
```

Outputs:



```
Path found using BFS: [a,c,f]
Path found using BFS: [a,b]
Path found using BFS: [c,a,b,d]
Path found using BFS: [b,a,c,f]
Path found using BFS: [e,b,a]
Path found using BFS: [d,b,a,c]
Path found using BFS: [b,a]
Nodes a and z are not connected.
Nodes a and g are not connected.
```

Explanation:

Using Breadth-First Search (BFS) as example, the Prolog implementation displayed is intended to find the shortest path among the set of all paths between two nodes in a given graph. The definition of graph can be implemented using predicates for the edges between different nodes and the BFS algorithm has been implemented recursively. BFS represents a graph traversal algorithm that explores the graph iteratively, visiting nodes level by level. We use queue as tree to manage our traversal. Also, we have implemented BFS in our project for both directed and undirected graphs. To test our BFS implementation, we come up with several test cases that given a graph and a pair of nodes as ‘input’ and plot out the shortest path as ‘output’. These test cases cover the most cases in our mind regarding such problems, and all of them unanimously yield correct results, indicating the correctness of our implementation. In general, such a powerful tool as Prolog has demonstrated its flexibility, features as well as effectiveness to act as the core in our quest of tackling graph traversal problems.