Assignment 2 Documentation

(In this Assignment, i have chosen option(a) ,i.e; Depth-First search and Best-First Search)

Source code with comments provided :-

(The comments are marked with bold italics in blue and start with %)

```
% Out program does option(a). That is, it does depth-first search(dfs) and
best-first search(bfs)

%Name: Rohit Roy.
%Roll Number: 2018259


fact_statements :-
        % We have preprocessed the cities to get My_Heuristics.csv file
        % Similarly, we have preprocessed the roaddistance.csv to get
Edge_values.csv file
        % My_Heuristics.csv contains 3 columns, the 1st is city1, the 2nd is city2,
the 3rd is the heuristic value between them
        % Edge_values.csv contains 3 columns, the 1st is city1, the 2nd is city2,
the 3rd is the edge cost between them

            csv_read_file('My_Heuristics.csv', Rows1, [functor(h)]),
            maplist(assert, Rows1),
            csv_read_file('Edge_values.csv', Rows2, [functor(edge)]),
            maplist(assert, Rows2).


% this clause makes the edges between any 2 city bi-directional

edge1(X, Y, Length):-
        edge(X,Y,Length) ; edge(Y,X,Length).


start :-
        retractall(h(_,_,_)),
        retractall(edge(_,_,_)),
        fact_statements,
```

```
        write("Hello, welcome to our search engine"),nl,
        write("This code supports DFS and Best-First Search algorithm"),nl,
        writeln("Enter your start city : "), read(Start),
        writeln("Enter your goal city : "), read(Goal),
        writeln("Enter 1 to select DFS algo and 2 to select Best-First Search Algo"),
        writeln("Your option :- "), read(Option),
        (
                Option==1 -> callDFS(Start, Goal);
                callBestFirstSearch(Start, Goal)
        ).


% print clause takes a list as an argument and prints all its elements in one line

print([]) :-
        writeln(' ').
print(L) :-
        [H|T] = L,
        write(H),write(' '),
        print(T).

% the dfs clause is used to traverse over the city map and find the path between
start and goal city

% Below is a base case of the dfs clause where start and goal city are same. It
means that we have reached the goal city
% and we need to terminate our recursion here.

dfs(Path, Goal, Goal, Solution, CurrLength, Distance, MaxRecDepth) :-
        Solution = [Goal | Path],
        Distance is CurrLength.

% This is the main recursive clause of dfs function. It takes Node as the current
city, Goal as the goal city, Path as the path of cities reached till now,
% CurrLength as a temporary variable to store the cost of reaching current node,
% Distance as the cost of reaching the goal city, MaxRecDepth as the maximum
recursion depth the user allows.

dfs(Path, Node, Goal, Solution, CurrLength, Distance, MaxRecDepth) :-
        MaxRecDepth>0,
        edge1(Node, Children, Dis),
        Dis>0,
        \+member(Children, Path),
        TemMaxDepth is MaxRecDepth-1,
```

```
        Tempdis is CurrLength+Dis,
        dfs([Node|Path], Children, Goal, Solution, Tempdis, Distance, TemMaxDepth).


% This is the caller clause of dfs.

callDFS(Start, Goal) :-
        writeln("This is DFS function"),
        writeln("Enter the maximum recursion depth required for dfs : "),
read(MaxRecDis),
        (Start==Goal -> (Distance is 0, Solution = []); dfs([], Start, Goal, Solution, 0,
Distance, MaxRecDis)),
        write("The solution path is : "),
        reverse(Solution, ReverseSolution),
        print(ReverseSolution),
        write("The total length of the path found is : "),
        writeln(Distance).

% This is the caller clause of Best-First Search (bfs)

callBestFirstSearch(Start, Goal) :-
        writeln("This is BFS function"),
        (Start==Goal -> (Distance is 0, Solution = []); (h(Start, Goal, Value),
bfs([],Start,Goal,Solution,[Value-Start],[],0, Distance))),
        write("The solution path is : "),
        reverse(Solution, ReverseSolution),
        print(ReverseSolution),
        write("The total length of the path found is : "),
        writeln(Distance).


% The bfs clause is used to traverse over the city map while selecting the child
node with least heuristic cost

% Below is a base case of bfs clause where the start and goal city are same. It
means that we have reached our goal city
% and we need to terminate our recursion.

bfs(Path,Goal,Goal,Solution,OpenList,ClosedList,CurrLength,Distance) :-
        Solution = [Goal|Path],
        Distance is CurrLength.

% Below is a base case of bfs clause where the OPEN list is empty. This means
that there are no more cities left to expand over, and that we
```

```
% have failed to find a solution.

bfs(Path,Start,Goal,Solution,[],ClosedList,CurrLength,Distance) :-
        Solution = [],
        Distance is CurrLength,
        writeln("Solution hasn't been found, try for different cities next time !").


% This is the main recursive clause of bfs function. Start is the starting city, Goal is the goal(destination) city, Path as the path of cities reached till now,
% Solution as the path till goal node, OpenList as a list of nodes(cities) which can be expanded, ClosedList as the cities which have already been expanded,
% CurrLength as the cost of path till the current node and Distance is the cost of reaching goal.

bfs(Path,Start,Goal,Solution,OpenList,ClosedList,CurrLength,Distance) :-
        [Head|Tail] = OpenList,
        Temp-Currnode = Head,
        findall(Value-Children,
(edge1(Currnode,Children,Dis),\+member(Children,ClosedList),\+(Children==Currnode),
h(Children,Goal,Value)), ChildList),
        append(ChildList, Tail, AllChildList),
        sort(AllChildList, OpenList2),
        [_-NewCurrNode|_] = OpenList2,
        edge1(Currnode,NewCurrNode,Weight),
        Temp2 is Weight+CurrLength,


bfs([Start|Path],NewCurrNode,Goal,Solution,OpenList2,[Start|ClosedList],Temp2,Distanc
e).
```

Explanation of Code :-

First, during the starting of code, we have read all the data from the csv file of edge distance between any 2 cities as well as the heuristic values between them. Then we used assert statements to convert them into facts.

Then, we read the start city and goal city as inputs from the user. After that, we asked the user for an option to choose the dfs function or bfs (best-first search) function.
If the user enters ‘1’, then we choose dfs, else we choose bfs. We invoked these options using a caller clause for dfs/bfs.

In the dfs caller clause , we further asked the user for entering the maximum recursion depth allowed. After that, we invoked the dfs clause.
In the dfs clause, we kept a base condition where if the current node and goal node are the same, then we stop the recursion there and print the cost and solution path. Other than that, there was the usual recursive condition where we kept exploring the children of the current node recursively in dfs.

In the bfs caller clause, we took no further input and directly invoked the bfs clause.
In the bfs clause, we kept 2 base conditions. One was when the current node and goal node are the same, then we return the cost and solution path. In the other case, if the OPEN list is empty, then we declare that we have found no solution.
In the bfs recursive clause, we find all the children of the current node, add it to the OPEN list and sort it. Then we find the node with least heuristic value out of it and expand on it recursively.

Working Screenshots :-

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- 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).

1 ?- start.
Hello, welcome to our search engine
This code supports DFS and Best-First Search algorithm
Enter your start city :
|: 'Ahmedabad'.
Enter your goal city :
|: 'Hubli'.
Enter 1 to select DFS algo and 2 to select Best-First Search Algo
Your option :-
|: 1.
This is DFS function
Enter the maximum recursion depth required for dfs :
|: 3.
The solution path is : Ahmedabad Bangalore Bhubaneshwar Hubli
The total length of the path found is : 4648
true .
```

```
2 ?- start.
Hello, welcome to our search engine
This code supports DFS and Best-First Search algorithm
Enter your start city :
|: 'Agartala'.
Enter your goal city :
|: 'Hubli'.
Enter 1 to select DFS algo and 2 to select Best-First Search Algo
Your option :-
|: 2.
This is BFS function
The solution path is : Agartala Panjim Hubli
The total length of the path found is : 3697
true .
```