AI: Assignment 2

Arjun Temura
2020497

OUTPUT:

1. **DFS : Depth First Search**

   Workflow:
   a) Data is read from the csv file and stored in the form a 2D list
   b) Preprocessing of this data to assert facts in form (city1, city2, distance)
   c) helper_dfs() takes source, destination, temporary path, final path, temporary cost, final cost as arguments to implement DFS.
   d) dfs() called by helper_dfs() . Handles edge cases if source is the destination or if source and destination are directly connected. If not connected directly, an Intermediate node is selected and dfs() is called recursively with this node as the source.
   e) Path and Cost found is returned

Source is same as destination

```
?- main.
##################################################
--------------------Welcome----------------------
##################################################
   Cities Distance Directory
Source City: 'Agra'.
Destination City: |: 'Agra'.
What algorithm do you wish to test: 1. Depth First Search 2. Best First Search|1.
Algorithm chosen is Depth First Search on source

The Depth First Search path from Agra to Agra is [Agra]

Estimated Cost from Agra to Agra is 0
true .
```

Directly connected cities

```
?- main.
#####################################################
----------------------Welcome-----------------------
#####################################################
   Cities Distance Directory
Source City: 'Agartala'.
Destination City: |: 'Hyderabad'.
What algorithm do you wish to test: 1. Depth First Search 2. Best First Search|1.
Algorithm chosen is Depth First Search on source

The Depth First Search path from Agartala to Hyderabad is [Agartala,Hyderabad]

Estimated Cost from Agartala to Hyderabad is 3330
true .
```

Cities not directly connected

```
?- main.
#####################################################
----------------------Welcome-----------------------
#####################################################
   Cities Distance Directory
Source City: 'Agra'.
Destination City: |: 'Calicut'.
What algorithm do you wish to test: 1. Depth First Search 2. Best First Search|1.
Algorithm chosen is Depth First Search on source

The Depth First Search path from Agra to Calicut is [Agra,Ahmedabad,Calicut]

Estimated Cost from Agra to Calicut is 2526
true .

?-
```

2. Best First Search

A heuristic function f(n)=h(n) is chosen and assigned to each city to the destination.
The heuristic function chosen is :
Let n be a node,

h(n)=minimum(
for all intermediate nodes 'intermediate_node' of n and destination
{distance(n,intermediate_node) + distance(intermediate_node, destination) }
)

This heuristic function is admissible and consistent as

1) It does not overestimate the distance from the source to destination.
2) For each node, h(n)<=cost(n,n')+h(n')

Workflow:

a) Data is read from the csv file and stored in the form a 2D list
b) Preprocessing of this data to assert facts in form (city1, city2, distance)
c) bestfs() takes source, destination, final path, temporary cost and final cost as arguments.
d) Heuristics are assigned from each city to the destination by assign_heurestic()
e) best_first() is used to calculate the best path from source to destination as per the heuristics.

f) best_first() extends the source node and appends the child nodes at the end of the queue. The queue is sorted based on the heuristic values and a recursive call is made with the new queue.
g) cost_estimator() estimates the cost to travel on this path and the values are returned

Source is same as destination

```
?- main.
#####################################################
----------------------Welcome-----------------------
#####################################################
   Cities Distance Directory
Source City: 'Agra'.
Destination City: |: 'Agra'.
What algorithm do you wish to test: 1. Depth First Search 2. Best First Search|2.
Algorithm chosen is Best First Search on source

The Best First Search path from Agra to Agra is [Agra]

Estimated Cost from Agra to Agra is 0
```

Source and destination are directly connected

```
?- main.
#####################################################
----------------------Welcome-----------------------
#####################################################
   Cities Distance Directory
Source City: 'Delhi'.
Destination City: |: 'Hyderabad'.
What algorithm do you wish to test: 1. Depth First Search 2. Best First Search|2.
Algorithm chosen is Best First Search on source
Calculated Heuristics from Delhi


The Best First Search path from Delhi to Hyderabad is [Delhi,Hyderabad]

Estimated Cost from Delhi to Hyderabad is 1477
true .
```

Source and destination are not directly connected

```
?- main.
#####################################################
----------------------Welcome-----------------------
#####################################################
   Cities Distance Directory
Source City: 'Bhopal'.
Destination City: |: 'Calicut'.
What algorithm do you wish to test: 1. Depth First Search 2. Best First Search|2.
Algorithm chosen is Best First Search on source
Calculated Heuristics from Bhopal


The Best First Search path from Bhopal to Calicut is [Bhopal,Cochin,Calicut]

Estimated Cost from Bhopal to Calicut is 2217
true .
```

```
?- main.
##################################################
----------------------Welcome-----------------------
##################################################
   Cities Distance Directory
Source City: 'Agra'.
Destination City: |: 'Shimla'.
What algorithm do you wish to test: 1. Depth First Search 2. Best First Search|2.
Algorithm chosen is Best First Search on source
Calculated Heuristics from Agra


The Best First Search path from Agra to Shimla is [Agra,Chandigarh,Shimla]

Estimated Cost from Agra to Shimla is 547
true .

?-
```

Source code :

```
%starts of the program
:-use_module(library(csv)).
:-use_module(library(lists)).
:-use_module(library(apply)).

%Process the csv data in the form of a 2d list

get_rows_data(File, Lists):- %Stored in 'Lists'
  csv_read_file(File, Rows, []),
  rows_to_lists(Rows, Lists).

rows_to_lists(Rows, Lists):-
  maplist(row_to_list, Rows, Lists).

row_to_list(Row, List):-
  Row =.. [row|List].



main:-

    %retract previous facts in the knowledge base
    retractall(child(_,_,_)),
    retractall(htics(_,_)),

    write("##################################################"), nl,
    write("----------------------Welcome-----------------------"),nl,
    write("##################################################"), nl,
```

```
    write("   Cities Distance Directory     "), nl,

    write("Source City: "),
    read(Src),
    assert(source(Src)),



    write("Destination City: "),
    read(Dst),
    assert(dest(Dst)),

    write("What algorithm do you wish to test: 1. Depth First Search 2.
Best First Search"),
    read(Choice),
    assert(is_choice(Choice)),
    (
    dfs_menu(Src, Dst, Choice);
    bestfs_menu(Src, Dst, Choice)
    ).



dfs_menu(Src, Dst, Choice):-

              Choice is 1,
              get_rows_data("roaddistance.csv",List),
              List=[_|Tail], %First element is the CSV header
              Tail=[Head1|Tail1], %Head1 is the first row, Tail1 is the
rest of the table
              write("Algorithm chosen is Depth First Search on source"),
              process(Head1,Tail1), %preprocessing of data stored in lists
to form facts
              helper_dfs(Src, Dst, [] , Path, 0, Cost),
              %user message
              nl,nl,write("The Depth First Search path from
"),write(Src),write(" to "),write(Dst),write(" is "), write(Path),
              nl,nl,write("Estimated Cost from "),write(Src),write(" to
"),write(Dst),write(" is "),write(Cost).

bestfs_menu(Src, Dst, Choice):-
```

```
            Choice is 2,
            get_rows_data("roaddistance.csv", List),

            List=[_|Tail],
            Tail=[Head1|Tail1],
            write("Algorithm chosen is Best First Search on source"),
            process(Head1,Tail1),
            bestfs(Src, Dst, Path, 0, Cost),
            %user message
            nl,nl,write("The Best First Search path from
"),write(Src),write(" to "),write(Dst),write(" is "), write(Path),
            nl,nl,write("Estimated Cost from "),write(Src),write(" to
"),write(Dst),write(" is "),write(Cost).


%Preprocessing and Storing of csv data as facts

process(_,[]).
process(Head1, Tail1):-

    Tail1=[Head2|Tail2],

    Head1=[_|FirstCol], %head has empty data
    FirstCol=[_|NeighbourCity], % here head is the first cell of the table
    Head2=[_|Distances], %head is empty data
    Distances=[CityName|Distance], %CityName is the name of first city in
the first column i.e. Ahmedabad
    traverse_and_assert(CityName, NeighbourCity, Distance), %assert facts
for each CityName across the row
    process(Head1,Tail2).

%helper function for list traversal
traverse_and_assert(CityName, NeighbourCity, Distances):-
    list_traverse(CityName,NeighbourCity,Distances).

%base case
list_traverse(_,[H1],[H2]).
```

```
%asserting facts about the data stored in the 2d list in the form (city1,
city2, distance)
list_traverse(CityName,[H1|T1],[H2|T2]):-

    assert(child(CityName,H1,H2)),
    assert(child(H1,CityName,H2)),
    assert(cities(CityName)),
    list_traverse(CityName,T1,T2).


%List operations

%append a list to another list
append_list([], L2, L2).

append_list([X | L1], L2, [X | L3]) :-
    append_list(L1, L2, L3).

% insert an element at the end of a list
insert_end(L, X, NewL) :-
    append_list(L, [X], NewL).

%to find the minimal value in a list
minimal(X,Y,X) :- X < Y.
minimal(X,Y,Y) :- X > Y.

list_min_elem([X],X).
list_min_elem([X,Y|Rest],Mini) :-
   list_min_elem([Y|Rest],MiniRest),
   minimal(X,MiniRest,Mini).

% Helper Function of DFS

helper_dfs(Cur,Next, L, Path, Cost, NetCost):-
    insert_end(L,Cur,NewList),  %insert the source node in the list
    dfs(Cur, Next, NewList, Path, Cost, NetCost).

%DFS Function

dfs(Source,Source,[Source],[Source],0,0). %if Source is Dest
```

```
dfs(Cur, Next, L, Path, Cost, NetCost):-

  (
  (
  %Base case
  child(Cur,Next,Dist),
  insert_end(L,Next,Path),
  NetCost is Cost+Dist,!

  );
  (
    %else execute this
    child(Cur, Inter, Dist),
   \+ member(Inter, L),          % avoid the visited nodes
   NewCost is Cost + Dist,
   insert_end(L, Inter, UpdatedList),
   dfs(Inter, Next, UpdatedList, Path, NewCost, NetCost)
  )
  ).

% Best First Search Function

best_first([[Goal|Path]|_], Goal, [Goal|Path]).
best_first([Path|Queue], Goal, FinalPath):-
    extend_and_add(Path, NewPaths, Queue, NewQueue),
    sorter(NewQueue, NewerQueue),
    best_first(NewerQueue, Goal, FinalPath).

%Adds child conections to the Source node in the Queue
extend_and_add([Node|Path], NewPaths, Queue, NewQueue):-
    findall([NewNode, Node|Path],
            (child(Node, NewNode, _),
            \+ member(NewNode, Path)),
            NewPaths),
    append_list(Queue, NewPaths, NewQueue).

%Sorts the paths based on heuristic values

sorter(NewQueue, NewerQueue):-
```

```
    swapping(NewQueue, AuxQueue), !,
    sorter(AuxQueue, NewerQueue).
sorter(NewQueue, NewQueue).

%Swapping of data based on heurestic values

swapping([[A1|B1], [A2|B2]|T], [[A2|B2], [A1|B1]|T]):-
    htics(A1, W1),
    htics(A2, W2),
    W1>W2.
swapping([X|T], [X|V]):-
    swapping(T, V).


%Best First Search : Helping functions

bestfs(Source, Source,[Source], 0, 0).
bestfs(Source, Dest, Path, InitCost, NetCost):-

    findall(X, cities(X), CityDirectory),
    assign_heuristic(Source, CityDirectory, Dest),nl,
    write("Calculated Heuristics from "), write(Source),nl,
    best_first([[Source]], Dest, TempPath),
    reverse(TempPath, Path),

    cost_estimator(Path, 0, NetCost).

% returns the heurestic value for the particular city
comparator(City, Dest, Cost):-
    child(City, Inter, D1),
    child(Inter, Dest, D2),
    Cost is D1 + D2.

% Estimates the cost from Source to Dest for that Path
cost_estimator([A|[B|C]], CurCost, NetCost):-
   ( C==[] ->
    child(A, B, Dist),
    NetCost is CurCost + Dist
   );
   ( child(A, B, Dist),
```

```
    NewCost is CurCost + Dist,
    cost_estimator([B|C], NewCost, NetCost)
   ).

% Assigns the heurestics

assign_heuristic(Source, [], Dest).
assign_heuristic(Source, [H|T], Dest):-
    findall(X, comparator(H, Dest, X), L),
    min_list(L,Min),
    forall(\+ htics(H,Min) , assert(htics(H, Min))),
    assign_heuristic(Source, T, Dest).
```