Assignment 2

Instructions to run-

- Open a terminal and go to the location where the code is kept along with .csv files.
- Type the following commands-
  - swipl.
  - consult(“main.pl”).
  - [‘main.pl’].
  - start.

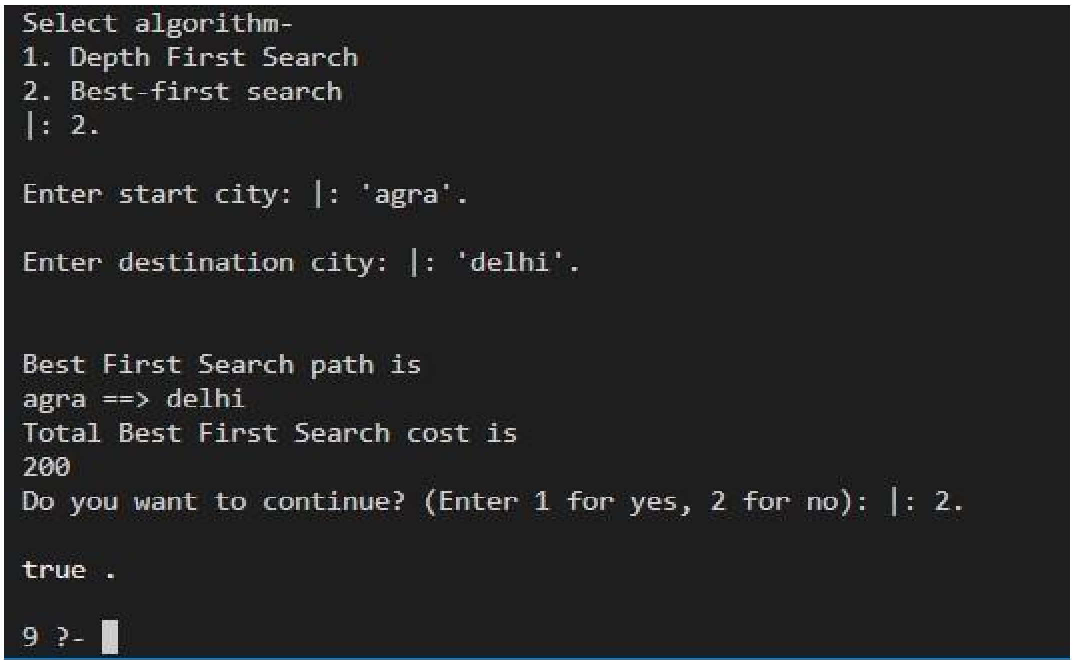
Code-

```
% Commands to run-
% swipl
% consult("refer.pl").
% ['refer.pl'].
% start

list_sum([Item], Item).
list_sum([Item1, Item2 | T], Total) :- list_sum([Item1+Item2 | T],
Total).

total_sum([], 0).
total_sum([H|T], Sum) :- total_sum(T,S), Sum is H+S.

print_line(Line):- write(Line), nl.


% Depth First Search
next_node_in_path(Current, Next, Path) :- distance(Current, Next,
Dist), not(member(Next, Path)), assertz(cost(Dist)).

print_path_for_DFS():- make_a_list(Path, 1), nl, print_line('DFS path
is '), write(Path).

print_cost_for_DFS():- make_a_list(Costsum, 2), total_sum(Costsum,
Cost), nl, nl, print_line('Total DFS cost is '),
      list_sum(Costsum, X), write(X), nl, write(" = "), write(Cost).

dfs_helper_function(End, End, _) :- assertz(city_name(End)),
assertz(cost(0)), print_path_for_DFS(), print_cost_for_DFS().
dfs_helper_function(Start, End, Visited) :-
    next_node_in_path(Start, Next, Visited), assertz(city_name(Start)),
assertz(city_name(" ==> ")),
    dfs_helper_function(Next, End, [Next|Visited]).

% Method = 1 is for distance;  Method = 2 is for cost;  Method = 3 is
for bfs_cost.
make_a_list([H|T], Method):-
    Method=:=1, retract(city_name(H)), make_a_list(T, Method);
    Method=:=2, retract(cost(H)), make_a_list(T, Method);
    Method=:=3, retract(best_first_search_cost(H)), make_a_list(T,
Method).
make_a_list([], _).

% starting point for DFS
depth_first_search(Start, End) :- dfs_helper_function(Start, End,
[Start]).


% Best First Search helper functions
print_node_in_path_BFS(Start) :- write(Start).
print_cost_for_BFS() :- print_line('Total Best First Search cost is '),
make_a_list(Cost, 3), total_sum(Cost, NetCost), write(NetCost).
print_connection_between_path() :- write(' ==> ').
print_list([]).
print_list([H | T]) :- write(H), print_connection_between_path(),
print_list(T).
print_path_for_BFS(Path) :- write(Path), nl.
print_connection_and_path(NBNode) :- print_connection_between_path(),
print_node_in_path_BFS(NBNode).
add_in_priority_queue(List, T, SortedOpenQueue) :- append(List, T,
UpdatedOpenQueue), keysort(UpdatedOpenQueue, SortedOpenQueue).
get_list_according_to_rule(SNode, ClosedPath, End, List) :-
findall(_, (distance(SNode, NNode, _), SNode \== NNode,
not(member(NNode, ClosedPath)), heuristicdistance(NNode, End, _)),
List).


% Best First Search main processing function
bestfs_helper_function(X, X, _, _, _) :- nl, print_cost_for_BFS().
bestfs_helper_function(_ , _, [], _, _):- print_line('Open List
empty!').

bestfs_helper_function(Start, End, OpenQueue, ClosedQueue, Path) :-
[H | T] = OpenQueue, _-StartNode = H,
findall(Value-NextNode, (distance(StartNode, NextNode, _), StartNode
\== NextNode, not(member(NextNode, ClosedQueue)),
heuristicdistance(NextNode, End, Value)), List),
add_in_priority_queue(List, T, SortedOpenQueue), [HeadNode | _] =
SortedOpenQueue,   _-NBNode = HeadNode,
print_connection_and_path(NBNode), distance(Start, NBNode, Dist),
assert(best_first_search_cost(Dist)),
bestfs_helper_function(NBNode, End, SortedOpenQueue, [Start |
ClosedQueue], Path).

% Best First Search calling function
best_first_search(Start, End) :- heuristicdistance(Start, End,
HeuristicValue), print_line('Best First Search path is '),
print_node_in_path_BFS(Start),
bestfs_helper_function(Start, End, [HeuristicValue-Start], [],
[Start]).


% program starts from here by calling start.
start :- build_knowledge_base, welcome_msg(_).

build_knowledge_base :- print_line('Building knowledge base...'), nl,
knowledgebase, nl, print_line('Knowledge base built'), nl.

welcome_msg(_) :-
write('----------------------------------------------'), nl,
write('Welcome to the path finder algoithms'), nl,

write('----------------------------------------------'), nl, nl,
solve().

solve() :- write('Select algorithm-\n1. Depth First Search\n2.
Best-first search'), nl, read(Algo), nl, option_selected(Algo).


option_selected(Algo) :- Algo =:= 1, write('Enter start city: '),
read(S), nl, write('Enter destination city: '), read(D), nl, nl,
depth_first_search(S, D), ask_end();
                              Algo =:= 2, write('Enter start city: '),
read(S), nl, write('Enter destination city: '), read(D), nl, nl,
best_first_search(S, D), ask_end();
                              write('Enter 1 or 2: '), read(X), nl,
option_selected(X).
```

```
ask_end() :-  nl, write('Do you want to continue? (Enter 1 for yes, 2
for no): '), read(Answer), (Answer =:= 1, solve(); Answer =:= 2).

second_option_selected(Algo, S, D) :- Algo =:= 1,
breadth_first_search(S, D); Algo =:= 2, a_star_search(S, D);
                                    write('Enter 1 or 2: '), read(X),
nl, second_option_selected(X, S, D).


% Reading the data from csv in the knowledge base
knowledgebase :-  csv_read_file('knowledgebase.csv', Distances,
[functor(distance)]),
maplist(assertz, Distances),
write(Distances), nl,
csv_read_file('heuristic.csv', Heuristics,
[functor(heuristicdistance)]),
maplist(assertz, Heuristics), nl,
write(Heuristics), nl.
```

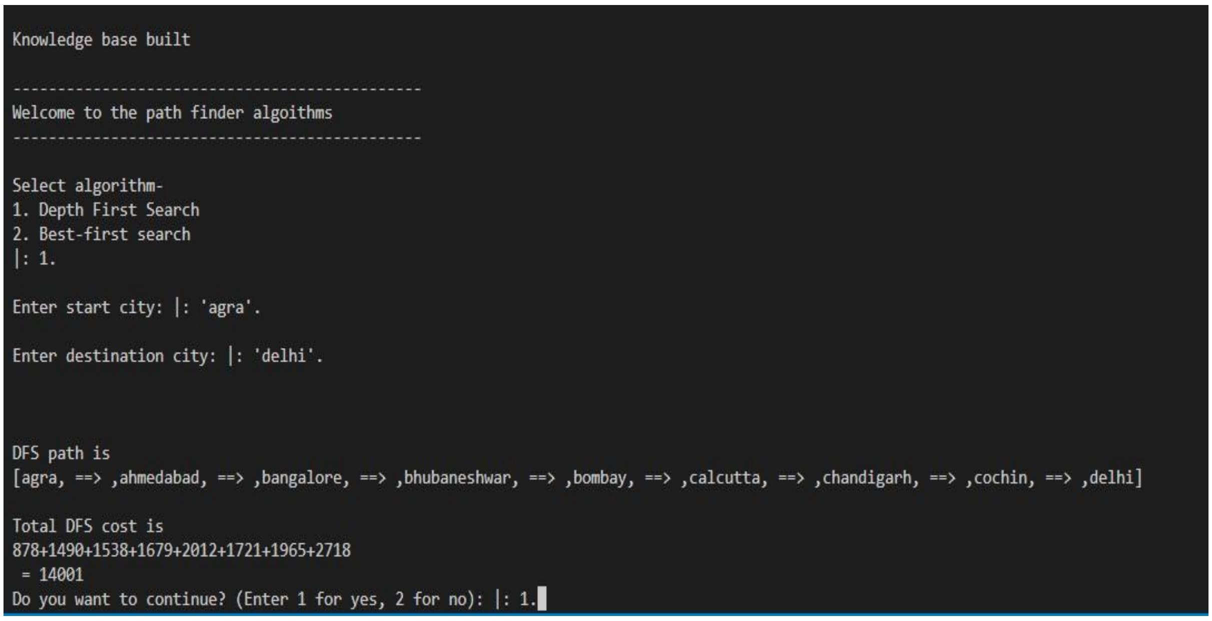
Output Screenshots -

Best First Search



```
Select algorithm-
1. Depth First Search
2. Best-first search
|: 2.

Enter start city: |: 'agra'.

Enter destination city: |: 'delhi'.


Best First Search path is
agra ==> delhi
Total Best First Search cost is
200
Do you want to continue? (Enter 1 for yes, 2 for no): |: 2.

true .

9 ?-
```

Depth First Search



```
Knowledge base built

--------------------------------------------
Welcome to the path finder algoithms
--------------------------------------------

Select algorithm-
1. Depth First Search
2. Best-first search
|: 1.

Enter start city: |: 'agra'.

Enter destination city: |: 'delhi'.



DFS path is
[agra, ==> ,ahmedabad, ==> ,bangalore, ==> ,bhubaneshwar, ==> ,bombay, ==> ,calcutta, ==> ,chandigarh, ==> ,cochin, ==> ,delhi]

Total DFS cost is
878+1490+1538+1679+2012+1721+1965+2718
 = 14001
Do you want to continue? (Enter 1 for yes, 2 for no): |: 1.
```