CSE 643: Artificial Intelligence
Assignment 1

Meetakshi Setiya, 2019253

I have implemented Depth First Search and Best First Search.

Implementation Details

I wrote a python script “create-rules.py” that reads the given roaddistance.csv file and creates rules about the actual road distance between two cities as mentioned in the CSV file. These facts are dumped into the “distances.pl” file.
Since best first search requires the use of heuristics, I have taken the air/straight-line distance between two cities as the plausible heuristic. The python script creates the file “heuristics.pl” and dumps the relevant facts into it. The straight-line/air distance between two cities is found by making a HTTP GET request to https://www.distance24.org at the URL
`https://www.distance24.org/route.json?stops=city1|city2`

The prolog file “find-path.pl” contains the actual search implementation. It consults “distances.pl” and “heuristics.pl”, asks the origin, destination and the search algorithm from the user. After the user has sent their input, the program runs the algorithm and displays the path and the path cost (in this case, the distance from origin to destination via the displayed path) if it exists else it displays the requisite error message or “false”.

I have used Prolog’s built in min-heap data structure as the priority queue for best first search. I have followed the algorithm and pseudocode taught in class for my implementation.

Run the Code:

The python script, the CSV file, the road distance and heuristic fact files along with the search algorithms have all been provided.

- To re-create the distances.pl and heuristics.pl file, run the python script “python create-rules.py”.
- Consult the find-path.pl file to your prolog workspace.
- Simply type ‘start.’ in the command line interface.
- Enter the start point, the end point and the method of search (Best first search or DFS).
- Once the program is done evaluating it will either list the path and the path cost (if a path exists) or ‘false or an appropriate failure message.’

Code

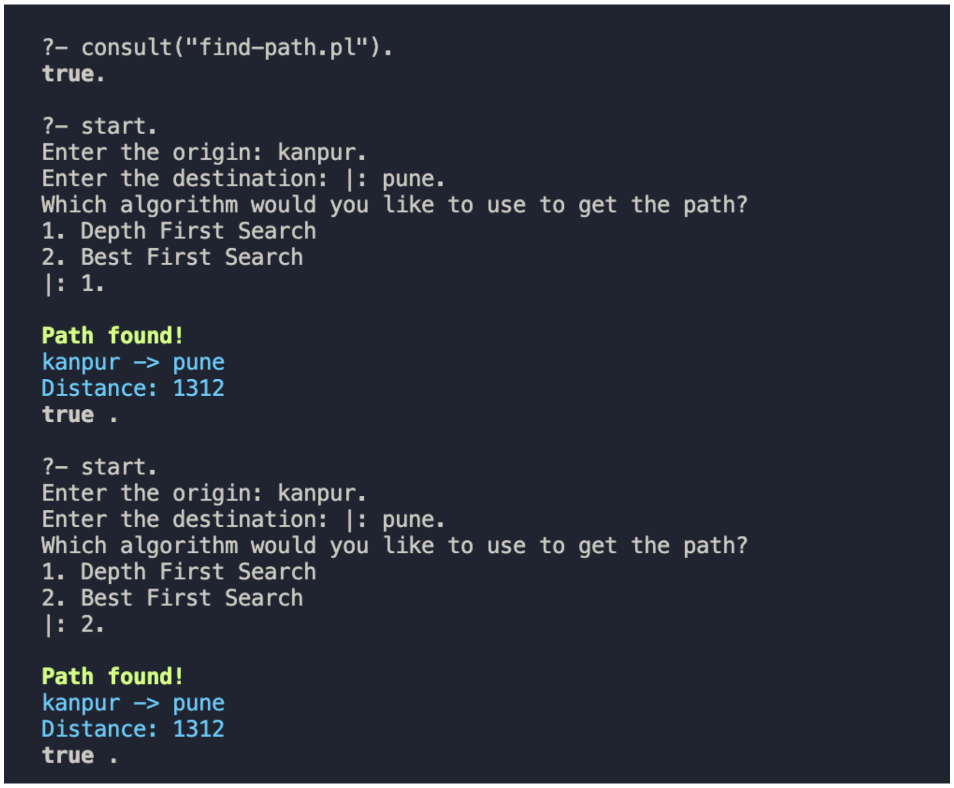
```
%Meetakshi Setiya, 2019253
%AI Assignment 2
%Best First Search and Depth First Search

:- consult(distances).
:- consult(heuristics).
:- dynamic path/3.
:- dynamic flag/1.

printPath([Stop|[]]) :-
   ansi_format([fg(blue)], '~w', [Stop]).
printPath([Stop|Path]) :-
   ansi_format([fg(blue)], '~w ~w ', [Stop, '->']),
   printPath(Path).

setLinks(_, []) :- !.       %add path and distance from current to each 
neighbour
setLinks(CurrentNode, [ChildNode|Neighbours]) :-
   road(CurrentNode, ChildNode, Distance),
   path(PathtoCurrent, CurrentNode, DistancetoCurrent),
   DistancetoChild is Distance + DistancetoCurrent,
   assert(path([ChildNode|PathtoCurrent], ChildNode, DistancetoChild)),
   setLinks(CurrentNode, Neighbours).

appendHeuristics(_, _, [], PQueue, PQueue).         %add heuristic to the 
priority queue
appendHeuristics(Destination, CurrentNode, [ChildNode|Neighbours], PQueue, 
FinalPQueue) :-
   (ChildNode == Destination ->
       (assert(flag(1)), appendHeuristics(Destination, CurrentNode, [], 
PQueue, FinalPQueue), !);
       (heuristic(Destination, ChildNode, DirectDistance),
       add_to_heap(PQueue, DirectDistance, ChildNode, NewPQueue),
       appendHeuristics(Destination, CurrentNode, Neighbours, NewPQueue, 
FinalPQueue))
       ).


%---------
```

```
performDfs(_, [], _) :-
   nl, ansi_format([bold, fg(red)], '~w', ['No path found.']), nl.

performDfs(Destination, [Destination|_], _) :-
   path(Path, Destination, Distance),
   reverse(Path, ActualPath), nl,
   ansi_format([bold, fg(green)], '~w', ['Path found!']),nl,
   printPath(ActualPath), nl,
   ansi_format([fg(blue)], '~w ~w ', ['Distance:', Distance]), !.

performDfs(Destination, [Current|Stack], Visited) :-
   findall(Next, (road(Next, Current, _ ), \+ (member(Next, Visited))),
Neighbours),
   append(Neighbours, Stack, NewStack),
   setLinks(Current, Neighbours),
   performDfs(Destination, NewStack, [Current|Visited]).


%---------

performBestfs(_, PQueue, _) :-
   empty_heap(PQueue),
   nl, ansi_format([bold, fg(red)], '~w', ['No path found.']), nl.

performBestfs(Destination, _, _) :-
   (flag(X), X==1),
   path(Path, Destination, Distance),
   reverse(Path, ActualPath), nl,
   ansi_format([bold, fg(green)], '~w', ['Path found!']),nl,
   printPath(ActualPath), nl,
   ansi_format([fg(blue)], '~w ~w ', ['Distance:', Distance]), !.

performBestfs(Destination, PQueue, Visited) :-
   get_from_heap(PQueue, _, Current, NewPQueue),
   findall(Next, (road(Next, Current, _), \+ (member(Next, Visited))),
Neighbours),
   appendHeuristics(Destination, Current, Neighbours, NewPQueue,
FinalPQueue),
   setLinks(Current, Neighbours),
   performBestfs(Destination, FinalPQueue, [Current|Visited]).


%----------
```

```
depthFirstSearch(Start, Destination) :-
   performDfs(Destination, [Start], []).

bestFirstSearch(Start, Destination) :-
   empty_heap(Init),
   (heuristic(Start, Destination, Distance) -> (
       add_to_heap(Init, Distance, Start, PQueue),
performBestfs(Destination, PQueue, []));
       nl, ansi_format([bold, fg(red)], '~w', ['Heuristic does not exist.
Exiting.']), nl).

handleSearch(Start, Destination, Choice) :-
   assert(path([Start], Start, 0)), % path going to Start using path
[Start] of distance=0
   (Choice==1 ->
       depthFirstSearch(Start, Destination);
       Choice==2 ->
           bestFirstSearch(Start, Destination);
       write("Wrong choice")).

start:-
   retractall(path(_,_,_)),
   retractall(flag(_)),
   write("Enter the origin: "), read(Start),
   write("Enter the destination: "), read(Destination),
   write("Which algorithm would you like to use to get the path?\n1. Depth
First Search\n2. Best First Search\n"), read(Choice),
   handleSearch(Start, Destination, Choice), nl.
```

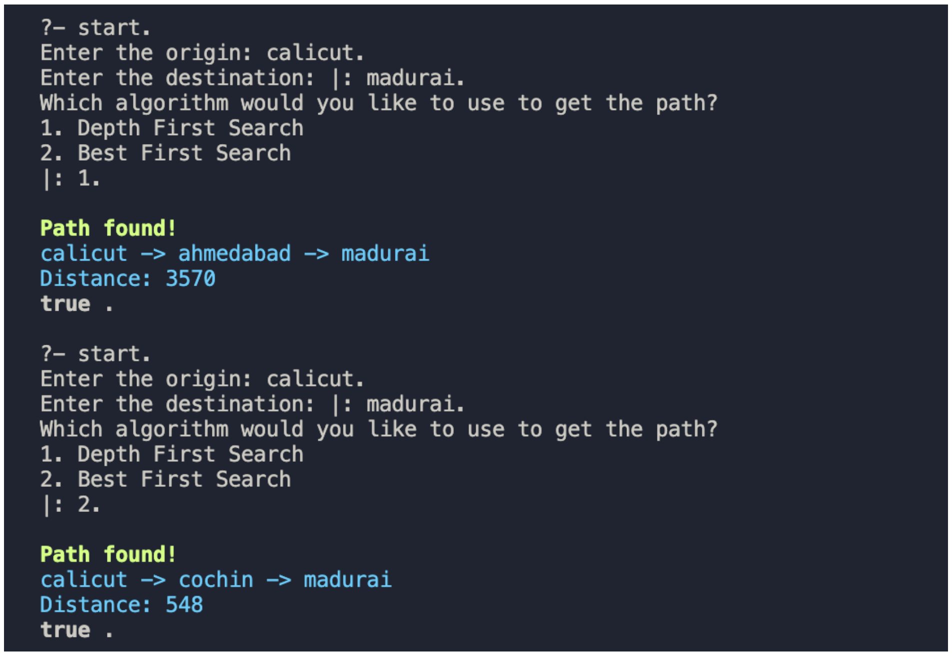
Screenshots

1. From Kanpur to Pune (directly connected).



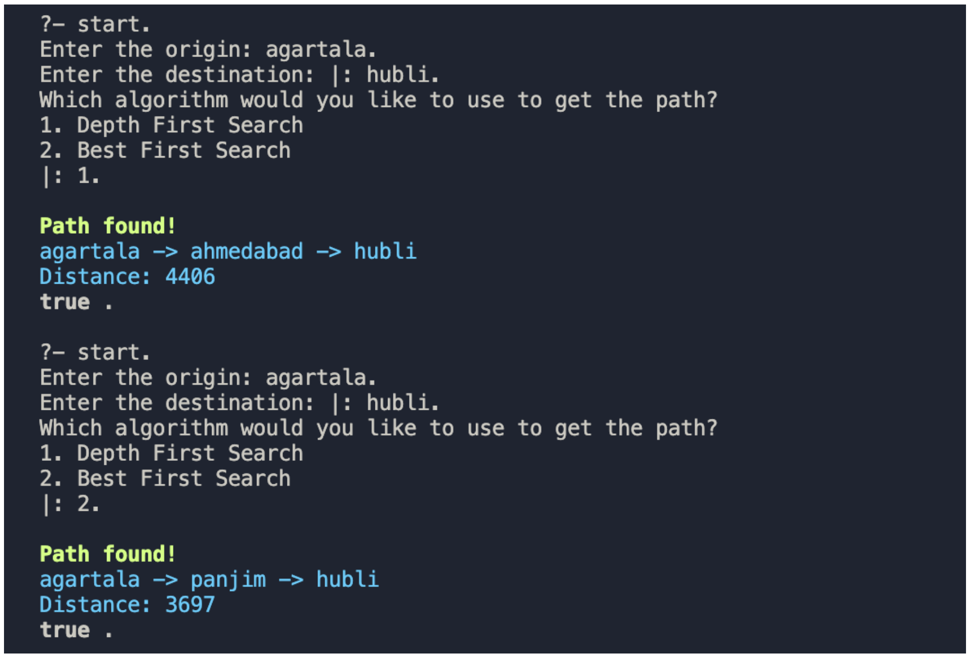
```
?- consult("find-path.pl").
true.

?- start.
Enter the origin: kanpur.
Enter the destination: |: pune.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 1.

Path found!
kanpur -> pune
Distance: 1312
true .

?- start.
Enter the origin: kanpur.
Enter the destination: |: pune.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 2.

Path found!
kanpur -> pune
Distance: 1312
true .
```

2. From Calicut to Madurai (not directly connected).



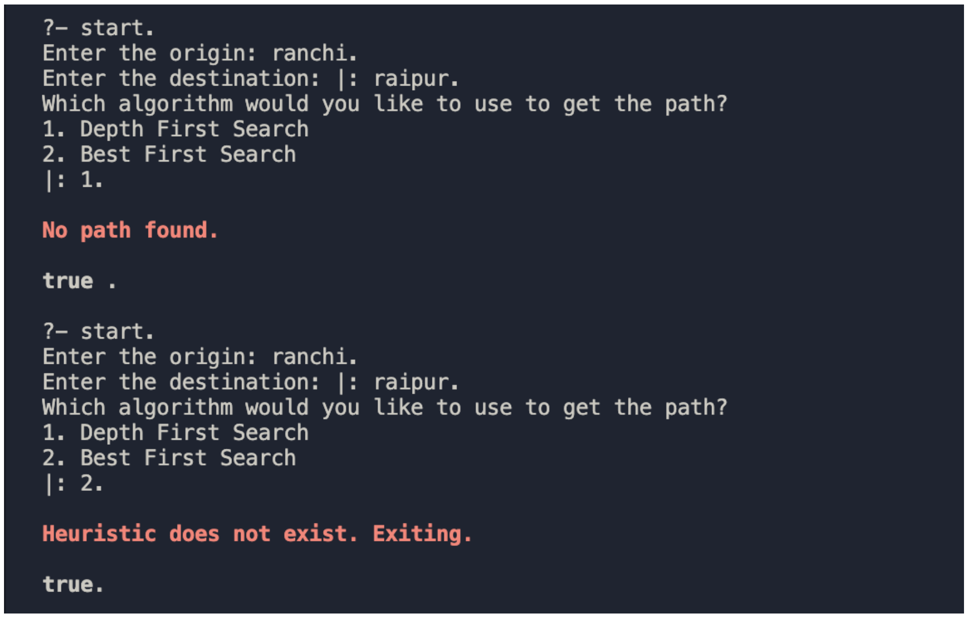
```
?- start.
Enter the origin: calicut.
Enter the destination: |: madurai.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 1.

Path found!
calicut -> ahmedabad -> madurai
Distance: 3570
true .

?- start.
Enter the origin: calicut.
Enter the destination: |: madurai.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 2.

Path found!
calicut -> cochin -> madurai
Distance: 548
true .
```

3. From Agartala to Hubli (not directly connected)



```
?- start.
Enter the origin: agartala.
Enter the destination: |: hubli.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 1.

Path found!
agartala -> ahmedabad -> hubli
Distance: 4406
true .

?- start.
Enter the origin: agartala.
Enter the destination: |: hubli.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 2.

Path found!
agartala -> panjim -> hubli
Distance: 3697
true .
```

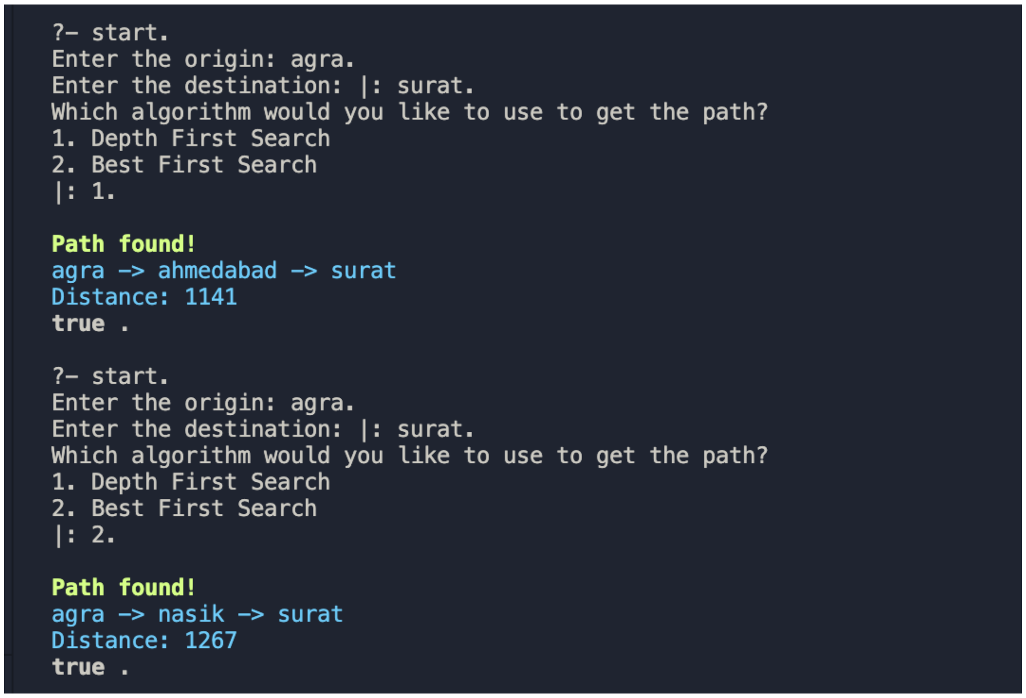
4. From Ranchi to Raipur (Raipur is not a valid city as per the given data).



```
?- start.
Enter the origin: ranchi.
Enter the destination: |: raipur.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 1.

No path found.

true .

?- start.
Enter the origin: ranchi.
Enter the destination: |: raipur.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 2.

Heuristic does not exist. Exiting.

true.
```

Caveats

It might look surprising at the first glance but there might be cases when DFS might return a better path than best-first search for the same source and destination. Please note that neither of these search methods are guaranteed to return the most optional path. The path that the naïve DFS returns is solely based on the chance encounter of an optimal intermediate node (and the way the nodes are arranged in the stack for that matter). For Best first search however, selection of a node at each step solely on the heuristic i.e. the direct distance from that node to the destination. This might lead to a case such as this:


```
?- start.
Enter the origin: agra.
Enter the destination: |: surat.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 1.

Path found!
agra -> ahmedabad -> surat
Distance: 1141
true .

?- start.
Enter the origin: agra.
Enter the destination: |: surat.
Which algorithm would you like to use to get the path?
1. Depth First Search
2. Best First Search
|: 2.

Path found!
agra -> nasik -> surat
Distance: 1267
true .
```

Ahmedabad was chosen by DFS solely because it gets appended at the start of the stack (being Agra's first enlisted neighbour). Let us now see the heuristic and actual distance for Ahmedabad to Surat and Nasik to Surat:

	Actual (Road) Distance	**Heuristic (Air Distance)**
Nasik to Surat	262	167
Ahmedabad to Surat	263	207

Clearly enough, best first search would choose the via Nasik to Surat route because Nasik is the first on the priority queue. Now, best first search is not going to consider the actual cost of reaching Nasik from Agra. It is just concerned about choosing the node closest to the destination heuristically.

	Actual (Road) Distance
Agra to Nasik	1005
Agra to Ahmedabad	878

Nasik gets picked out of the priority queue before Ahmedabad based on the heuristic distance to Surat alone. Therefore, even though via Ahmedabad turns out to be a better route, the via Nasik route is chosen. Since best first search stops after finding the first solution and does not continue searching anymore, we are left with a good, but not the optimal solution.
