

# Assignment 2

## Search a road route from any city to any other city

First of all, it is essential to fetch the facts, after all entire work is based on that. To achieve that, the .xlsx file is converted to .csv using online converter because there is facility is prolog to deal with .csv files. After .csv file is ready (named: 'roaddistance.csv'), next task is to convert .csv file to facts and store all the facts in 'facts.pl'. A program can be written in prolog to do that work easily. Simply it has to iterate through all rows and columns and then assert the facts accordingly. To write facts in 'fact.pl' write

```
save:-
    tell('facts.pl'),
    listing(distance),
    told.
```

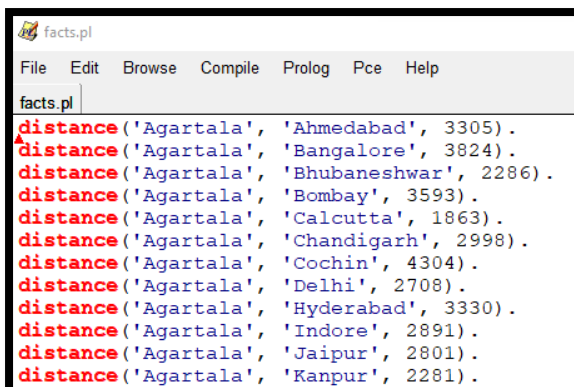
To run the program, type the following.

```
?- csv('roaddistance.csv').
true.

?- save.
true.
```

Now 'facts.pl' has all the facts which are given but distance(A,A,-) is not required as it is, but required after distance is made 0. Now, 'facts.pl' is completely ready to be used for input to algorithms for road route search.

The snapshot of 'facts.pl' is shown below.

A screenshot of a Prolog IDE window titled 'facts.pl'. The window has a menu bar with 'File', 'Edit', 'Browse', 'Compile', 'Prolog', 'Pce', and 'Help'. The main text area contains a list of facts for the 'distance' predicate, all starting with 'Agartala' as the first argument. The facts are: distance('Agartala', 'Ahmedabad', 3305)., distance('Agartala', 'Bangalore', 3824)., distance('Agartala', 'Bhubaneshwar', 2286)., distance('Agartala', 'Bombay', 3593)., distance('Agartala', 'Calcutta', 1863)., distance('Agartala', 'Chandigarh', 2998)., distance('Agartala', 'Cochin', 4304)., distance('Agartala', 'Delhi', 2708)., distance('Agartala', 'Hyderabad', 3330)., distance('Agartala', 'Indore', 2891)., distance('Agartala', 'Jaipur', 2801)., and distance('Agartala', 'Kanpur', 2281).

```
facts.pl
distance('Agartala', 'Ahmedabad', 3305).
distance('Agartala', 'Bangalore', 3824).
distance('Agartala', 'Bhubaneshwar', 2286).
distance('Agartala', 'Bombay', 3593).
distance('Agartala', 'Calcutta', 1863).
distance('Agartala', 'Chandigarh', 2998).
distance('Agartala', 'Cochin', 4304).
distance('Agartala', 'Delhi', 2708).
distance('Agartala', 'Hyderabad', 3330).
distance('Agartala', 'Indore', 2891).
distance('Agartala', 'Jaipur', 2801).
distance('Agartala', 'Kanpur', 2281).
```

Out of the two choices given, I would like to go for (a) choice which is Depth First Search and Best First Search. This report will firstly discuss Depth First Search and then later on Best First Search.

## Depth First Search (DFS)

DFS explores the nodes depth wise. So simply traversing from root to the leaf would work and then there is inherent backtracking, so that would help.

Now, the path can be stored into list and distance can be calculated along with.

```
% DFS solution
dfsroute(SRC,DST,[SRC|Further],Distance):-
    connected(SRC,DST,Further,Distance).
% rules for direct path or indirect path
connected(SRC,DST,[DST],Distance):-
    dist(SRC,DST,Distance).
connected(SRC,DST,[Intermediate|L],Distance):-
    dist(SRC,Intermediate,Distance1),
    connected(Intermediate,DST,L,Distance2),
    Distance is Distance1 + Distance2.
```

One thing to notice is that facts had distance(X,Y,230) format while here, the fact of format dist(X,Y,230) is used. This question is answered in figure below. This is done to handle bidirectionality.

```
dist(X,Y,D):-
    distance(X,Y,D).
dist(X,Y,D):-
    distance(Y,X,D).
```

Find the screenshot for only the Depth First Search as of now below.

-----  
As per Depth First Search

-----  
Your Road Route should be: [Baroda,Ahmedabad,Hubli]  
The distance for the above path is: 1220

## Best First Search

Best first search makes use of only the heuristics, to reach the goal. Hence, it is utmost important to create heuristics. Heuristics can be created by the code as under.

```
% iterate through list
itr([],DST).
itr([H|T],DST) :- process(H,DST), itr(T,DST).

process(H,DST):-
    dfsroute(H,DST,_,TheDist),
    HeurDist is TheDist - 10,
    assert(heur(H,DST,HeurDist)).

% function to create heuristics for Best First Search
createheuristics(DST):-
    ListforHeur = ['Agartala','Agra','Ahmedabad','Allahabad','Amritsar','Asansol',
        'Bangalore','Bhopal','Bhubaneshwar','Bombay','Calcutta',
        'Calicut','Chandigarh','Cochin','Coimbatore',
        'Delhi','Gwalior','Hubli','Hyderabad','Imphal','Indore',
        'Jabalpur','Jaipur','Jamshedpur','Jullundur','Kanpur','Kolhapur',
        'Lucknow','Ludhiana','Madras','Madurai','Meerut','Nagpur','Nasik',
        'Panjim','Patna','Pondicherry','Pune','Ranchi','Shillong','Shimla',
        'Surat','Trivandrum','Varanasi','Vijayawada','Vishakapatnam'],
    itr(ListforHeur,DST),
    tell('heuristics.pl'),
    listing(heur),
    told.
```

Firstly, the destination has to be passed to createheuristics(DST). There, the entire list of cities is maintained and later on itr(ListforHeur,DST) is called whose main function is to iterate through elements of list and then create heuristics through process(H,DST) where heuristics are calculated through DFS distance between head of list and the destination city. To underestimate, it is subtracted with 10 so that heuristic always underestimates. Also, since it is made from original distance in DFS fashion, so heuristic is consistent as well.

Snapshot for heuristics.pl is as under.

```
heur('Agartala', 'Baroda', 3414).
heur('Agra', 'Baroda', 987).
heur('Ahmedabad', 'Baroda', 109).
heur('Allahabad', 'Baroda', 1360).
heur('Amritsar', 'Baroda', 1465).
heur('Asansol', 'Baroda', 1951).
heur('Bangalore', 'Baroda', 1398).
heur('Bhopal', 'Baroda', 632).
heur('Bhubaneshwar', 'Baroda', 1594).
heur('Bombay', 'Baroda', 423).
heur('Calcutta', 'Baroda', 1927).
heur('Calicut', 'Baroda', 1757).
heur('Chandigarh', 'Baroda', 1171).
```

Now, comes the code for Best First Search. The concepts are taken from Dr. Albert Cruz. As, is the algorithm, step one is to find successors and then store them in priority queue. There are two queues, closed and open. It is somewhat like that.

```
bestfirstsearchroute([ [DST|Path] | _ ], DST, [DST|Path], 0) .
bestfirstsearchroute([Path|Queue], DST, FinalPath, DistanceBFS) :-
    findsuccesortoexpand(Path, AdditionalPath),
    append(Queue, AdditionalPath, Queue1),
    sort_pqueue_one(Queue1, NewQueue),
    bestfirstsearchroute(NewQueue, DST, FinalPath, Dist),
    DistanceBFS is Dist+1.
```

To find the successor, it checks direct distance given by  $\text{dist}(X,Y,D)$  and also, so that it doesn't go back to cities which are already taken into path, it checks through negation of `member()`.

```
findsuccesortoexpand([Node|Path], AdditionalPath) :-
    findall([NewNode, Node|Path],
        (dist(Node, NewNode, _),
         \+ member(NewNode, Path)),
        AdditionalPath) .
```

Rest is the code so as to maintain the priority queue.

See the output for Best First Search below.

-----  
As per Best First Search

-----  
Your Road Route should be (Reverse Order): [Baroda,Ahmedabad,Allahabad]  
The approximate distance for the above path is: 1360

The complete output is shown below.

?- | start.

-----  
Welcome to Ricerca - Your Road Route Search Guide  
-----

Dear User! Please help me to help you by providing your  
Source: 'Allahabad'.  
Destination: |: 'Baroda'.

-----  
As per Depth First Search  
-----

Your Road Route should be: [Allahabad,Ahmedabad,Baroda]  
The distance for the above path is: 1370

-----  
As per Best First Search  
-----

Your Road Route should be (Reverse Order): [Baroda,Ahmedabad,Allahabad]  
The approximate distance for the above path is: 1360  
true .

?- start.

-----  
Welcome to Ricerca - Your Road Route Search Guide  
-----

Dear User! Please help me to help you by providing your  
Source: 'Jullundur'.  
Destination: |: 'Kanpur'.

-----  
As per Depth First Search  
-----

Your Road Route should be: [Jullundur,Kanpur]  
The distance for the above path is: 855

-----  
As per Best First Search  
-----

Your Road Route should be (Reverse Order): [Kanpur,Jullundur]  
The approximate distance for the above path is: 845  
true .