# ASSIGNMENT-2

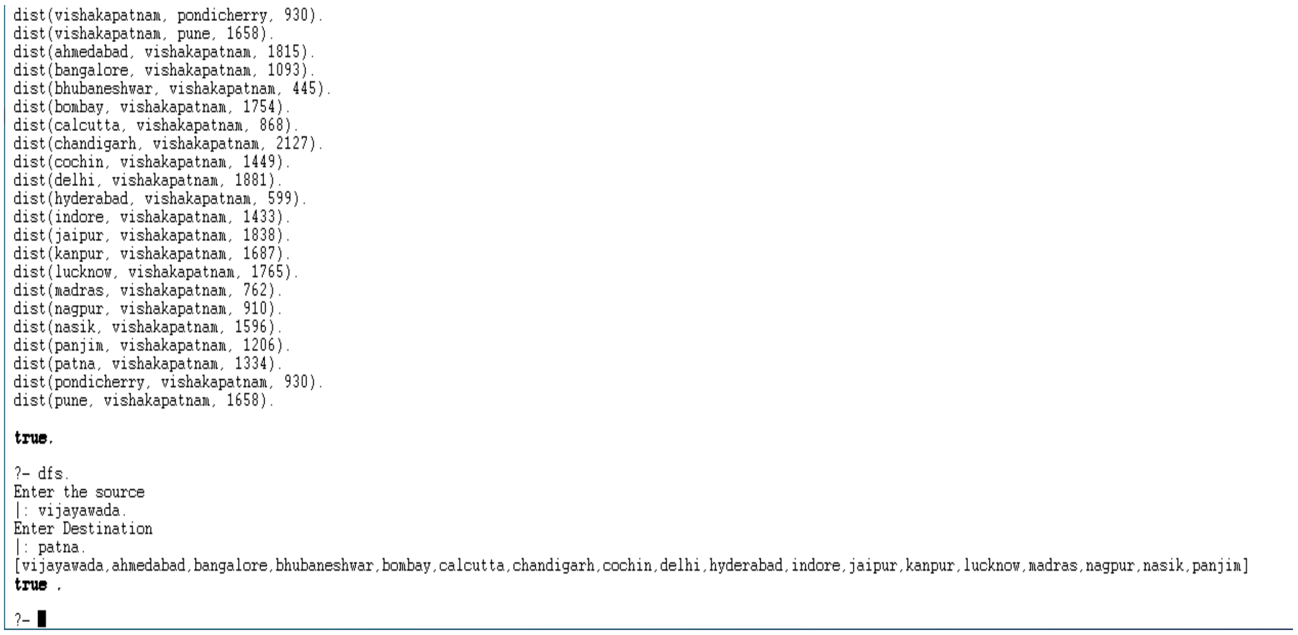
# DFS AND BFS IMPLEMENTATION

# REPORT

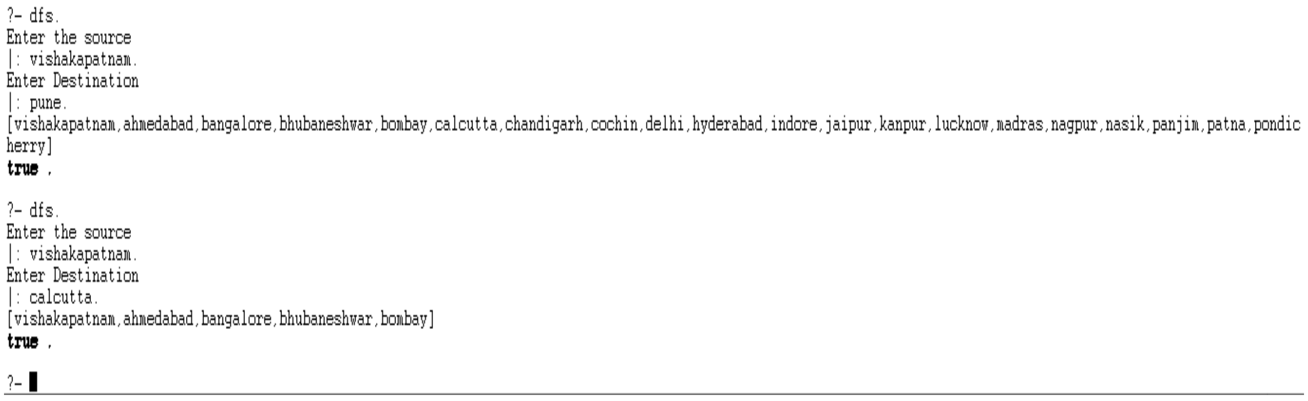
## EXPLANATION :-

In this project we are trying to implement depth first search and best first search for a given csv file. In our project we are given a roaddistance.csv file and using that file we are creating facts which have distance between two cities and also its heuristics between these two cities. Using these facts that we created, we are trying to find the shortest path between any two given nodes taken as input from the user using Depth First Search(DFS) and Best First Search(BFS) algorithms.

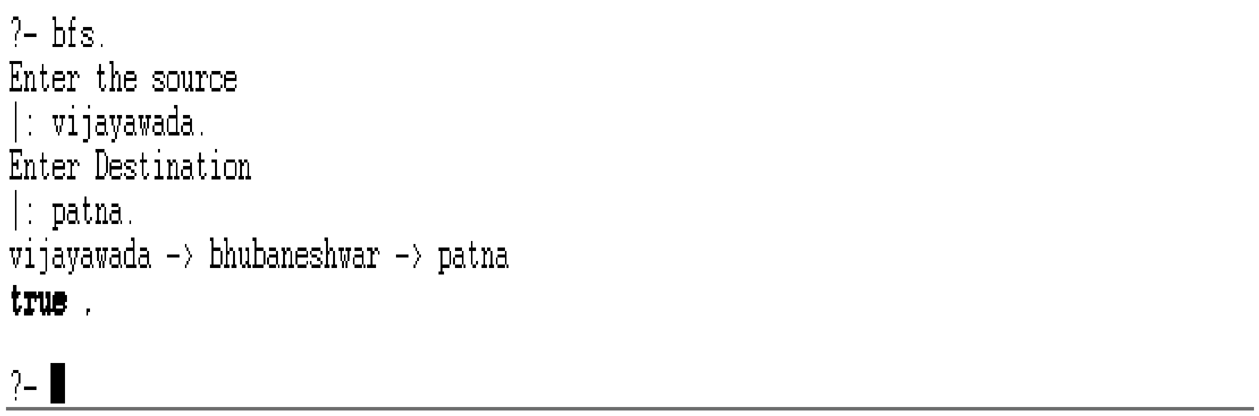
# <u>OUTPUT</u> :-



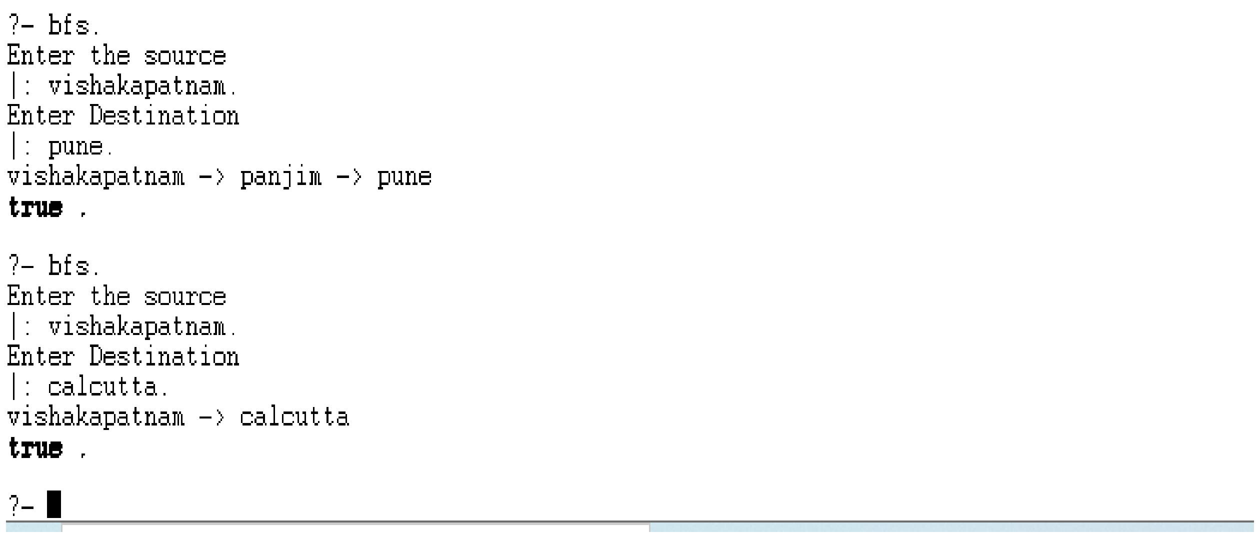
```
dist(vishakapatnam, pondicherry, 930).
dist(vishakapatnam, pune, 1658).
dist(ahmedabad, vishakapatnam, 1815).
dist(bangalore, vishakapatnam, 1093).
dist(bhubaneshwar, vishakapatnam, 445).
dist(bombay, vishakapatnam, 1754).
dist(calcutta, vishakapatnam, 868).
dist(chandigarh, vishakapatnam, 2127).
dist(cochin, vishakapatnam, 1449).
dist(delhi, vishakapatnam, 1881).
dist(hyderabad, vishakapatnam, 599).
dist(indore, vishakapatnam, 1433).
dist(jaipur, vishakapatnam, 1838).
dist(kanpur, vishakapatnam, 1687).
dist(lucknow, vishakapatnam, 1765).
dist(madras, vishakapatnam, 762).
dist(nagpur, vishakapatnam, 910).
dist(nasik, vishakapatnam, 1596).
dist(panjim, vishakapatnam, 1206).
dist(patna, vishakapatnam, 1334).
dist(pondicherry, vishakapatnam, 930).
dist(pune, vishakapatnam, 1658).

true.

?- dfs.
Enter the source
|: vijayawada.
Enter Destination
|: patna.
[vijayawada,ahmedabad,bangalore,bhubaneshwar,bombay,calcutta,chandigarh,cochin,delhi,hyderabad,indore,jaipur,kanpur,lucknow,madras,nagpur,nasik,panjim]
true .

?-
```



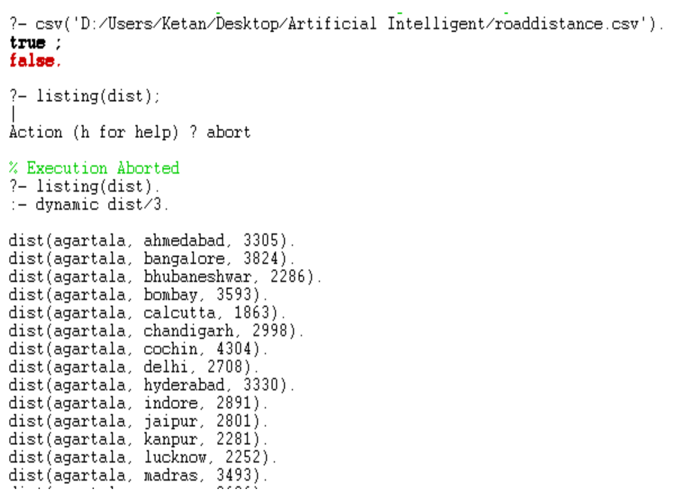
```
?- dfs.
Enter the source
|: vishakapatnam.
Enter Destination
|: pune.
[vishakapatnam,ahmedabad,bangalore,bhubaneshwar,bombay,calcutta,chandigarh,cochin,delhi,hyderabad,indore,jaipur,kanpur,lucknow,madras,nagpur,nasik,panjim,patna,pondic
herry]
true .

?- dfs.
Enter the source
|: vishakapatnam.
Enter Destination
|: calcutta.
[vishakapatnam,ahmedabad,bangalore,bhubaneshwar,bombay]
true .

?-
```



```
?- bfs.
Enter the source
|: vijayawada.
Enter Destination
|: patna.
vijayawada -> bhubaneshwar -> patna
true .

?-
```



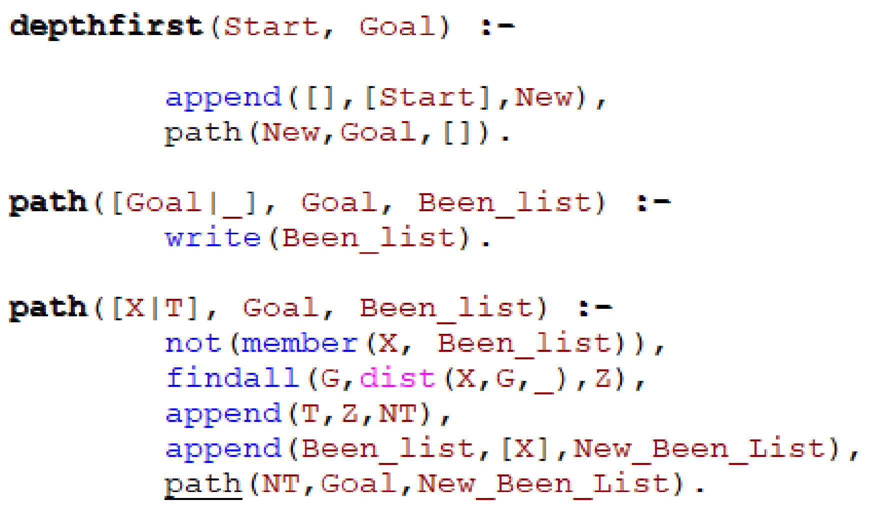
```
?- bfs.
Enter the source
|: vishakapatnam.
Enter Destination
|: pune.
vishakapatnam -> panjim -> pune
true .

?- bfs.
Enter the source
|: vishakapatnam.
Enter Destination
|: calcutta.
vishakapatnam -> calcutta
true .

?-
```



```
?- csv('D:/Users/Ketan/Desktop/Artificial Intelligent/roaddistance.csv').
true ;
false.

?- listing(dist);
|
Action (h for help) ? abort

% Execution Aborted
?- listing(dist).
:- dynamic dist/3.

dist(agartala, ahmedabad, 3305).
dist(agartala, bangalore, 3824).
dist(agartala, bhubaneshwar, 2286).
dist(agartala, bombay, 3593).
dist(agartala, calcutta, 1863).
dist(agartala, chandigarh, 2998).
dist(agartala, cochin, 4304).
dist(agartala, delhi, 2708).
dist(agartala, hyderabad, 3330).
dist(agartala, indore, 2891).
dist(agartala, jaipur, 2801).
dist(agartala, kanpur, 2281).
dist(agartala, lucknow, 2252).
dist(agartala, madras, 3493).
```

## FEATURES USED :-

### 1. LISTS :-



```
depthfirst(Start, Goal) :-

        append([],[Start],New),
        path(New,Goal,[]).

path([Goal|_], Goal, Been_list) :-
        write(Been_list).

path([X|T], Goal, Been_list) :-
        not(member(X, Been_list)),
        findall(G,dist(X,G,_),Z),
        append(T,Z,NT),
        append(Been_list,[X],New_Been_List),
        path(NT,Goal,New_Been_List).
```

### 2. RECURSION :-

```
%Sorting Function
start(New1,Dest,List,Main,Sorted):-
    New1=[H|T],
    dist(H,Dest,K),
    toList(K,Dist),
    append(List,Dist,New),
    start(T,Dest,New,Main,Sorted).

start(New1,Dest,List,Main,Sorted):-
    New1=[_|T],
    start(T,Dest,List,Main,Sorted).
```

```
rm_duplicates([], []).

rm_duplicates([H|T],R) :-
    member(H,T), !,
    rm_duplicates(T,R).

rm_duplicates([H|T], [H|R]) :-
    rm_duplicates(T, R).
```

## 3. **BINDING** :-

```
distance_sort(_,[],_,Temp,Temp):-!.

distance_sort(List,Main,Dest,Temp,Sorted):-
    Main=[H|T],
    dist(A,Dest,H),
    toList(A,Node),
    append(Temp,Node,Temp1),
    distance_sort(List,T,Dest,Temp1,Sorted).
```

## 4. <u>BACKTRACKING</u> :-

```
path2(Src,Src,_,Closed):-
    rm_duplicates(Closed,New),
    rev(New,New1,[]),
    pr(New1),
    write(Src),
    write('\n').

path2(Src,Dest,Open,Closed):-
    hlist(Src,Nodes),
    append(Nodes,Open,New1),
    toList(Src,Done),
    append(Done,Closed,Close),
    sort1(New1,Dest,Sorted),
    path3(Src,Dest,New1,Close,Sorted).

path3(_,Dest,Open,Closed,Sorted):-
    Sorted=[H|_],
    path2(H,Dest,Open,Closed).
```

## 5. <u>ASSERT</u> :-

```
(loop) csv_ (HEADERs,[ROW|ROWss])
:-
row__to__list(ROW,ROWs) ,
lists:nth1(1,ROWs,CITY_A) ,
QUERY_A=(lists:nth1(NTH,ROWs,DISTANCE)) ,
QUERY_B=(NTH > 1) ,
QUERY_C=(lists:nth1(NTH,HEADERs,CITY_B)) ,
QUERY=(QUERY_A,QUERY_B,QUERY_C) ,
ASSERT=assertz(dist(CITY_A,CITY_B,DISTANCE)) ,
forall(QUERY,ASSERT) ,
As1=assertz(dist(CITY_B,CITY_A,DISTANCE)) ,
forall(QUERY,As1) ,
(loop) csv_ (HEADERs,ROWss)
.
```

## <u>RESULT:-</u>

**SEARCHING USING DEPTH FIRST SEARCH AND BEST FIRST SEARCH HAS BEEN SUCCESSFULLY IMPLEMENTED.**