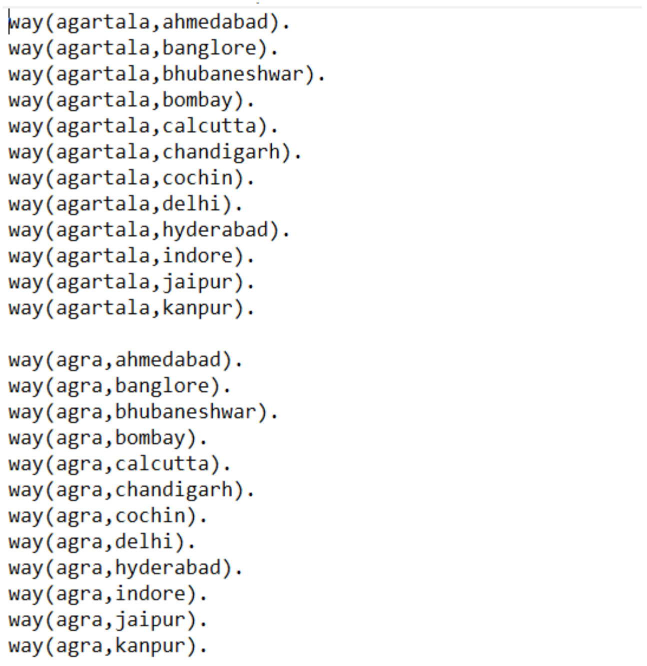
Given: Road distance map. Distance between the cities.

To find: Path from one city to another using:

1. Breadth-first-search
2. A* Algorithm

Breadth First Search

1. Clauses: - clauses here are the routes/ways from one city to another. How can we reach a destination city from source given? This is made according to the road map given. The distance between the source and destination has been also given.
   way(agra,bombay,1202).
   Where to reach bombay from agra the distance is 1202.

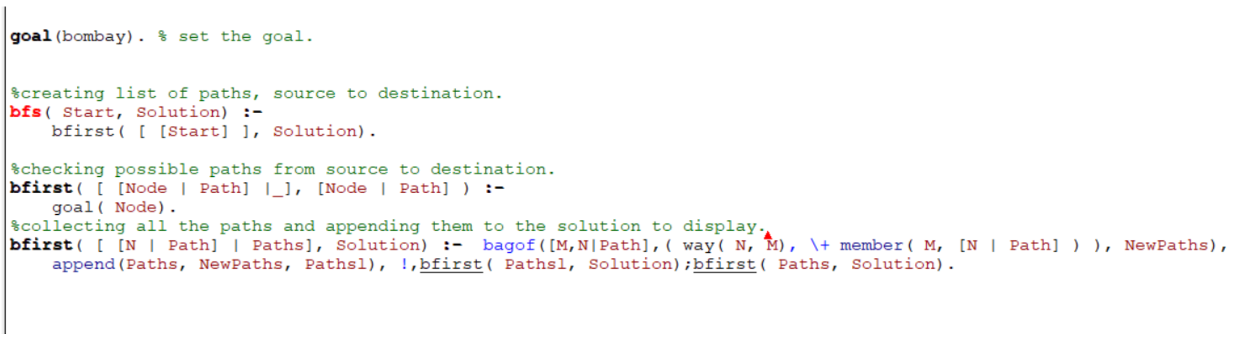


```
way(agartala,ahmedabad).
way(agartala,banglore).
way(agartala,bhubaneshwar).
way(agartala,bombay).
way(agartala,calcutta).
way(agartala,chandigarh).
way(agartala,cochin).
way(agartala,delhi).
way(agartala,hyderabad).
way(agartala,indore).
way(agartala,jaipur).
way(agartala,kanpur).

way(agra,ahmedabad).
way(agra,banglore).
way(agra,bhubaneshwar).
way(agra,bombay).
way(agra,calcutta).
way(agra,chandigarh).
way(agra,cochin).
way(agra,delhi).
way(agra,hyderabad).
way(agra,indore).
way(agra,jaipur).
way(agra,kanpur).
```

2. Applying Breadth First Search
3. Idea:
   - Maintaining a list of paths from source to destination.
   - Checking for all the possible paths by visiting each path from source to destination.
   - bagof: collecting all the possible paths to append to the solution.

- Displaying the paths from source to destination.

Code:

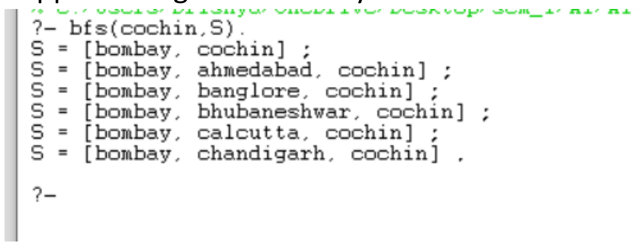


```
goal(bombay). % set the goal.

%creating list of paths, source to destination.
bfs( Start, Solution) :-
    bfirst( [ [Start] ], Solution).

%checking possible paths from source to destination.
bfirst( [ [Node | Path] |_], [Node | Path] ) :-
    goal( Node).
%collecting all the paths and appending them to the solution to display.
bfirst( [ [N | Path] | Paths], Solution) :-  bagof([M,N|Path],( way( N, M), \+ member( M, [N | Path] ) ), NewPaths),
    append(Paths, NewPaths, Paths1), !,bfirst( Paths1, Solution);bfirst( Paths, Solution).
```

Output:

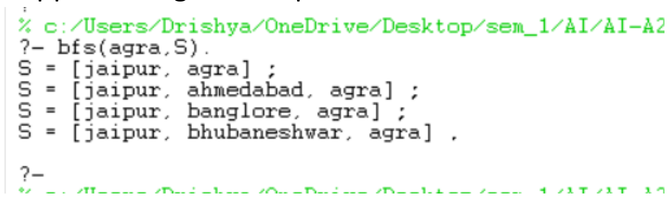
The goal has been set by default in the program.

1. Suppose the goal is Bombay.



```
?- bfs(cochin,S).
S = [bombay, cochin] ;
S = [bombay, ahmedabad, cochin] ;
S = [bombay, banglore, cochin] ;
S = [bombay, bhubaneshwar, cochin] ;
S = [bombay, calcutta, cochin] ;
S = [bombay, chandigarh, cochin] .

?-
```

2. Suppose the goal is Jaipur.

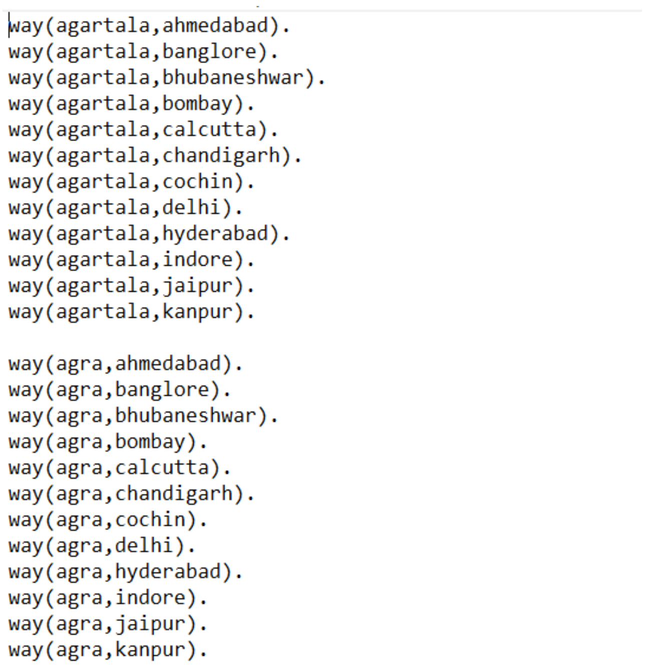


```
% c:/Users/Drishya/OneDrive/Desktop/sem_1/AI/AI-A2
?- bfs(agra,S).
S = [jaipur, agra] ;
S = [jaipur, ahmedabad, agra] ;
S = [jaipur, banglore, agra] ;
S = [jaipur, bhubaneshwar, agra] .

?-
```

The bfs is going to show all the possible routes.

<u>A* Search Algorithm</u>

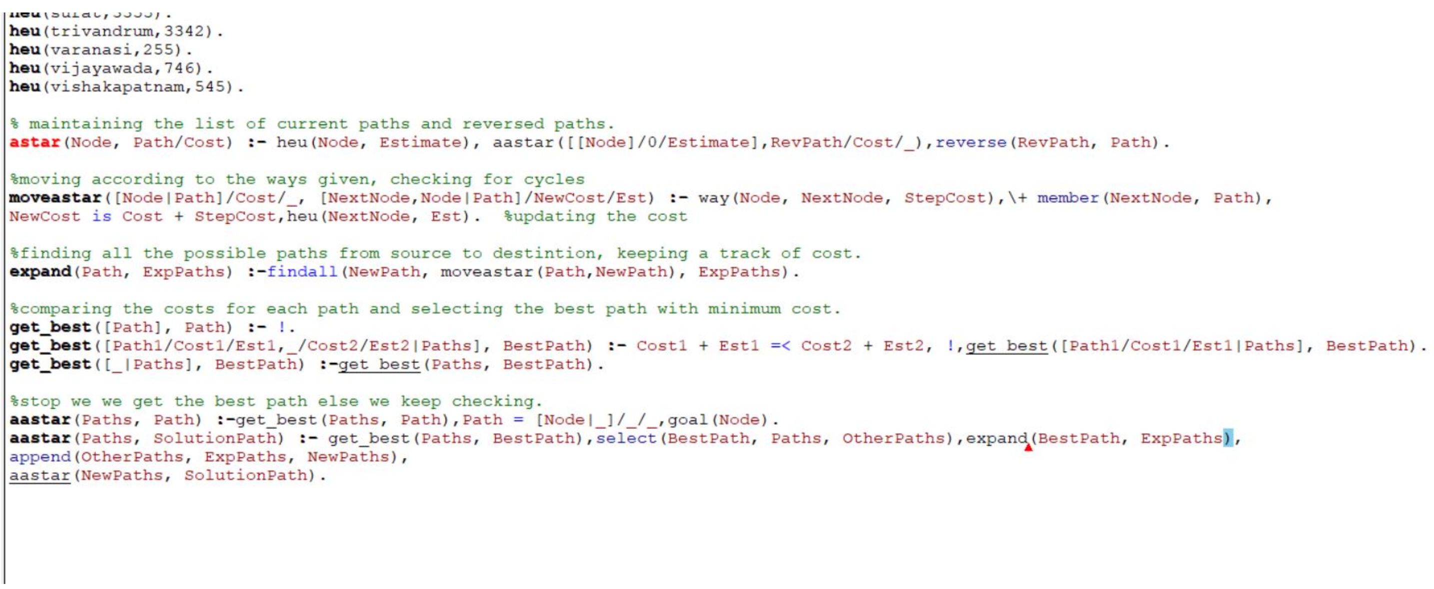
1. Clauses: - clauses here are the routes/ways from one city to another. How can we reach a destination city from source given. This is made according to the road map given. The distance between the source and destination has been also given.
   way(agra,bombay,1202).
   Where to reach bombay from agra the distance is 1202.



```
way(agartala,ahmedabad).
way(agartala,banglore).
way(agartala,bhubaneshwar).
way(agartala,bombay).
way(agartala,calcutta).
way(agartala,chandigarh).
way(agartala,cochin).
way(agartala,delhi).
way(agartala,hyderabad).
way(agartala,indore).
way(agartala,jaipur).
way(agartala,kanpur).

way(agra,ahmedabad).
way(agra,banglore).
way(agra,bhubaneshwar).
way(agra,bombay).
way(agra,calcutta).
way(agra,chandigarh).
way(agra,cochin).
way(agra,delhi).
way(agra,hyderabad).
way(agra,indore).
way(agra,jaipur).
way(agra,kanpur).
```

2. Applying A* Algorithm
   Idea:
   (a) Estimating the heuristic function to reach goal.
   (b) Taking into consideration the cost of path so far.
       We know:
       f(n) = g(n) + h(n).
       where n is the node,
       g(n) is cost of moving from initial node to n.
       h(n) is estimated cost of reaching a goal node from n.
   (c) We have to implement the code as:
       - Given the current state we have to move to next state and check it with cost. All possible ways should be generated by backtracking.
       - If we reach the goal state then it's a success.
       - Estimating the cost how we can reach the goal state.
   (d) We maintain a list of paths and also reversed paths so that we can keep track of every node and their cost.
   (e) We make a move function that actually checks for any cycles and updates the cost of current path and checks estimate.
   (f) Implementing the search for A* by getting the best-path. We have the list of paths; we compare the current cost and estimate and return the minimum sum f(n).
   (g) We stop if we get the best path else, we keep checking.
   (h) For different destinations the values of h(n) should be different but in this case I have considered a common value and the code is more general.

Code:

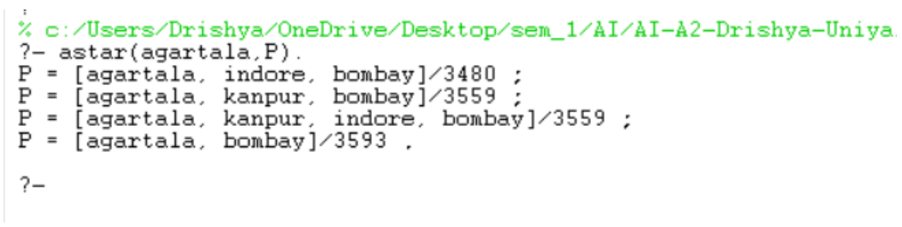


```
heu(trivandrum,3342).
heu(varanasi,255).
heu(vijayawada,746).
heu(vishakapatnam,545).

% maintaining the list of current paths and reversed paths.
astar(Node, Path/Cost) :- heu(Node, Estimate), aastar([[Node]/0/Estimate],RevPath/Cost/_),reverse(RevPath, Path).

%moving according to the ways given, checking for cycles
moveastar([Node|Path]/Cost/_, [NextNode,Node|Path]/NewCost/Est) :- way(Node, NextNode, StepCost),\+ member(NextNode, Path),
NewCost is Cost + StepCost,heu(NextNode, Est).  %updating the cost

%finding all the possible paths from source to destintion, keeping a track of cost.
expand(Path, ExpPaths) :-findall(NewPath, moveastar(Path,NewPath), ExpPaths).

%comparing the costs for each path and selecting the best path with minimum cost.
get_best([Path], Path) :- !.
get_best([Path1/Cost1/Est1,_/Cost2/Est2|Paths], BestPath) :- Cost1 + Est1 =< Cost2 + Est2, !,get_best([Path1/Cost1/Est1|Paths], BestPath).
get_best([_|Paths], BestPath) :-get_best(Paths, BestPath).

%stop we we get the best path else we keep checking.
aastar(Paths, Path) :-get_best(Paths, Path),Path = [Node|_]/_/_,goal(Node).
aastar(Paths, SolutionPath) :- get_best(Paths, BestPath),select(BestPath, Paths, OtherPaths),expand(BestPath, ExpPaths),
append(OtherPaths, ExpPaths, NewPaths),
aastar(NewPaths, SolutionPath).
```

Output:

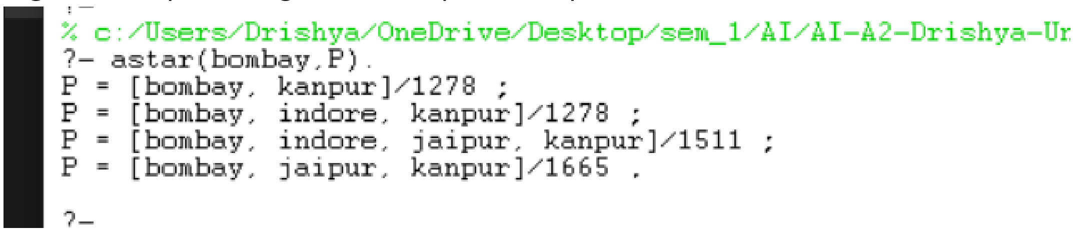
By default the goal has been given in the code itself so we need to pass the source.

1. If goal is Bombay say. The best path is at top with cost 3480.



```
% c:/Users/Drishya/OneDrive/Desktop/sem_1/AI/AI-A2-Drishya-Uniya
?- astar(agartala,P).
P = [agartala, indore, bombay]/3480 ;
P = [agartala, kanpur, bombay]/3559 ;
P = [agartala, kanpur, indore, bombay]/3559 ;
P = [agartala, bombay]/3593 .

?-
```

2. If goal is Kanpur. We get the best path at top.



```
% c:/Users/Drishya/OneDrive/Desktop/sem_1/AI/AI-A2-Drishya-Ur
?- astar(bombay,P).
P = [bombay, kanpur]/1278 ;
P = [bombay, indore, kanpur]/1278 ;
P = [bombay, indore, jaipur, kanpur]/1511 ;
P = [bombay, jaipur, kanpur]/1665 .

?-
```