

CSPC 54 : Prolog Assignment-3

Name : Rajneesh Pandey,

Roll no. 106119100,

Class : CSE-B

Instructions

Implement, DFS, BFS, Best-first search and A* search algorithms in Prolog. Compare the algorithms and justify your results. For A* and Best-first search you may use Straight line distance as the heuristics function.

Try an alternate heuristic function and compare that with the SLD.

Apply the above algorithms to solve the TSP problem

Code :

```
/* city with its corresponding index */
city(1,'City A').
city(2,'City B').
city(3,'City C').
city(4,'City D').
city(5,'City E').
city(6,'City F').
city(7,'City G').

/* Distance between cities */
road(1,2,436).    /* From City A to City B the distance is 436 kilometers */
road(1,7,600).
road(1,3,78).
road(2,4,399).
road(2,5,85).
road(3,7,260).
road(3,6,227).
road(3,4,175).
road(5,7,241).
road(4,6,390).
road(4,5,481).

/* Heuristic that was used */
/* distance with straight line (assumed) */
h(1,300).
h(2,250).
h(3,250).
h(4,100).
h(5,80).
h(6,50).
```

```

h(7,50).
h(8,200).

/* highway between cities */
highway(1,2,'N 163').
highway(1,3,'N 343 e 358').
highway(2,4,'N 163').
highway(4,5,'N 163').
highway(3,4,'N 246').
highway(3,7,'N 246, 364 e 163').
highway(3,6,'N 358 e 364').
highway(5,7,'N 163').
highway(2,5,'N 163').
highway(4,6,'N 358').

find_shortest_path(Origin, Destination):-
    city(C1,Origin),
    city(C2,Destination),
    a_star([[0,C1]],C2,ReversePath),
    reverse(ReversePath, Path),
    write('The best/shortest Path is: '), print_path(Path,Highways),
    write('Highway to be traveled will be: '),print_highways(Highways),!
.

find_shortest_path(_,_):- write('There was an error with origin or destination
city, please type again').

find_all(Origin, Destination):-
    city(C1,Origin),
    city(C2,Destination),
    a_star([[0,C1]],C2,ReversePath),
    reverse(ReversePath, Path),
    write('A Path was found: '), print_path(Path,Highways),
    write('Highway to be traveled will be: '),print_highways(Highways),fail.
find_all(_,_):- write('That is all!').

a_star(Paths, Dest, [C,Dest|Path]):-
    member([C,Dest|Path],Paths),
    decide_best(Paths, [C1|_]),
    C1 == C.

a_star(Paths, Destination, BestPath):-
    decide_best(Paths, Best),
    delete(Paths, Best, PreviousPaths),
    expand_border(Best, NewPaths),
    append(PreviousPaths, NewPaths, L),
    a_star(L, Destination, BestPath).

decide_best([X],X):-!.
decide_best([[C1,Ci1|Y],[C2,Ci2|_]|Z], Best):-
    h(Ci1, H1),
    h(Ci2, H2),
    H1 + C1 <= H2 + C2,
    decide_best([[C1,Ci1|Y]|Z], Best).
decide_best([[C1,Ci1|_],[C2,Ci2|Y]|Z], Best):-

```

```

h(Ci1, H1),
h(Ci2, H2),
H1 + C1 > H2 + C2,
decide_best([[C2,Ci2|Y]|Z], Best).

expand_border([Cost,City|Path],Paths):-
    findall([Cost,NewCity,City|Path],
        (road(City, NewCity,_),
         not(member(NewCity,Path))),
        L),
    change_costs(L, Paths).

change_costs([],[]):-!.
change_costs([[Total_Cost,Ci1,Ci2|Path]|Y],[[NewCost_Total,Ci1,Ci2|Path]|Z]):-
    road(Ci2, Ci1, Distance),
    NewCost_Total is Total_Cost + Distance,
    change_costs(Y,Z).

print_path([Cost],[]):- nl, write('The total cost of the path is: '), write(Cost),
write(' kilometers'),nl.
print_path([City,Cost],[]):- city(City, Name), write(Name), write(' '), nl, write('The total cost of the path is: '), write(Cost), write(' kilometers'),nl.
print_path([City,City2|Y],Highways):-
    city(City, Name),
    highway(City,City2,Highway),
    append([Highway],R,Highways),
    write(Name),write(', '),
    print_path([City2|Y],R).

print_highways([X]):- write(X), nl, nl.
print_highways([X|Y]):-
    write(X),write(' - '),
    print_highways(Y).

```

INPUT: find_shortest_path('City A','City E').

OUTPUT:

The best/shortest Path is: City A, City B, City E

The total cost of the path is: 521 kilometers

Highway to be traveled will be: N 163 - N 163

Screenshots:

The screenshot shows the SWISH Prolog IDE interface. On the left, a code editor displays a Prolog program for finding the shortest path between cities. The program defines cities, roads with distances, and a heuristic function. On the right, a console window shows the execution of the query `find_shortest_path('City A','City E').`. The output indicates the best/shortest path is City A, City B, City E, with a total cost of 521 kilometers. The highway to be traveled is N 163 - N 163. Below the console, a query input field shows `?- find_shortest_path('City A','City E').` and buttons for Examples, History, and Solutions are visible.

```
1 /* city with its corresponding index */
2 city(1,'City A').
3 city(2,'City B').
4 city(3,'City C').
5 city(4,'City D').
6 city(5,'City E').
7 city(6,'City F').
8 city(7,'City G').
9
10 /* Distance between cities */
11 road(1,2,436). /* From City A to City B the distance is 436 kilometers */
12 road(1,7,600).
13 road(1,3,78).
14 road(2,4,399).
15 road(2,5,85).
16 road(3,7,260).
17 road(3,6,227).
18 road(3,4,175).
19 road(5,7,241).
20 road(4,6,390).
21 road(4,5,481).
22
23 /* Heuristic that was used */
24 /* distance with straight line (assumed) */
25 h(1,300).
26 h(2,250).
27 h(3,250).
28 h(4,100).
29 h(5,80).
30 h(6,50).
31 h(7,50).
32 h(8,200).
```

find_shortest_path('City A','City E').

The best/shortest Path is: City A, City B, City E
The total cost of the path is: 521 kilometers
Highway to be traveled will be: N 163 - N 163

true

?- find_shortest_path('City A','City E').

Examples History Solutions

☐ table results Run

BFS and DFS in Prolog

```
%-----%
% Depth-first search by using a stack %
% call: depth_first(+[[Start]],+Goal,-Path,-ExploredNodes). %
%-----%

depth_first([ [Goal|Path] | _ ], Goal, [Goal|Path], 0).
depth_first([Path|Queue], Goal, FinalPath, N) :-
    extend(Path, NewPaths),
    append(NewPaths, Queue, NewQueue),
    depth_first(NewQueue, Goal, FinalPath, M),
    N is M+1.

extend([Node|Path], NewPaths) :-
    findall([NewNode,Node|Path],
        (arc(Node,NewNode,_),
         \+ member(NewNode,Path)), % for avoiding loops
        NewPaths).
```

```
%-----%
%   Breadth-first search                                     %
%   call: breadth_first(+[[Start]],+Goal,-Path,-ExploredNodes).%
%-----%
breadth_first([[Goal|Path]|_],Goal,[Goal|Path],0).
breadth_first([Path|Queue],Goal,FinalPath,N) :-
    extend(Path,NewPaths),
    append(Queue,NewPaths,NewQueue),
    breadth_first(NewQueue,Goal,FinalPath,M),
    N is M+1.
```