```prolog
%%% List of the stations serving more than one line
multiple_lines(S):-
    stop(L1,_,S),
    stop(L2,_,S),
    L1\==L2.


%%% Computing terminals
first_stop(L,S):-
    line(L),
    stop(L,1,S).
not_last_stop(L,S):-
    line(L),
    stop(L,N,S),
    stop(L,N1,_),
    N1>N.
last_stop(L,S):-
    line(L),
    stop(L,_,S),
    \+not_last_stop(L,S).
termini(L,S1,S2):-
    first_stop(L,S1),
    last_stop(L,S2).


%%% List of all the stations of a line (stations are ordered)
next_stop(L,S1,S2):-
    line(L),
    stop(L,N1,S1),
    N2 is N1+1,
    stop(L,N2,S2).
list_stops_helper(L,ACC,RES):-
    ACC = [S|_],
    first_stop(L,S),
    RES = ACC.
list_stops_helper(L,ACC,RES):-
    ACC = [S|_],
    next_stop(L,P,S),
    list_stops_helper(L,[P|ACC],RES).
list_stops(L,List):-
    line(L),
    last_stop(L,S),
    list_stops_helper(L,[S],List).


%%% Path from a station to another. Ths solutions fulfills both proerty (a) and
proerty (b)

%%% Set of the stations traversed
stations_in_segment(segment(_,S,S),[S]).
stations_in_segment(segment(L,S1,S2),Result):-
    next_stop(L,S1,T),
    stations_in_segment(segment(L,T,S2),Temp),
    Result = [S1|Temp].
stations_traversed([],[]).
stations_traversed([segment(L,S1,S2)|Tail],Result):-
    stations_traversed(Tail,Temp),
    stations_in_segment(segment(L,S1,S2),TempFirst),
    union(TempFirst,Temp,Result).

%%% Cyclic segment
segment_adds_cycle(segment(_,_,_),[]):-false.
segment_adds_cycle(segment(L,S1,S2),Path):-
    stations_traversed(Path,StPath),
    next_stop(L,S1,T),
    stations_in_segment(segment(L,T,S2),StSeg),
    \+intersection(StPath,StSeg,[]).

%%% Test whether the path uses a specific line
uses_line(_,[]):-false.
uses_line(L,[segment(L,_,_)|_]).
uses_line(L,[_|Rest]):-uses_line(L,Rest).
```

```prolog

%%% Finally the predicate for the path
path(S1,S2,List):-pathHelper(S1,S2,List,[]).
pathHelper(S1,S2,List,ATTEMPT):-
    stop(X,N1,S1),
    stop(X,N2,S2),
    N1<N2,
    \+uses_line(X,ATTEMPT),
    \+segment_adds_cycle(segment(X,S1,S2),ATTEMPT),
    List=[segment(X,S1,S2)].
pathHelper(S1,S2,List,ATTEMPT):-
    stop(X,N1,S1),
    stop(X,N_Med,S_Med),
    N1<N_Med,
    \+uses_line(X,ATTEMPT),
    \+segment_adds_cycle(segment(X,S1,S_Med),ATTEMPT),
    pathHelper(S_Med,S2,L_Temp,[segment(X,S1,S_Med)|ATTEMPT]),
    List=[segment(X,S1,S_Med)|L_Temp].


%%% Path with minimum numer of changes
list_length([],0):-!.
list_length([_|R],N):-
    list_length(R,N1),
    N is N1+1.

non_minimum_path(S1,S2,P):-
    path(S1,S2,P),
    path(S1,S2,P2),
    list_length(P,N1),
    list_length(P2,N2),
    N2<N1.
minimum_path(S1,S2,P):-
    path(S1,S2,P),
    \+non_minimum_path(S1,S2,P).

```