```
/* Demonstration of using search through state space to
   generate an executable plan for getting from A to B.

   Michael E. Sparks, 12 Nov 2020

   SAMPLE USAGE:

?- soln_DFSid([[e,c],[f,a,d,g],[b]],Plan).

  [[e,c],[f,a,d,g],[b]]
  [[b,e,c],[f,a,d,g],[]]
  [[b,e,c],[a,d,g],[f]]
  [[b,e,c],[d,g],[a,f]]
  [[e,c],[d,g],[b,a,f]]
  [[c],[e,d,g],[b,a,f]]
  [[],[e,d,g],[c,b,a,f]]
  [[e],[d,g],[c,b,a,f]]
  [[d,e],[g],[c,b,a,f]]
  [[c,d,e],[g],[b,a,f]]
  [[b,c,d,e],[g],[a,f]]
  [[a,b,c,d,e],[g],[f]]

Plan = [[[a, b, c, d, e], [g], [f]], [[b, c, d, e], [g], [a, f]], [[c, d, e], [g
], [b, a, f]], [[d, e], [g], [c, b|...]], [[e], [d, g], [c|...]], [[], [e|...],
[...|...]], [[c], [...|...]|...], [[...|...]|...], [...|...]|...] .
*/

% main user interface, uses a depth-first/ breadth-first
% hybrid searching approach
soln_per_depth_first_search_with_iterative_deepening(Init,Plan) :-
    goal(Goal),
    path(Init,Goal,Plan),
    reverse(Plan,Plan1),
    display(Plan1).

% create alias for full predicate name
soln_DFSid(Init,Plan) :-
    soln_per_depth_first_search_with_iterative_deepening(Init,Plan).

% We have three slots with unique letters randomly
% distributed into each as a starting point.
% The goal is to place chars [a-e] in sorted order in
% the first slot, and we're indifferent about placement
% of residual chars in the last two.
goal(State) :-
    State = [[a,b,c,d,e]|_].

% returns a legal path from initial node to goal node
path(N,N,[N]).

path(Init,Goal,[Goal|Path]) :-
    path(Init,Penultimate,Path),
    succ(Penultimate,Goal),
    \+ member(Goal,Path).

% Define what constitutes a viable successor in the state space.
% That is, these are the rules governing "node expansion."
succ([S1,S2,S3],[SS1,[Top1|S2],S3]) :- S1 = [Top1|SS1].
succ([S1,S2,S3],[SS1,S2,[Top1|S3]]) :- S1 = [Top1|SS1].
succ([S1,S2,S3],[[Top2|S1],SS2,S3]) :- S2 = [Top2|SS2].
succ([S1,S2,S3],[S1,SS2,[Top2|S3]]) :- S2 = [Top2|SS2].
succ([S1,S2,S3],[[Top3|S1],S2,SS3]) :- S3 = [Top3|SS3].
succ([S1,S2,S3],[S1,[Top3|S2],SS3]) :- S3 = [Top3|SS3].

% pretty printing routine for the plans generated
display([]) :-
    nl, nl.

display([Move|Rest]) :-
```



```
    nl,
    tab(2),
    write(Move),
    display(Rest).
```