Notes on DAGS and TopOrder, §3.6

This is the approach followed in class. I changed some names of variables to match the assignment. The input is a directed graph G with the Adjacency list representation. There is a Boolean Variable `NOPROBLEMO` which will be true if G is not a DAG and (therefore) there is a cycle. There is a *stack* `GOODORDER` which, if G is a DAG, will give a topological ordering. There is an array `COLOR`$[v]$ which takes on the values `white`, `grey`, `black`. We'll call the whole algorithm `GRAPHTEST`

```
GRAPHTEST
(* Initialization *)
GOODORDER ← ∅
For all vertices v
      COLOR[v] ← white
(* End Initialization *)
For each vertex v
      If COLOR[v] ← white
            DFS[v]
      EndIf
EndFor
```

The center of the algorithm is `DFS`$[v]$

```
DFS[v]
COLOR[v] ← grey (* v has been found *)
For all w ∈ Adj[v]
      If COLOR[w] ← grey (* Cycle found *)
            NOPROBLEMO ← false
            EXIT GRAPHTEST
      EndIf
      If COLOR[w] ← white (* New point found*)
            DFS[w] (* recursive call *)
      EndIf
      (* Ignore black vertices *)
Endfor
COLOR[v] ← black (* v has been completed *)
ADDTOSTACK[GOODORDER,v]
(* stack will have v in reverse order of completion *)
```

That does it. It helps to think of `DFS`$[v]$ at the start of `GRAPHTEST` when all vertices are white. It will explore all vertices w reachable from v and call `DFS`$[w]$ for those w and all those w will be come `black` and then when `DFS`$[v]$ finishes, v itself becomes `black`. At this point some (sometimes all, in which case we're done) of the vertices are `black` but none are `grey`. When we find another white vertex v' and call `DFS`$[v']$ it is as if the `black` vertices weren't there.