Algorithm for finding strongly connected components

The idea of the algorithm is as follows:

1. call DFS(G) to compute the finishing time $f[v]$ for every vertex $v$, sort the vertices of $G$ in decreasing order of their finishing time;

2. compute the transpose $G^T$ of $G$;

3. Perform another DFS on $G$, this time in the main for-loop we go through the vertices of $G$ in the decreasing order of $f[v]$;

4. output the vertices of each tree in the DFS forest (formed by the second DFS) as a separate strongly connected component.

There are two issues. First, on line 1, we want to sort the vertices. We don’t want $\Theta(n \ln n)$ algorithm for sorting here. Instead, we put the vertices in a linked list as they are colored Black (thus $v$ is inserted into the list at time $f[v]$). (This is similar to what we do in the Topological sort algorithm, except for we do not check for back edges here.)

Second, to output the components (on line 4) we can use a unique identifier for each strongly connected component.

For the first DFS, we use colors White, Gray, Black as usual. For the second DFS, we will use Black, Blue, Red. (SCC stands for “strongly connected component”.)

```
SCC(G)
1. % initialization for the first DFS
2. for each $u$ in $V$ do
3. \hspace{1em} color[$u$] ← White
4. end for
5. Linked list $L$ ← $\emptyset$ \hspace{1em} % $L$ contains the vertices of $G$ in decreasing order or $f[v]$
6. % now the main loop of the first DFS
7. for each $u$ in $V$ do
8. \hspace{1em} if $color[u] = $ White do
9. \hspace{2em} SCC-Visit1($G, u, L$)
10. \hspace{1em} end if
11. end for \hspace{1em} % end of the first DFS
12. % now compute $G^T$ by reversing the edges of $G$: $B[v]$ is the adjacency list of $v$ in the new graph $G'$
```
for \( v \in V \) do
  for \( u \in \text{Adj}[v] \) do
    insert \( v \) into \( B[u] \)
  end for
end for % end of computing \( G^T \)

% now the second DFS
% initialization for the second DFS
for \( v \) from 1 to \(|V|\) do
  \( \text{SCC}[v] \leftarrow 0 \) % new array, \( \text{SCC}[v] \) is the SCC identifier for \( v \).
end for

c \leftarrow 0 % c is the identifier for the current strongly connected component
% the main loop of the second DFS
for each \( u \) in \( L \) do
  if \( \text{color}[u] = \text{Black} \) do
    \( c \leftarrow c + 1 \)
    \( \text{SCC-Visit2}(G^T, u, c) \)
  end if
end for
The procedure SCC-Visit1 is similar to Topo-Visit. Once a vertex is colored Black we insert it into the linked list $L$. So at the end $L$ contains the vertices of $G$ in decreasing order of $f$ (finishing time).

SCC-Visit1($G, u, L$):

1. stack $S \leftarrow \emptyset$  % initialize $S$ to the empty stack
2. $push(S, u)$
3. while $S$ is not empty do
4.   $x \leftarrow pop(S)$
5.   if $color[x] = White$ do
6.     $time \leftarrow time + 1$
7.     $s[x] \leftarrow time$
8.     $color[x] \leftarrow Gray$
9.     $push(S, x)$
10.    for each $v$ in $Adj[x]$ do
11.      if $color[v] = White$ do
12.         $p[v] \leftarrow x$
13.         $push(S, v)$
14.      else if $color[v] = Gray$ do
15.         return false
16.      end if
17.    end for
18.   else if $color[x] = Gray$ do
19.     $time \leftarrow time + 1$
20.     $f[x] \leftarrow time$
21.     $color[x] \leftarrow Black$
22.     insert $x$ to $L$
23.   end if
24. end while
The procedure SCC-Visit2 is similar to DFS-Visit, except now the triple of colors are (Black, Blue, Red). Also, we give each vertex encountered during this search the SCC identifier $c$.

SCC-Visit2($G, u, c, SCC$):
1. stack $S \leftarrow \emptyset$  \hspace{1em}  \% initialize $S$ to the empty stack
2. $push(S, u)$
3. while $S$ is not empty do
4.  $x \leftarrow pop(S)$
5.  $SCC[x] \leftarrow c$
6.  if $color[x] = Black$ do
7.    $color[x] \leftarrow Blue$
8.    $push(S, x)$
9.    for each $v$ in $Adj[x]$ do
10.   if $color[v] = Black$ do
11.      $p[v] \leftarrow x$
12.     $push(S, v)$
13.     end if
14.   end for
15.  else if $color[x] = Blue$ do
16.   $color[x] \leftarrow Red$
17.  end if
18. end while