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. $color[u] \leftarrow White$
4. end for
5. Linked list $L \leftarrow \varnothing$ % 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.   if $color[u] = White$ do
9.   SCC-Visit1(G, u, L)
10.   end if
11. end for   % 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 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
    SCC[v] ← 0  % new array, SCC[v] is the SCC identifier for v.
end for
c ← 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 color[u] = Black do
        c ← c + 1
        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):

```
stack S ← ∅   % initialize S to the empty stack
push(S, u)
while S is not empty do
    x ← pop(S)
    if color[x] = White do
        time ← time + 1
        s[x] ← time
        color[x] ← Gray
        push(S, x)
        for each v in Adj[x] do
            if color[v] = White do
                p[v] ← x
                push(S, v)
            else if color[v] = Gray do
                return false
            end if
        end for
    else if color[x] = Gray do
        time ← time + 1
        f[x] ← time
        color[x] ← Black
        insert x to L
    end if
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):

```
stack S ← ∅   % initialize S to the empty stack
push(S, u)
while S is not empty do
    x ← pop(S)
    SCC[x] ← c
    if color[x] = Black do
        color[x] ← Blue
        push(S, x)
        for each v in Adj[x] do
            if color[v] = Black do
                p[v] ← x
                push(S, v)
            end if
        end for
    else if color[x] = Blue do
        color[x] ← Red
    end if
end while
```