Solutions, Class 03/16: SCC & Topological Sort

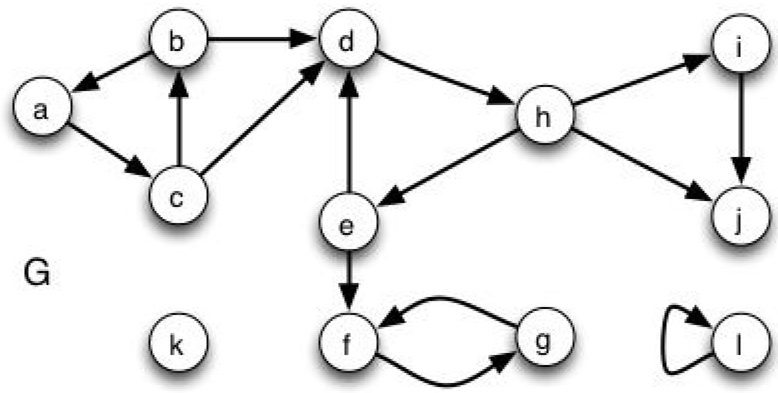
Finding SCC of a graph following the SCC and DFS algorithms of the textbook:

```
STRONGLY-CONNECTED-COMPONENTS(G)
1  call DFS(G) to compute finishing times u.f for each vertex u
2  compute G^T
3  call DFS(G^T), but in the main loop of DFS, consider the vertices
       in order of decreasing u.f (as computed in line 1)
4  output the vertices of each tree in the depth-first forest formed in line 3 as a
       separate strongly connected component
```

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G,u)
```

```
DFS-VISIT(G,u)
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each v ∈ G.Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G,v)
8  u.color = BLACK
9  time = time + 1
10 u.f = time
```

1. <u>Run DFS on this graph</u>. To make grading easier, <u>visit vertices in alphabetical order</u> (both in the main loop of DFS and the adjacency list loop of DFS-Visit).



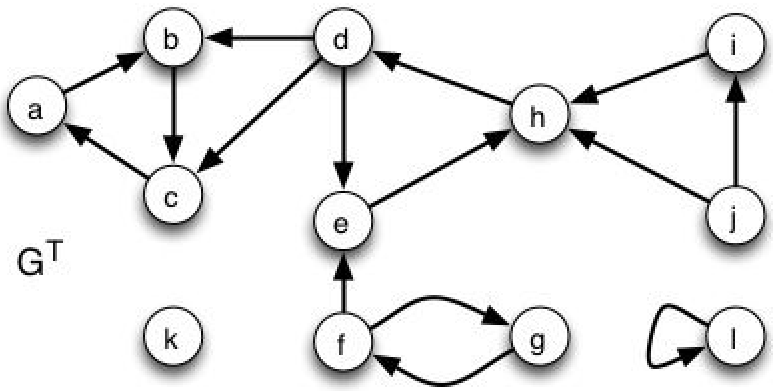
For each vertex, show values d (discovery), f (finish), and π (parent).

	a	b	c	d	e	f	g	h	i	j	k	l
d	1	3	2	4	6	7	8	5	12	13	21	23
f	20	18	19	17	11	10	9	16	15	14	22	24
π	NIL	c	a	b	h	e	f	d	h	i	NIL	NIL

Write the vertices in order from largest to smallest finish time:

l, k, a, c, b, d, h, i, j, e, f, g

2. Now <u>run DFS on the transpose graph, visiting vertices in order of finish time</u> (largest to smallest) from the DFS of step 1 (as required by the SCC algorithm). Again, show values d (discovery), f (finish), and π (parent).



	a	b	c	d	e	f	g	h	i	j	k	l
d	5	6	7	11	12	21	22	13	17	19	3	1
f	10	9	8	16	15	24	23	14	18	20	4	2
π	NIL	a	b	NIL	d	NIL	f	e	NIL	NIL	NIL	NIL

3. List the strongly connected components you found by first <u>listing the tree edges</u> in the transpose graph that define the SCC, and then <u>listing the vertices</u> in the SCC (the first is shown):

SCC 1: Tree edges: { }; Vertices: {l}
SCC 2: Tree edges: { }; Vertices: {k}
SCC 3: Tree edges: {(a,b) (b,c) }; Vertices: {a, b, c}
SCC 4: Tree edges: {(d, e) (e, h) }; Vertices: {d, e, h}
SCC 5: Tree edges: { }; Vertices: {i}
SCC 6: Tree edges: { }; Vertices: {j}
SCC 7: Tree edges: {(f,g)}; Vertices: {f, g}

4. (Extra Credit): The DFS-Visit procedure as written uses three colors; WHITE, GRAY, and BLACK. Show that we can get by with two colors, and hence only one bit of storage per vertex. Do this by saying how you would modify or delete (your choice!) lines 3 or 8 of DFS-Visit, and arguing that the algorithm still works.

The only <u>tests</u> of color are in line 5 of DFS and line 5 of DFS-Visit. These tests only care whether or not the vertex is white. In the existing algorithm, Gray and Black will lead to the same outcome in these tests. Therefore we need only one color.

The non-white color indicates that a node has just been discovered, so should not be processed again in lines 5-7 of DFS-Visit. Therefore we should mark the node non-white as soon as it is visited, in line 3. (If line 3 were not there, one might get back to u via a cycle and try to process it again.) It does not matter whether we call it “gray” or “black, as long as the color becomes non-white in line 3. On the other hand, line 8 serves no purpose and can be deleted.

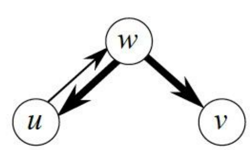
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each v ∈ G.Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G, v)
8  u.color = BLACK
9  time = time + 1
10 u.f = time
```

Note that the instructor’s manual for this problem says that line 3 can be deleted. I am certain that they are wrong. We would go around a white cycle endlessly.

5. Additional extra credit!

(a) Give a counterexample to the conjecture that if a directed graph G contains a path from u to v, and if u.d < v.d in a depth-first search of G, then v is a descendant of u in the depth-first forest produced.

	d	f
w	1	6
u	2	3
v	4	5



(b) Give a counterexample to the conjecture that if a directed graph G contains a path from u to v, then any depth-first search must result v.d <= u.f.

The above also works.