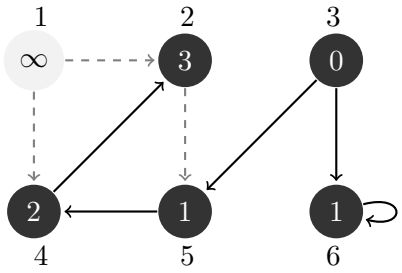
Chapter 22: Elementary Graph Algorithms

22.2-1) Show the d and π values that result from running breadth-first search on the directed graph of Figure 22.2(a), using vertex 3 as the source.

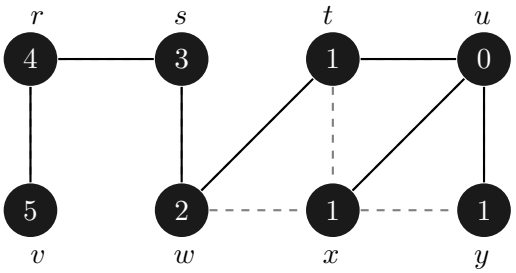
Solution:



	d	π
1	∞	NIL
2	3	4
3	0	NIL
4	2	5
5	1	3
6	1	3

22.2-2) Show the d and π values that result from running breadth-first search on the undirected graph of Figure 22.3, using vertex u as the source.

Solution:



	d	π
r	4	s
s	3	w
t	1	u
u	0	NIL
v	5	r
w	2	t
x	1	u
y	1	u

22.2-6) There are two types of professional wrestlers: "good guys" and "bad guys." Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an $O(n + r)$-time algorithm that determines whether it is possible to designate some of the wrestlers as good guys and the remainder as bad guys such that each rivalry is between a good guy and a bad guy. If is it possible to perform such a designation, your algorithm should produce it.

Solution:
Let $G = (V, E)$ be a graph with each vertex representing a professional wrestler and each edge representing a rivalry (good vs bad). For n wrestlers and r rivalries, $V = n$ and $E = r$

Algorithm:

1. Perform Breath-First-Search on G. This runs in $O(n+r)$.
2. For each vertex whose distance label from the source is even, label g, good guy, otherwise, label b, bad guy. Each vertex is visited once $\implies$ running time is $O(n)$.

The total running time is then, $O(n+r)+O(n)=O(n+r)$.

22.3-11) Show that a depth-first search of an undirected graph G can be used to identify the connected components of G, and that the depth-first forest contains as many trees as G has connected components. More precisely, show how to modify depth-first search so that each vertex v is assigned an integer label $cc[v]$ between 1 and k, where k is the number of connected components of G, such that $cc[u]=cc[v]$ if and only if u and v are in the same connected component.

Solution:

DFS-VISIT(G,u) explores all the reachable vertices from u. The result is a spaning tree. DFS(G) on the other hand will explore all vertices by calling DFS-VISIT(G,u) iteratively until there are no WHITE vertices left (unvisited vetices). This means that each call to DFS-VISIT within DFS is a call for the exploration of a different connected component (a different spanning tree with root at the next WHITE node, u). This works well given the fact that G is an undirected graph, in particular, we don't have to worry about unreachable vertices being connected to a tree. Thus, in order to number the connected components, all we need is to update the component number each time a new tree is explored (below in line 10 in DFS) and label the visited vertices in DFS-VISIT the current component number (line 4 in DFS-VISIT).

DFS(G)

```
1  for each vertex u ∈ G.V
2    u.color = WHITE
3    u.π = NIL
4    cc[u]=0;                              \\added
5  time = 0
6  component = 0                           \\added
7  for each vertes u ∈ G.V
8    if u.color == WHITE
9      component = component + 1           \\added
10     DFS–VISIT(G, u, component)
```

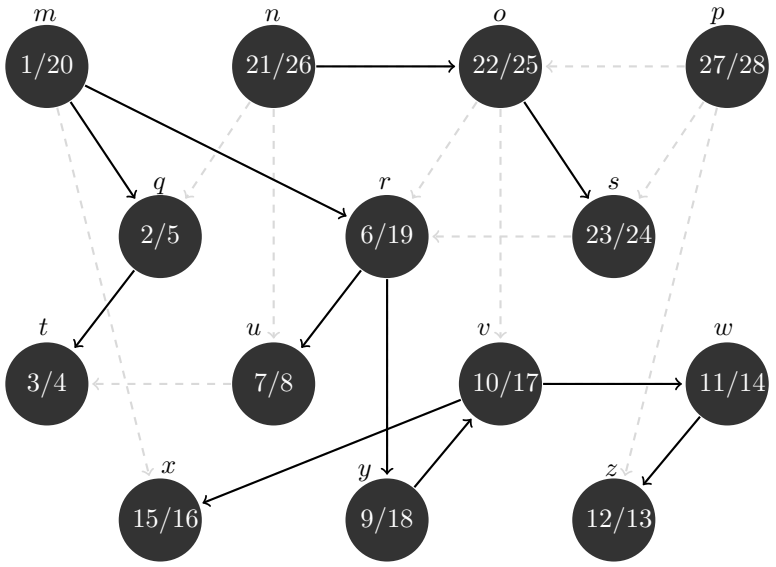
DFS-VISIT$(G,u,component)$

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

22.4-1) Show the ordering of vertices produced by TOPOLOGICAL-SORT when it is run on the dag of Figure 22.8, under the assumption of Exercise 22.3-2.

TOPOLOGICAL-SORT(G)

1. call DFS to compute finishing times $f[v]$ for each vertex v



v	d	f
m	1	20
n	21	26
o	22	25
p	27	28
q	2	5
r	6	19
s	23	24
t	3	4
u	7	8
v	10	17
w	11	14
x	15	16
y	9	18
z	12	13

2. as each vertex is finished, insert it onto the front of the linked list

p →	n →	o →	s →	m →	r →	y →	v →	x →	w →	z →	u →	q →	t
27/28	21/26	22/25	23/24	1/20	6/19	9/18	10/17	15/16	11/14	12/13	7/8	2/5	3/4

22.5-1) How can the number of strongly connected components of a graph change if a new edge is added?

Solution:

case1: the number of strongly connected components is unchanged

This occurs when the edge is added between vertices belonging to the same strongly connected component.

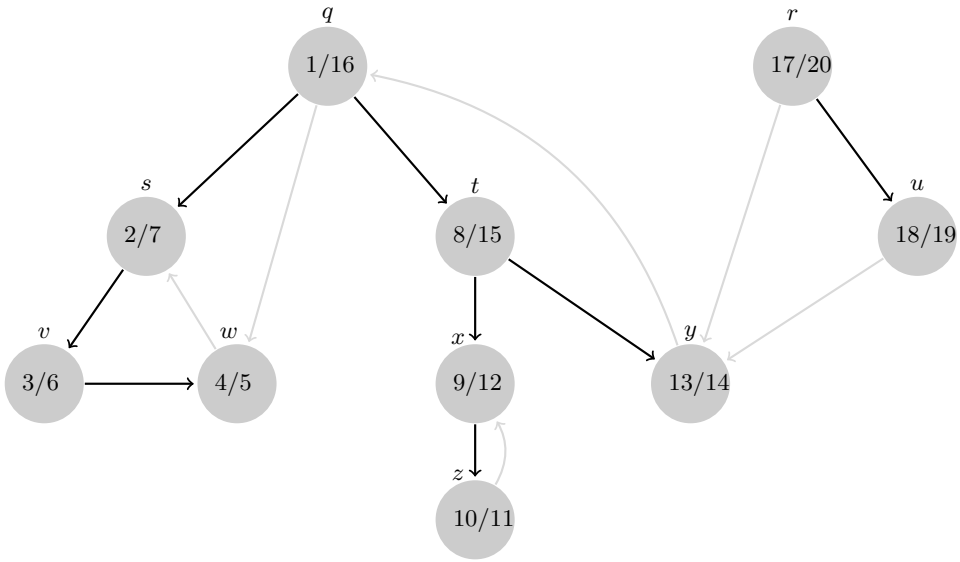
case2: the number of strongly connected components is decreased by one

This case occurs when the edge is added between different vertices belonging to different strongly connected components.

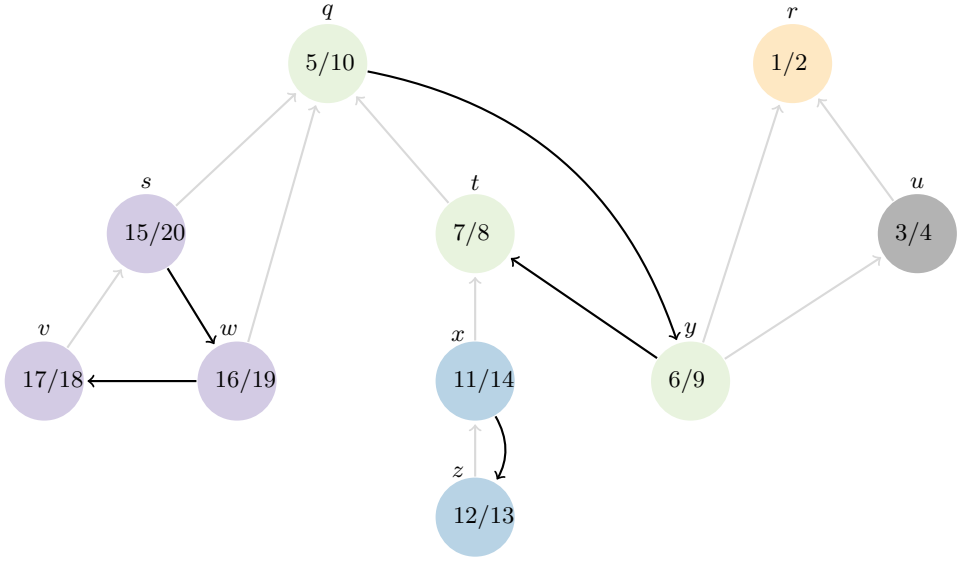
22.5-2) Show how the procedure STRONGLY-CONNECTED-COMPONENTS works on the graph of Figure 22.6. Specifically, show the finishing times computed in line 1 and the forest produced in line 3. Assume that the loop of lines 5-7 of DFS considers vertices in alphabetical order and that the adjacency lists are in alphabetical order.

Solution:

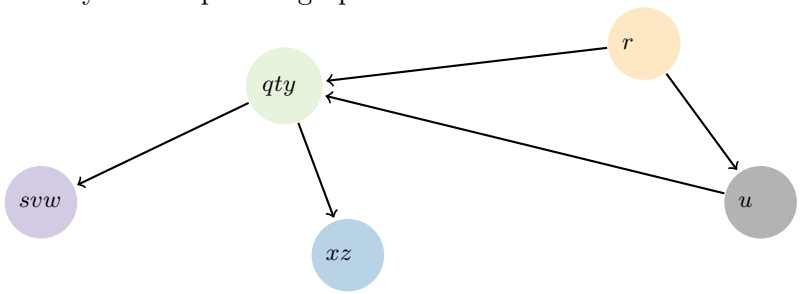
1. call DFS(G) to compute finishing times $f[u]$ for each vertex u
   each node u contains discover/finishing time
   For example, in the graph shown below, node q was discovered at time 1 and finished at time 16 ($d[q] = 1$, $f[q] = 16$)



2. compute G^T
3. call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing $f[u]$ (as computed in (1))

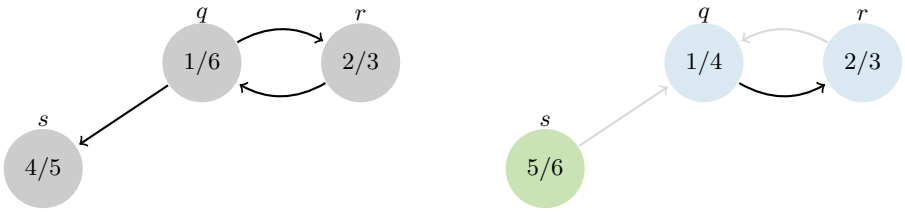


4. the acyclic component graph G^{SCC}



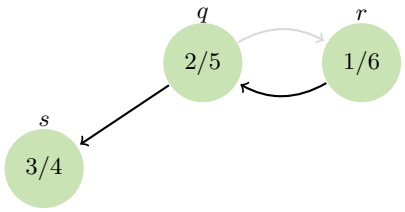
22.5-3) Professor Deaver claims that the algorithm for strongly connected components can be simplified by using the original (instead of the transpose) graph in the second depth-first search and scanning the vertices in order of increasing finishing times. Is the professor correct?
Solution:

No, the resulting components are not the same.
An example is shown below. The graph on the left and the graph on the right are the resulting graphs of running DFS and STRONGLY-CONNECTED-COMPONENTS(G) respectively. As it can be seen, G clearly has two strongly connected components.



On the other hand, if the algorithm was modified to run a second depth-first search on the original graph and scan the vertices in the order of increasing finishing times, we would have to start with vertex r, resulting in only one strongly connected component.

$f[r] = 3,\ f[s] = 5,\ f[q] = 6$



[2]

References

[1] Cormen, Thomas. H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, Second Edition*. MIT Press, Cambridge, MA, 2009.

[2] http://jt-web-site.tripod.com/MSCS/COMP510/Assignment12/Assignment12.htm