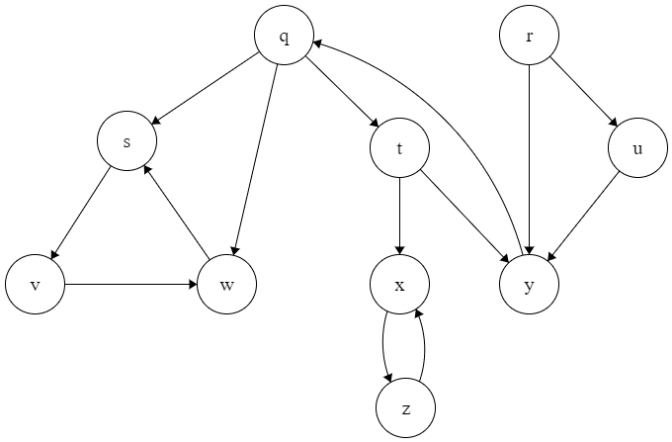
CMPS 101
Homework Assignment 7

1. p.610: 22.3-2
   Show how depth-first search works on the graph of Figure 22.6 (p.611). Assume that the **for** loop of lines 5-7 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discover and finishing times for each vertex, and show the classification of each edge.

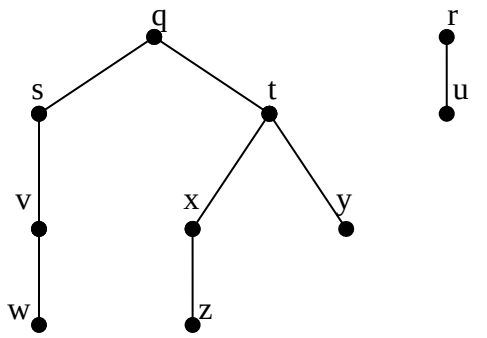
Solution:



Vertex	Adj	Discover	Finish
q	s t w	1	16
r	u y	17	20
s	v	2	7
t	x y	8	15
u	y	18	19
v	w	3	6
w	s	4	5
x	z	9	12
y	q	13	14
z	x	10	11

Edge	Classification
(q, s)	tree
(q, t)	tree
(q, w)	forward
(r, u)	tree
(r, y)	cross
(s, v)	tree
(t, x)	tree
(t, y)	tree
(u, y)	cross
(v, w)	tree
(w, s)	back
(x, z)	tree
(y, q)	back
(z, x)	back

DFS Forest



2. p.610: 22.3-1
Make a 3-by-3 chart with row and column labels WHITE, GRAY, and BLACK. In each cell (i, j), indicate whether, at any point during a depth-first search of a directed graph, there can be an edge from a vertex of color i to a vertex of color j. For each possible edge, indicate what types it can be.

Solution:

	WHITE	GRAY	BLACK
WHITE	Yes: tree, back, forward, cross	Yes: back, cross	Yes: cross
GRAY	Yes: tree, forward	Yes: tree, back, forward	Yes: tree, forward, cross
BLACK	No	Yes: back	Yes: tree, back, forward, cross

3. p.612: 22.3-10
Modify the pseudocode for depth-first search so that it prints out every edge in the directed graph together with its type. (Hint: use the result stated in the last paragraph of page 609, and the result of problem 22.3-5.)

Solution:

DFS(G)

```
1.  for each u ∈ V[G]
2.      color[u] = white
3.      p[u] = nil
4.  time = 0
5.  for each u ∈ V[G]
6.      if color[u] == white
7.          ModifiedVisit(u)
```

ModifiedVisit(u)

```
1.  color[u] = gray
2.  d[u] = (++time)
3.  for all v ∈ Adj[u]      // Explore edge (u,v)
4.          if color[v] == white
5.                  print (u, v) " is a Tree edge"
6.                  p[v] = u
7.                  ModifiedVisit(v)
8.          else if color[v] == gray
9.                  print (u,v) " is a Back edge"
10.         else if d[u] > d[v]
11.                 print (u,v) " is a Cross edge"
12.         else
13.                 print (u,v) " is a Forward edge"
14. color[u] = black
15. f[u] = (++time)
```

4. p.612: 22.3-12

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:

Let $cc[u]$ be another attribute associated with each vertex u in the graph, and let k be a new variable which is local to DFS, but static over all recursive calls to Visit (similar to `time`.) We alter DFS() and Visit() as follows.

DFS(G)

1) for each $u \in V(G)$
2) color$[u]$ = white
3) $\pi[u]$ = nil
4) time = 0
5) $k = 0$
6) for each $u \in V(G)$
7) if color$[u]$ == white
8) $k++$
9) Visit(u)

Visit(u)

1) color$[u]$ = gray
2) $d[u] = (++\text{time})$
3) $cc[u] = k$
4) for each $v \in$ adj$[u]$
5) if color$[v]$ == white
6) $\pi[v] = u$
7) Visit(v)
8) color$[u]$ = black
9) $f[u] = (++\text{time})$

Observe that k is incremented only when a new root in a DFS tree is established, and that that root and all its descendents will have the same cc value. Thus when DFS() is complete, k will equal the number of trees in the DFS forest, and all vertices in each tree will have the same cc value. Theorem 22.10 on page 610 says that after running DFS() on an undirected graph G, all edges are classified as either tree or back. Thus there are no cross edges, and therefore no edges joining distinct DFS trees. Hence if vertices x and y lie in distinct DFS trees, then G contains no x-y path, and therefore x and y lie in different connected components. On the other hand, if x and y lie in the same DFS tree, then G does contain an x-y path, namely the unique path consisting of tree edges. In this case x and y lie in the same component. Thus each DFS tree spans a connected component of G, and in particular, the number of trees in a DFS forest is the same as the number of connected components in G. So after this modified version of DFS() is run, k will equal the number of connected components in G, and for any two vertices x and y, we will have $cc[x] = cc[y]$ if and only if x and y lie in the same connected component. ///