Exercise VII, Algorithms 2024-2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. There are many problems on this set, solve as many as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

Graph Representation

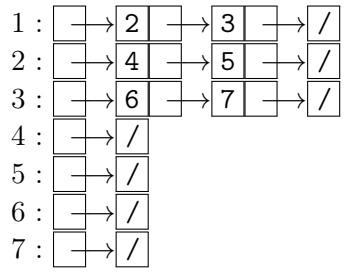
1 (Adaptation of Exercise 22.1-1) Given an adjacency-list representation of a directed graph, how long does it take to compute the out-degree of a given vertex v? How long does it take to compute the in-degree of a given vertex v?

Solution: The out-degree of a vertex v is obtained by computing the number of successors v has, that is the size of v's adjacency list. Assuming that the adjacency list of v is a simple linked list, it takes $\Theta(\text{degree}(v))$ to compute the out-degree of v (i.e., iterating over v's successors and counting them). Note that if the implementation of the adjacency list allows computing the size of a list in $\Theta(1)$ (e.g., the size of the list is stored and updated upon addition and deletion operations), so is the time needed to compute the out-degree of a vertex. For the in-degree, one needs to count the number of vertices that have v in their adjacency lists. This requires to iterate over all the vertices ($\Theta(|V|)$ operations) and, for each vertex u, to search for v in its adjacency list ($O(\text{degree}(u))$ operations). Therefore it takes $O(|V| + |E|)$ to compute the in-degree of a vertex.

2 (Exercise 22.1-2) Give an adjacency-list representation for a complete binary tree on 7 vertices. Give an equivalent adjacency-matrix representation. Assume that vertices are numbered from 1 to 7 as in a binary heap.

Solution:

Adjacency-list representation:

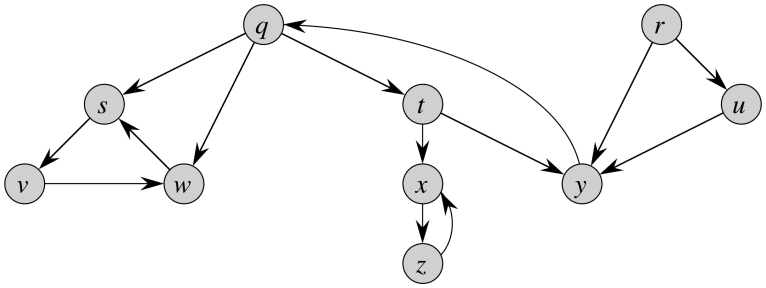


Adjacency-matrix representation:

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	0	0	0	1	1	0	0
3	0	0	0	0	0	1	1
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

BFS and DFS

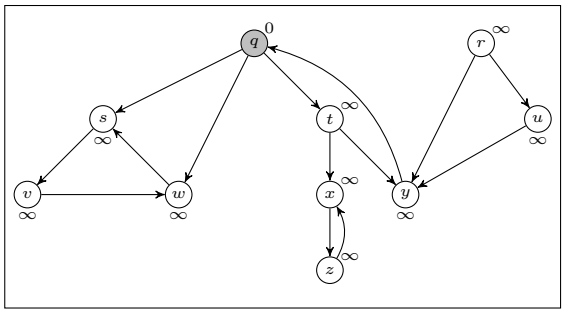
3 (Adaptation of Exercise 22.3-2) Consider the following directed graph:



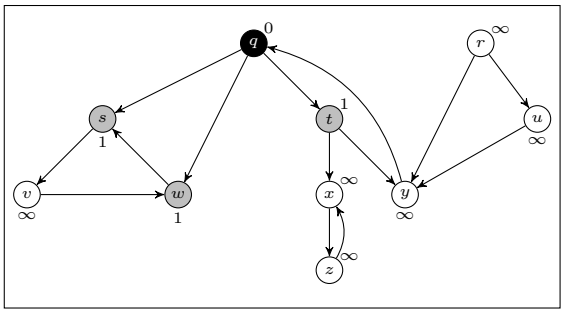
3a Show how breadth-first search works on the graph if we start with q as a source and each adjacency list is ordered alphabetically. Show the length of the shortest path from the source to each vertex as the algorithm progresses.

Solution: As BFS progresses, every vertex has a color:

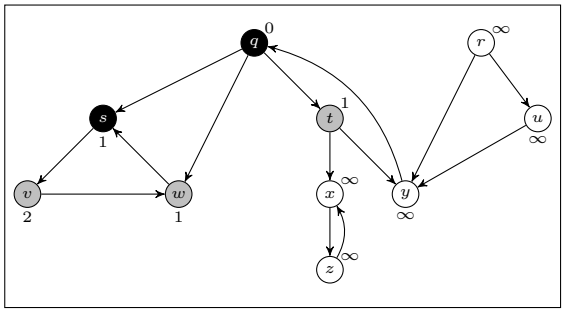
- WHITE = undiscovered
- GRAY = discovered, still adjacent to white vertices
  (representing the "frontier" of the BFS)
- BLACK = finished
  (have discovered everything reachable from it)



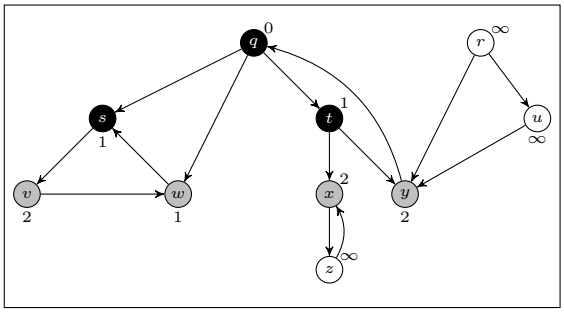
Queue $Q = q$



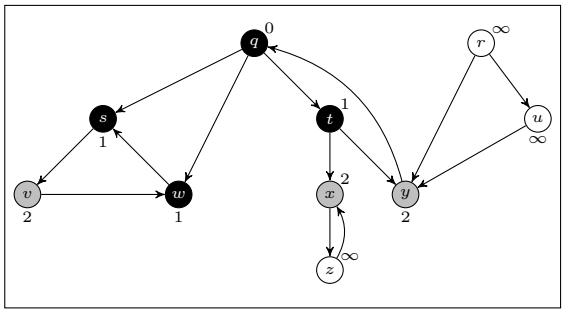
Queue $Q = s, t, w$



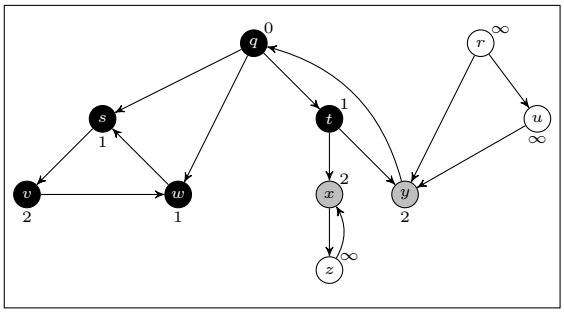
Queue $Q = t, w, v$



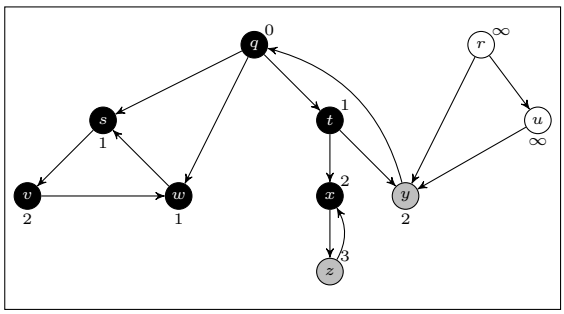
Queue $Q = w, v, x, y$



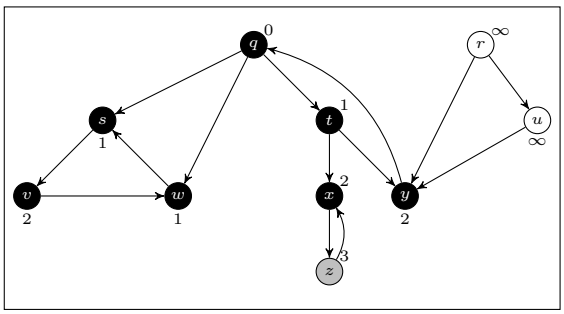
Queue $Q = v, x, y$



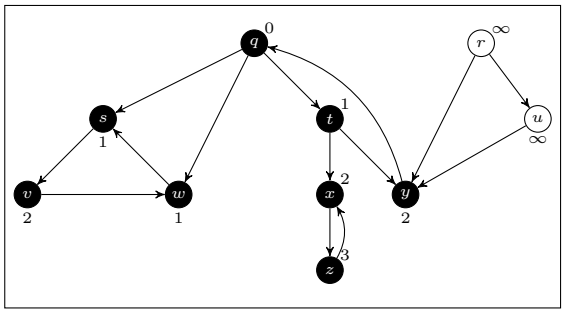
Queue $Q = x, y$



Queue $Q = y, z$



Queue $Q = z$



Queue $Q = \emptyset$

3b Show how depth-first search works on the graph. Assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.

Solution: As DFS progresses, every vertex (and edge) has a color:

- A vertex has color

  WHITE if it is undiscovered;

  GRAY discovered but not finished (not done exploring from it);

  BLACK = finished (have found everything reachable from it).

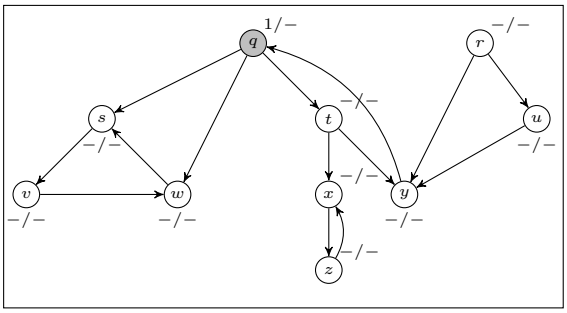
- An edge has color

  ORANGE: if it is a tree edge;

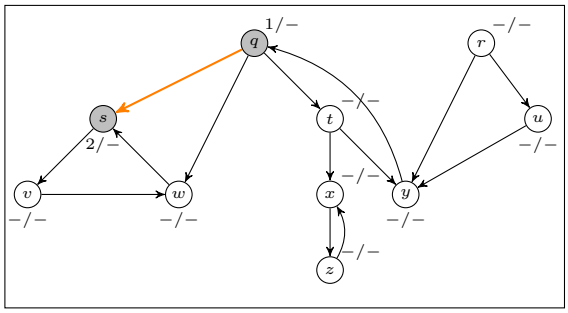
  BLUE: if it is a back edge;

  RED: if it is a foward edge;

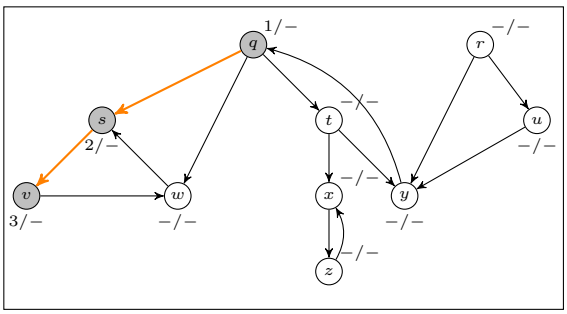
  GREEN: if it is a cross edge.



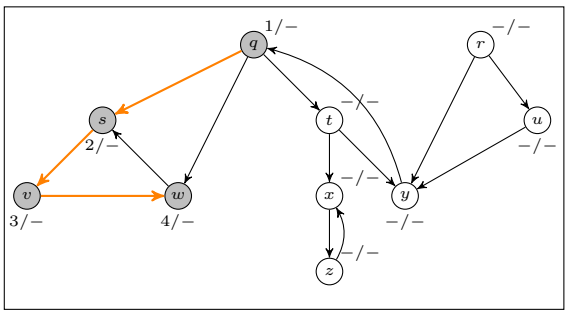
Time 1



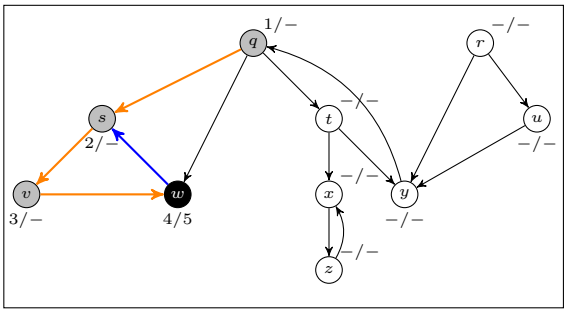
Time 2



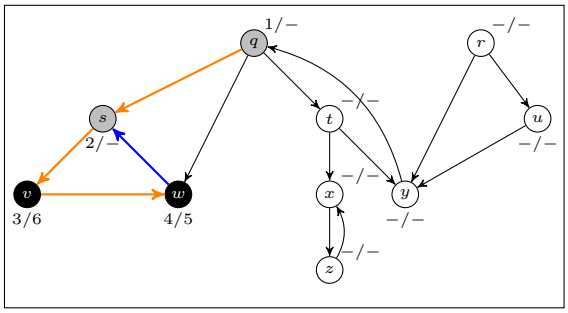
Time 3



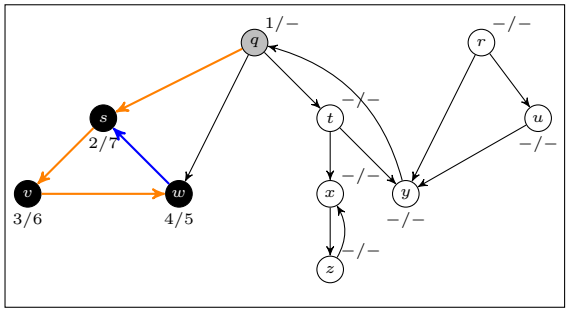
Time 4



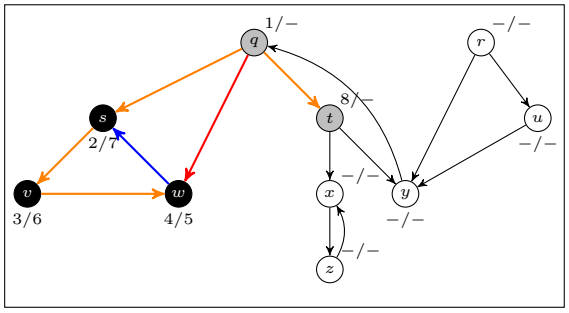
Time 5



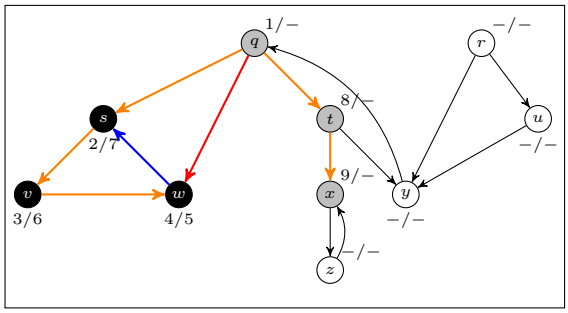
Time 6



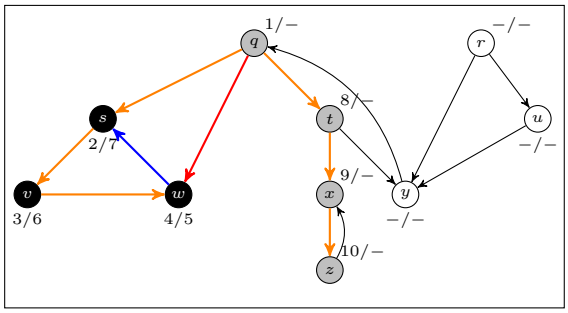
Time 7



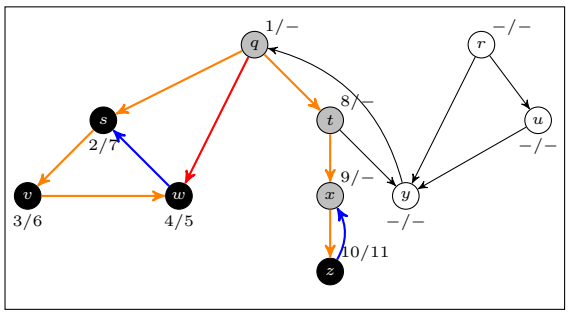
Time 8



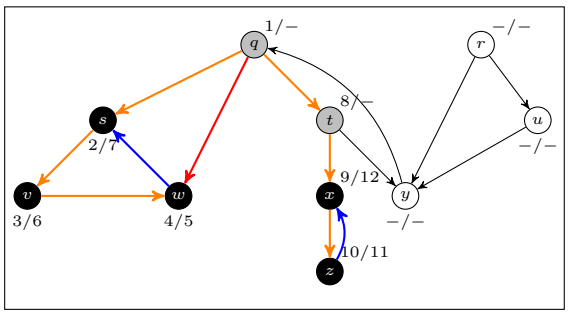
Time 9



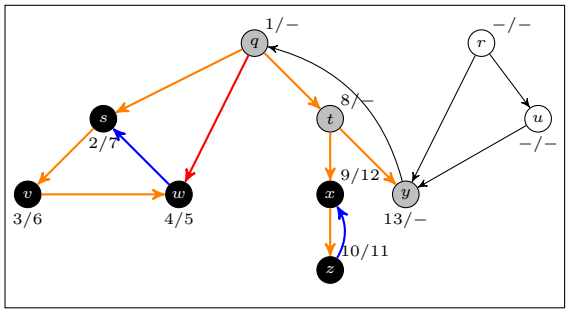
Time 10



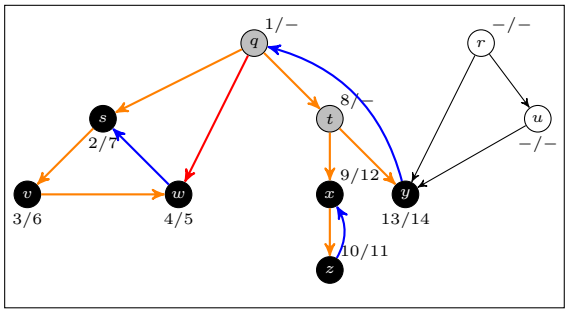
Time 11



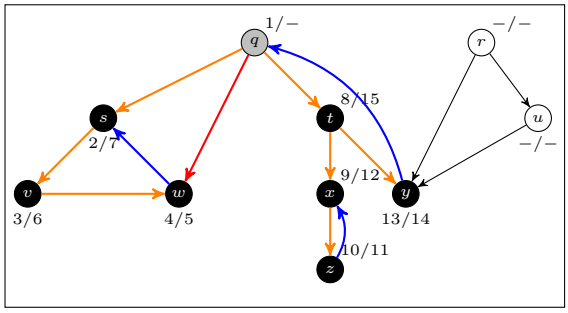
Time 12



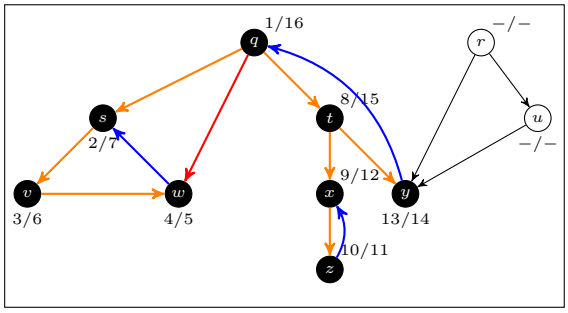
Time 13



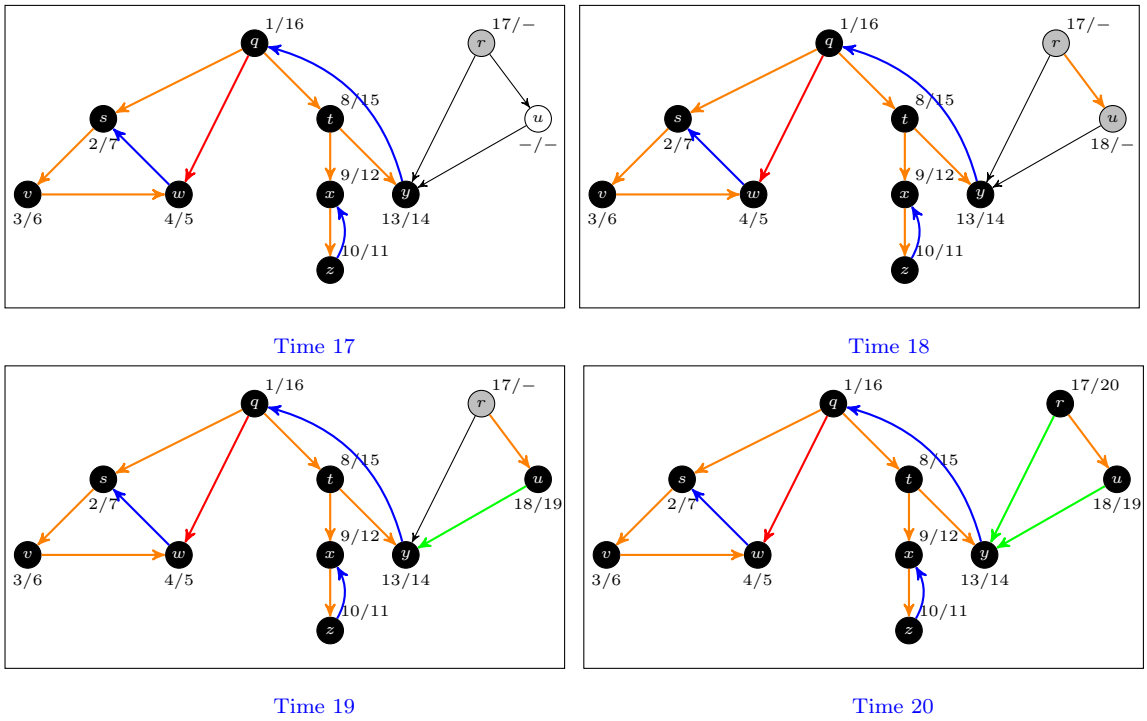
Time 14



Time 15



Time 16



Note: If there was an edge from t to w it would have been a *cross-edge*.

4 (Exercise 22.2-4) What is the running time of BFS if we represent its input graph by an adjacency matrix and modify the algorithm to handle this form of input?

Solution: Each vertex is examined at most once. When examining a vertex v, BFS iterates over all the vertices adjacent to v, which takes $\Theta(|V|)$ using a adjacency-matrix representation (i.e., scanning through an entire row of the adjacency matrix). Therefore, the running time of BFS is $O(|V|^2)$. To handle inputs represented by an adjacency matrix, the instruction "for each $v \in G.Adj[u]$" must be replaced by "for each $v \in V$" and "if $v.d == \infty$" must be replaced by "if $v.d == \infty$ and $G.Adj[u, v] == 1$".

BFS(V, E, s)
1 **for** each $u \in V - \{s\}$
2 $\quad u.d = \infty$
3 $s.d = 0$
4 $Q = \emptyset$
4 ENQUEUE(Q, s)
5 **while** $Q \neq \emptyset$
5 $u =$ DEQUEUE(Q)
5 $\quad$ **for** each $v \in V$
6 $\quad\quad$ **if** $v.d == \infty$ **and** $G.Adj[u, v] == 1$
7 $\quad\quad\quad v.d = u.d + 1$
4 $\quad\quad\quad$ ENQUEUE(Q, v)

---

5 (Exercise 22.3-12) Show that we can use a depth-first search of an undirected graph G 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 it assigns to each vertex v an integer label $v.cc$ between 1 and k, where k is the number of connected components of G, such that $u.cc = v.cc$ if and only if u and v are in the same connected component.

Solution: We modify the algorithm for DFS from CLRS to assign a label from 1 to k to each vertex, where k is the number of connected components in G.

Before line 5 in DFS(V, E) is executed, note that all vertices are either WHITE (unvisited) or BLACK (visited). All WHITE vertices have not yet been visited and therefore have been assigned label 0. The next WHITE vertex u on which DFS-Visit(u,label) (in line 8) is called is assigned the next consecutive label. In a single call to DFS-Visit(u,label) on line 8 of DFS(V, E), each vertex v in the connected component with vertex u are (recursively) visited by a call to DFS-Visit(v,label) on line 7 of DFS-Visit and are therefore each assigned the same label as vertex u.

(The modified pseudocode for depth-first search is above.)

```
DFS(V, E)
1   for each u ∈ V
2     do color[u] ← WHITE, cc[u] ← 0
3   time ← 0
4   label ← 0
5   for each u ∈ V
6     if color[u] = WHITE
7       then label = label + 1
8            DFS-Visit(u, label)
```

```
DFS-Visit(u, label)
1   color[u] ← GRAY
2   time ← time + 1
3   d[u] ← time
4   cc[u] ← label
5   for each v ∈ Adj[u]
6     if color[v] = WHITE
7       then DFS-Visit(v, label)
8   color[u] ← BLACK
9   time ← time + 1
10   f[u] ← time
```