问题与反馈

2014.10.15

22.1-8

Suppose that instead of a linked list, each array entry $Adj[u]$ is a hash table containing the vertices v for which $(u, v) \in E$. If all edge lookups are equally likely, what is the expected time to determine whether an edge is in the graph? What disadvantages does this scheme have? Suggest an alternate data structure for each edge list that solves these problems. Does your alternative have disadvantages compared to the hash table?

```
BFS(G, s)
1  for each vertex u ∈ G.V − {s}
2      u.color = WHITE
3      u.d = ∞
4      u.π = NIL
5  s.color = GRAY
6  s.d = 0
7  s.π = NIL
8  Q = ∅
9  ENQUEUE(Q, s)
10 while Q ≠ ∅
11     u = DEQUEUE(Q)
12     for each v ∈ G.Adj[u]
13         if v.color == WHITE
14             v.color = GRAY
15             v.d = u.d + 1
16             v.π = u
17             ENQUEUE(Q, v)
18     u.color = BLACK
```

22.2-3

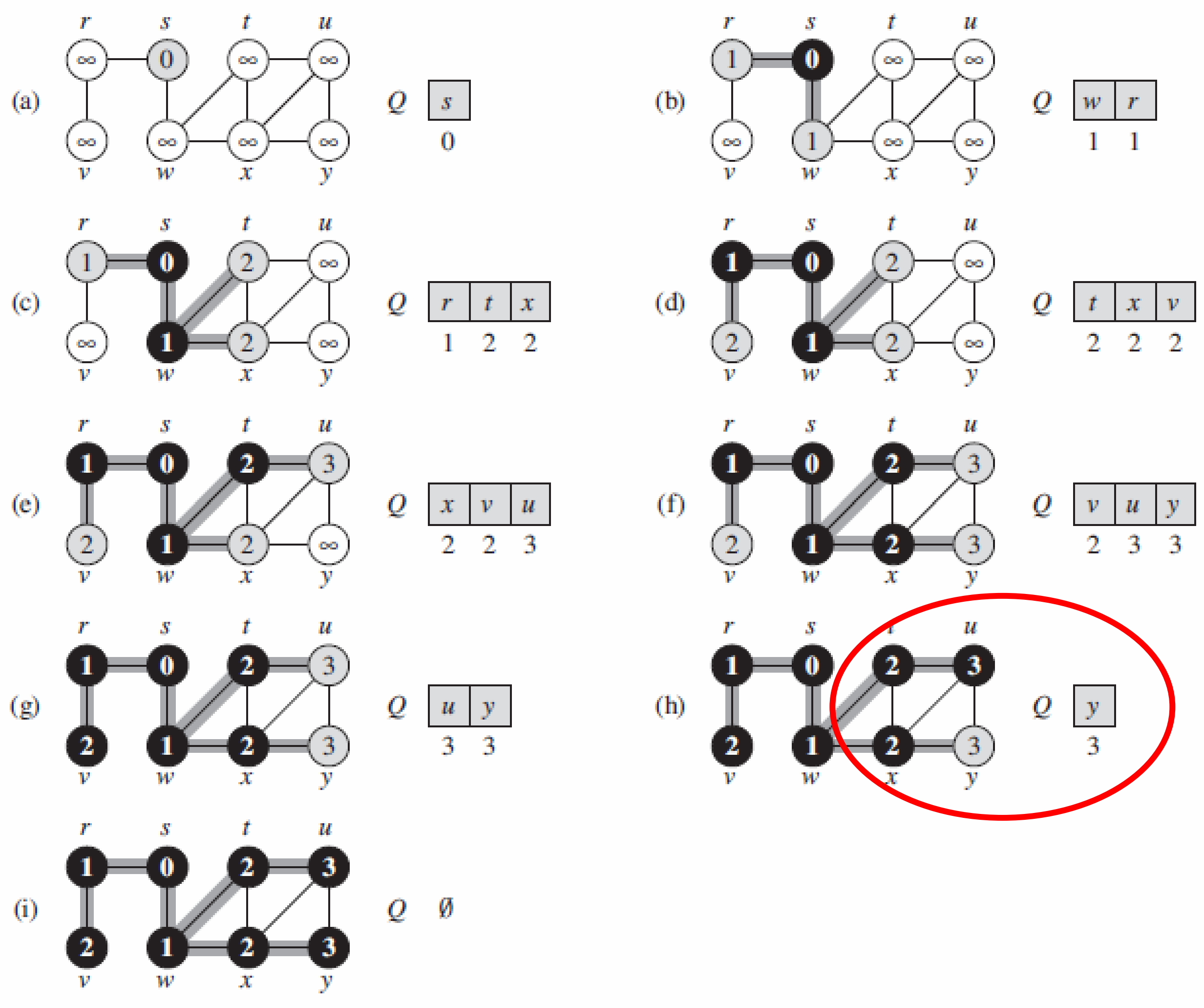
Show that using a single bit to store each vertex color suffices by arguing that the BFS procedure would produce the same result if lines 5 and 14 were removed.

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?

22.2-5

Argue that in a breadth-first search, the value $u.d$ assigned to a vertex u is independent of the order in which the vertices appear in each adjacency list. Using Figure 22.3 as an example, show that the breadth-first tree computed by BFS can depend on the ordering within adjacency lists.



(a)
r s t u
v w x y
Q s
0
(b)
Q w r
1 1
(c)
Q r t x
1 2 2
(d)
Q t x v
2 2 2
(e)
Q x v u
2 2 3
(f)
Q v u y
2 3 3
(g)
Q u y
3 3
(h)
Q y
3
(i)
Q ∅

22.3-6

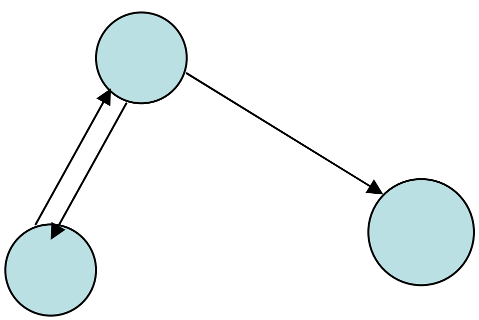
Show that in an undirected graph, classifying an edge (u, v) as a tree edge or a back edge according to whether (u, v) or (v, u) is encountered first during the depth-first search is equivalent to classifying it according to the ordering of the four types in the classification scheme.

22.3-7

Rewrite the procedure DFS, using a stack to eliminate recursion.

22.3-8

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.



```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1              // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]        // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK              // blacken u; it is finished
time = time + 1
u.f = time
```

22.3-9

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 in $v.d \leq u.f$.

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 search so that it as where k is the num and only if u and v

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1                  // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]            // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                  // blacken u; it is finished
time = time + 1
u.f = time
```

22.4-2

Give a linear-time algorithm that takes as input a directed acyclic graph $G = (V, E)$ and two vertices s and t, and returns the number of simple paths from s to t in G. For example, the directed acyclic graph of Figure 22.8 contains exactly four simple paths from vertex p to vertex v: pov, $poryv$, $posryv$, and $psryv$. (Your algorithm needs only to count the simple paths, not list them.)

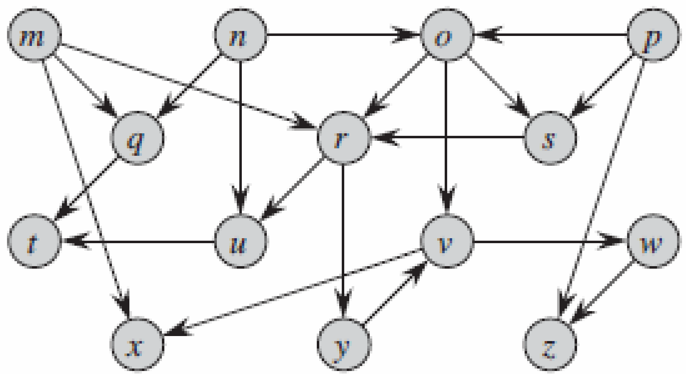


Figure 22.8 A dag for topological sorting.

22.4-3

Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time, independent of $|E|$.

22.5-5

Give an $O(V + E)$-time algorithm to compute the component graph of a directed graph $G = (V, E)$. Make sure that there is at most one edge between two vertices in the component graph your algorithm produces.

22.5-7

A directed graph $G = (V, E)$ is ***semiconnected*** if, for all pairs of vertices $u, v \in V$, we have $u \leadsto v$ or $v \leadsto u$. Give an efficient algorithm to determine whether or not G is semiconnected. Prove that your algorithm is correct, and analyze its running time.