Selected Solutions for Chapter 22: Elementary Graph Algorithms

Solution to Exercise 22.1-7

$$BB^T(i,j) = \sum_{e \in E} b_{ie} b^T_{ej} = \sum_{e \in E} b_{ie} b_{je}$$

- If $i = j$, then $b_{ie} b_{je} = 1$ (it is $1 \cdot 1$ or $(-1) \cdot (-1)$) whenever e enters or leaves vertex i, and 0 otherwise.
- If $i \neq j$, then $b_{ie} b_{je} = -1$ when $e = (i, j)$ or $e = (j, i)$, and 0 otherwise.

Thus,

$$BB^T(i,j) = \begin{cases} \text{degree of } i = \text{in-degree} + \text{out-degree} & \text{if } i = j \,, \\ -(\text{\# of edges connecting } i \text{ and } j) & \text{if } i \neq j \,. \end{cases}$$

Solution to Exercise 22.2-5

The correctness proof for the BFS algorithm shows that $u.d = \delta(s,u)$, and the algorithm doesn't assume that the adjacency lists are in any particular order.

In Figure 22.3, if t precedes x in $Adj[w]$, we can get the breadth-first tree shown in the figure. But if x precedes t in $Adj[w]$ and u precedes y in $Adj[x]$, we can get edge (x, u) in the breadth-first tree.

Solution to Exercise 22.3-12

The following pseudocode modifies the DFS and DFS-VISIT procedures to assign values to the *cc* attributes of vertices.

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
counter = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        counter = counter + 1
        DFS-VISIT(G, u, counter)
```

```
DFS-VISIT(G, u, counter)
u.cc = counter                // label the vertex
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v, counter)
u.color = BLACK
time = time + 1
u.f = time
```

This DFS increments a counter each time DFS-VISIT is called to grow a new tree in the DFS forest. Every vertex visited (and added to the tree) by DFS-VISIT is labeled with that same counter value. Thus $u.cc = v.cc$ if and only if u and v are visited in the same call to DFS-VISIT from DFS, and the final value of the counter is the number of calls that were made to DFS-VISIT by DFS. Also, since every vertex is visited eventually, every vertex is labeled.

Thus all we need to show is that the vertices visited by each call to DFS-VISIT from DFS are exactly the vertices in one connected component of G.

- All vertices in a connected component are visited by one call to DFS-VISIT from DFS:

  Let u be the first vertex in component C visited by DFS-VISIT. Since a vertex becomes non-white only when it is visited, all vertices in C are white when DFS-VISIT is called for u. Thus, by the white-path theorem, all vertices in C become descendants of u in the forest, which means that all vertices in C are visited (by recursive calls to DFS-VISIT) before DFS-VISIT returns to DFS.

- All vertices visited by one call to DFS-VISIT from DFS are in the same connected component:

  If two vertices are visited in the same call to DFS-VISIT from DFS, they are in the same connected component, because vertices are visited only by following paths in G (by following edges found in adjacency lists, starting from some vertex).

Solution to Exercise 22.4-3

An undirected graph is acyclic (i.e., a forest) if and only if a DFS yields no back edges.

- If there's a back edge, there's a cycle.
- If there's no back edge, then by Theorem 22.10, there are only tree edges. Hence, the graph is acyclic.

Thus, we can run DFS: if we find a back edge, there's a cycle.

- Time: $O(V)$. (Not $O(V + E)$!)
  If we ever see $|V|$ distinct edges, we must have seen a back edge because (by Theorem B.2 on p. 1174) in an acyclic (undirected) forest, $|E| \le |V| - 1$.

Solution to Problem 22-1

a.
1. Suppose (u, v) is a back edge or a forward edge in a BFS of an undirected graph. Then one of u and v, say u, is a proper ancestor of the other (v) in the breadth-first tree. Since we explore all edges of u before exploring any edges of any of u's descendants, we must explore the edge (u, v) at the time we explore u. But then (u, v) must be a tree edge.
2. In BFS, an edge (u, v) is a tree edge when we set $v.\pi = u$. But we only do so when we set $v.d = u.d + 1$. Since neither $u.d$ nor $v.d$ ever changes thereafter, we have $v.d = u.d + 1$ when BFS completes.
3. Consider a cross edge (u, v) where, without loss of generality, u is visited before v. At the time we visit u, vertex v must already be on the queue, for otherwise (u, v) would be a tree edge. Because v is on the queue, we have $v.d \le u.d + 1$ by Lemma 22.3. By Corollary 22.4, we have $v.d \ge u.d$. Thus, either $v.d = u.d$ or $v.d = u.d + 1$.

b.
1. Suppose (u, v) is a forward edge. Then we would have explored it while visiting u, and it would have been a tree edge.
2. Same as for undirected graphs.
3. For any edge (u, v), whether or not it's a cross edge, we cannot have $v.d > u.d + 1$, since we visit v at the latest when we explore edge (u, v). Thus, $v.d \le u.d + 1$.
4. Clearly, $v.d \ge 0$ for all vertices v. For a back edge (u, v), v is an ancestor of u in the breadth-first tree, which means that $v.d \le u.d$. (Note that since self-loops are considered to be back edges, we could have $u = v$.)