Algorithms: Solutions 10

Problem 1
Write algorithms for converting (a) an adjacency-list representation of a graph into an adjacency matrix and (b) an adjacency matrix into adjacency lists.

We denote the adjacency list of a vertex u by *Adj-List*$[u]$, and the adjacency-matrix element for vertices u and v by *Adj-Matrix*$[u, v]$. The time complexity of both algorithms is $\Theta(V^2)$.

(a) Converting adjacency lists into a matrix.

```
LISTS-TO-MATRIX(G)      ▷ G is represented by adjacency lists
for each u ∈ V[G]
    do for each v ∈ V[G]
            do Adj-Matrix[u, v] ← 0
       for each v ∈ Adj-List[u]
            do Adj-Matrix[u, v] ← 1
```

(b) Converting an adjacency matrix into lists.

```
MATRIX-TO-LISTS(G)      ▷ G is represented by an adjacency matrix
for each u ∈ V[G]
    do initialize an empty list Adj-List[u]
       for each v ∈ V[G]
            do if Adj-Matrix[u, v] = 1
                    then add v to Adj-List[u]
```

Problem 2
Consider a directed graph with n vertices, represented by an adjacency matrix $M[1..n, 1..n]$. A vertex is called a *sink* if it has $(n-1)$ incoming edges and no outgoing edges. Give an algorithm that finds the sink and returns its number; if the graph has no sink, return 0.

```
FIND-SINK(M, n)
i ← 1
j ← 1
while i < n and j ≤ n      ▷ find a sink candidate i
    do if M[i, j] = 0
            then j ← j + 1
            else i ← i + 1
for k ← 1 to n      ▷ check whether i is a sink
    do if M[i, k] = 1
            then return 0
       if k ≠ i and M[k, i] = 0
            then return 0
return i
```

Problem 3
Describe a data structure for representing a graph that supports the following operations:

- Check the presence of an edge between two given vertices, in $O(\lg V)$ time.
- Add an edge between two given vertices, in $O(\lg V)$ time.
- Perform the breadth-first search, in $O(V + E)$ time.

We use the "adjacency-list" representation with red-black trees instead of linked lists; that is, for each vertex in the graph, we keep a red-black tree with adjacent vertices.