Analysis of Algorithms: Solutions 6

```
                                         X
                                         X
                                         X
                                         X
                                         X
                                         X
                                         X
                                         X
                                         X
                          X        X     X
number of                 X        X  X  X
homeworks                 X        X  X  X
                    X     X  X     X  X  X
                    X     X  X  X  X  X  X
                    X     X  X  X  X  X  X
           --------------------------------
           0  1  2  3  4  5  6  7  8  9  10
                        grades
```

The histogram shows the distribution of grades for the homeworks submitted on time.

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. Give the time complexity of your algorithms.

We denote an adjacency list of a vertex u by *Adj-List*$[u]$, and an 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) $\quad \triangleright$ G is represented by adjacency lists.
for each $u \in V[G]$
  do for each $v \in V[G]$
    do *Adj-Matrix*$[u, v] \leftarrow 0$
for each $u \in V[G]$
  do for each $v \in$ *Adj-List*$[u]$
    do *Adj-Matrix*$[u, v] \leftarrow 1$

(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 u ∈ V[G]
    do for each v ∈ V[G]
            do if Adj-Matrix[u, v] = 1
                    then add v to Adj-List[u]
```

Problem 2 (3 points)
Using Figure 23.3 in the textbook as a model, illustrate the steps of breadth-first search on the directed graph of Figure 23.2(a), with vertex 3 as the source.

The order of painting the vertices is as follows:

gray 3	black 5
gray 5	black 6
gray 6	gray 2
black 3	black 4
gray 4	black 2

Problem 3
The depth-first search algorithm may be used to identify the connected components of an *undirected* graph. Write a modified version of DFS for performing this task.

We use the *component* field of a vertex instead of the color. Initially, this field is set to 0. When the DFS algorithm discovers the vertex, it replaces 0 with the component number.

```
DFS-COMPONENTS(G)
k ← 0   ▷ Component counter.
for each u ∈ V[G]
    do component[u] ← 0
for each u ∈ V[G]
    do if component[u] ≠ 0
            then k ← k + 1
                 DFS-VISIT(u, k)
return k
```

```
DFS-VISIT(u, k)
component[u] ← k   ▷ u has just been discovered.
for each v ∈ Adj[u]
    do if component[v] = 0
            then DFS-VISIT(v, k)
```