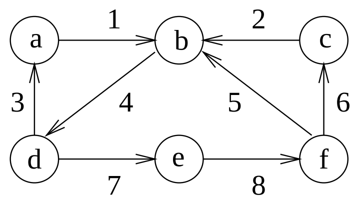
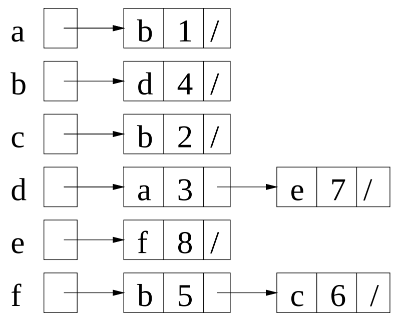
Analysis of Algorithms: Solutions 8

```
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
              X
number of     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
    X X X X X X
    ------------
    5 6 7 8 9 10
       grades
```

Problem 1

Give the adjacency-list and adjacency-matrix representation of the following weighted graph.





	a	b	c	d	e	f
a	0	1	0	0	0	0
b	0	0	0	4	0	0
c	0	2	0	0	0	0
d	3	0	0	0	7	0
e	0	0	0	0	0	8
f	0	5	6	0	0	0

Problem 2

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) $\quad\triangleright$ 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) $\quad\triangleright$ G is represented by an adjacency matrix
for each $u \in V[G]$
  do initialize an empty list *Adj-List*$[u]$
for each $u \in V[G]$
  do for each $v \in V[G]$
    do if *Adj-Matrix*$[u, v] = 1$
      then add v to *Adj-List*$[u]$

Problem 3
Consider a directed graph with n vertices, represented by a 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 vertex; if the graph has no sink, it should return 0.

The following algorithm consists of two parts: the first loop finds a vertex i that *may* be a sink, and ensures that no other vertex is a sink; the second loop tests whether i is indeed a sink. The running time is $\Theta(n)$.

```
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
```