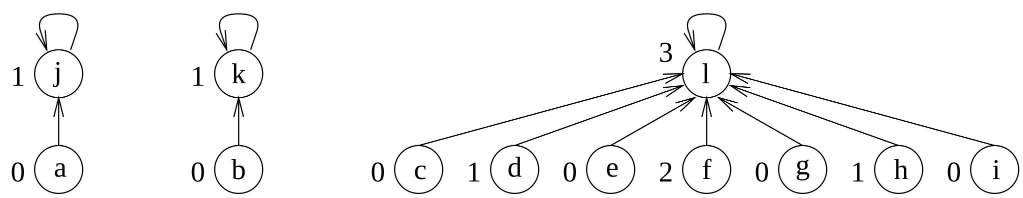
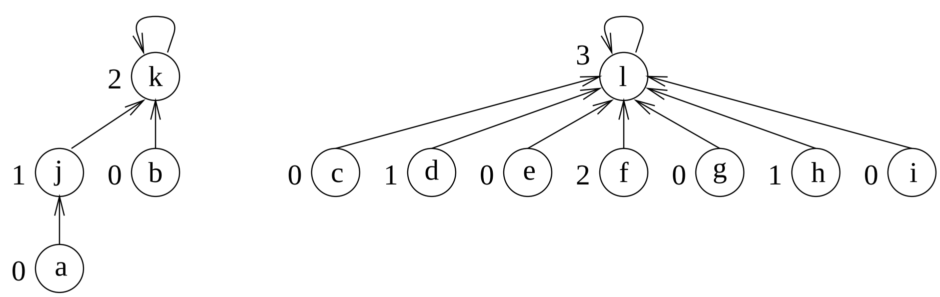
# Analysis of Algorithms: Solutions 7

```
                                    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       X   X   X   X   X
            -------------------------------
            3   4   5   6   7   8   9   10
                         grades
```

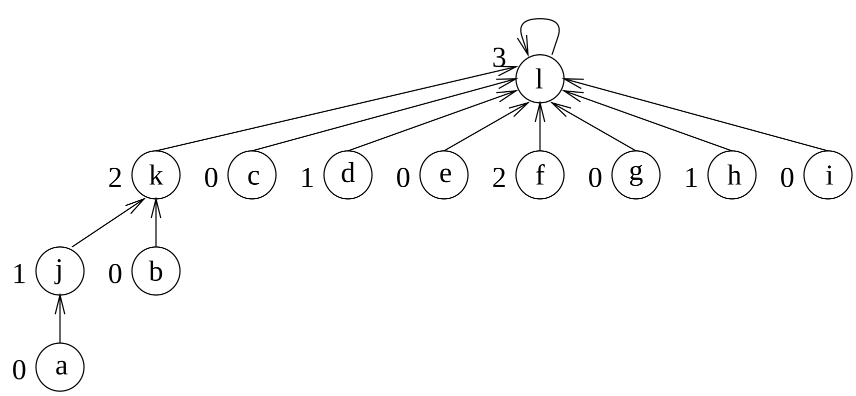
**Problem 1**
Consider the disjoint-set forest below, where numbers are the ranks of elements, and suppose that you apply three successive operations: UNION$(a, b)$, UNION$(b, c)$, and FIND-SET$(a)$. Give a picture of the disjoint forest after each of these operations.



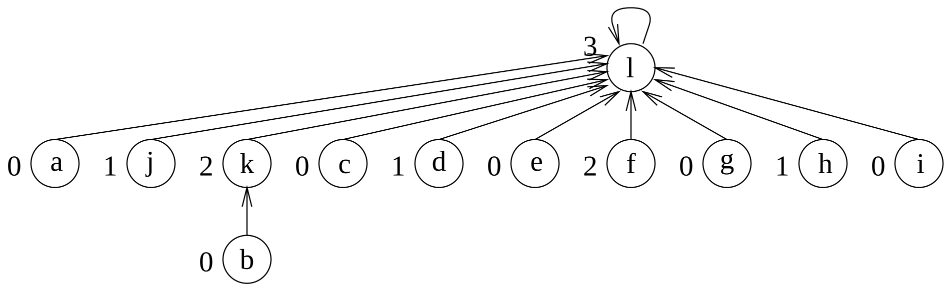

UNION$(a, b)$:




UNION$(b, c)$:



FIND-SET($a$):




**Problem 2**

Write pseudocode for MAKE-SET, FIND-SET, and UNION, using the linked-list representation of disjoint sets.

We use four fields for each element $x$ of a linked list:

> *next*[$x$]: pointer to the next element of the list; NIL if $x$ is the last element
> *rep*[$x$]: pointer to the set representative, that is, to the first element of the list
> *last*[$x$]: if $x$ is the first element of a list, then this field points to the last element
> *size*[$x$]: if $x$ is the first element, then this field contains the size of the list

If $x$ is not the first element of a list, then the algorithms do *not* use its *last* and *size* fields, and the information in these fields may be incorrect.

```
MAKE-SET(x)
next[x] ← NIL
rep[x] ← x
last[x] ← x
size[x] ← 1

FIND-SET(x)
return rep[x]

UNION(x, y)
if size[rep[x]] > size[rep[y]]
    then APPEND(rep[x], rep[y])
    else APPEND(rep[y], rep[x])

APPEND(x, y)
next[last[x]] ← y
size[x] ← size[x] + size[y]
z ← y
while z ≠ NIL        ▷ change the rep pointers in the second list
    do rep[z] ← x
       z ← next[z]
```

**Problem 3**

Suppose that $A[1..n]$ and $B[1..m]$ are sorted arrays, and $n \leq m$. Write an algorithm that finds their smallest common element; if they have no common elements, it should return 0.

The intuitive idea is to divide $B[1..m]$ into segments, each of size $k = m/n$, and perform binary search in each segment. We need to use a version of binary search, BIN-SEARCH$(B, p, r, k)$, which searches for an element $k$ in a segment $B[p..r]$. If this version finds $k$, it returns the corresponding index of $B$; if not, it returns the index of the next larger element. For example, if $k = 6$ and $B[p..r] = \langle 3, 5, 7, 9 \rangle$, the search returns the index of 7. The following algorithm calls BIN-SEARCH on $k$-element segments of $B$.

```
COMMON-ELEMENT(A, B, n, m)
k ← ⌊m/n⌋
i ← 1
j ← 1
while i ≤ n and j ≤ m
    do  if A[i] = B[j]
          then return A[i]
        if A[i] < B[j]
          then i = i + 1
          else repeat j = j + k
                 until j > m or A[i] ≤ B[j]
               j ← BIN-SEARCH(B, j − k + 1, min(j, m), A[i])
return 0
```

The running time of COMMON-ELEMENT is $O(n \cdot (1 + \lg \frac{m}{n}))$. In particular, if $A$ and $B$ are of about the same size, then the time is $O(m)$. On the other hand, if $A$ is much smaller than $B$, the running time is significantly better than $O(m)$.