Analysis of Algorithms: Solutions 6

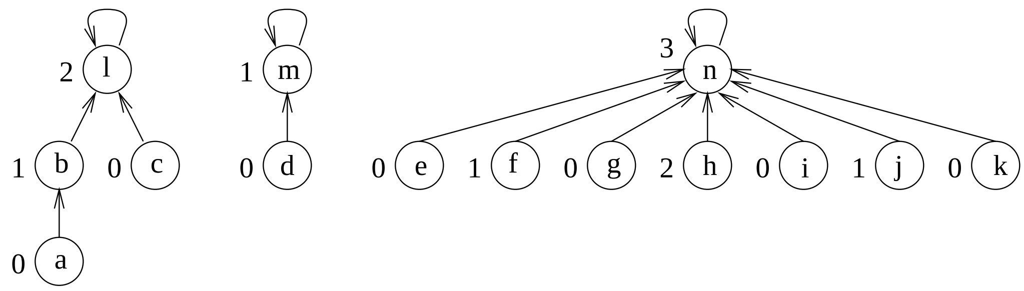
```
                X           X
                X           X
                X           X
                X       X   X
                X   X   X   X
                X   X   X   X       X
        X       X   X   X   X   X   X
X       X   X   X   X   X   X   X   X
------------------------------------
2   3   4   5   6   7   8   9   10  11
              grades
```

Problem 1

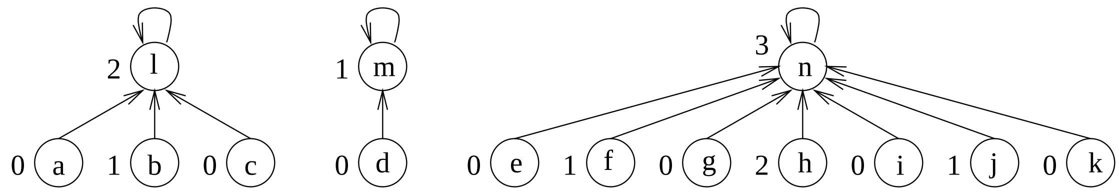
Suppose we apply the Connected-Components algorithm to an undirected graph G, with vertices $G[V] = \{a, b, c, d, e, f, g, h, i, j, k\}$, and its edges $E[G]$ are processed in the following order: $(e, g), (a, d), (i, k), (c, g), (b, f), (b, h), (f, k), (a, k), (f, h), (d, i)$. Using Figure 22.1 in the textbook as a model, illustrate the steps of Connected-Components on this graph.

Edge	Disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$	$\{k\}$
(e, g)	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e, g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$	$\{k\}$
(a, d)	$\{a, d\}$	$\{b\}$	$\{c\}$		$\{e, g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$	$\{k\}$
(i, k)	$\{a, d\}$	$\{b\}$	$\{c\}$		$\{e, g\}$	$\{f\}$		$\{h\}$	$\{i, k\}$	$\{j\}$	
(c, g)	$\{a, d\}$	$\{b\}$	$\{c, e, g\}$			$\{f\}$		$\{h\}$	$\{i, k\}$	$\{j\}$	
(b, f)	$\{a, d\}$	$\{b, f\}$	$\{c, e, g\}$					$\{h\}$	$\{i, k\}$	$\{j\}$	
(b, h)	$\{a, d\}$	$\{b, f, h\}$	$\{c, e, g\}$						$\{i, k\}$	$\{j\}$	
(f, k)	$\{a, d\}$	$\{b, f, h, i, k\}$	$\{c, e, g\}$							$\{j\}$	
(a, k)	$\{a, b, d, f, h, i, k\}$		$\{c, e, g\}$							$\{j\}$	
(f, h)	$\{a, b, d, f, h, i, k\}$		$\{c, e, g\}$							$\{j\}$	
(d, i)	$\{a, b, d, f, h, i, k\}$		$\{c, e, g\}$							$\{j\}$	

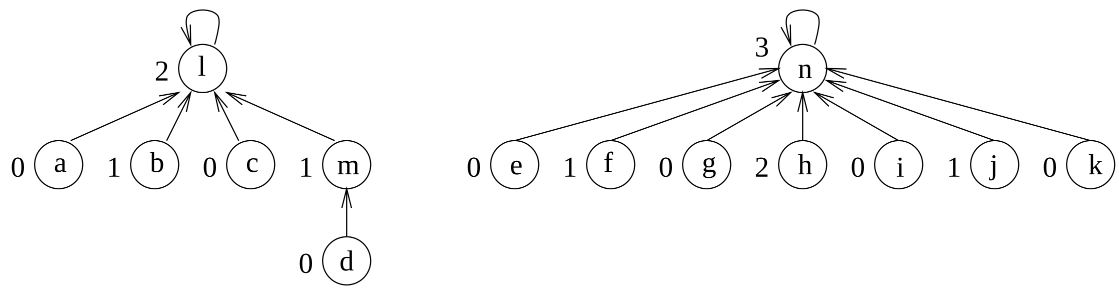
Problem 2
Consider the disjoint-set forest shown below, where numbers are the ranks of elements, and suppose that you apply three successive operations to this forest: FIND-SET(a), UNION(l, d), and UNION(d, e). Give a picture of the disjoint forest after each of these operations.



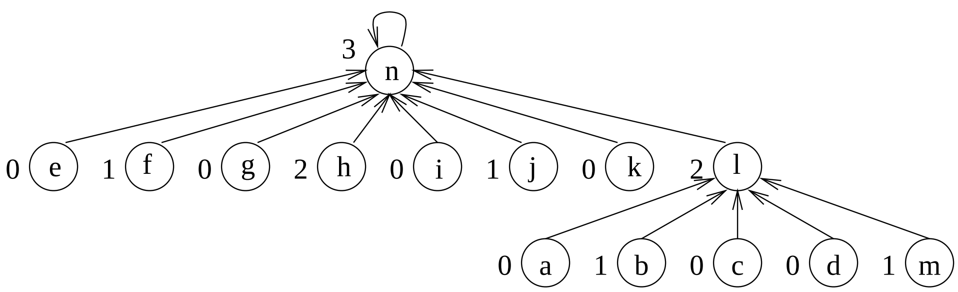
FIND-SET(a):



UNION(l, d):



UNION(d, e):



Problem 3

The *transpose* of a graph G is the result of reversing all edges in G. Write an algorithm that computes the transpose of a given graph.

We denote the array of initial adjacency lists by *Adj-Initial* and the array of transposed adjacency lists by *Adj-Transpose*. The time complexity of the algorithm is $\Theta(V+E)$.

```
TRANSPOSE(G)
for each u ∈ V[G]
    do initialize an empty list Adj-Transpose[u]
for each u ∈ V[G]
    do for each v ∈ Adj-Initial[u]
            do add u to Adj-Transpose[v]
```

Problem 4

Suppose that the rank of each node in a disjoint-set forest must be the exact height of the node, rather than an upper bound on the height. Then, FIND-SET has to change the ranks of the nodes on the compressed path.

Describe a modified representation of the disjoint-set forest, which supports this operation, and the corresponding modifications of MAKE-SET, UNION, and FIND-SET. What is the time complexity of the resulting implementation, in terms of m and n?

The modified representation is similar to the standard disjoint-set forest. The difference is that each node x includes a list *children*$[x]$, which contains all children of x. The running time of this implementation is $O(m \cdot n)$; it is much slower than the standard implementation.

```
MAKE-SET(x)
parent[x] ← x
rank[x] ← 0
initialize an empty list children[x]

UNION(x, y)
LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
if rank[x] > rank[y]
   then parent[y] ← x
        add y to children[x]
   else parent[x] ← y
        add x to children[y]
if rank[x] = rank[y]
   then rank[y] ← rank[y] + 1
```

```
FIND-SET(x)
y ← parent[x]
if y ≠ parent[y]
    ▷ neither x nor parent[x] is the root
   then remove x from children[y]
        RECOMPUTE-RANK[y]
        parent[x] ← FIND-SET(y)
        add x to children[parent[x]]
        if rank[parent[x]] < rank[x] + 1
           then rank[parent[x]] = rank[x] + 1
return parent[x]

RECOMPUTE-RANK(y)
rank[y] ← 0
for each z ∈ children[y]
    do if rank[y] < rank[z] + 1
          then rank[y] ← rank[z] + 1
```