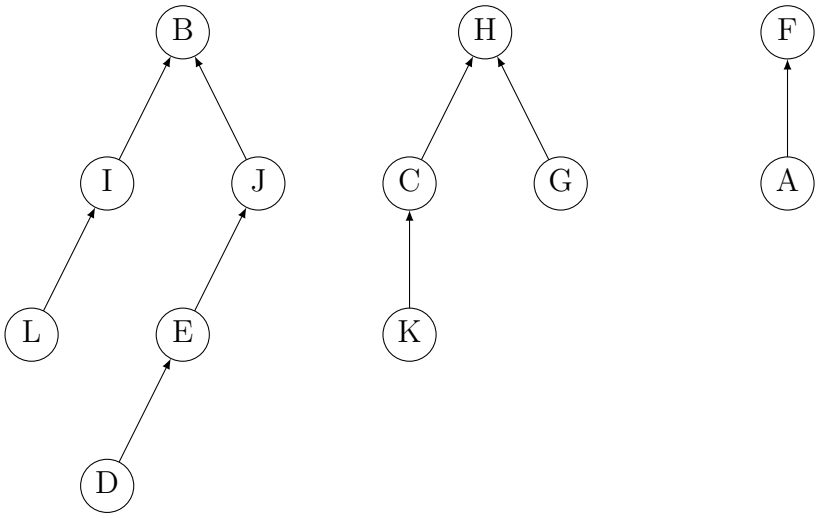
CIS 575. Introduction to Algorithm Analysis
Material for March 27, 2024

Union-Find with Path Compression

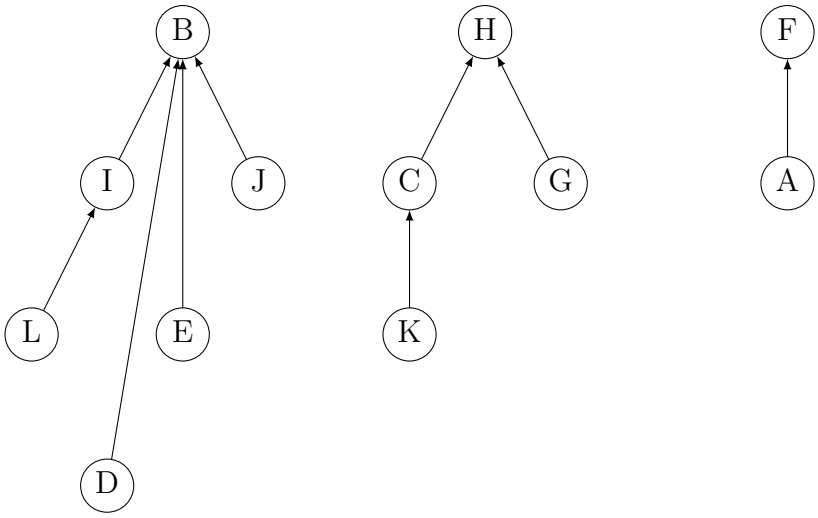
The topic of this note is covered in *Cormen*'s Section 19.3.

1 Path Compression

Recall that we have seen that UNION and FIND can be implemented to run in time logarithmic in the number of nodes. To further cut down the running time, we may apply a simple optimization named *path compression*: whenever FIND is called, all nodes on the path towards the representative will be made to point directly to that representative. Recall our example:



After a call to FIND on D, the tree rooted by B would be "flattened" into



and actually its height would *decrease*. With path compression, it will be too expensive to keep track of the actual height of a tree, and instead we shall only record an upper bound, known as the **rank**. In our example, the height of the tree rooted by B decreases from 3 to 2, but its rank will still be recorded as 3.

Even with this optimization, a *few* operations could take time proportional to the logarithm of the number of nodes. But the *amortized* cost of an operation is much less; here "amortized" is a technical term that can be thought of as "average". With n elements, the cost of doing m operations will be in $O(m \cdot \alpha(n))$ where α is a function that grows *extremely slowly* and which is for all practical purposes a constant. We shall therefore say that such an algorithm runs in *almost linear* time.

You may ask: how is this extremely **slow**ly growing function α defined? It is essentially (cf. Section 19.4 in *Cormen*) the *inverse* of an extremely **fast** growing function known as the **Ackermann** function (possibly defined and explored in a later assignment).