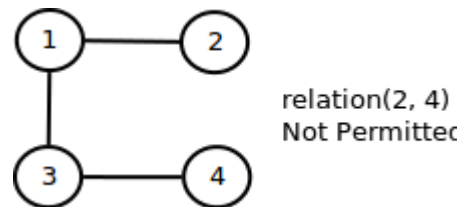
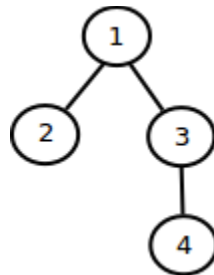


1.5 Case Study: Union-Find

In this case, we actually deal with graph representation. We used array to store each Vertex and dynamically create an edge using function **relation(p, q)** that will update value of one of them. This case called as Union-Find because we need to find corresponding vertex for each p and q, then did merge or union operation. Kind of graph which will be built is **undirected graph** and followed **transitive** rule to define vertex connectivity. Transitive rule is, if p connected to q and q connected to r, then p connected to r. An image below demonstrate transitive rule:



Since vertex (2, 4) could be connected through path 2-1-3-4, then we avoided connecting (2, 4) directly. The final result of union() operation is a tree with a root. A path in the image above may be represented as a **rooted tree** in the image below:



There are five algorithms that are used to implement Union-Find, detail of each algorithm will be described later:

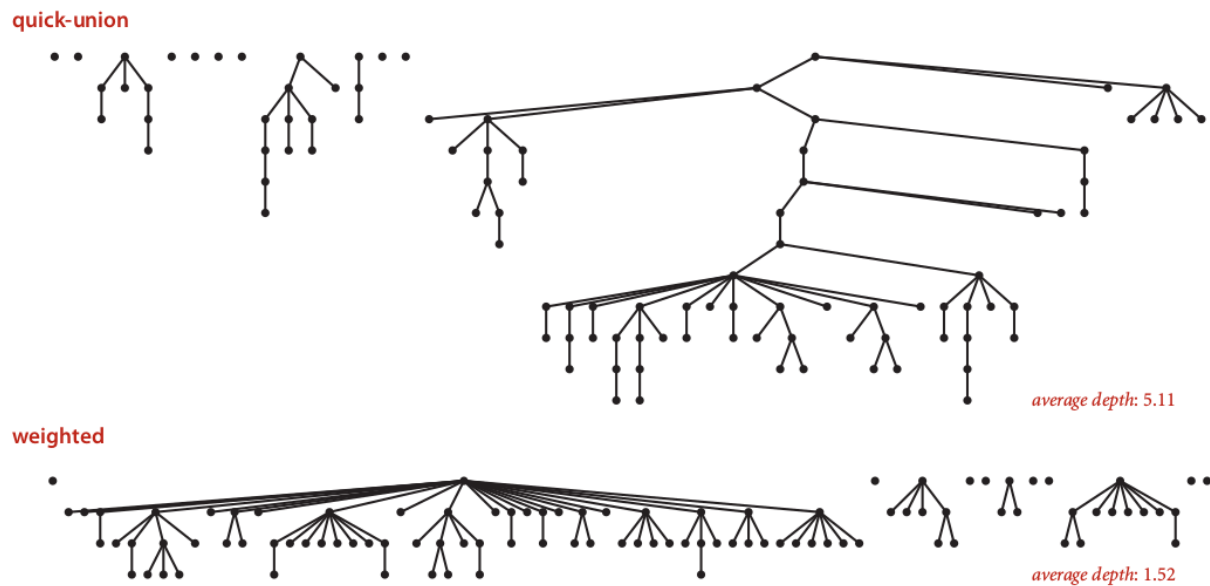
Algorithm	Preprocessing	Union	Find
Quick-Find	N	N	1
Quick-Union	N	1	Tree height
Weighted Quick-Union	N	$\lg N$	$\lg N$
Weighted Quick-Union with Path Compression	N	$\lim_{x \rightarrow 1} f(x)$	

Quick-Find

For example, we want to connect (5, 9) in the given vertex below:

Right tree of root 8 smaller than left tree of root 1, then we need to decide that right tree should be connected as child of left tree.

Weighted Quick-Union amazingly best algorithm so far in practice. The experiment using 100 vertex proof that weighted scheme offered better union performance.



Quick-union and weighted quick-union (100 sites, 88 union() operations)

Weighted Quick-Union with Path Compression

Path compression is a method that each node in a tree should be connected directly to the new root.