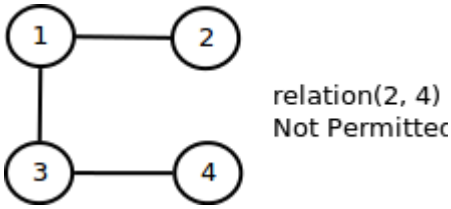
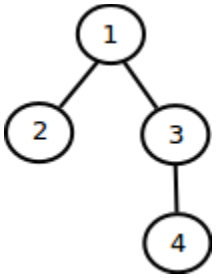
1.5 Case Study: Union-Find

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



There are five algorithm that used to implement Union-Find, detail of each algorithm will 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	$\lim_{x \to 1} f(x)$	

Quick-Find

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

p	q		0	1	2	3	4	5	6	7	8	9
5	9		1	1	1	8	8	0	1	1	8	8
Find								0				8
Union			8	8	8	8	8	8	8	8	8	8

To find each vertex, Quick-Find only need lookup its index directly that cost O(1) complexity. But, for one connectivity, Quick-Union need to update almost all rows. This algorithm does not aware may 5 is a root or child of 0, so this algorithm just traversing any vertex that connected to 5 and change all of them as child of 9. Because if this scheme, union() take N operation for each connectivity or N^2 for all connectivity in worst case scenario.

Quick Union

To increase efficiency in union() operation, we need to aware to construct proper tree representation by find root of each vertex. Take a look for table below:

p	q		0	1	2	3	4	5	6	7	8	9
5	9		1	1	1	8	8	0	1	1	8	8
Find			-	1				-			8	-
Union				8								

We need to find each parent of each vertex until found its root. Since height of each tree varies, than the find() operation cost equal to tree height. The union() operation cost only O(1) since we have already know each root of tree, then union() only need to connecting root of tree in the left into tree in the right.

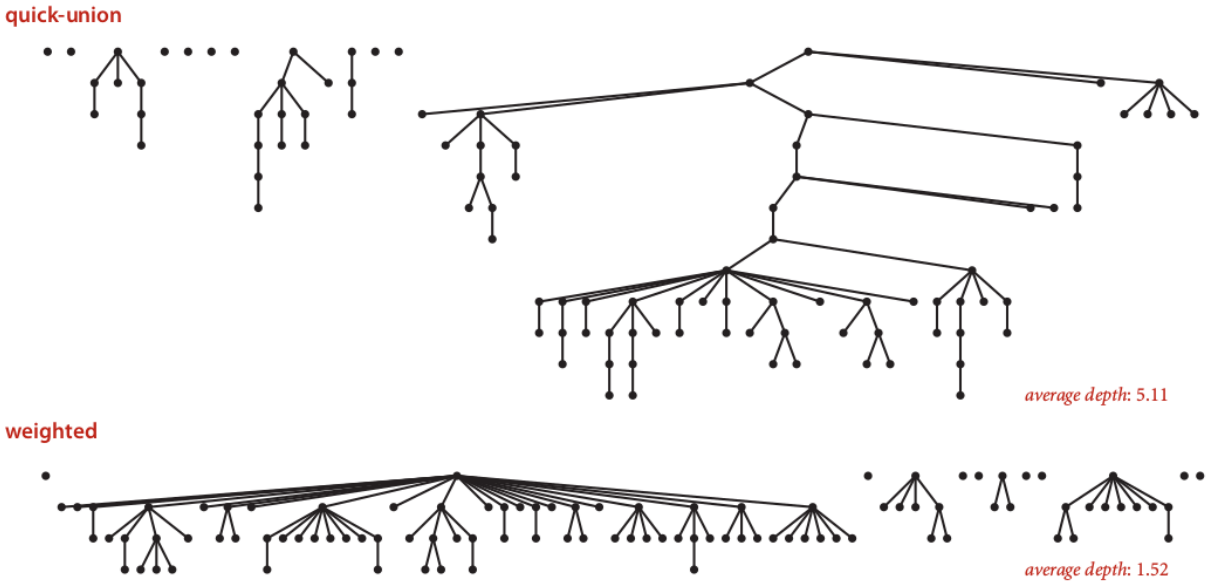
Weighted Quick Union

Quick-Union offer faster union operation, but find operation has strong correlation with tree height. This means, we need to avoiding build very height tree representation. To do that, we just need very small tweak of Quick-Union algorithm. Rather than just connection tree in the left to another tree in the right, we need to evaluate better connectivity by always connecting smaller tree into bigger tree. Take a look for table below:

p	q		0	1	2	3	4	5	6	7	8	9
5	9		1	1	1	8	8	0	1	1	8	8
Find			-	1				-			8	-
Union											1	

Right tree of root 8 smaller that left tree of root 1, then we need to decided 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.