Algorithms and Data Structures

COMP261
Disjoint Set

Yi Mei

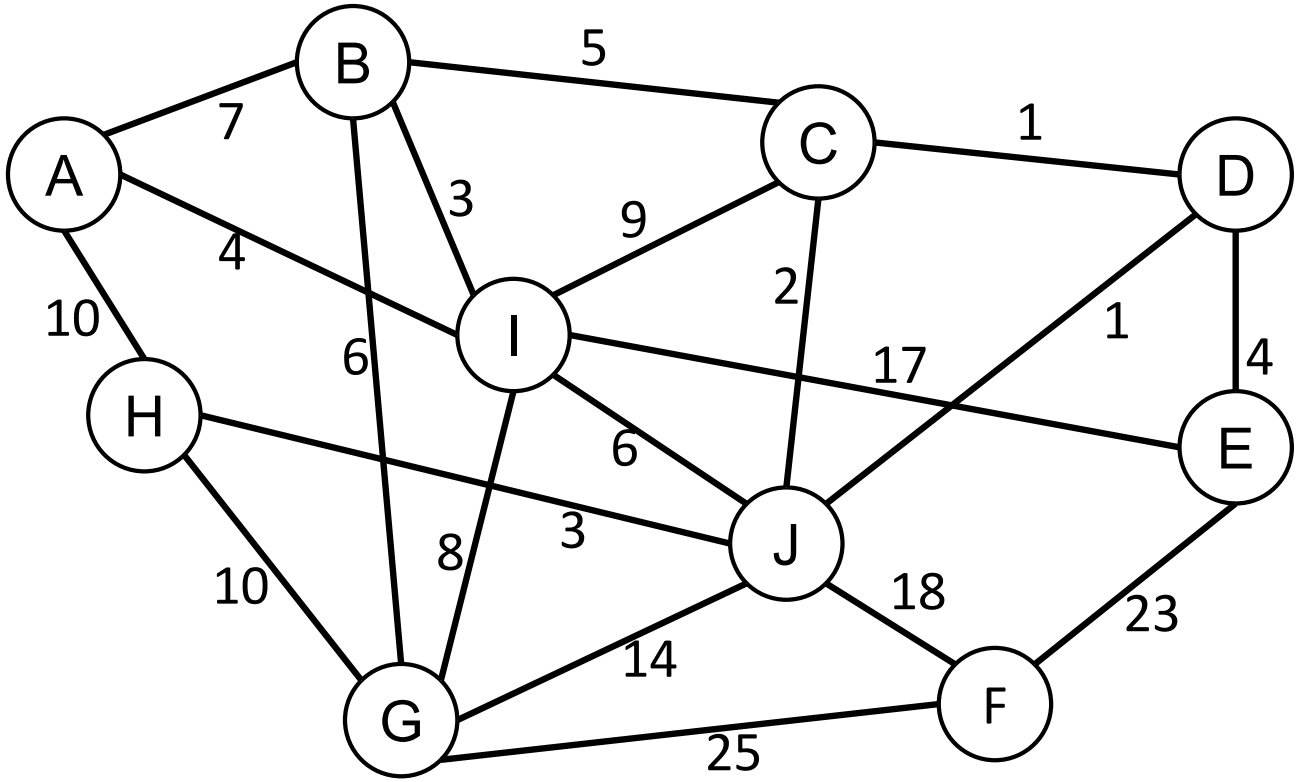
yi.mei@ecs.vuw.ac.nz

Outline

- Why Disjoint Set
- Disjoint Set
- Operations
  - Find
  - Union

Kruskal's Algorithm

- Merge trees
  - Initially, each node is a single-node tree
  - At each step, merge two trees into one
  - The merge cost is the minimum (min-cost edge)



Kruskal's Algorithm Revisited

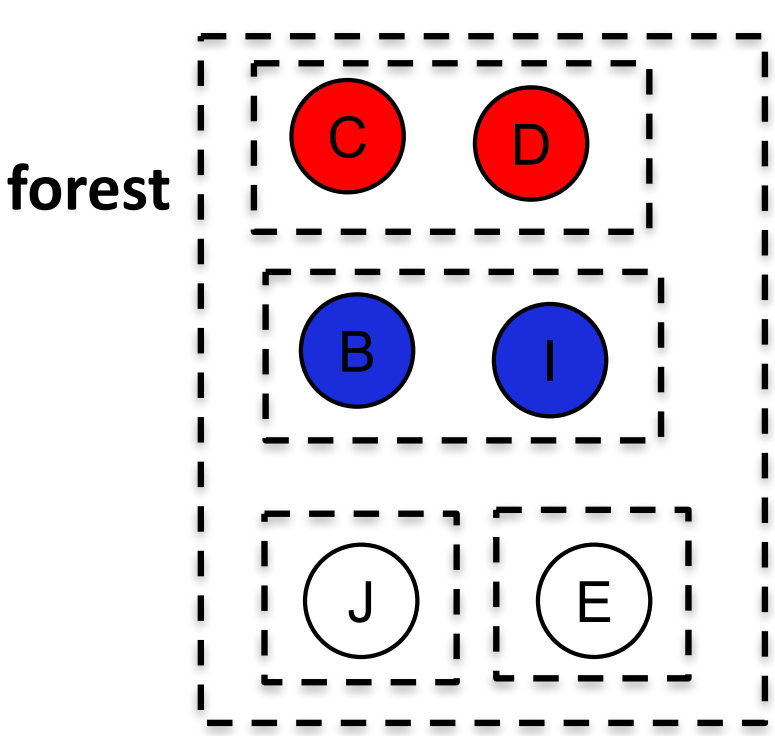
Given: a connected undirected weight graph (*N* nodes, *M* edges)

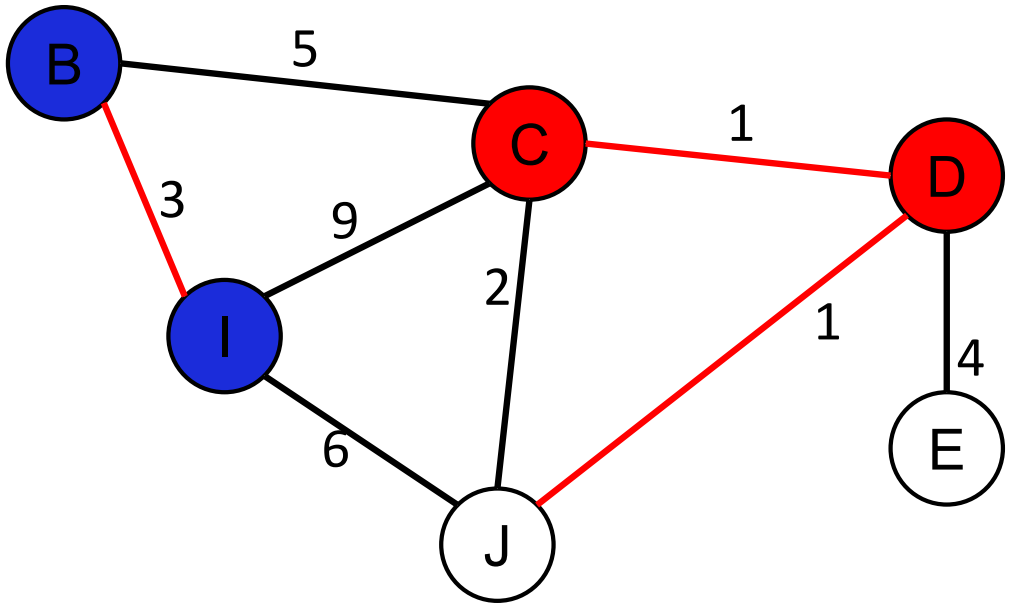
Set *forest* as *N* node sets, each containing a node;
Set *fringe* as a priority queue of all the edges ⟨n1, n2, length⟩;
Set *tree* as an empty set of edges;
Repeat until *forest* contains only one tree or edges is empty {
  Get and remove *⟨ n1*, n2*, length*⟩* as the edge with minimum length from fringe;
  if (*n1** and *n2** are in different sets in forest) {
    Merge the two sets in *forest*;
    Add the edge to *tree*;
  }
}
return *tree*;

Complexity of Kruskal's Algorithm

- **Find**: Determine whether two elements belong to the same set
- **Union**: merge two sets into one

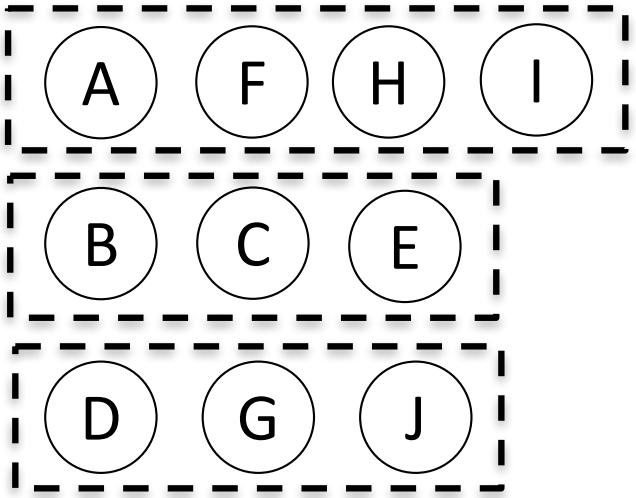
- The cost of Find and Union depends on the data structure of the forest: set of sets





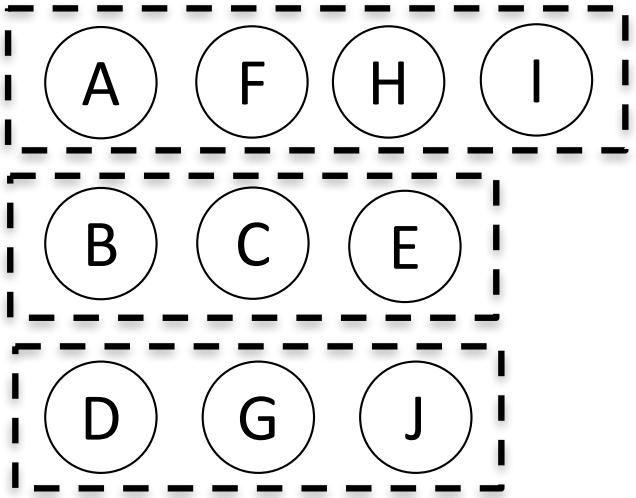
Set of Sets: Data Structures

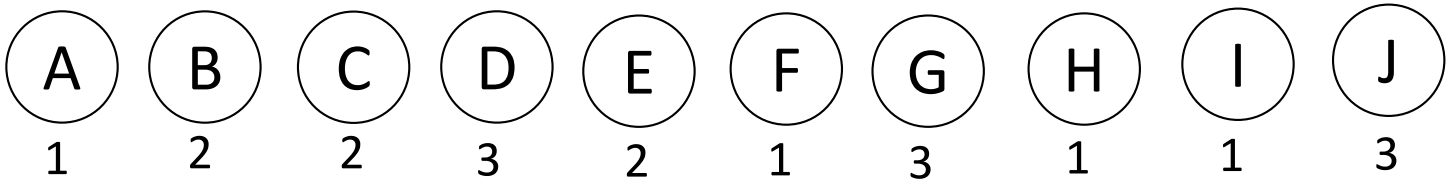
- Option 1: set of sets (e.g. `HashSet<HashSet<Node>>`)
  - Cost of find: iterate over all sets,
  - Cost of union: add all the elements from one set to another,



Set of Sets: Data Structures

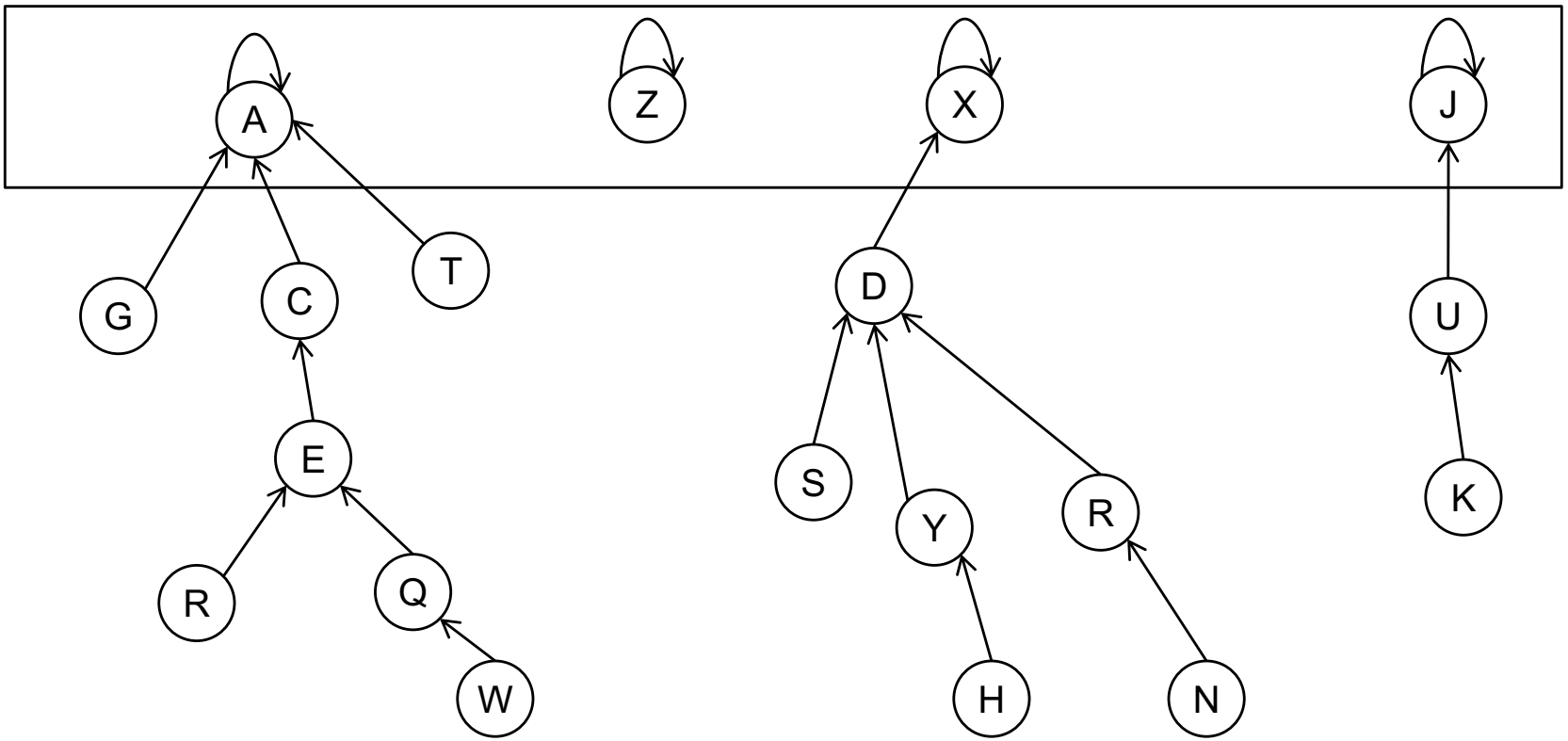
- Option 2: mark each node with set ID
  - Cost of find: check whether the two elements have the same set ID,
  - Cost of union: iterate all the nodes, change the set ID of one set to another





Set of Sets: Data Structures

- Option 3 (the best): disjoint-set (union-find) data structure
  - Set of inverted trees
  - Each set is represented by a linked tree with links pointing **towards** the root
  - **Forest** = set of root nodes



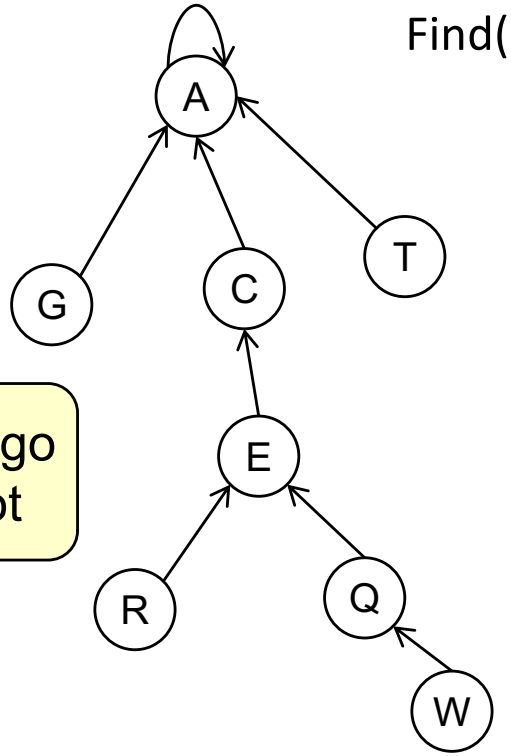
Disjoint Set

```
// make a new set with element x
MakeSet(x) {
    x.parent = x;
    add x to forest;
}

// find the root of the set that x belongs to
Find(x) {
    if (x.parent == x) { // x is the root
        return x;
    } else {
        root = Find(x.parent);
        return root;
    }
}
```

Recursively go up to the root

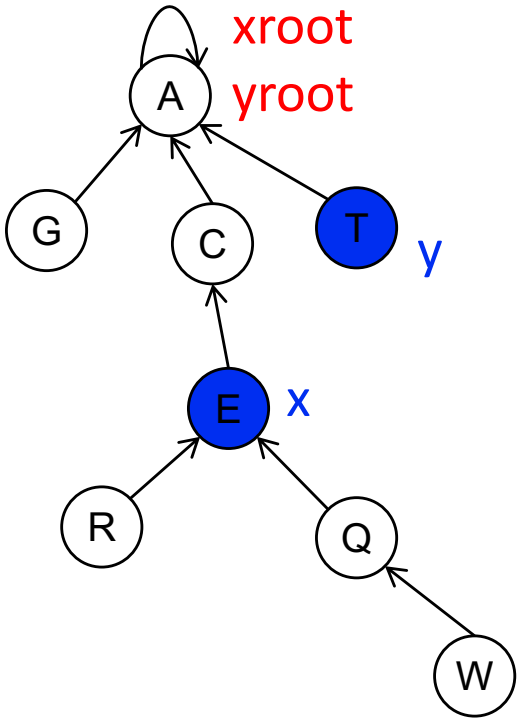
Find(A) = A
Find(G) = A
Find(E) = A
Find(W) = A
…



Disjoint Set

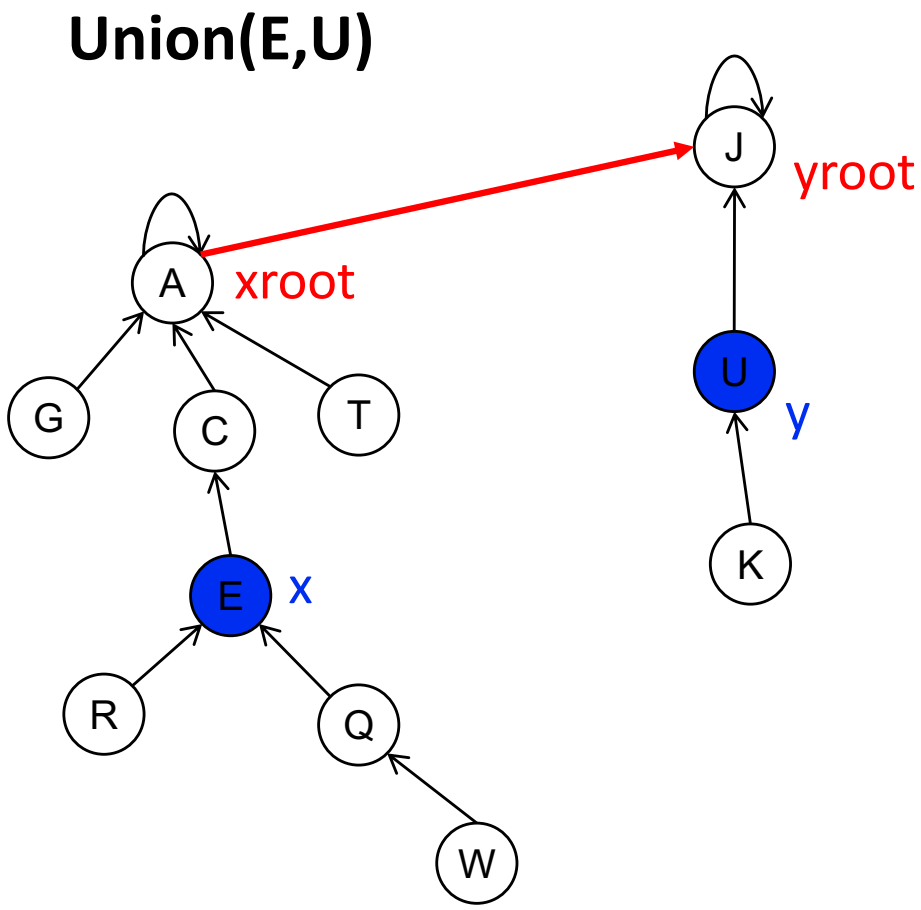
```
// union the sets of x and y
Union(x, y) {
    xroot = Find(x);
    yroot = Find(y);
    if (xroot == yroot) {
        // x and y belong to
        // the same set
        return;
    } else {
        xroot.parent = yroot;
        remove xroot from forest;
}}
```

Union(E,T)



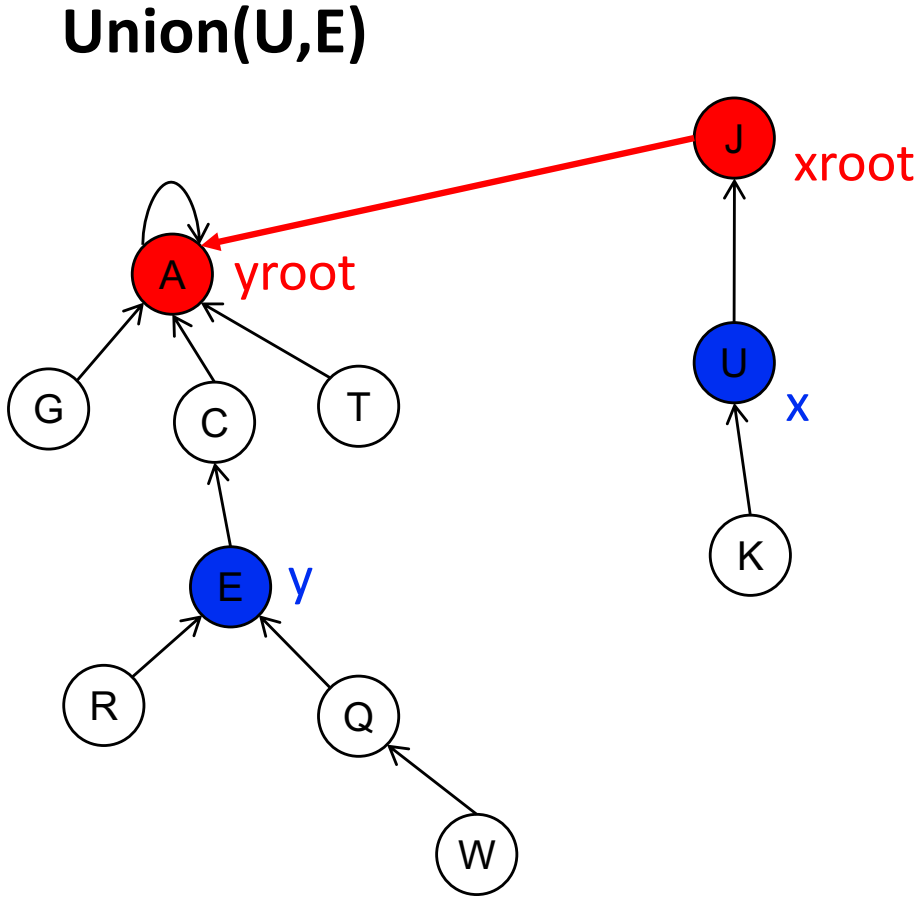
Disjoint Set

```
// union the sets of x and y
Union(x, y) {
    xroot = Find(x);
    yroot = Find(y);
    if (xroot == yroot) {
        // x and y belong to
        // the same set
        return;
    } else {
        xroot.parent = yroot;
        remove xroot from forest;
}
```



Disjoint Set

```
// union the sets of x and y
Union(x, y) {
    xroot = Find(x);
    yroot = Find(y);
    if (xroot == yroot) {
        // x and y belong to
        // the same set
        return;
    } else {
        xroot.parent = yroot;
        remove xroot from forest;
}
```



Order can change the depth of the resultant tree

Disjoint Set

- To reduce complexity, always merge shorter trees into deeper ones

```
MakeSet(x) {
    x.parent = x;
    x.depth = 0;
    add x to forest;
}

Find(x) {
    if (x.parent == x) {
        return x;
    } else {
        root = Find(x.parent);
        return root;
    }
}
```

```
Union(x, y) {
    xroot = Find(x);
    yroot = Find(y);
    if (xroot == yroot) {
        return;
    } else {
        if (xroot.depth < yroot.depth) {
            xroot.parent = yroot;
            remove xroot from forest;
        } else {
            yroot.parent = xroot;
            remove yroot from forest;
            if (xroot.depth == yroot.depth)
                xroot.depth ++;
        }
}
```

Summary

- Disjoint set for representing set of sets
- Very efficient (Almost constant time) for
  - Determine whether two elements belong to the same set
  - Merge two sets into one
  - Kruskal's algorithm for MST