Disjoint Sets

Disjoint sets (also called Merge-Find sets):

Elements 1…n

Initially each element is in a singleton set:
$S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3\}$, …, $S_n = \{n\}$

Two operations:

Find (x) = return the index k of the set S_k that currently contains element x

Merge (a, b) = replace the two sets S_a and S_b by their union, and choose a name (either S_a or S_b) for this union

Example: $n = 7$

$S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3\}$, $S_4 = \{4\}$, $S_5 = \{5\}$, $S_6 = \{6\}$, $S_7 = \{7\}$

Find (1) = 1, Find (2) = 2, etc.

Merge (1, 3), choose name $S_1 \Rightarrow S_1 = \{1, 3\}$
Merge (4, 7), choose name $S_4 \Rightarrow S_4 = \{4, 7\}$
Merge (5, 6), choose name $S_6 \Rightarrow S_6 = \{5, 6\}$

Find (3) = 1, Find (7) = 4, Find (5) = 6

Note: Find (x) == Find (y) if and only if x and y in same set

Merge (1, 4), choose name $S_4 \Rightarrow S_4 = \{1, 3, 4, 7\}$
Merge (2, 6), choose name $S_6 \Rightarrow S_6 = \{2, 5, 6\}$

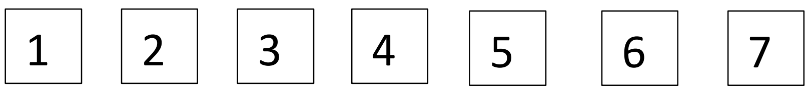
Find (1) = 4, Find (3) = 4, Find (2) = 6

Merge (6, 4), choose name $S_4 \Rightarrow S_4 = \{1, 2, 3, 4, 5, 6, 7\}$

Find (2) = 4, Find (5) = 4, Find (6) = 4

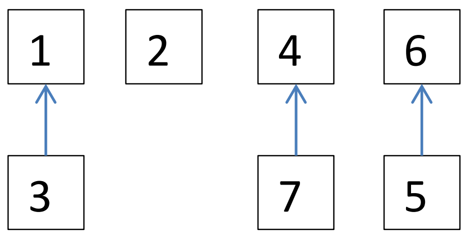
Data structure: Disjoint set forest

- Each set is a tree
- If the root element is k then the set name is S_k
- Represent the forest using a parent array
- Use 0 to represent null



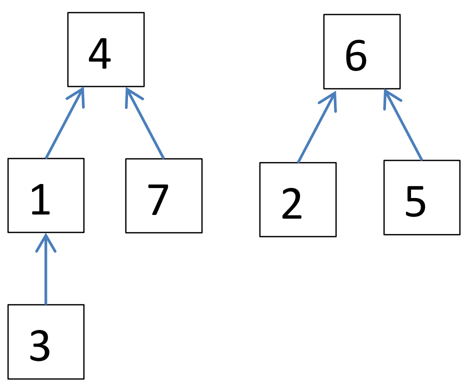
	1	2	3	4	5	6	7
p	0	0	0	0	0	0	0

Merge (1, 3)
Merge (4, 7)
Merge (5, 6)
Find (3) = 1



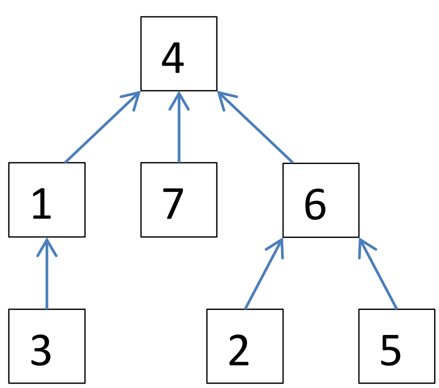
	1	2	3	4	5	6	7
p	0	0	1	0	6	0	4

Merge (1, 4)
Merge (2, 6)
Find (2) = 6
Find (3) = 4



	1	2	3	4	5	6	7
p	4	6	1	0	6	0	4

Merge (6, 4)
Find (2) = 4
Find (5) = 4



	1	2	3	4	5	6	7
p	4	6	1	0	6	4	4

```
class DisjointSets {
	int p[1…n];			// parent
	DisjointSets( ) {
		for (int k=1; k<=n; k++)
			p[k] = 0;
	}
	int Find (int x) {
		while (p[x] != 0)
			x = p[x];
		return x;
	}
	void Merge (int a, int b) {
		if (p[a]!=0 || p[b]!=0)  throw exception;
		p[b] = a;			// alternatively, p[a] = b;
	}
}
```

Worst-case analysis:

- DisjointSets constructor takes $\theta(n)$ time, but it's only called once.
- Merge (a,b) takes $\theta(1)$ time.
- Find (x) takes $\theta(h)$ time if tree has height h, so it can take $\theta(n)$ time if we build a tree of height $\theta(n)$.

How to make this more efficient?

Union-by-height heuristic:

The shorter tree becomes a subtree of the taller tree

```
class DisjointSets {
    int p[1…n];                 // parent
    int h[1…n];                 // height of tree
    DisjointSets( ) {
        for (int k=1; k<=n; k++)
            { p[k] = 0;  h[k] = 0; }
    }
    int Find (int x) {
        while (p[x] != 0)
            x = p[x];
        return x;
    }
    void Merge (int a, int b) {
        if (p[a]!=0 || p[b]!=0)  throw exception;
        if (h[a] < h[b])  swap (a, b);       // union-by-height
        p[b] = a;
        if (h[a] == h[b])  h[a] ++;
    }
}
```

With union-by-height, the height of every tree is ≤ lg n,
so Find(x) takes θ(lg n) time in worst-case.
Also, Merge (a,b) still takes θ(1) time.

Union-by-size heuristic:

The smaller tree becomes a subtree of the larger tree

```
class DisjointSets {
    int p[1…n];                 // parent
    int s[1…n];                 // size of tree
    DisjointSets( ) {
        for (int k=1; k<=n; k++)
            { p[k] = 0;  s[k] = 1; }
    }
    int Find (int x) {
        while (p[x] != 0)
            x = p[x];
        return x;
    }
    void Merge (int a, int b) {
        if (p[A]!=0 || p[B]!=0)  throw exception;
        if (s[a] < s[b])  swap (a, b);        // union-by-size
        p[b] = a;
        s[a] += s[b];
    }
}
```

With union-by-size, the height of every tree is $\leq \lg n$,
so Find(x) takes $\theta(\lg n)$ time in worst-case.
Also, Merge (a,b) still takes $\theta(1)$ time.

Efficient implementation and analysis of Kruskal MST algorithm:

First sort edge weights using heap sort or merge sort:

θ(m lg m) time

But $n-1 \le m < n^2$ because graph is connected, undirected

$\lg(n-1) \le \lg m < \lg n^2 = 2 \lg n$, so lg m is θ(lg n)

Therefore $\theta(m \lg m) = \theta(m \lg n)$ time

Next use disjoint sets to represent the components (trees):

```
for each edge (x,y) in this order {
    a = Find (x);  b = Find (y);
    if (a != b) {
        Merge (a, b);
        add edge (x,y);
    }
}
```

Each Find operation takes O(lg n) time

Each Merge operation takes O(1) time

2m Find operations $\Rightarrow$ O(m lg n) time

n–1 Merge operations $\Rightarrow \theta(n)$ time $\Rightarrow$ O(m) time

Total time for disjoint sets is O(m lg n) time

So total time for Kruskal's algorithm is θ(m lg n)