MST and Union Find

Neeyanth Kopparapu

October 2019

1 Introduction - The Problem

Suppose we have a set of trees, something we call a *forest*. [1] We want to know if two nodes are part of the same component.

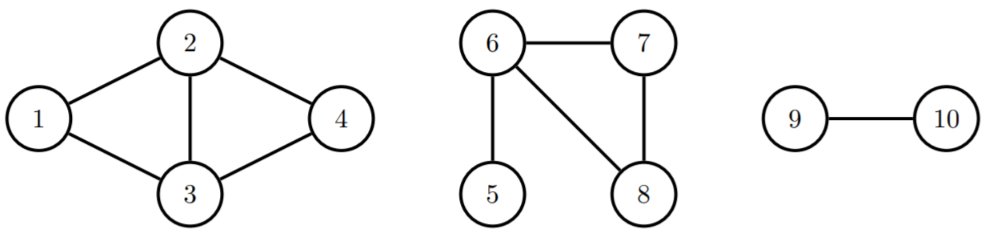


Figure 1: Your Typical Graph

We would do this in the following way:

- *find(node)* would give us the **label** of the component that it belonged to. (This is all we would need if the graph didn't change)
- *union*(n_1, n_1) connect the components that contain n_1 and n_2 and would have the same label.

We are gonna go through some ways to optimize the performance of this and some applications.

[1] If you don't know what a tree is, go to the other lecture.

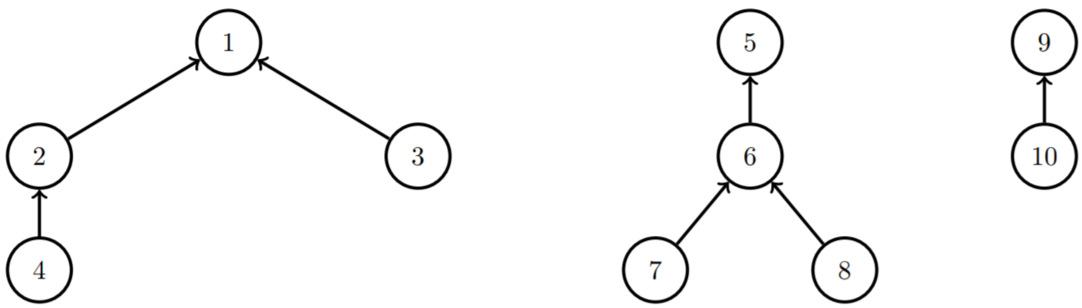
2 Union Find

2.1 Naive Solutions

We would first think of two solutions:

2.1.1 Pointer Representation

In a graph, keep pointers to parents of nodes. (Arbitrarily choose parents)



-1	1	1	2	-1	5	6	6	-1	9
1	2	3	4	5	6	7	8	9	10

Figure 2: Your Typical Graph

We can then get *find* by keep tracing parents until we get to -1, implying we hit the parent node of the component. **Question:** What is *find(8)*? Which nodes did you have to trace to get there?

Doing the union is also really easy. If we had to do *union(3,6)*, all we essentially have to do is set the *find* of the root of 3 to the *find* of 6. **Why does this work?**

However, we run into a problem - *find* starts to grow linearly. This is a problem in our runtime, we are going for approximately **constant** complexity.

2.1.2 List Representation

In a typical graph, we just store (in an array) the label given the vertex. Clearly *find* is linear, but this also makes *union* become linear. **Why?**

2.2 Optimizations

We are going to go back to the pointer representation, because that is what we will optimize. We can make two fixes:

- The first is to always add the shorter tree to the taller tree, as we want to minimize the maximum height. An easy heuristic for the height of the tree is simply the number of elements in that tree. We can keep track of the size of the tree with a second array.
- The second is a bit more tricky. Assign the pointer associated with *node* to be *find(node)* at the end of the find operation. We can design *find(node)* to recursively call *find* on the pointer associated with *node*, so this fix sets pointers associated with nodes along the entire chain from *node* to *find(node)* to be *find(node)*.

It turns out that these two optimizations turn the run-time of both operations to $O(\alpha(V))$, where $\alpha(V)$ is the inverse Ackerman function, and for all intensive purposes $\alpha(V) \leq 5$, so this makes is approximately constant.

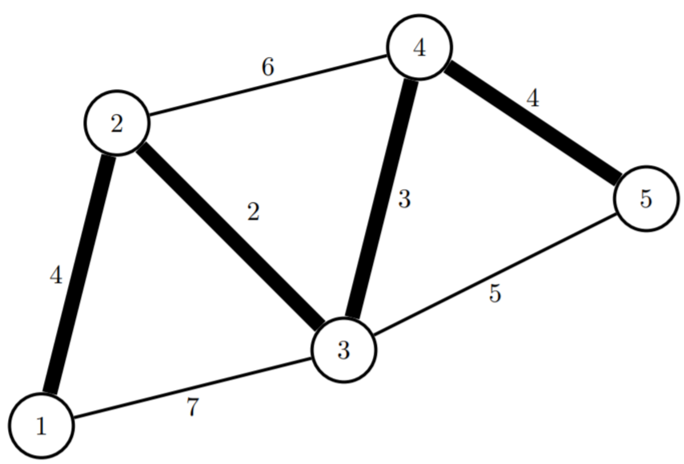
Algorithm 1 Union-Find

```
function Find(v)
    if v is the root then
        return v
    parent(v) ← Find(parent(v))
    return parent(v)
function Union(u, v)
    uRoot ← Find(u)
    vRoot ← Find(v)
    if uRoot = vRoot then
        return
    if size(uRoot) < size(vRoot) then
        parent(uRoot) ← vRoot
        size(vRoot) ← size(uRoot) + size(vRoot)
    else
        parent(vRoot) ← uRoot
        size(uRoot) ← size(uRoot) + size(vRoot)
```

With that, we can start with its applications!

3 Minimum Spanning Tree

Consider a connected, undirected graph. A *spanning tree* is a subgraph that is a tree and contains every vertex in the original graph. A *minimum spanning tree* is a spanning tree such that the sum of the edge weights of the tree is minimized. Finding the minimum spanning tree uses many of the same ideas discussed earlier.



I won't discuss **Prim's Algorithm** but it is a way to do this. Instead, we are going to talk about **Kruskal's Algorithm**, essentially an HC of Union-Find in this application.

First, we need to sort the edges. Kruskal's algorithm greedy selectes edges that would contribute to the MST until all the nodes are accounted for. Union-Find is the perfect way to make sure all the nodes are accounted for!!

Algorithm 6 Kruskal

```
for all edges (u, v) in sorted order do
    if FIND(u) ≠ FIND(v) then
        add (u, v) to spanning tree
        UNION(u, v)
```

The complexity is thus $O(E \log E)$. Note that most Union-Find problems have the union-find as an intermediate step - mostly in flood fill or MST or something else. Like always, algorithms aren't meant to be memorized! Most problems will not be solved if you know the algorithm, but can be easily solved if you understand whats going on in the background.

4 Problems

- USACO 2014 March Contest, Silver Problem 1. Watering the Fields
- USACO 2014 January Contest, Gold Problem 3. Ski Course Rating
- USACO 2016 December Contest, Gold Problem 1. Moocast
- **USACO 2011 December Contest, Gold Division Problem 2. Simplifying the Farm**