Union Find

George Tang

February 2018

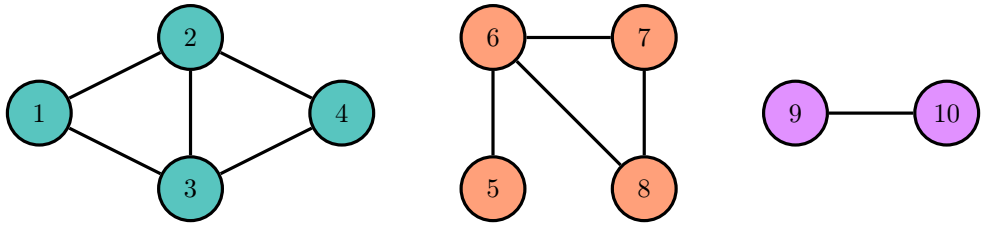
1 Introduction

Given a set nodes, where some are connected (forming a group) and some are not, we want to accomplish:

- *find(u)*: return the group of node u
- *union(u, v)*: merge the groups of u and v

2 Components

In a group of nodes, every node is reachable from every other node via edges in the group. More formally, this is known as a component. The figure below shows three components.

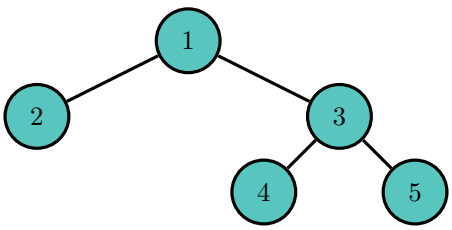


Credit: Samuel Hsiang

Previously, we have discussed methods such as DFS/Flood Fill to find connected components. Consider the flood fill implementation: we perform a O(N) complete search to determine if two nodes are in the same component. Merging two components also takes O(N), since we must manually change the component value of each node.

3 Quick Find

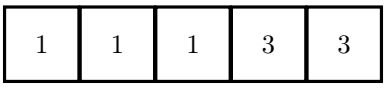
Imagine that nodes in the same components are arranged in a tree such that the root represents the component. Each node will have one pointer, pointing to the root of the component, and the root node will point to itself. If root[u]==root[v], u and v will be in the same component. To merge components, we can have to change the root of the nodes in the other component. Thus, the complexity of find will be O(1) and union O(N).



For instance, in this figure of a component tree, node 1 is the root and represents the component.

4 Quick Union

Instead of updating the root of the component every time we perform a union, we can just keep track of the parent of each node. Then, the pointer array will be as follows, where index 0 stores the parent of node 1 and index n, of node n+1:



4.1 Union by Rank

When combining many component trees, we may end up with a large tree. For optimal performance, we must limit the depth of the tree. We can do this by keeping another array that keeps track of the size of each component tree. Every time we merge two component trees, we can attach the the smaller tree to the larger tree.

4.2 Path Compression

Because now find needs to traverse the tree to find the root, the worst case scenario is O(logN). We can avoid tracing multiple parent pointers if every time we trace a pointer, we set the parent of the node to the root.

5 Implementation

Sample implementation of weighted quick-union with path compression. *Credit: Samuel Hsiang*

Algorithm 1 Sample Union Find by Samuel Hsiang

```
1:  function FIND(u)
2:      if parent[u]==u then
3:          return u
4:      parent[u] = find(parent[u])
5:      return parent[u]
6:  function UNION(u, v)
7:      uRoot = FIND(u)
8:      vRoot = FIND(v)
9:      if uRoot == vRoot then
10:         return
11:     if size[u]<size[v] then
12:         parent[uRoot] = vRoot
13:         size[vRoot] = size[uRoot] + size[vRoot]
14:     else
15:         parent[vRoot] = uRoot
16:         size[uRoot] = size[uRoot] + size[vRoot]
```