Overview of Disjoint Set

Report Issue

Given the vertices and edges between them, how could we quickly check whether two vertices are connected? For example, Figure 5 shows the edges between vertices, so how can we efficiently check if 0 is connected to 3, 1 is connected to 5, or 7 is connected to 8? We can do so by using the "disjoint set" data structure, also known as the "union-find" data structure. Note that others might refer to it as an algorithm. In this Explore Card, the term "disjoint set" refers to a data structure.

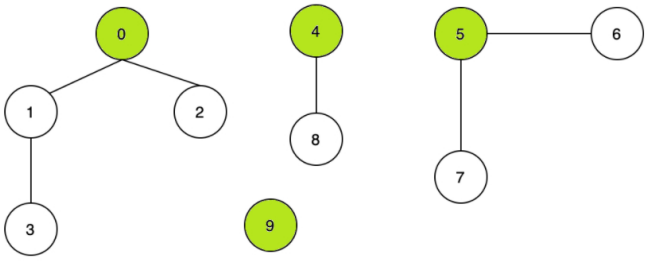

Figure 5. Each graph consists of vertices and edges. The root vertices are in green

The primary use of disjoint sets is to address the connectivity between the components of a network. The "network" here can be a computer network or a social network. For instance, we can use a disjoint set to determine if two people share a common ancestor.

Terminologies

- Parent node: the direct parent node of a vertex. For example, in Figure 5, the parent node of vertex 3 is 1, the parent node of vertex 2 is 0, and the parent node of vertex 9 is 9.
- Root node: a node without a parent node; it can be viewed as the parent node of itself. For example, in Figure 5, the root node of vertices 3 and 2 is 0. As for 0, it is its own root node and parent node. Likewise, the root node and parent node of vertex 9 is 9 itself. Sometimes the root node is referred to as the head node.

Introduction to Disjoint Sets

Summary of video content:

1. How do "disjoint sets" work.
2. Solving the connectivity question in Figure 5.

Implementing “disjoint sets”

Summary of video content:

1. How to implement a “disjoint set”.
2. The `find` function of a disjoint set.
3. The `union` function of a disjoint set.

The two important functions of a “disjoint set.”

In the introduction videos above, we discussed the two important functions in a “disjoint set”.

- **The `find` function** finds the root node of a given vertex. For example, in Figure 5, the output of the find function for vertex 3 is 0.
- **The `union` function** unions two vertices and makes their root nodes the same. In Figure 5, if we union vertex 4 and vertex 5, their root node will become the same, which means the union function will modify the root node of vertex 4 or vertex 5 to the same root node.

There are two ways to implement a “disjoint set”.

- Implementation with Quick Find: in this case, the time complexity of the `find` function will be $O(1)$. However, the `union` function will take more time with the time complexity of $O(N)$.
- Implementation with Quick Union: compared with the Quick Find implementation, the time complexity of the `union` function is better. Meanwhile, the `find` function will take more time in this case.

Next, we will learn these two implementations and two common strategies to optimize a disjoint set.