What is union-find?

Union-Find is a data structure with two key operations, as its name suggests: union and find

- `union(x, y)` — Merge two elements into the same group. For example, in the island problem on LeetCode, we can merge adjacent pieces of land into one island.
- `find(x)` — Quickly find which group (or "leader") a particular element belongs to. It tells you the "root" or representative of the group.

Two classic optimizations:

- **Path Compression**: When searching for a representative of an element, all points on the search path are directly linked to the root node, making subsequent searches faster.
- **Union by Rank (or Size)**: When merging two trees, always attach the shorter tree under the taller one. This keeps the trees shallow, improving performance.

With both optimizations, each operation becomes nearly constant time **O(α(n))**, where α is the inverse Ackermann function.

Starting from the “simplest idea”

Method 1: use a dictionary to map group IDs to their members

```
group_dict = {
    1: [0, 3, 5],
    2: [1, 4],
    3: [2]
}
```

scan the whole dictionary:

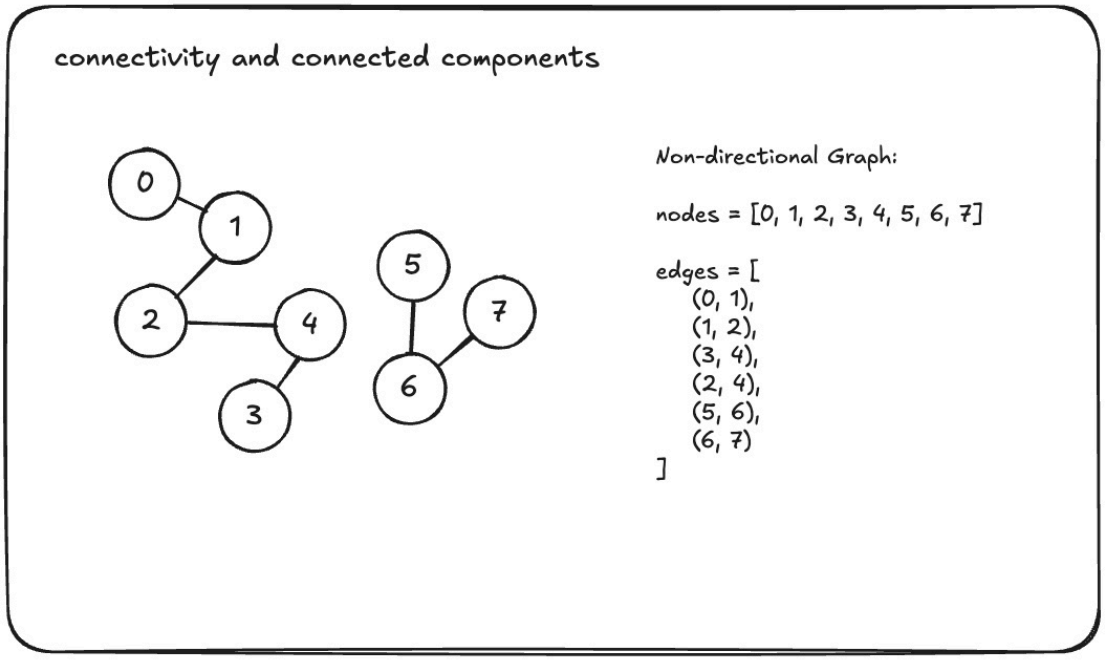
```
for group_id, members in group_dict.items():
    if 3 in members:
        print(group_id)
```

Method 2:

```
belong = [0, 0, 1, 0, 2, 1]
```

Understand Process

Connectivity & Connected Component



We want to know how to use code to:
1. Group these nodes → find your biggest boss

2. Check if node a and node b are connected?
   How many connected components are in the graph? → if biggest boss is the same, these 2 persons are belong to the same gang

What is Connectivity?

In an **undirected graph**, two nodes are **connected** if there's a path between them — even if the path goes through multiple jumps. If nodes can reach each other, we say they belong to the same **connected region**.

What is a Connected Component?

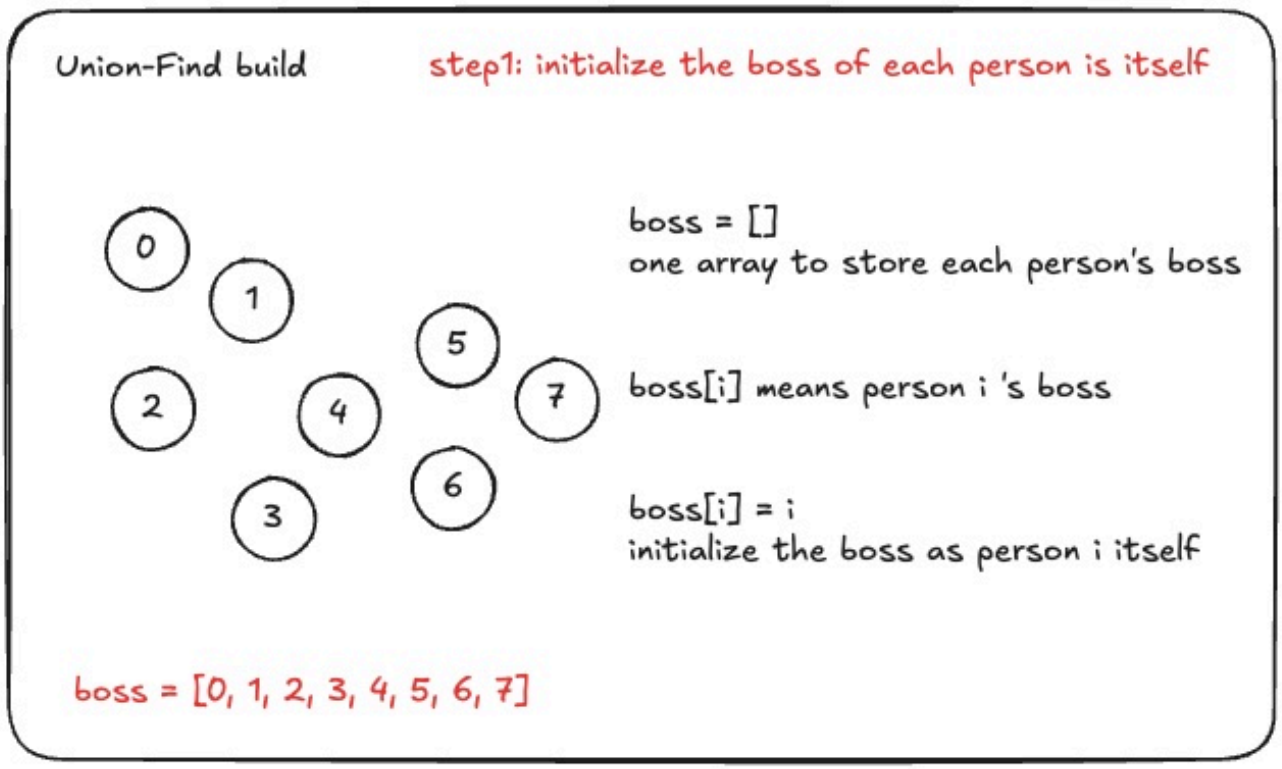
A **connected component** is a group of nodes that are all reachable from one another. A graph may have several components — like islands, isolated groups, or subnetworks.

In real problems, we often care about two main things:

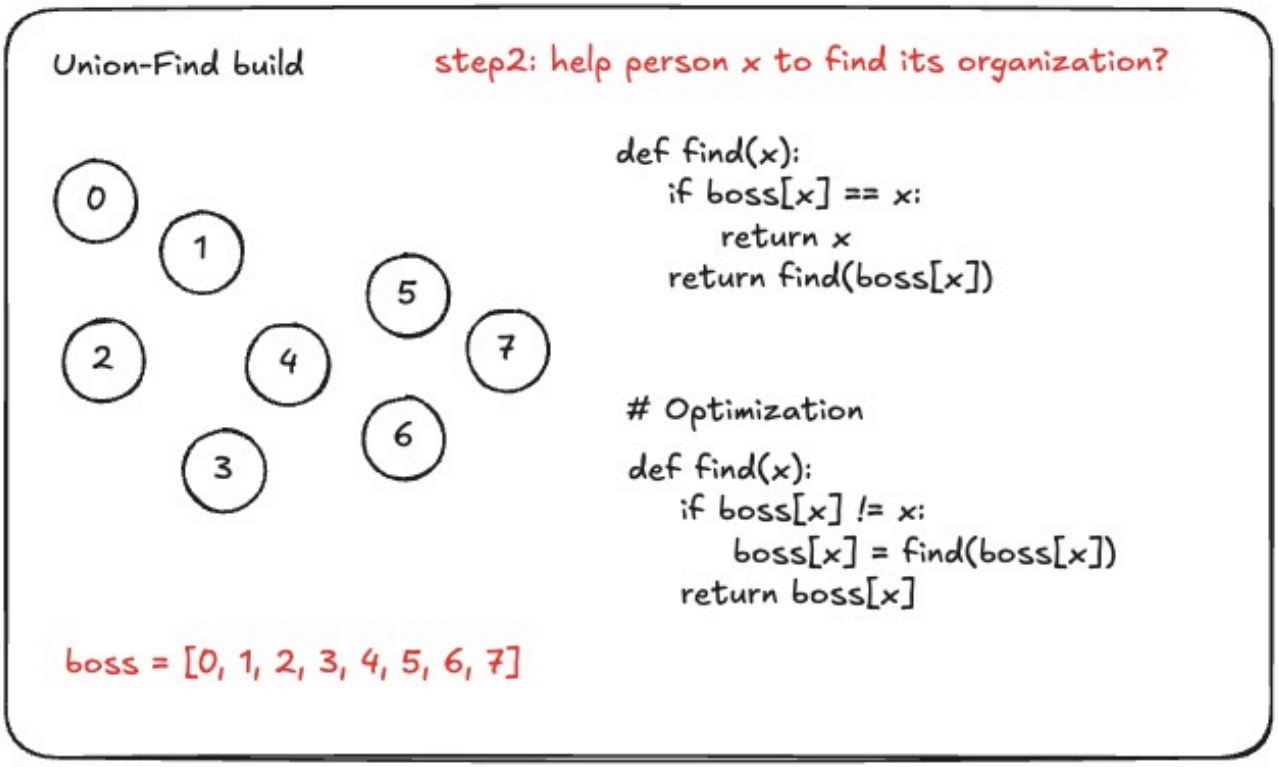
1. **How to group connected nodes together**
2. **How to check if two nodes belong to the same group (component)**

Explain Codes

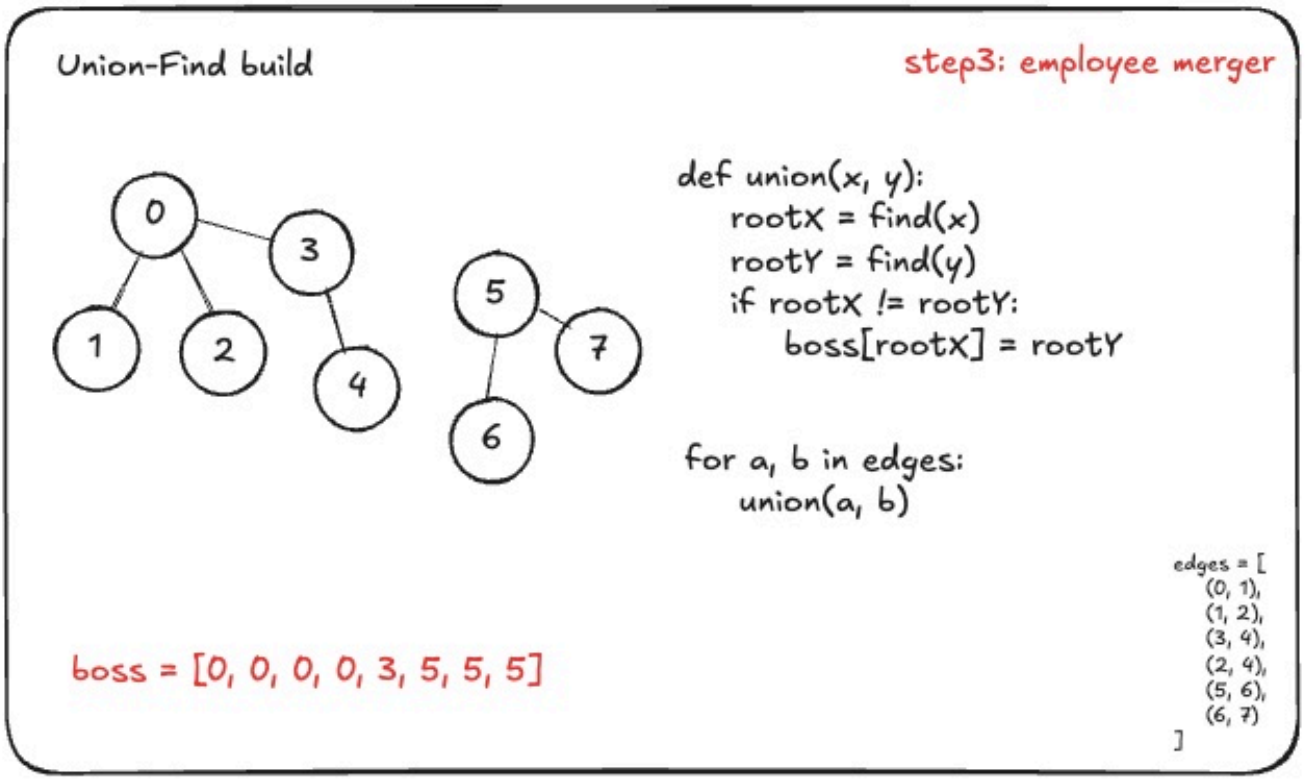
Step 1: Initialization



Step 2: The Find Operation

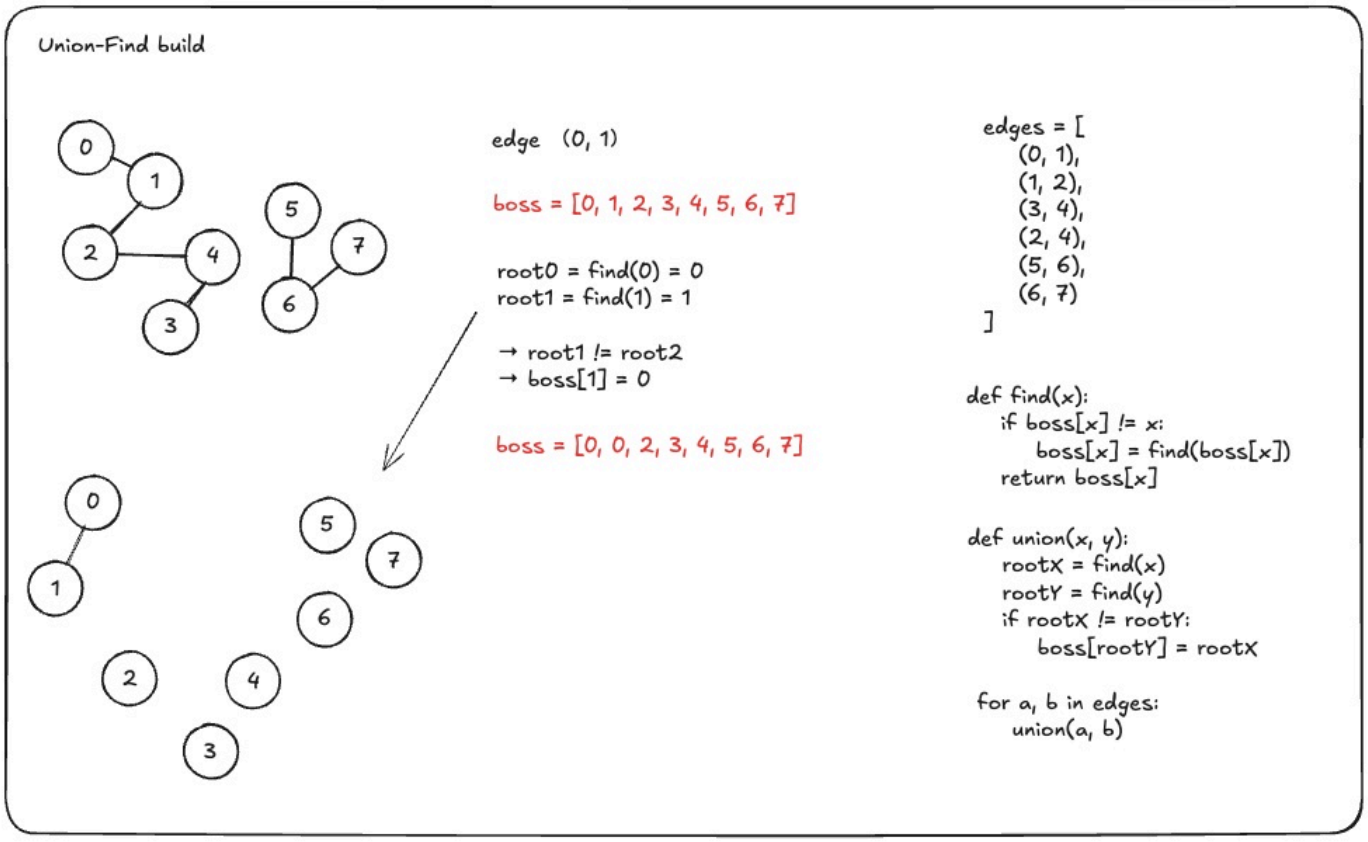


Step 3: The Union Operation



How to build Union-Find?

Let's walk through a concrete example.



Step 1: No Connections Yet

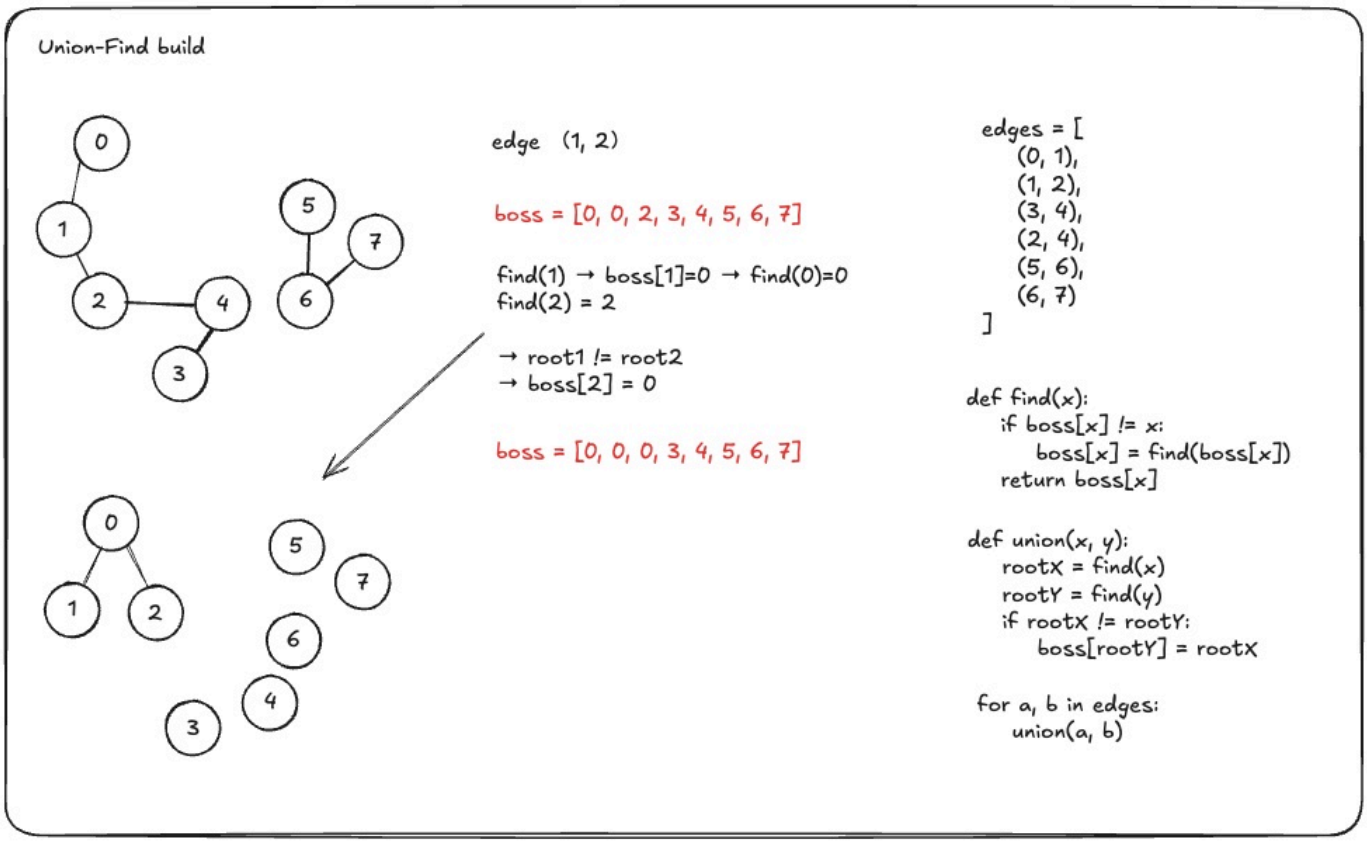
Initially, all nodes are their own bosses: Each node is its own little company. `boss[i] = i`。

Step 2: Connect (0, 1)

`find(0) = 0`, `find(1) = 1` → different roots

Merge: make 1 report to 0 → `boss[1] = 0`

We got a new array.

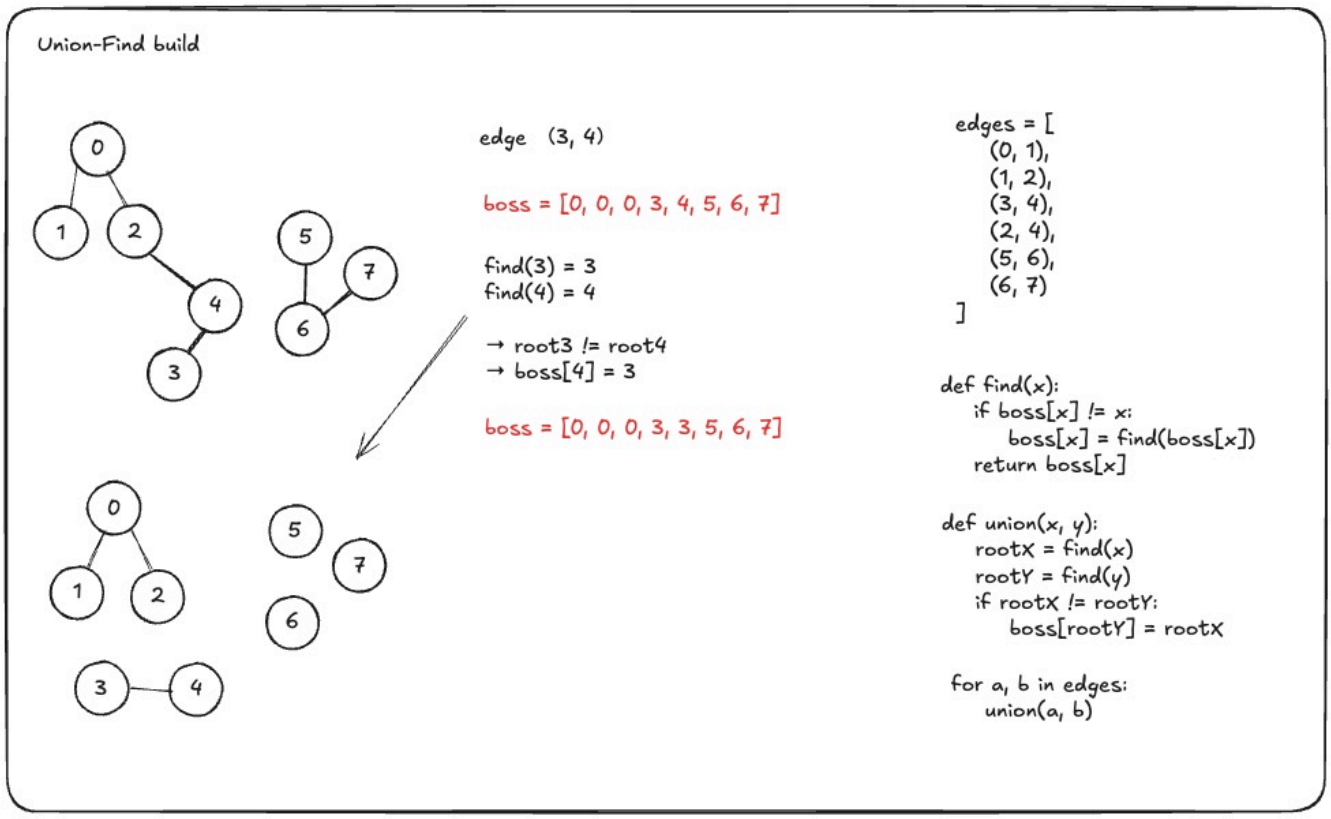


Step 3: Connect (1, 2)

- `find(1) → boss[1] = 0`, and `find(0) = 0` → root is 0
- `find(2) = 2`

Merge: `boss[2] = 0`

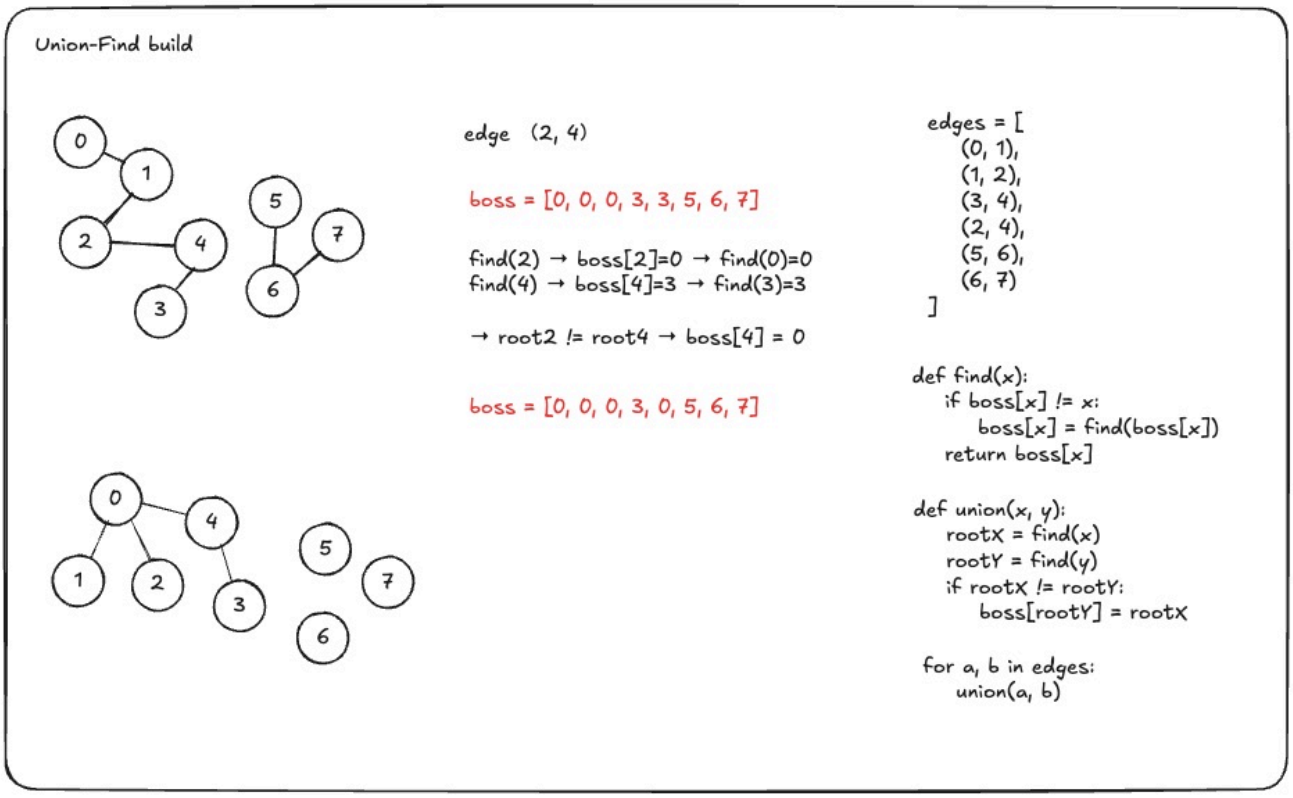
Now we've merged 0, 1, and 2 into one group: `boss = [0, 0, 0, 3, 4, 5, 6, 7]`.



Step 4: Connect (3, 4)

- `find(3) = 3`
- `find(4) = 4`

Merge: `boss[4] = 3`

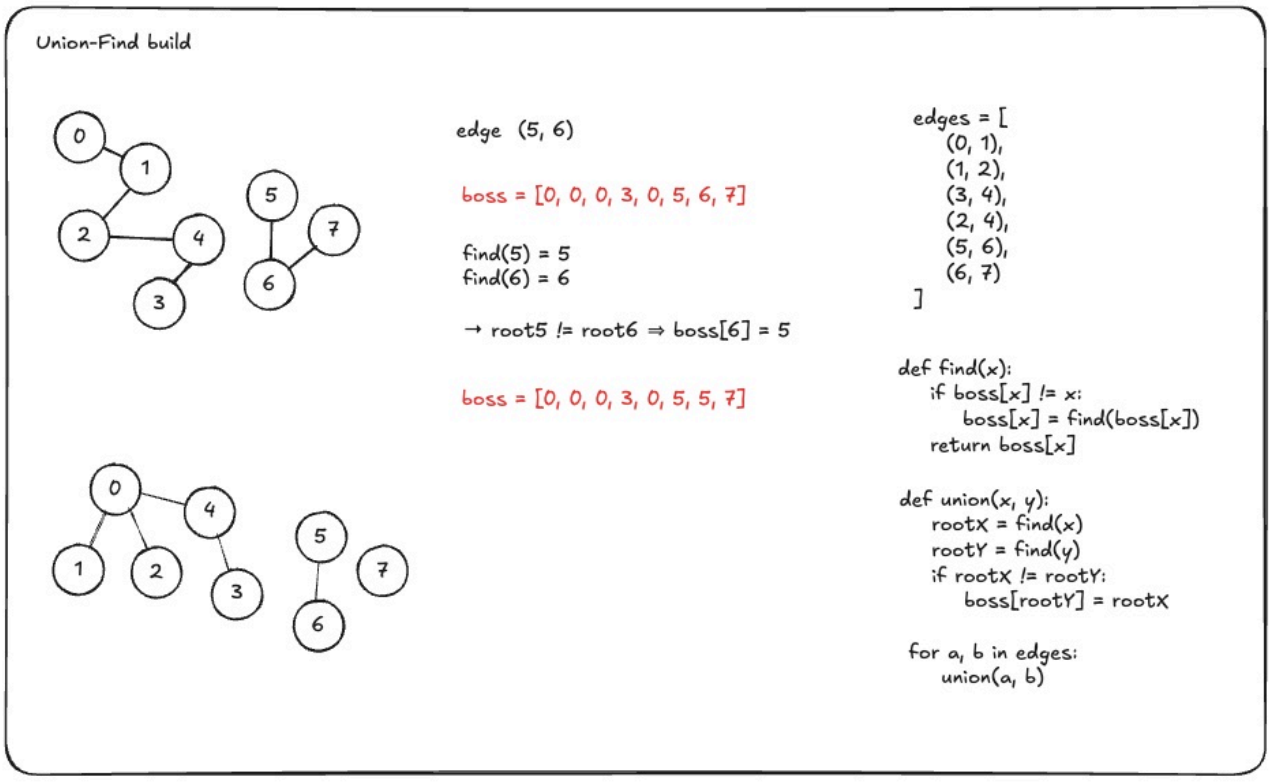


Step 5: Connect (2, 4)

- `find(2)` → 0
- `find(4) → boss[4]=3 → find(3) = 3 → root is 3`

Merge: `boss[3] = 0` → now 3 and 4 are under 0 too

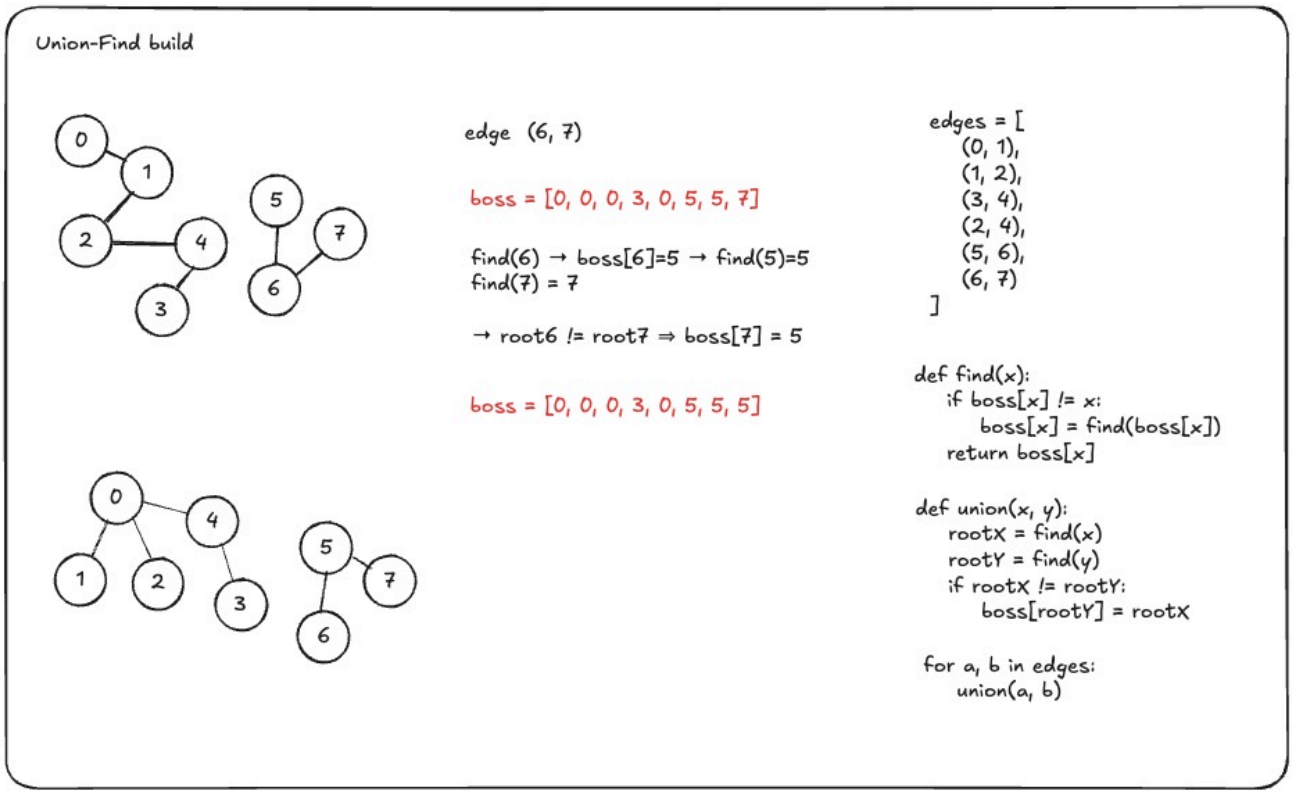
This subgraph already has 5 members connected together.



Step 6: Connect (5, 6)

- `find(5) = 5`
- `find(6) = 6`

Merge: `boss[6] = 5` → now 5 and 6 are under 5



Step 7: Connect (6, 7)

- `find(6)` → boss[6]=5 → the root 5
- `find(7) = 7`

Merge → `boss[7] = 5` → now 5, 6 and 7 are in the same group

Final structure: `boss = [0, 0, 0, 0, 3, 5, 5, 5]`

We now have two major connected components:

- Group represented by 0: includes nodes 0, 1, 2, 3, 4
- Group represented by 5: includes nodes 5, 6, 7

When to use it?

- Does the problem talk about **groups**, **regions**, or **connectivity**?
- Are we checking whether two things are in the same group?
- Are we merging groups or building connections dynamically?
- Are there a lot of queries like “are these connected?”
- Do the keywords include: **undirected edges**, **components**, **connectivity**?