COMP S265F Design and Analysis of Algorithms
Lab 10: Union-Find Disjoint Sets

In this lab, we will apply the union-find disjoint sets to solve a LeetCode problem "547. Number of Provinces": https://leetcode.com/problems/number-of-provinces/

1. The problem

There are n cities. Some of them are connected, while some are not. If city a is connected directly with city b, and city b is connected directly with city c, then city a is connected indirectly with city c.

A province is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an $n \times n$ matrix `isConnected` where `isConnected[`i`][`j`] = 1` if the ith city and the jth city are directly connected, and `isConnected[`i`][`j`] = 0` otherwise.

Return the total number of provinces.

```
class Solution:
    def findCircleNum(self, isConnected: List[List[int]]) -> int:
```

The problem has given the following examples and constraints:

- **Example 1.**
  Input: `isConnected = [[1,1,0],[1,1,0],[0,0,1]]`
  Output: `2`
- **Example 2.**
  Input: `isConnected = [[1,0,0],[0,1,0],[0,0,1]]`
  Output: `3`

Constraints:

- `1 <= n <= 200`
- `n == isConnected.length`
- `n == isConnected[`i`].length`
- `isConnected[`i`][`j`]` is 1 or 0.
- `isConnected[`i`][`i`] == 1`
- `isConnected[`i`][`j`] == isConnected[`j`][`i`]`

2. Problem formulation

The input `isConnected` is actually an adjacency matrix. Therefore, we can initialize the data structures for the union-find disjoint sets (`T`, `next`, `size`), and the number of provinces `numP`, as follows:

```
def findCircleNum(self, isConnected: List[List[int]]) -> int:
    n = len(isConnected)
    self.T = list(range(n))
    self.next = list(range(n))
    self.size = [1 for i in range(n)]
    self.numP = n  # no. of components

    # to be completed by you
```

Then, we can implement the union-find disjoint sets, as follows:

```
def findSet(self, x):
    return self.T[x]

```

```
# update the name of the set containing b
def union(self, a, b):
    i = b
    while True:
        self.T[i] = self.T[a]
        i = self.next[i]
        if i == b:
            break
    self.next[a], self.next[b] = self.next[b], self.next[a]

def unionBySize(self, a, h):
    if self.size[a] > self.size[h]:
        self.union(a, h)
        self.size[a] += self.size[h]
    else:
        self.union(h, a)
        self.size[h] += self.size[a]
```

Your task. Complete the above method `findCircleNum`. You are advised to type this method and the three methods `findSet`, `union`, and `unionBySize` to strengthen your understanding.