2092. Find All People With Secret

Hard | Topics | Companies | Hint

You are given an integer `n` indicating there are `n` people numbered from `0` to `n - 1`. You are also given a **0-indexed** 2D integer array `meetings` where `meetings[i] = [$x_i$, $y_i$, $time_i$]` indicates that person `$x_i$` and person `$y_i$` have a meeting at `$time_i$`. A person may attend **multiple meetings** at the same time. Finally, you are given an integer `firstPerson`.

Person `0` has a **secret** and initially shares the secret with a person `firstPerson` at time `0`. This secret is then shared every time a meeting takes place with a person that has the secret. More formally, for every meeting, if a person `$x_i$` has the secret at `$time_i$`, then they will share the secret with person `$y_i$`, and vice versa.

The secrets are shared **instantaneously**. That is, a person may receive the secret and share it with people in other meetings within the same time frame.

Return *a list of all the people that have the secret after all the meetings have taken place.* You may return the answer in **any order**.

Example 1:

```
Input: n = 6, meetings = [[1,2,5],[2,3,8],[1,5,10]], firstPerson = 1
Output: [0,1,2,3,5]
Explanation:
At time 0, person 0 shares the secret with person 1.
At time 5, person 1 shares the secret with person 2.
At time 8, person 2 shares the secret with person 3.
At time 10, person 1 shares the secret with person 5.
Thus, people 0, 1, 2, 3, and 5 know the secret after all the
meetings.
```

Example 2:

```
Input: n = 4, meetings = [[3,1,3],[1,2,2],[0,3,3]], firstPerson = 3
Output: [0,1,3]
Explanation:
At time 0, person 0 shares the secret with person 3.
At time 2, neither person 1 nor person 2 know the secret.
At time 3, person 3 shares the secret with person 0 and person 1.
Thus, people 0, 1, and 3 know the secret after all the meetings.
```

Example 3:

```
Input: n = 5, meetings = [[3,4,2],[1,2,1],[2,3,1]], firstPerson = 1
Output: [0,1,2,3,4]
Explanation:
At time 0, person 0 shares the secret with person 1.
At time 1, person 1 shares the secret with person 2, and person 2
shares the secret with person 3.
Note that person 2 can share the secret at the same time as
receiving it.
At time 2, person 3 shares the secret with person 4.
Thus, people 0, 1, 2, 3, and 4 know the secret after all the
meetings.
```

Constraints:

- $2 <= n <= 10^5$
- $1 <= meetings.length <= 10^5$
- $meetings[i].length == 3$
- $0 <= x_i, y_i <= n - 1$
- $x_i != y_i$
- $1 <= time_i <= 10^5$
- $1 <= firstPerson <= n - 1$

Python:

```
class Solution:
    def findAllPeople(self, n: int, meetings: List[List[int]], firstPerson: int) -> List[int]:
        groups = [i for i in range(n)]
        groups[firstPerson] = 0

        meetings.sort(key=lambda x: x[2])

        size = len(meetings)
        i = 0
        while i < size:
            current_time = meetings[i][2]
            temp = []
```

```
            while i < size and meetings[i][2] == current_time:
                g1 = self.find(groups, meetings[i][0])
                g2 = self.find(groups, meetings[i][1])
                groups[max(g1, g2)] = min(g1, g2)
                temp.extend([meetings[i][0], meetings[i][1]])
                i += 1
            for j in temp:
                if self.find(groups, j) != 0:
                    groups[j] = j

        result = []
        for j in range(n):
            if self.find(groups, j) == 0:
                result.append(j)

        return result

    def find(self, groups: List[int], index: int) -> int:
        while index != groups[index]:
            index = groups[index]
        return index
```

JavaScript:

```
/**
 * @param {number} n
 * @param {number[][]} meetings
 * @param {number} firstPerson
 * @return {number[]}
 */
var findAllPeople = function(n, meetings, firstPerson) {
    const find = (groups, index) => {
        while (index !== groups[index]) {
            index = groups[index];
        }
        return index;
    };

    let groups = new Array(100000).fill(0).map((_, i) => i);
    let result = [];

    groups[firstPerson] = 0;

    meetings.sort((a, b) => a[2] - b[2]);
```

```
    let i = 0;
    while (i < meetings.length) {
        let currentTime = meetings[i][2];
        let temp = [];
        while (i < meetings.length && meetings[i][2] === currentTime) {
            let g1 = find(groups, meetings[i][0]);
            let g2 = find(groups, meetings[i][1]);
            groups[Math.max(g1, g2)] = Math.min(g1, g2);
            temp.push(meetings[i][0]);
            temp.push(meetings[i][1]);
            ++i;
        }
        for (let j = 0; j < temp.length; ++j) {
            if (find(groups, temp[j]) !== 0) {
                groups[temp[j]] = temp[j];
            }
        }
    }

    for (let j = 0; j < n; ++j) {
        if (find(groups, j) === 0) {
            result.push(j);
        }
    }

    return result;
};
```

Java:

```
class Solution {
    public List<Integer> findAllPeople(int n, int[][] meetings, int firstPerson) {
        int[] groups = new int[100000];
        List<Integer> result = new ArrayList<>();
        List<Integer> temp = new ArrayList<>();

        for (int i = 0; i < n; ++i) groups[i] = i;
        groups[firstPerson] = 0;

        Arrays.sort(meetings, (a, b) -> Integer.compare(a[2], b[2]));

        int i = 0;
        while (i < meetings.length) {
            int currentTime = meetings[i][2];
```

```
            temp.clear();
            while (i < meetings.length && meetings[i][2] == currentTime) {
                int g1 = find(groups, meetings[i][0]);
                int g2 = find(groups, meetings[i][1]);
                groups[Math.max(g1, g2)] = Math.min(g1, g2);
                temp.add(meetings[i][0]);
                temp.add(meetings[i][1]);
                ++i;
            }
            for (int j = 0; j < temp.size(); ++j) {
                if (find(groups, temp.get(j)) != 0) {
                    groups[temp.get(j)] = temp.get(j);
                }
            }
        }

        for (int j = 0; j < n; ++j) {
            if (find(groups, j) == 0) result.add(j);
        }

        return result;
    }

    private int find(int[] groups, int index) {
        while (index != groups[index]) index = groups[index];
        return index;
    }
}
```