Python simple Union Find Solution

sukeyzhou ★ 22 Last Edit: July 28, 2019 12:04 AM 1.5K VIEWS

18

1. Sort `connections` by cost
2. Iterate `connections`, if `union(city1,city2)`, add `cost` to `ans`
3. Check if there is only one group of connected components

- Time Complexity: O(N log N)
  Note that Union operation takes constant time when UnionFind is implemented with both path compression and union by rank.
- Space Complexity: O(N)

```
def minimumCost(self, N: int, connections: List[List[int]]) -> int:
    parents = [x for x in range(N)]
    ranks = [1] * N

    def find(u):
        while u != parents[u]:
            parents[u] = parents[parents[u]]
            u = parents[u]
        return u

    def union(u, v):
        root_u, root_v = find(u), find(v)
        if root_u == root_v: return False
        if ranks[root_v] > ranks[root_u]:
            root_u, root_v = root_v, root_u
        parents[root_v] = root_u
        ranks[root_u] += ranks[root_v]
        return True

    connections.sort(key = lambda x: x[2])
    ans = 0
    for u, v, val in connections:
        if union(u-1, v-1): ans += val
    groups = len({find(x) for x in parents})
    return ans if groups == 1 else -1
```

Comments: 7 Best Most Votes Newest to Oldest Oldest to Newest

Type comment here... (Markdown is supported)

Post

Merciless ★ 489 August 1, 2019 8:50 AM

Slightly simpler one

```
class Solution:
    def minimumCost(self, N: int, connections: List[List[int]]) -> int:
        memo = {i:i for i in range(1,N + 1)}

        def find(x):
            if memo[x] != x:
                memo[x] = find(memo[x])
            return memo[x]
```

Read More

3 Reply

cool_shark ★ 414 July 27, 2019 10:44 PM

I had a very similar solution.

```
class UnionFind:
    def __init__(self, n):
```