305. Number of Islands II

作者：qianrong wu

思路

为了解决陆地合并，最好能将每个陆地都标记出其属于哪个岛屿，这样就会方便我们统计岛屿个数。这种群组类问题，很适合使用Union Find 做。root可以用数组或者HashMap来表示，如果个体是数字的话，那么数组就OK，如果个体是字符串的话，可能就需要用HashMap了。root数组的初始化可以有两种，可以均初始化为-1，或者都初始化为不同的数字。getRoot函数的写法也可用递归或者迭代的方式.

此题跟经典的UF使用场景有一点点的区别，因为一般的场景中两个个体之间只有两种关系，属于一个群组或者不属于同一个群组，而这道题里面由于water的存在，就多了一种情况，我们只需要事先检测一下当前位置是不是岛屿就行了。一般来说我们的root数组都是使用一维数组，方便一些，那么这里就可以将二维数组encode为一维的，于是我们需要一个长度为m*n的一维数组来标记各个位置属于哪个岛屿，我们假设每个位置都是一个单独岛屿，岛屿编号可以用其坐标位置表示，但是我们初始化时将其都赋为-1，这样方便我们知道哪些位置尚未变成岛屿。然后我们开始遍历陆地数组，将其岛屿编号设置为其坐标位置，然后岛屿计数加1，我们此时开始遍历其上下左右的位置，遇到越界或者岛屿标号为-1的情况直接跳过，遇到是water的地方直接跳过。否则我们用getRoot来查找邻居位置的岛屿编号，同时也用getRoot来查找当前点的编号，这一步就是经典的UF算法的操作了，因为当前这两个land是相邻的，它们是属于一个岛屿，所以其getRoot函数的返回值suppose应该是相等的，但是如果返回值不同，说明我们需要合并岛屿，将两个返回值建立关联，并将岛屿计数cnt减1。当我们遍历完当前点的所有邻居时，该合并的都合并完了，将此时的岛屿计数cnt存入结果中 # 解法

```
class Solution(object):
    def numIslands2(self, m, n, positions):
        """
        :type m: int
        :type n: int
        :type positions: List[List[int]]
        :rtype: List[int]
        """
        def node_id(node, n):
            return node[0] * n + node[1]

        def find_set(x):
           if set[x] != x:
               set[x] = find_set(set[x])  # path compression.
           return set[x]

        def union_set(x, y):
            x_root, y_root = find_set(x), find_set(y)
            set[min(x_root, y_root)] = max(x_root, y_root)

        numbers = []
        number = 0
        directions = [(0, -1), (0, 1), (-1, 0), (1, 0)]
        set = {}
        for position in positions:
            node = (position[0], position[1])
            set[node_id(node, n)] = node_id(node, n)
            number += 1

            for d in directions:
                neighbor = (position[0] + d[0], position[1] + d[1])
                if 0 <= neighbor[0] < m and 0 <= neighbor[1] < n and \
                   node_id(neighbor, n) in set:
                   if find_set(node_id(node, n)) != find_set(node_id(neighbor, n)):
                       # Merge different islands, amortised time: O(log*k) ~= O(1)
                       union_set(node_id(node, n), node_id(neighbor, n))
                       number -= 1
            numbers.append(number)

        return numbers
```

总结

Time complexity : O(m×n + L)O(m×n+L); Space complexity : O(m×n)O(m×n)