685. Redundant Connection II

one pass disjoint set solution with explain

2.4K
VIEWS

39

Last Edit: 2 hours ago

(/tyuan73) tyuan73 (/tyuan73) 630

This problem is tricky and interesting. It took me quite a few hours to figure it out. My first working solution was based on DFS, but I feel there might be better solutions. By spending whole night thinking deeply, I finally cracked it using Disjoint Set (also known as Union Find?). I want to share what I got here.

Assumption before we start: input "**edges**" contains a directed tree with one and only one extra edge. If we remove the extra edge, the remaining graph should make a directed tree - a tree which has one root and from the root you can visit all other nodes by following directed edges. It has features:

1. one and only one root, and root does not have parent;
2. each non-root node has exactly one parent;
3. there is no cycle, which means any path will reach the end by moving at most (n-1) steps along the path.

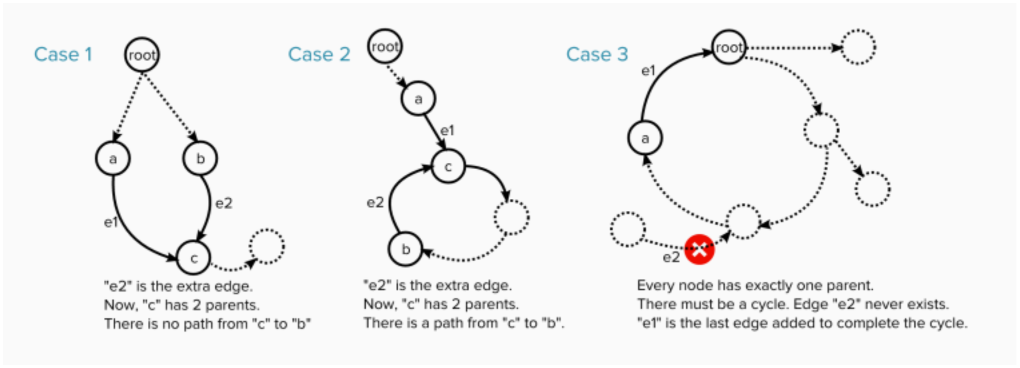
By adding one edge ***(parent->child)*** to the tree:

1. every node including root has exactly one parent, if ***child*** is root;
2. root does not have parent, one node (***child***) has 2 parents, and all other nodes have exactly 1 parent, if ***child*** is not root.

Let's check cycles. By adding one edge ***(a->b)*** to the tree, the tree will have:

1. a cycle, if there exists a path from ***(b->...->a)***; in particularly, if ***b == root***, (in other word, add an edge from a node to root) it will make a cycle since there must be a path ***(root->...->a)***.
2. no cycle, if there is no such a path ***(b->...->a)***.

After adding the extra edge, the graph can be generalized in 3 different cases:


Case 1
root
a
b
c
e1
e2
"e2" is the extra edge.
Now, "c" has 2 parents.
There is no path from "c" to "b"
Case 2
root
a
e1
c
e2
b
"e2" is the extra edge.
Now, "c" has 2 parents.
There is a path from "c" to "b".
Case 3
root
e1
a
e2
Every node has exactly one parent.
There must be a cycle. Edge "e2" never exists.
"e1" is the last edge added to complete the cycle.

`Case 1` : "c" is the only node which has 2 parents and there is not path (c->...->b) which means no cycle. In this case, removing either "e1" or "e2" will make the tree valid. According to the description of the problem, whichever edge added later is the answer.

`Case 2` : "c" is the only node which has 2 parents and there is a path(c->...->b) which means there is a cycle. In this case, "e2" is the only edge that should be removed. Removing "e1" will make the tree in 2 separated groups. Note, in input `edges` , "e1" may come after "e2".

`Case 3` : this is how it looks like if edge ***(a->root)*** is added to the tree. Removing any of the edges along the cycle will make the tree valid. But according to the description of the problem, the last edge added to complete the cycle is the answer. Note: edge "e2" (an edge pointing from a node outside of the cycle to a node on the cycle) can never happen in this case, because every node including root has exactly one parent. If "e2" happens, that make a node on cycle have 2 parents. That is impossible.

As we can see from the pictures, the answer must be:

1. one of the 2 edges that pointing to the same node in `case 1` and `case 2` ; there is one and only one such node which has 2 parents.
2. the last edge added to complete the cycle in `case 3` .

Note: both `case 2` and `case 3` have cycle, but in `case 2` , "e2" may not be the last edge added to complete the cycle.

Now, we can apply Disjoint Set (DS) to build the tree in the order the edges are given. We define `ds[i]` as the parent or ancestor of node `i` . It will become the root of the whole tree eventually if `edges` does not have extra edge. When given an edge (a->b), we find node `a` 's ancestor and assign it to `ds[b]` . Note, in typical DS, we also need to find node `b` 's ancestor and assign `a` 's ancestor as the ancestor of `b` 's ancestor. But in this case, we don't have to, since we skip the second parent edge (see below), it is guaranteed `a` is the only parent of `b`.

If we find an edge pointing to a node that already has a parent, we simply skip it. The edge skipped can be "e1" or "e2" in `case 1` and `case 2`. In `case 1`, removing either "e1" or "e2" will make the tree valid. In `case 3`, removing "e2" will make the tree valid, but removing "e1" will make the tree in 2 separated groups and one of the groups has a cycle. In `case 3`, none of the edges will be skipped because there is no 2 edges pointing to the same node. The result is a graph with cycle and "n" edges.

How to detect cycle by using Disjoint Set (Union Find)?
When we join 2 nodes by edge (a->b), we check `a`'s ancestor, if it is b, we find a cycle! When we find a cycle, we don't assign `a`'s ancestor as `b`'s ancestor. That will trap our code in endless loop. We need to save the edge though since it might be the answer in `case 3`.

Now the code. We define two variables (`first` and `second`) to store the 2 edges that point to the same node if there is any (there may not be such edges, see `case 3`). We skip adding `second` to tree. `first` and `second` hold the values of the original index in input `edges` of the 2 edges respectively. Variable `last` is the edge added to complete a cycle if there is any (there may not be a cycle, see `case 1` and removing "e2" in `case 2`). And it too hold the original index in input `edges`.

After adding all except at most one edges to the tree, we end up with 4 different scenario:

1. `case 1` with either "e1" or "e2" removed. Either way, the result tree is valid. The answer is the edge being removed or skipped (a.k.a. `second`)
2. `case 2` with "e2" removed. The result tree is valid. The answer is the edge being removed or skipped (a.k.a. `second`)
3. `case 2` with "e1" removed. The result tree is invalid with a cycle in one of the groups. The answer is the other edge (`first`) that points to the same node as `second`.
4. `case 3` with no edge removed. The result tree is invalid with a cycle. The answer is the `last` edge added to complete the cycle.

In the following code,
`last == -1` means "no cycle found" which is scenario 1 or 2
`second != -1 && last != -1` means "one edge removed and the result tree has cycle" which is scenario 3
`second == -1` means "no edge skipped or removed" which is scenario 4

```
    public int[] findRedundantDirectedConnection(int[][] edges) {
        int n = edges.length;
        int[] parent = new int[n+1], ds = new int[n+1];
        Arrays.fill(parent, -1);
        int first = -1, second = -1, last = -1;
        for(int i = 0; i < n; i++) {
            int p = edges[i][0], c = edges[i][1];
            if (parent[c] != -1) {
                first = parent[c];
                second = i;
                continue;
            }
            parent[c] = i;

            int p1 = find(ds, p);
            if (p1 == c) last = i;
            else ds[c] = p1;
        }

        if (last == -1) return edges[second]; // no cycle found by removing second
        if (second == -1) return edges[last]; // no edge removed
        return edges[first];
     }

     private int find(int[] ds, int i) {
         return ds[i] == 0 ? i : (ds[i] = find(ds, ds[i]));
     }
```

This solution past all these test cases and ACed.
Test case:
[[1,2],[1,3],[2,3]]
[[1,2], [2,3], [3,4], [4,1], [1,5]]
[[4,2],[1,5],[5,2],[5,3],[2,4]]
[[2,1],[3,1],[4,2],[1,4]]
[[4,1],[1,2],[1,3],[4,5],[5,6],[6,5]]
[[2,3], [3,4], [4,1], [1,5], [1,2]]
[[3,1],[1,4],[3,5],[1,2],[1,5]]
[[1,2],[2,3],[3,1]]

expected output:
[2,3]
[4,1]
[4,2]
[2,1]

[6,5]
[1,2]
[1,5]
[3,1]

Comments: 5

Sort By

Type comment here... (Markdown is supported)

Preview Post

marsleezm (/marsleezm) ★ 0 October 10, 2018 12:58 PM

(/marsleezm)

This really is THE post that should get the highest vote. Honestly, other posts that only give the code and a few lines of hints are really not helpful at all. I especially appreciate your effort in writing down the thought process, which really helped me to understand the solution :) :) !

0 Share Reply

Zhiyuan_Yao (/zhiyuan_yao) ★ 21 September 26, 2018 10:59 PM

(/zhiyuan_yao)

This is an excellent post! Thank you!
Small typo in the paragraph right ABOVE `How to detect cycle by using Disjoint Set (Union Find)?` : It should be:
`In case 2 (was "3" in the post), removing "e2" will make the tree valid, but removing "e1" will make the tree in 2 separated groups and one of the groups has a cycle.`

0 Share Reply

jocelynayoga (/jocelynayoga) ★ 130 September 15, 2018 9:45 PM

(/jocelynayoga)

another java version using HashMap for translation

Read More

0 Share Reply

jso (/jso) ★ 7 October 5, 2017 1:42 PM

(/jso)

Here is a python version.

```
def findRedundantDirectedConnection(self, edges):
    first, second, n = None, None, len(edges)
    ds = ''.join(map(chr, range(n+1)))
```

Read More

0 Share Reply

SHOW 1 REPLY

jso (/jso) ★ 7 October 5, 2017 1:04 PM

(/jso)

Nice solution, here is my interpretation: if there is a cycle and dual parents, then one of the dual edges is on the cycle. If you can complete the cycle without the second dual, then it must be the first, if you can't then it must be the second. If there is no cycle, you want the second dual anyway, so the answer is the same as if the cycle included the second dual.

0 Share Reply