

森林与并查集

胡船长

初航我带你，远航靠自己

本章题目

1-校招. Leetcode-128: 最长连续序列

2-校招. Leetcode-130: 被围绕的区域

3-校招. Leetcode-200: 岛屿数量

4-校招. Leetcode-547: 省份数量

5-竞赛. HZOJ-72: 练习题2-猜拳

6-竞赛. HZOJ-322: 程序自动分析

7-竞赛. HZOJ-327: 关押罪犯

本期内容

一. 什么是：连通性问题

二.Quick-Find 算法

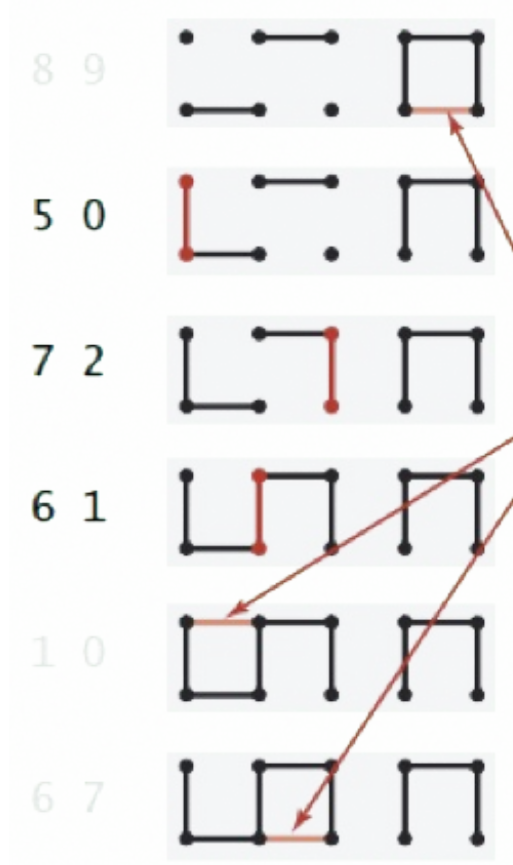
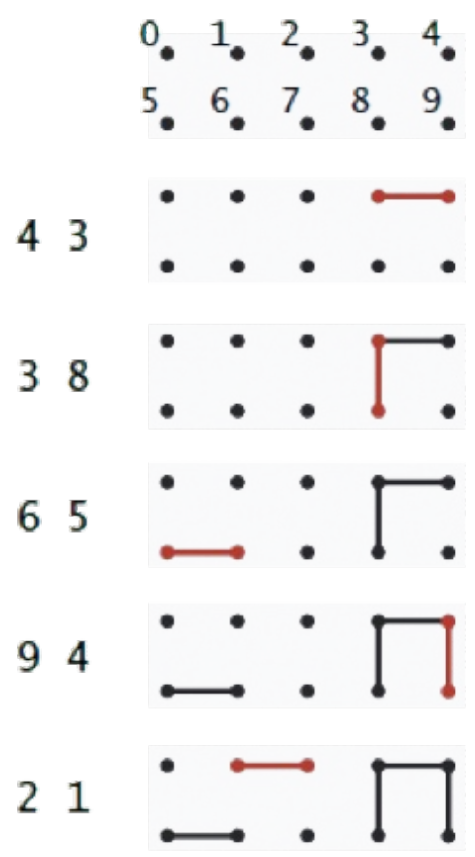
三.Quick-Union 算法

四.并查集的按秩优化

五.路径压缩优化

一. 什么是：连通性问题

连通性问题



*don't print
pairs that
are already
connected*

连通性问题

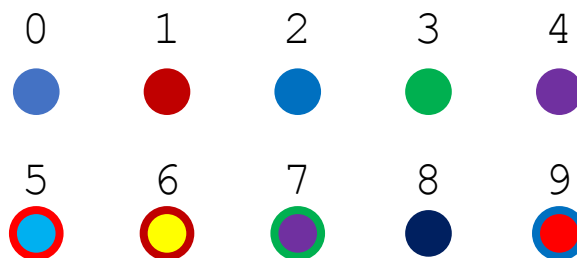
- 1、基于染色思想，一开始所有点的颜色不同
- 2、连接两个点的操作，可以看成将一种颜色的点染成另一种颜色
- 3、如果两个点颜色一样，证明联通，否则不联通
- 4、这种方法叫做并查集的：【Quick-Find 算法】

二. Quick-Find 算法

Quick-Find算法

now :

next : [4 -- 3]

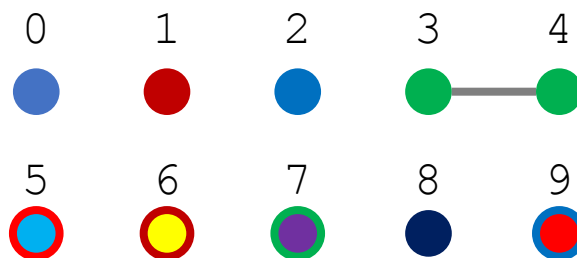


0	1	2	3	4
0	1	2	3	4
5	6	7	8	9
5	6	7	8	9

Quick-Find算法

now : [4 -- 3]

next : [4 -- 8]

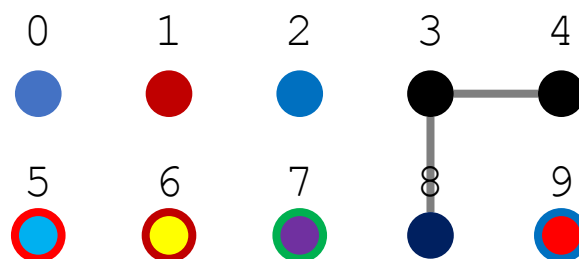


0	1	2	3	4
0	1	2	3	3
5	6	7	8	9
5	6	7	8	9

Quick-Find算法

now : [4 -- 8]

next : [6 -- 5]

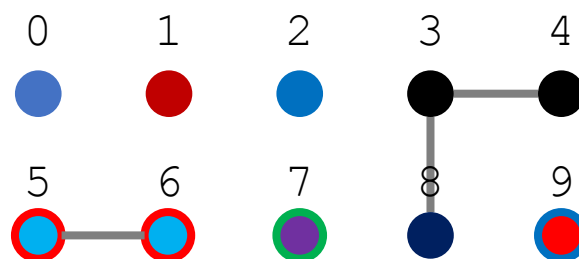


0	1	2	3	4
0	1	2	8	8
5	6	7	8	9
5	6	7	8	9

Quick-Find算法

now : [6 -- 5]

next : [9 -- 4]

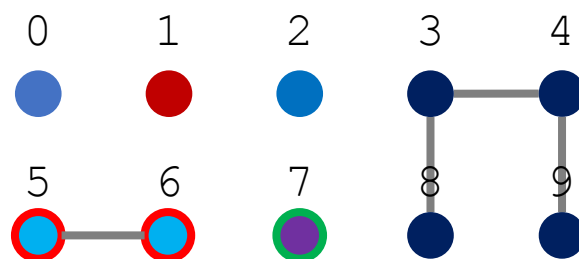


0	1	2	3	4
0	1	2	8	8
5	6	7	8	9
5	5	7	8	9

Quick-Find算法

now : [9 -- 4]

next : [2 -- 1]

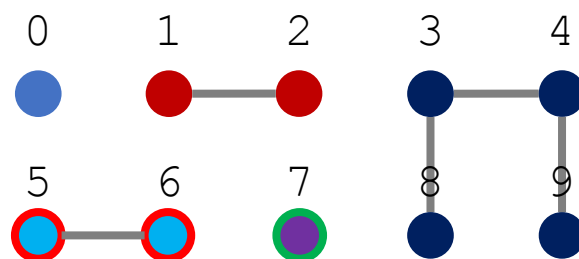


0	1	2	3	4
0	1	2	8	8
5	6	7	8	9
5	5	7	8	8

Quick-Find算法

now : [2 -- 1]

next : [5 -- 0]

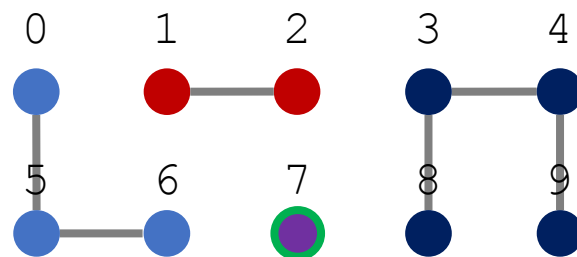


0	1	2	3	4
0	1	1	8	8
5	6	7	8	9
5	5	7	8	8

Quick-Find算法

now : [5 -- 0]

next : [7 -- 2]

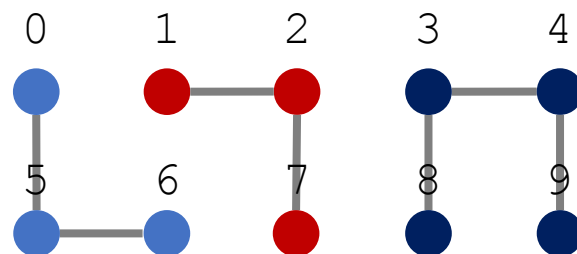


0	1	2	3	4
0	1	1	8	8
5	6	7	8	9
0	0	7	8	8

Quick-Find算法

now : [7 -- 2]

next : [6 -- 1]

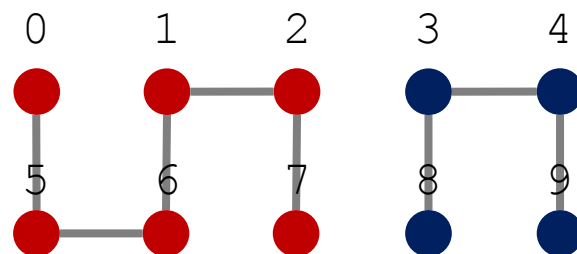


0	1	2	3	4
0	1	1	8	8
5	6	7	8	9
0	0	1	8	8

Quick-Find算法

now : [6 -- 1]

next :



0	1	2	3	4
1	1	1	8	8
5	6	7	8	9
1	1	1	8	8

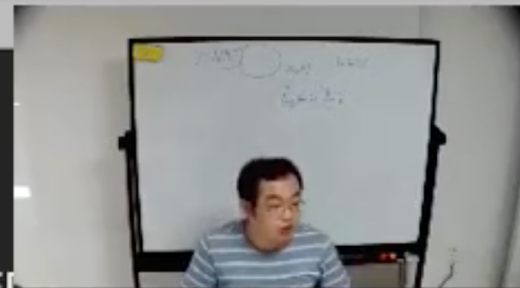
Quick-Find 算法总结

- 1、联通判断： $O(1)$
- 2、合并操作： $O(n)$

问题思考：

- 1、quick-find 算法的联通判断非常快，可是合并操作非常慢
- 2、本质上问题中只是需要知道一个点与哪些点的颜色相同
- 3、而若干点的颜色可以通过间接指向同一个节点
- 4、合并操作时，实际上是将一棵树作为另一棵树的子树

```
vim %1 bash %2 bash %3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51     }
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55     }
56 }
57
58
```



Quick-Find 算法：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

三. Quick-Union 算法

Quick-Find 算法总结

- 1、联通判断： $O(1)$
- 2、合并操作： $O(n)$

问题思考：

- 1、quick-find 算法的联通判断非常快，可是合并操作非常慢
- 2、本质上问题中只是需要知道一个点与哪些点的颜色相同
- 3、而若干点的颜色可以通过间接指向同一个节点
- 4、合并操作时，实际上是将一棵树作为另一棵树的子树

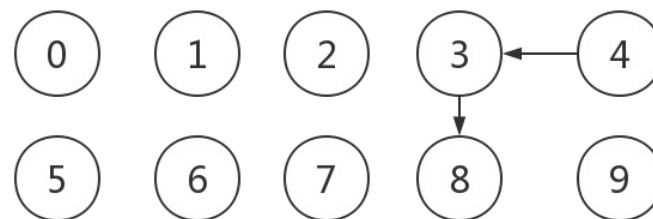
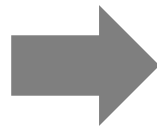
Quick-Union算法

now : [4 -- 8]

next : [6 -- 5]



0	1	2	3	4
0	1	2	8	8
5	6	7	8	9
5	6	7	8	9



0	1	2	3	4
0	1	2	8	3
5	6	7	8	9
5	6	7	8	9

随堂练习-1

有10个点的图，按照如下顺序进行连接，请分别写出：

- 1、quick-find 算法最终数组的结果
- 2、quick-union 算法最终数组的结果

1:[0, 1]、2:[1, 2]、3:[3, 4]、4:[2, 3]

5:[8, 9]、6:[9, 7]、7:[7, 6]、8:[1, 5]

随堂练习-1

有10个点的图，按照如下顺序进行连接，请分别写出：

- 1、quick-find 算法最终数组的结果
- 2、quick-union 算法最终数组的结果

1:[0, 1]、2:[1, 2]、3:[3, 4]、4:[2, 3]
5:[8, 9]、6:[9, 7]、7:[7, 6]、8:[1, 5]

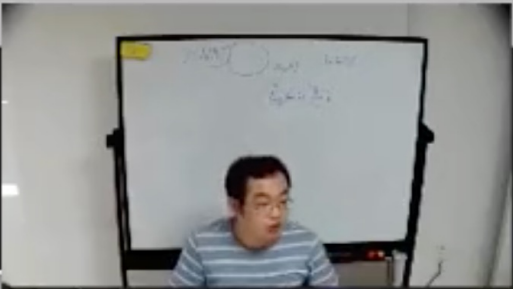
0	1	2	3	4
5	5	5	5	5
5	6	7	8	9
5	6	6	6	6

0	1	2	3	4
1	2	4	4	5
5	6	7	8	9
5	6	6	9	7

1. vim

vim %1 bash %2 bash %3

```
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55
56     }
57
58 }
```



Quick-Union 算法：代码演示

59

60

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

<-6班资料 / X.现场撸代码 / 15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

四. 并查集的按秩优化

Quick-Union 算法总结

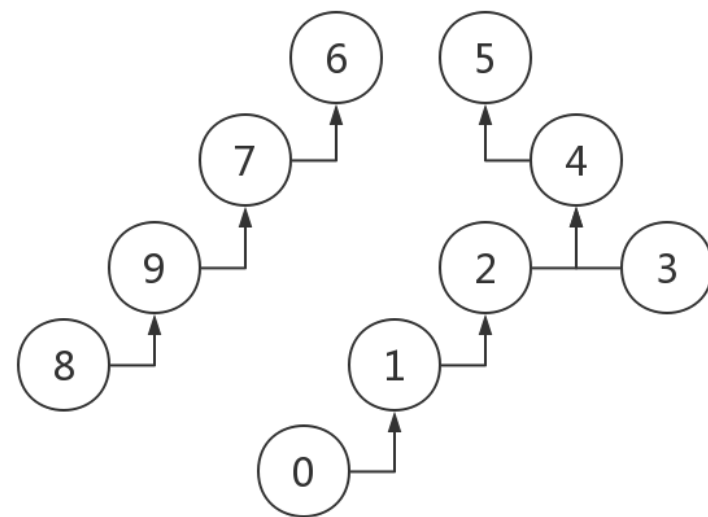
- 1、联通判断: tree-height 树高
- 2、合并操作: tree-height 树高

问题思考:

- 1、极端情况下会退化成一条链
- 2、将节点数量多的接到少的树上面, 导致了退化
- 3、将树高深的接到浅的上面, 导致了退化

随堂思考:

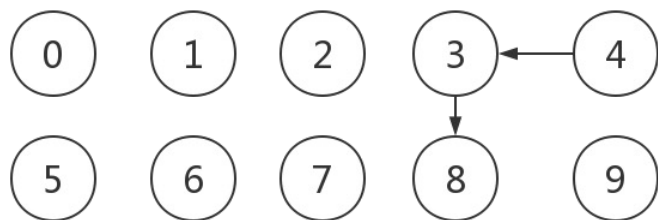
若要改进, 是按照节点数量还是按照树的高度为合并参考?



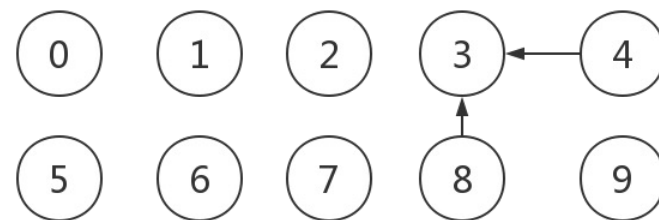
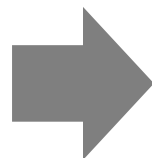
Weighted Quick-Union算法

now : [4 -- 8]

next : [6 -- 5]



0	1	2	3	4
0	1	2	8	3
5	6	7	8	9
5	6	7	8	9



0	1	2	3	4
0	1	2	3	3
5	6	7	8	9
5	6	7	3	9

随堂练习-2

有10个点的图，按照如下顺序进行连接，请分别写出：

- 1、 quick-union 算法最终数组的结果
- 2、 weighted quick-union 算法最终的数组结果

1:[0, 1]、2:[2, 3]、3:[4, 2]、4:[4, 0]

5:[8, 9]、6:[9, 7]、7:[7, 6]、8:[1, 5]

随堂练习-2

有10个点的图，按照如下顺序进行连接，请分别写出：

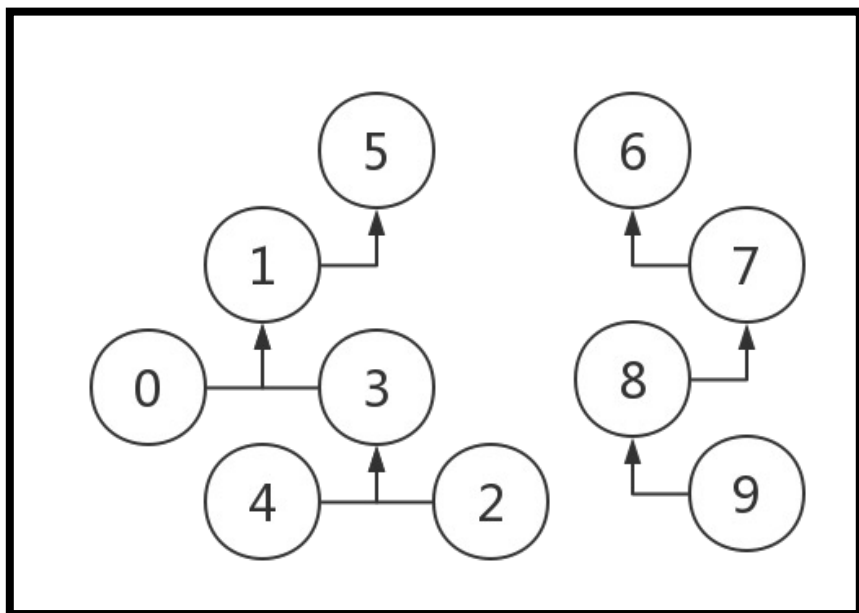
- 1、 quick-union 算法最终数组的结果
- 2、 weighted quick-union 算法最终的数组结果

1:[0, 1]、 2:[2, 3]、 3:[4, 2]、 4:[4, 0]
5:[8, 9]、 6:[9, 7]、 7:[7, 6]、 8:[1, 5]

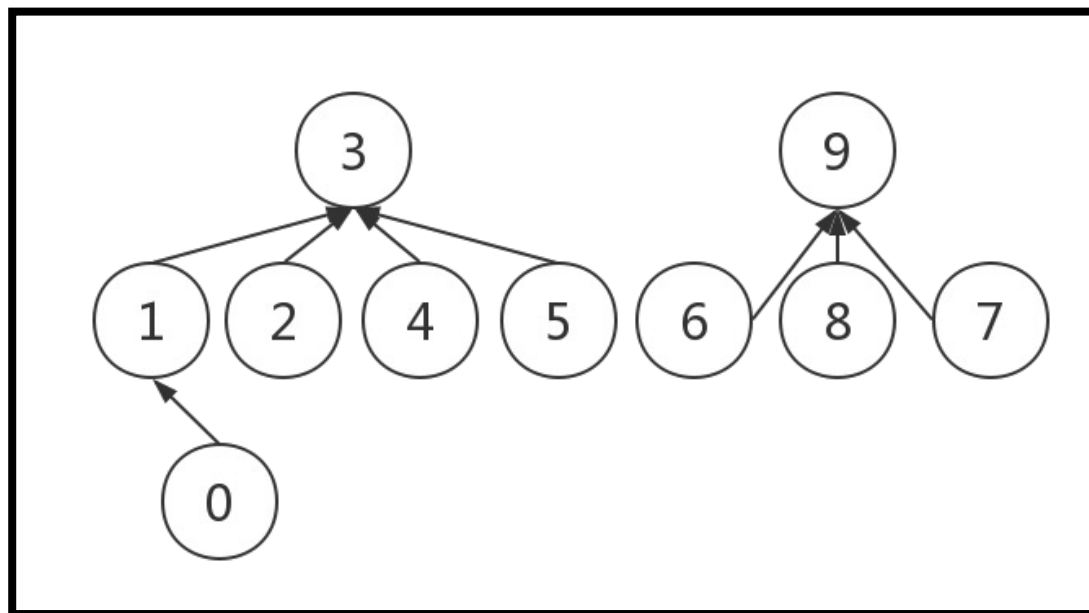
0	1	2	3	4
1	5	3	1	3
5	6	7	8	9
5	6	6	9	7

0	1	2	3	4
1	3	3	3	3
5	6	7	8	9
3	9	9	9	9

随堂练习-2



Quick-Union



Weighted Quick-Union

五. 路径压缩优化

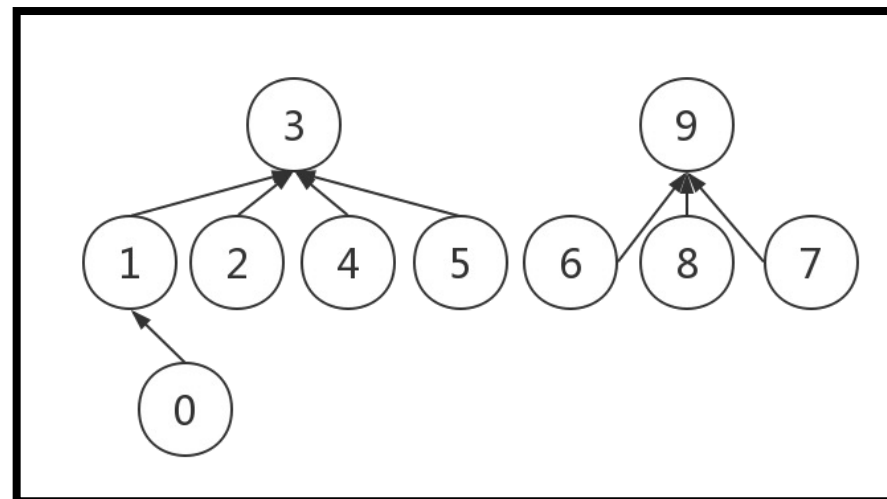
Weighted Quick-Union 算法总结

1、联通判断: $\log(N)$

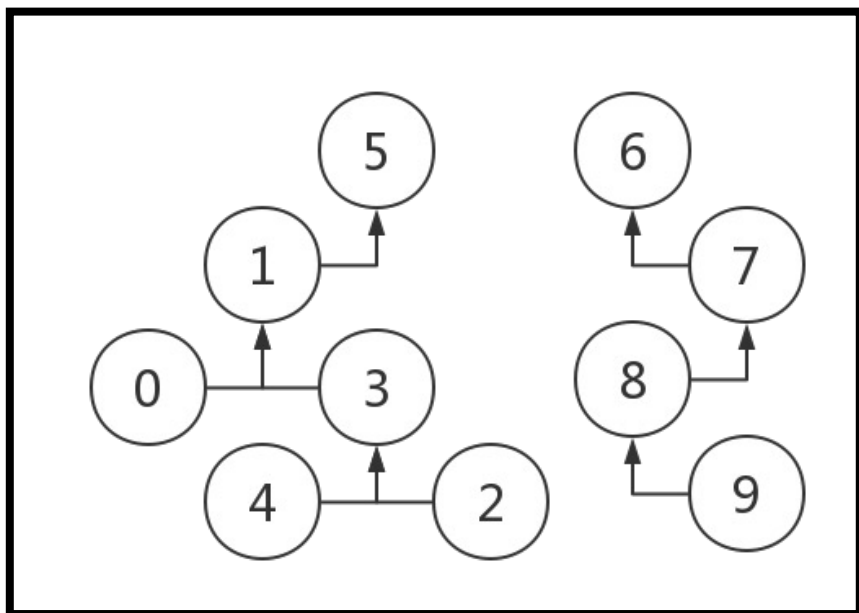
2、合并操作: $\log(N)$

问题最终优化:

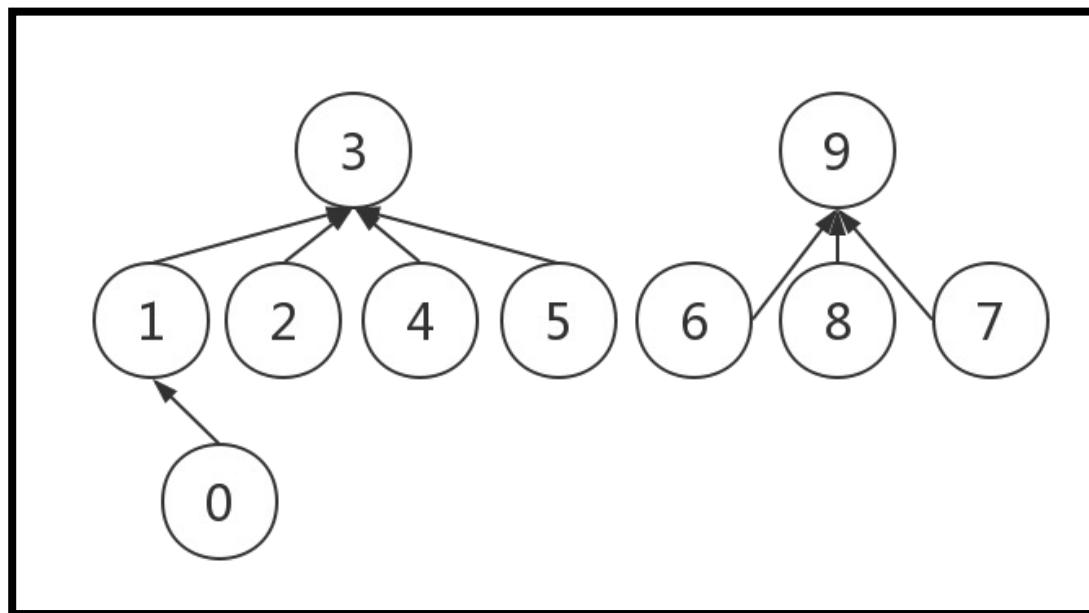
参考 quick-find 算法, 做【路径压缩】



随堂练习-2



Quick-Union



Weighted Quick-Union

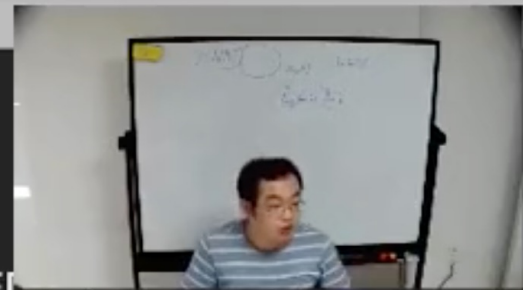
并查集总结

Algorithm	Constructor	Union	Find
Quick-Find	N	N	1
Quick-Union	N	Tree height	Tree height
Weighted Quick-Union	N	$\lg N$	$\lg N$
Weighted Quick-Union With Path Compression	N	Very near to 1 (amortized)	Very near to 1 (amortized)

课后阅读:

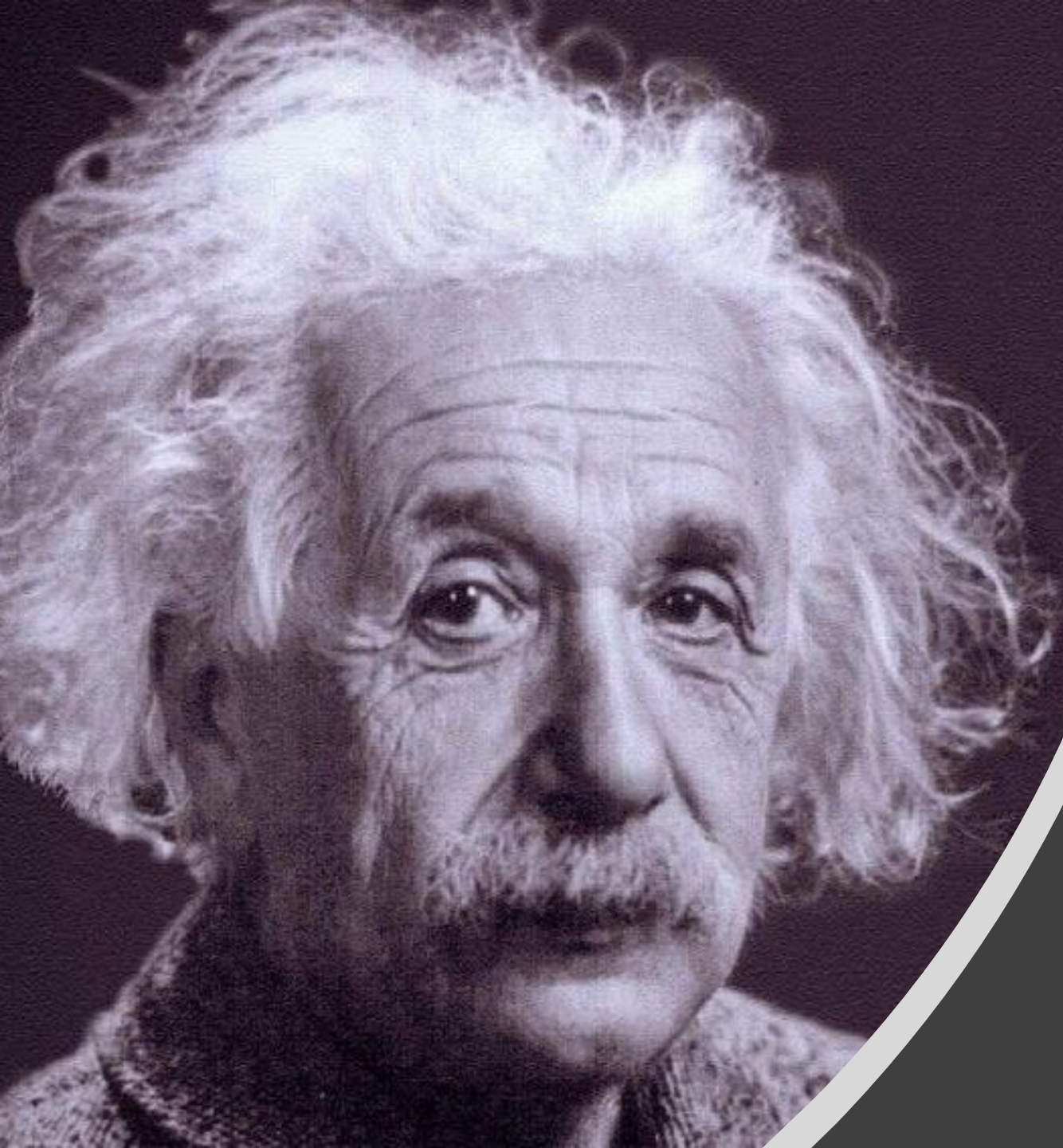
- 1、http://blog.csdn.net/dm_vincent/article/details/7655764
- 2、http://blog.csdn.net/dm_vincent/article/details/7769159

```
vim %1 bash %2 bash %3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51     }
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54     }
55 }
56
57
58
```



并查集优化：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```



为什么
会出一样的题目？