Union Query

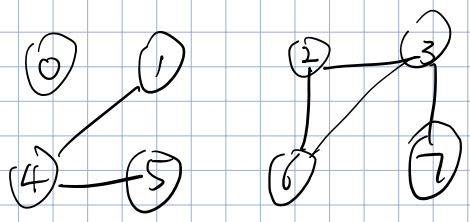
并查集

p is connected to p

p is connect to q ⇔ q is connected to p

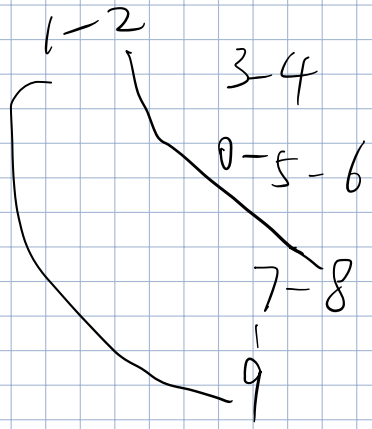
互通.

{0} {1, 4, 5} {2, 3, 6, 7}

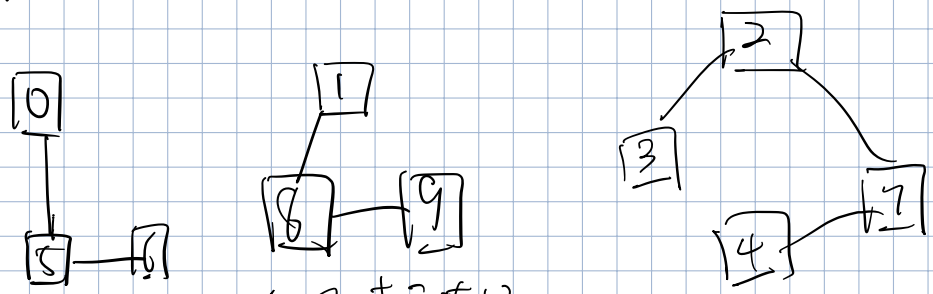


Find query: 找两个对象是否是在同一连通分量

Union Command: 连接分量，变通集合.



Quick find 贪心算法



id 相等 则表示连通

	0	1	2	3	4	5	6	7	8	9
id[]	1	6	7	7	7	1	1	7	6	6

Let id = new Array(10)

(p, q)

union(2, 3)

```
for( let i=0; i<N; i++)
{ if( id[i] == id[p])
    id[i] = id[q]; }
```

union(0, 1)

id. 把所有数 6

错误，因为未完成合并时 id[p] 已经改变，无法找原来的 id[p] 值

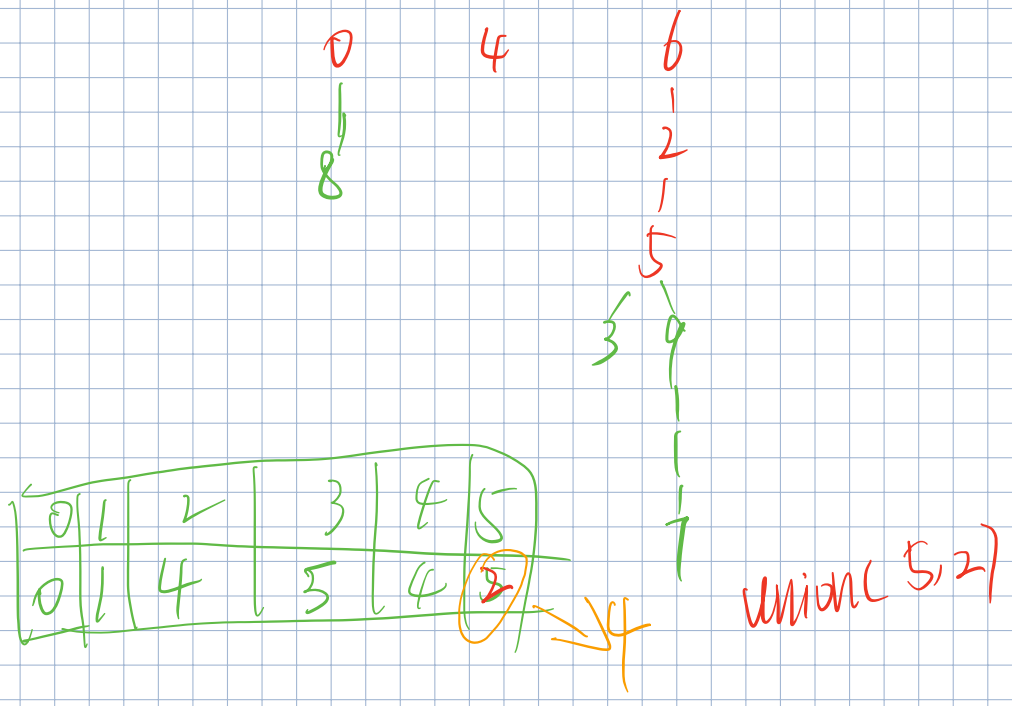
判断连通，O(1)

合并，N个对象 O(N²)

初始化：O(N)

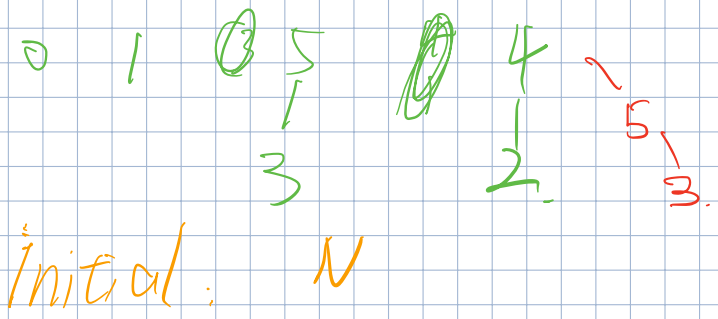
Quick Union 快速合并，"懒策略"

id保存某个节点的父节点



0	1	2	3	4	5
0	1	4	5	4	2

union(5,2)



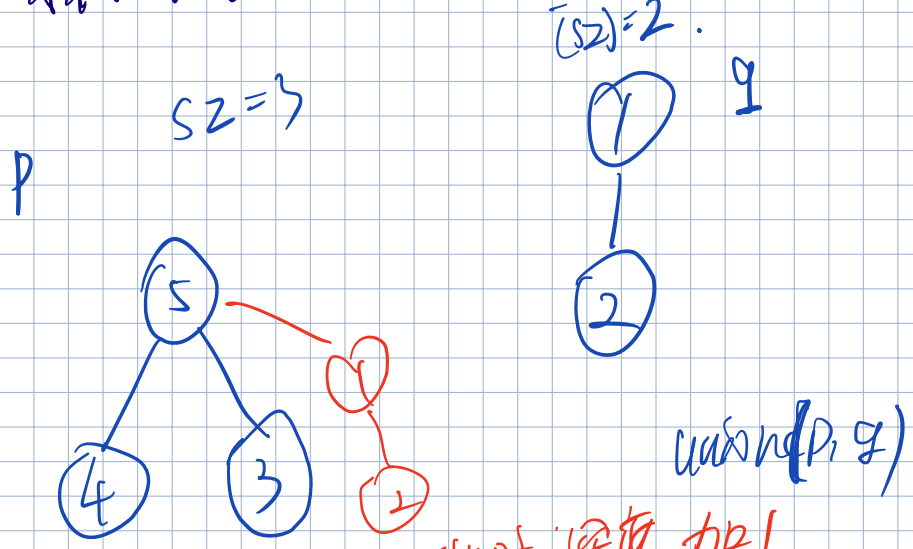
initial: N

Quick-Union Improvements.

Improvement 1:

Weighted quick-union.

- 避免生成过高的树
- 追踪每个树的尺寸.
- 将小树的根连接到大树的根.



union(p, q)

此时深度加1.
节点数至少翻倍.

翻倍 $\log_2 N$ 次, 得到所有点

$1 \times (2)^{\log_2 N} = N$

合并与查询都需 $O(\log N)$ 的时间复杂度

进一步优化：

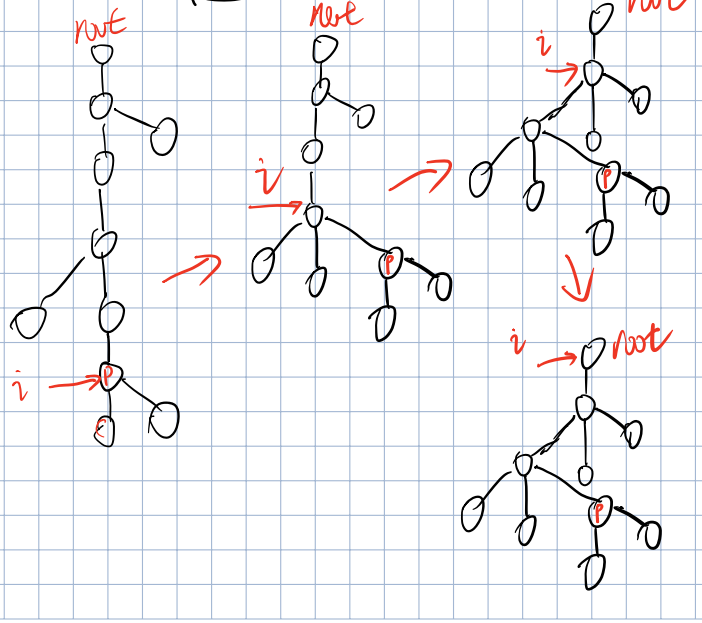
路径压缩

```
root = function (i) {
    while (i != id[i]) {
        id[i] = id[id[i]]
        i = id[i]
    }
    return
}
```

标准

指针指向各个节点

每次向上查找，将其放在各个节点

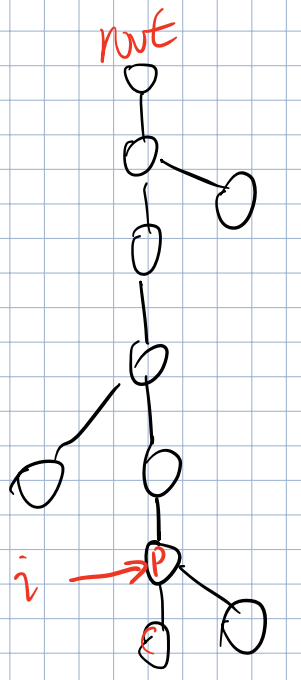


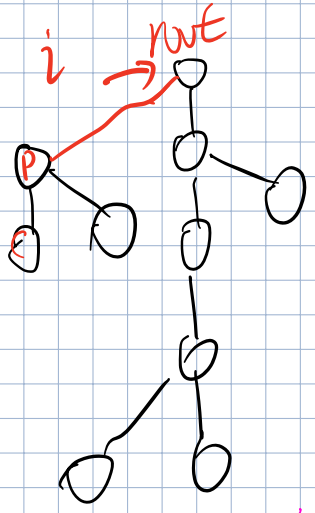
我的思路

```
root= function(i){
  let tmp=i
while( i != id[i]){
  i = id[i];
}
id[tmp]=i;
return i;
}
```

每次找到根节点时，将该节点直接添加到根节点下。

只需几行代码，常数次时间。





果然，大神给的解法更妙

union(2,4)

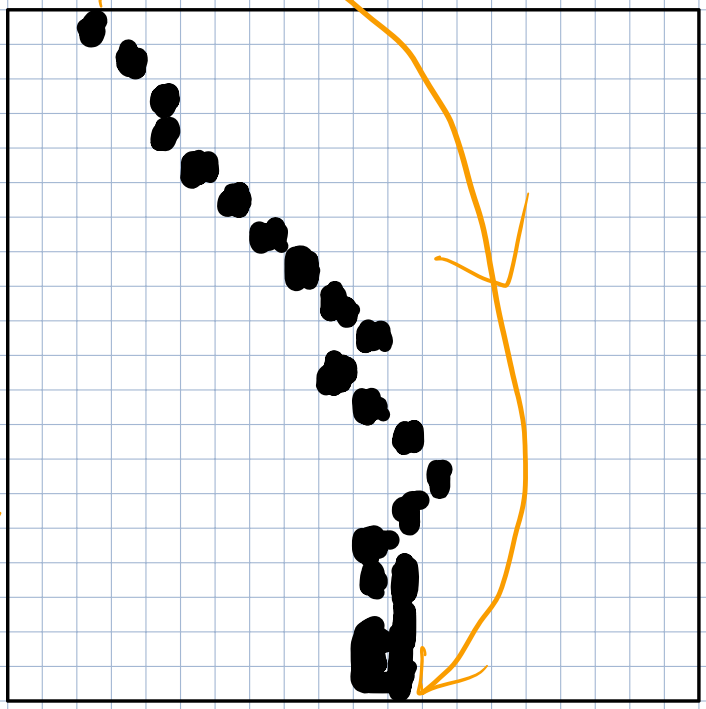
并查集不存在线性时间算法

algorithm	最差时间复杂度
quick-find	MN
quick-union	MN
weighted QU	$N + M \log N$
QU + path compression	$N + M \log N$
weighted QU + path compression	$N + M \lg N$

并查集不存在线性查找时间

N个对象，M次find或union操作

渗透模型



每个单元
以一定概率
发生贯通，
概率为P，
称为渗流，
从上到下贯通

以白色表示导通为例,

暴力枚举, N×N次,
对顶部每个点 (5)
检查与底部任一点是否连通
worst-time-complexity N^2. 5×5 25

解:
方法, 顶部和底部, 创建虚拟节点,
只需检查 top, bottom 结点是否连通

思路：对每个点先
① 概率判断，以p
决定是否开通，设置
标记为1，
② 对每个点进行归并
处理，一旦有标记为
1的在四周，进行
归并
③ 对 top-site
bottom-site 连通
性检查