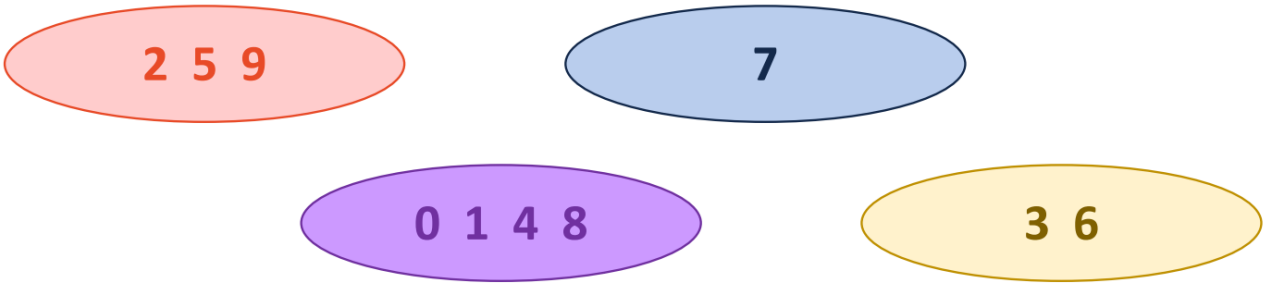
回顾一下MATH213

差不多就是每个元素只在一个子集里面



`find(0)==find(1)`

数据结构

在数据结构中，为了方便表示，对于每一个子集我们用一个代表元素（representive element）来表示

- 维护一个集合 $S = \{S_0, S_1, \ldots, S_k\}$
- 每个子集都有一个代表元素（representive element）
- 接口：
  - `void makeset(const T& t);`
  - `void union(const T& k1, const T& k2);`
  - `T& find(const T& k);`

实现

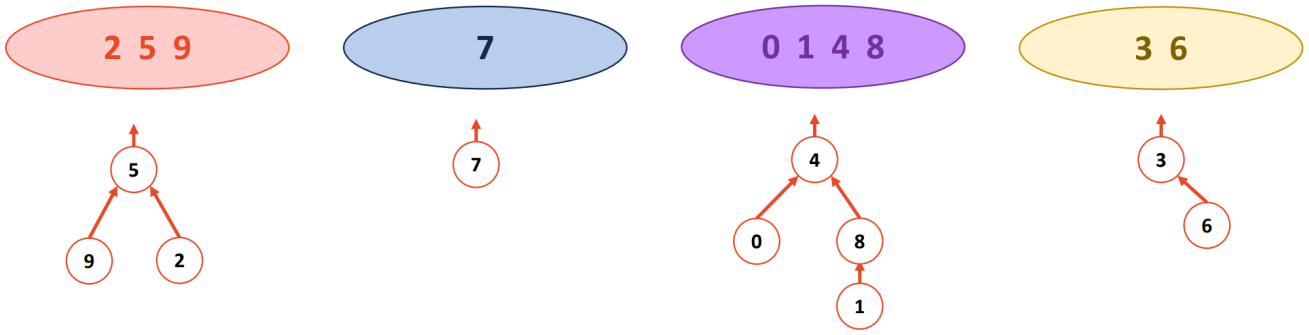
实现1

用数组实现，索引为元素，值为所属自己的代表元素

- `find(k)`：$O(1)$
- `union(k1, k2)`：$O(n)$

实现2：Uptrees

我们用一个树状结构来表示



为了实现这个结构，我们可以用一个数组，索引表示元素，值表示父节点的索引，根节点的值另外表示（图中为-1）

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

find

```
int DisjointSets:find(int i) {
    if (s[i] < 0) {
        return i;
    } else {
        return find(s[i]);
    }
}
```

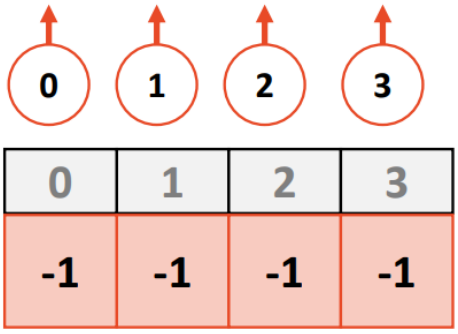
上述实现的时间复杂度在高度大的时候比较差

为了改善时间复杂度，我们可以在查询的把被路径上的节点全部指向代表节点，这样后续查询都会变成O(1)

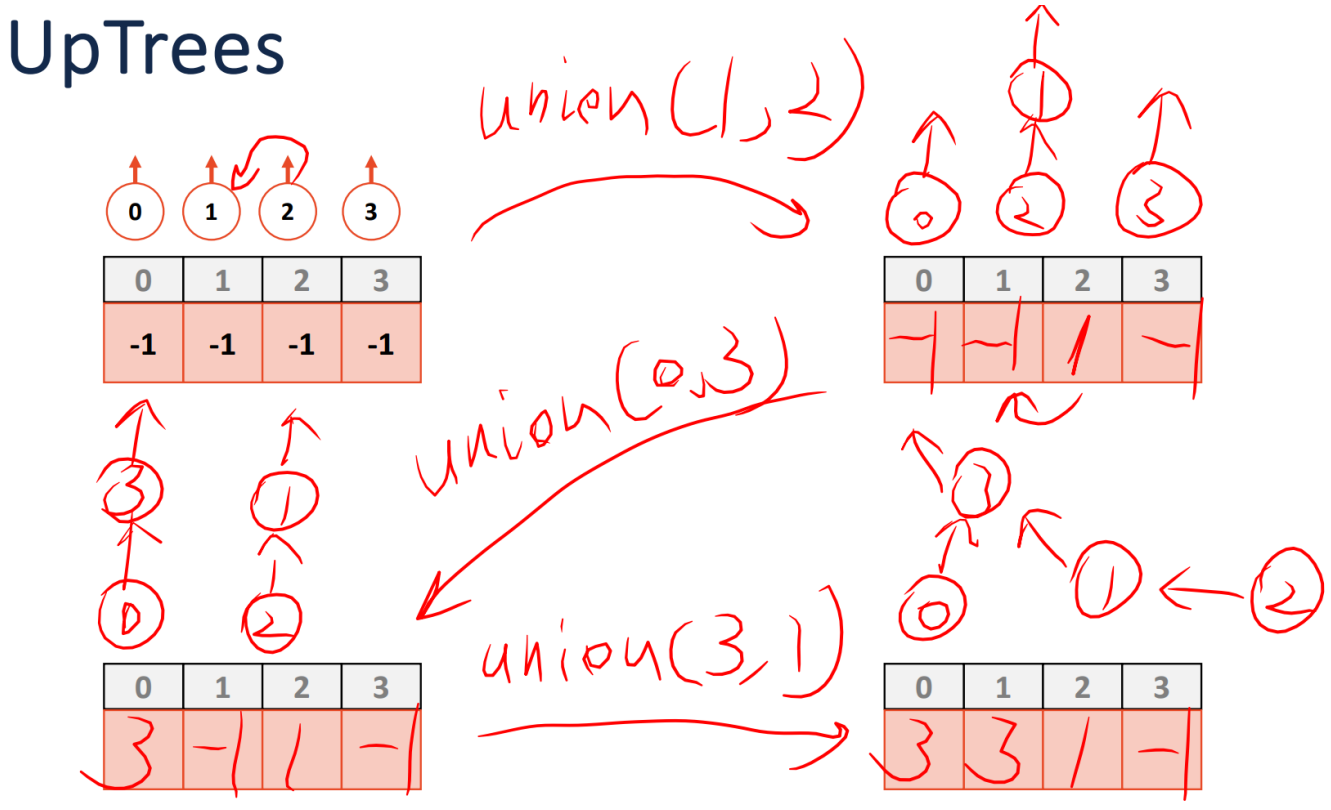
What is the ideal UpTree?

Union

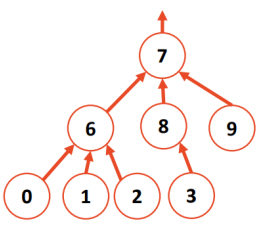
假设有下面这个数据

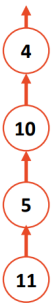


现在进行一些union操作



上述只是一个简单示意，实际操作的时候我们一般有两种策略：最小高度和最小尺寸





Union by height

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

***Idea**: Keep the height of the tree as small as possible.*

Union by size

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

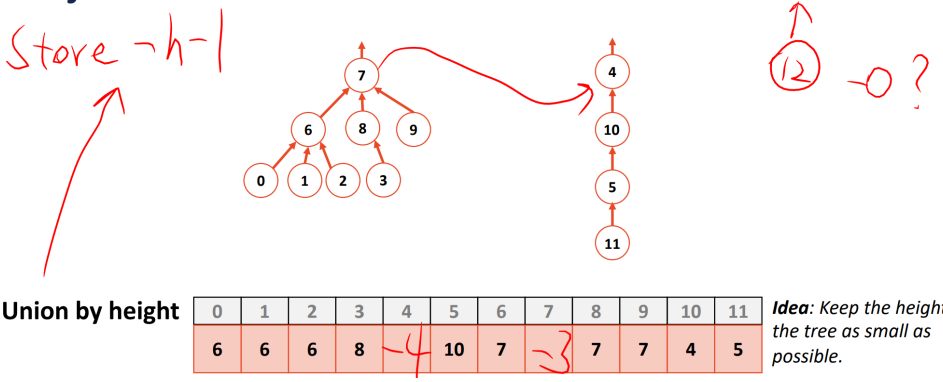
***Idea**: Minimize the number of nodes that increase in height*

两种策略都能保证树的高度为$\mathrm{O}(\log(n))$

Union by Height

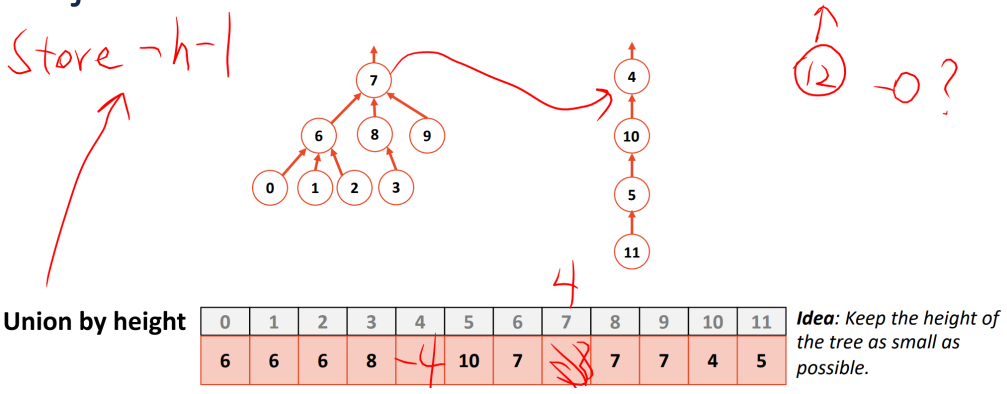
根节点的值为$-h-1$

Disjoint Sets – Smart Union



然后把较矮的树合并到较高的树的根节点

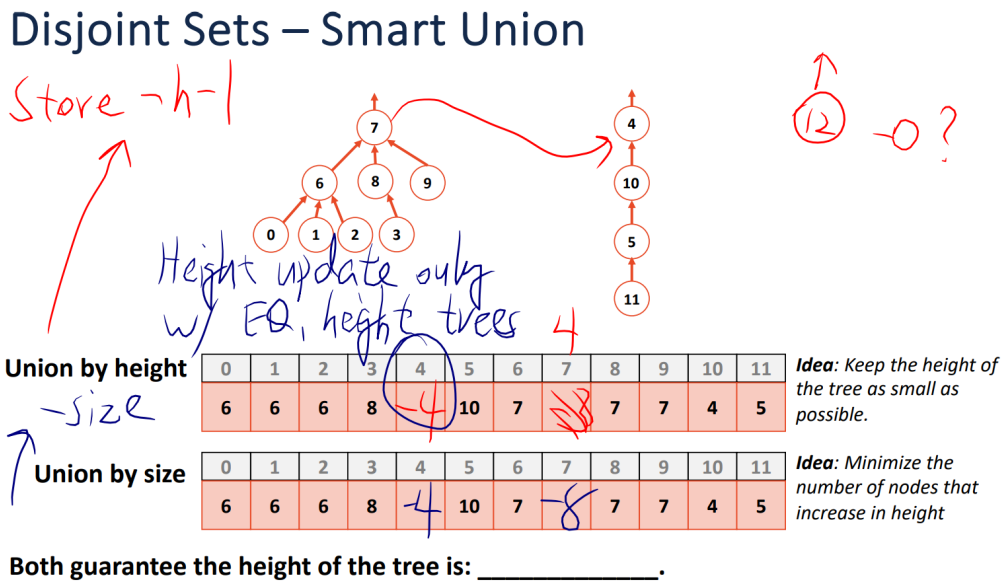
Disjoint Sets – Smart Union



然后只有在原来两树高度相同的时候需要更新高度

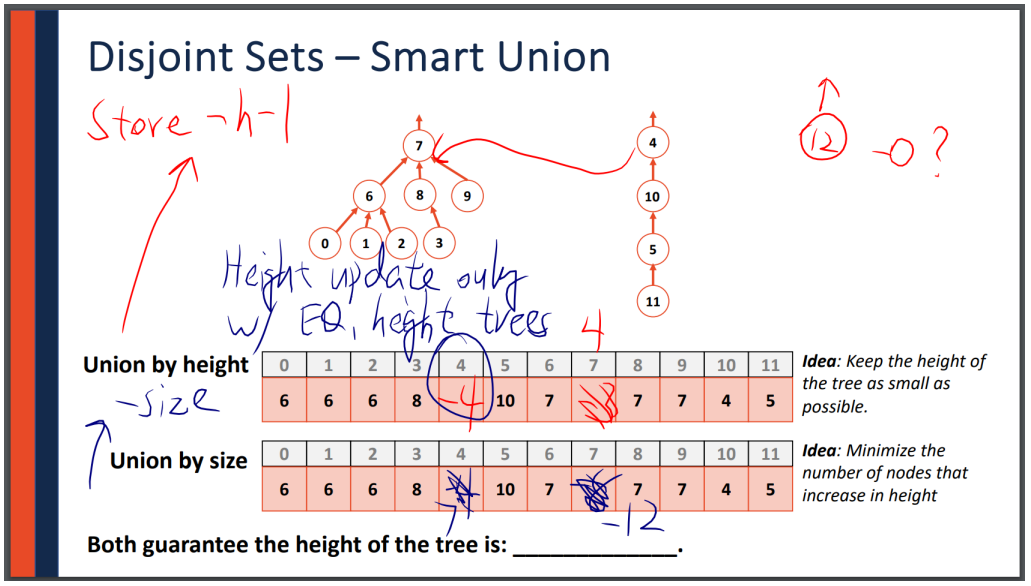
Union by Size

根节点的值为-size



然后把晓得合并到大的树的根节点

更新size信息为两者之和



Path Compression

原先上面的树可能比较高，查询的时候就要花比较长的时间，所以可以在查询的时候把路径上的节点全部连到根节点，这样理想情况

下时间复杂度是O(1)

如果我们要对Union by Size的策略使用路径压缩

我们只需要把查询函数从

```
int DisjointSetBySize::find(int x) {
    if (arr_[x] < 0) {
        return x;
    }
    return find(arr_[x]);
}
```

改为

```
int DisjointSetBySize::find(int x) {
    if (arr_[x] < 0) {
        return x;
    }
    return (arr_[x]=find(arr_[x]));
}
```

分析

迭代对数函数 Iterated log function

$$\log^*(n) = \begin{cases} o & n \leq 1 \\ 1 + \log^*(\log(n)) & n > 1 \end{cases}$$

意思就是连续算多少次对数之后值小于等于1，这里面的对数一般是以2为底

例子：$\log^*(2^{65536})$

$$\rightarrow 65536 \rightarrow 16 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

用了5次，所以结果是5

这里他的话很抽象，等会儿回来补

确实抽象，他并没有解释，记住就好

Disjoint Sets Analysis

In an Disjoint Sets implemented with smart **unions** and path compression on **find**:

Any sequence of **m union** and **find** operations result in the worse case running time of O($m \cdot \lg^*(n)$),
where **n** is the number of items in the Disjoint Sets.

Uptree

一个Uptree实际上表示一个等价类Equivalence Class

我们几乎可以在常数时间内找到一个元素的代表元素，但是无法很快的列出一个类里的所有元素。意思是在下图中，我们可以很快的找到5的代表元素，但是如果给定10，我们很难很快的列出整整棵树里的元素