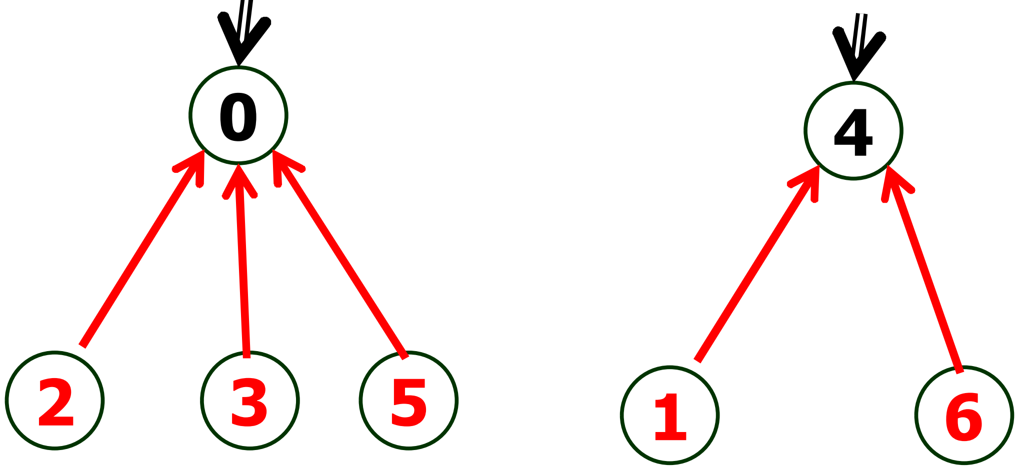
等价类的表示——树

- 一个等价类以一棵树(*in-Tree*)的形式表示，**树的根结点标记等价类**。
- 树采用双亲表示法存放，**A[*i*]**存放数据元素x_i在树中的双亲结点。

➢ S划分为2个等价类：

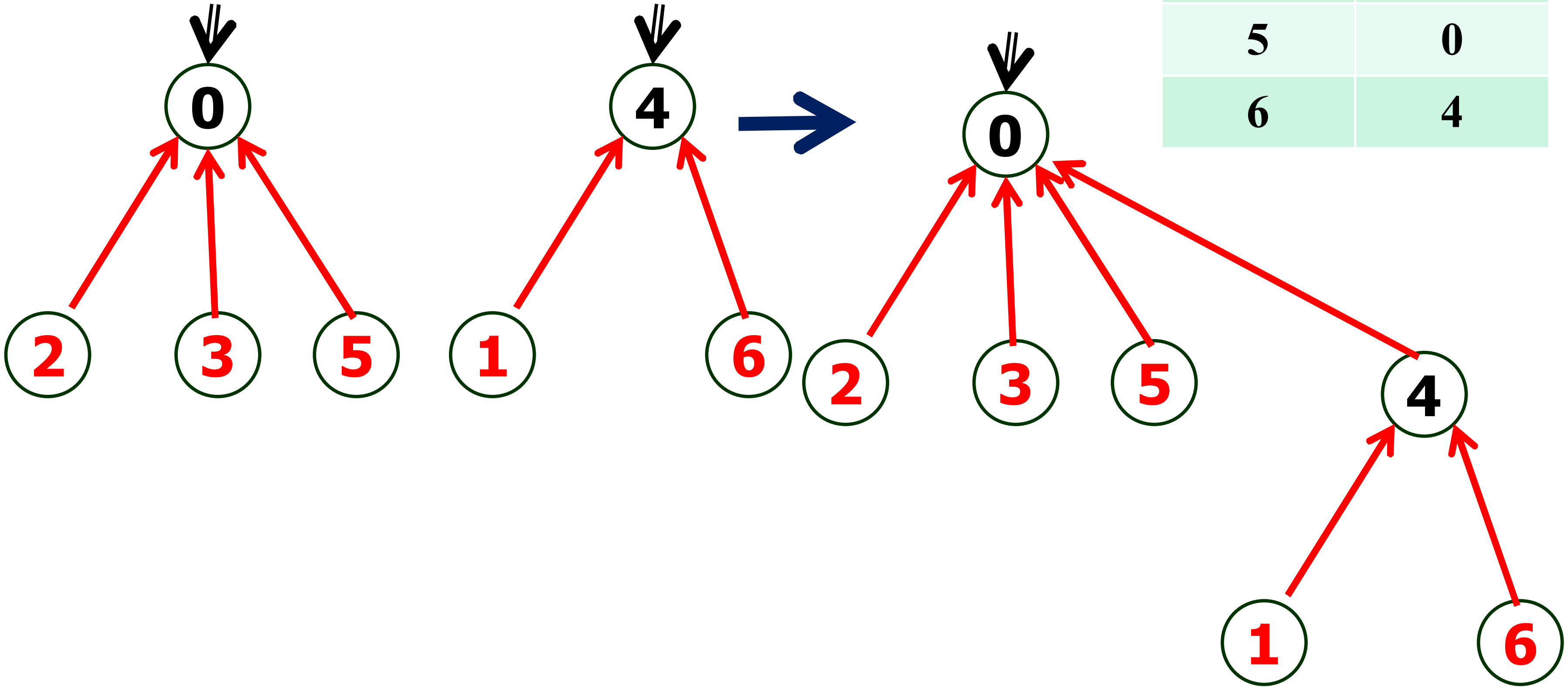
➢ $S_1=\{0,2,3,5\}$，$S_2=\{4,1,6\}$



data	Parent
0	-1
1	4
2	0
3	0
4	-1
5	0
6	4

{0,2,3,5}与{4,1,6}合并

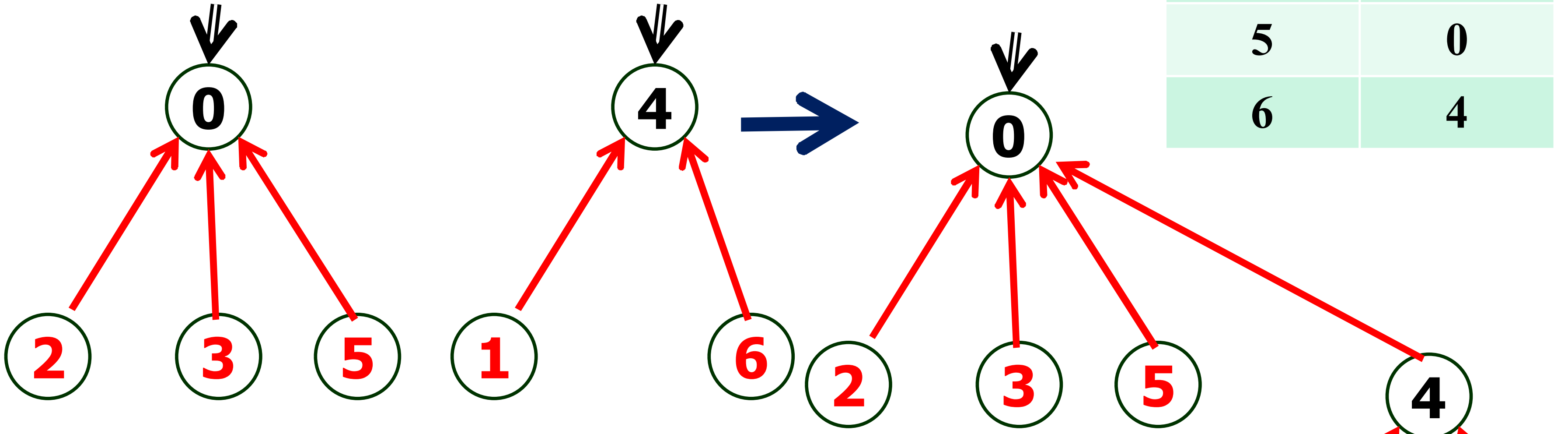
data	Parent
0	-1
1	4
2	0
3	0
4	-1
5	0
6	4



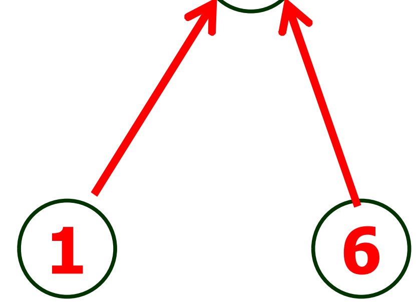
{0,2,3,5}与{4,1,6}合并

data	Parent
0	-1
1	4
2	0
3	0
4	0
5	0
6	4

union----O(1)



find---- d+1 次 parent lookup



The *depth* of the root is 0 and the depth of any other node is one plus the depth of its parent.[1] A *complete binary tree* is a binary tree in which all internal nodes have degree 2 and all leaves are at the same depth. The binary tree on the right in Figure 2.7 is complete.

Union-Find program

- 动态等价关系--一系列的Union和Find操作组成
- 等价类以树（in-tree）表示-- Union和Find操作主要通过访问双亲数组完成
- 对双亲数组的访问次数--衡量算法的时间复杂度
- 双亲数组访问--*lookup*，*assignment*--$O(1)$--*link operation*
- *Union*----$O(1)$
- *Find*---- $d+1$次 *parent lookup*
- ***link opeation***

We take the number of accesses to the parent array as the measure of work done; each access is either a *lookup* or an *assignment*, and we assume they each take time $O(1)$. (It will be clear that the total number of operations is proportional to the number of parent accesses.) Each makeSet or union does one parent assignment, and each find(i) does $d+1$ parent lookups, where d is the depth of node i in its tree. The parent assignments and lookups collectively will be called *link* operations.

*Union(u,t)*是将等价类*u*的根结点*u*作为等价类*t*的根结点*t*的孩子

1. *Union*(1, 2)
2. *Union*(2, 3)

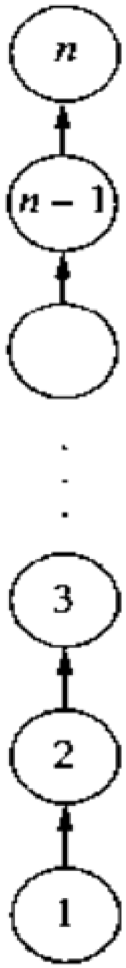
⋮

n − 1. *Union*(*n* − 1, *n*)

n. *Find*(1)

⋮

m. *Find*(1)



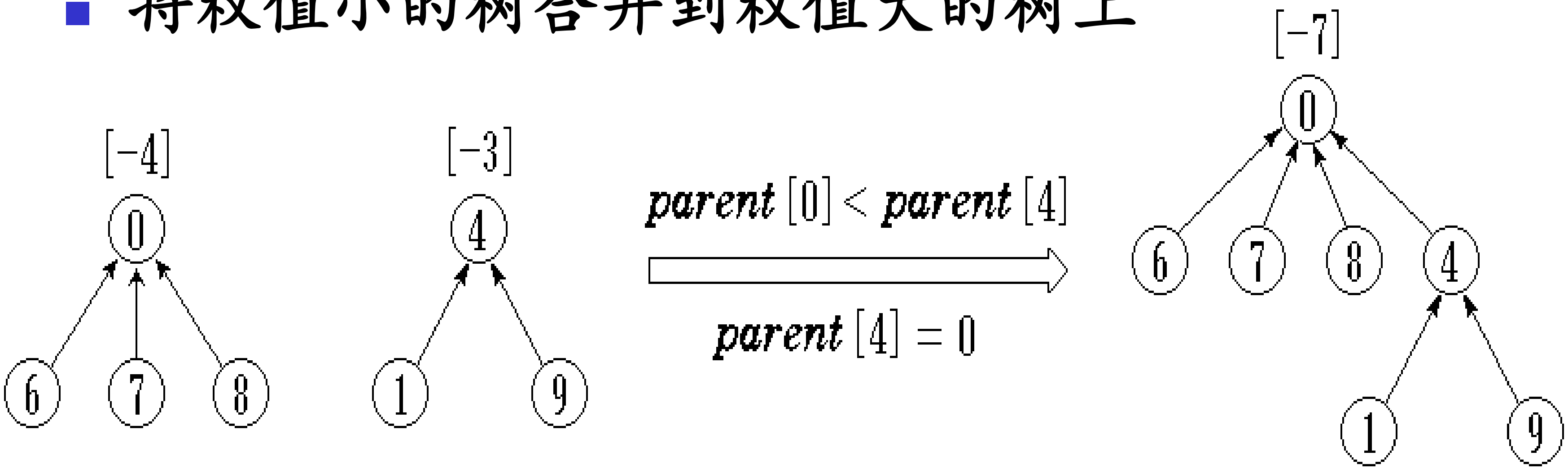
n-1+n (m-n+1) *link operations*

- n-1次的Union操作建成了一个单支树，树太高使得Find操作时间开销**大**
- 为提高Find操作的速度，应改进Union操作，使树建得**矮一些**

- 设树u的高度为h_1,设树t的高度为h_2，则2树合并后的高度=max$\{h_1+1, h_2\}$
- 设树u的结点数为x_1,设树t的结点数为x_2，则2树合并后，x_1个结点的Find操作*link operations*次数+1, x_2个结点的Find操作*link operations*次数不变

weighted Union

- 给树加权，权值的2种定义方式：
  - 树的结点数-----以**负数**形式存放于**根**结点
  - 树的高度-----以**负数**形式存放于**根**结点
- 将权值小的树合并到权值大的树上



weighted Union

- 再次考察下面程序段P:
  - Union(1,2)
  - Union(2,3)
  - …
  - Union(n-1,n)

- 采用wUnion?
  - wUnion(1,2)
  - wUnion(2,3)
  - …
  - wUnion(n-1,n)

wUnion(t,u)是将**权重小**的等价类合并到**权重大**的等价类,不能确定合并后的等价类对应的树的根结点为u。所以程序段P中的每一合并指令不能直接采用wUnion代替，而应为:

- s1=Find(t); s2=Find(u); wUnion(s1,s2)

weighted Union

1. *Union* (1, 2)

2. *Union* (2, 3)

⋮

$n-1$. *Union* ($n-1$, n)

n. *Find* (1)

⋮

m. *Find* (1)

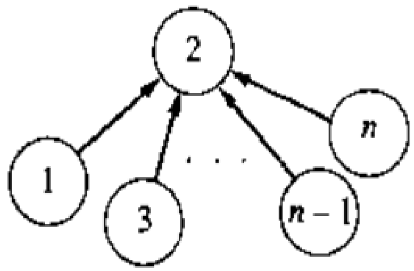
m-n+1个Find操作：2(m-n+1) link operations

n-1个wUnion操作:

n-1+2+2+3(n-3)=4n-6 link operations

所以共：2m+2n-4link operations

weighted union



(b) Tree for P', using weighted union

Lemma 6.6 If union(t, u) is implemented by wUnion—that is, so that the tree with root *u* is attached as a subtree of *t* if and only if the number of nodes in the tree with root *u* is smaller, and the tree with root *t* is attached as a subtree of *u* otherwise—then, after any sequence of union instructions, any tree that has *k* nodes will have height at most $\lfloor \lg k \rfloor$.

- 证明：采用数学归纳法。
- k=1时，只有一个结点的树的高度为0，$\lfloor lg\ 1 \rfloor$=0，引理成立。
- 假设0≤k<m时成立，考虑k=m的情况：一个树有m个结点，高度为h，见图6.21，他由树T_1和T_2通过Union操作建立。假设如图所示T_2的根u做T_1的根t的孩子。设T_1的结点数和高为k_1和h_1，T_2的结点数和高为k_2和h_2。
- 由推理假设$h_1 \le \lfloor lg\ k_1 \rfloor$，$h_2 \le \lfloor lg\ k_2 \rfloor$
- $h=\max(h_1, h_2+1)$, $h_1 \le \lfloor lg\ k_1 \rfloor < \lfloor lg\ m \rfloor$,
- $k_2 \le m/2$，$h_2 \le \lfloor \lg k_2 \rfloor \le \lfloor \lg m \rfloor - 1$
- $h \le \lfloor \lg m \rfloor$

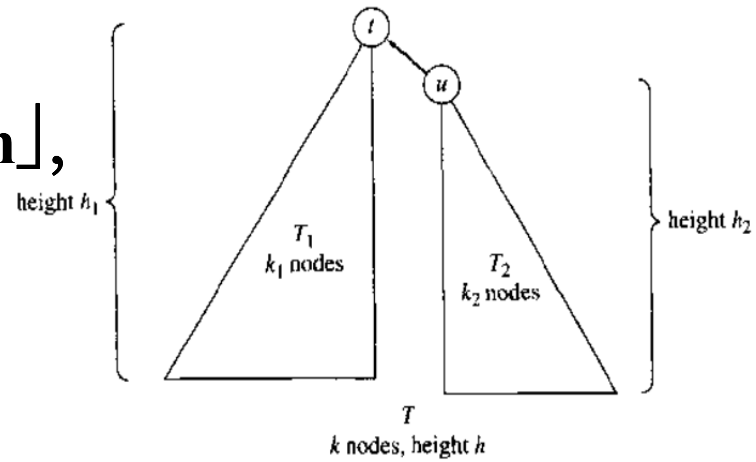
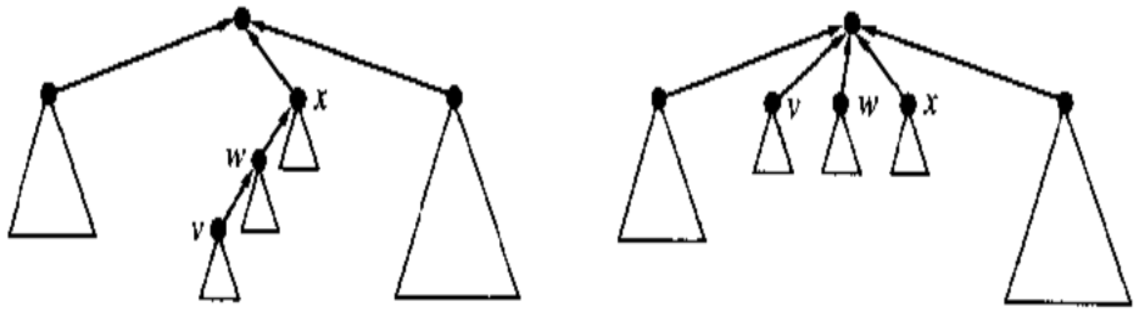


Figure 6.21 An example for the proof of Lemma 6.6

路径压缩—进一步优化



Before cFind (v)　　　　After cFind(v)

```
int cFind(int v)
    int root;
    int oldParent = parent[v];
    if (oldParent == -1)  // v is a root
        root = v;
    else
        root = cFind(oldParent);
        if (oldParent ≠ root)  // This if statement
            parent[v] = root);  // does path compression.
    return root;
```