并查集 P264-265

假设集合S由若干个元素组成，可以按照某一规则把集合S分成若干个互不相交的子集合，称之为**等价分类**。

◆ 例如，集合S={1,2,3,4,5,6,7,8,9,10}，可以分成如下三个不相交的子集合：

S1={1,2,4,7}

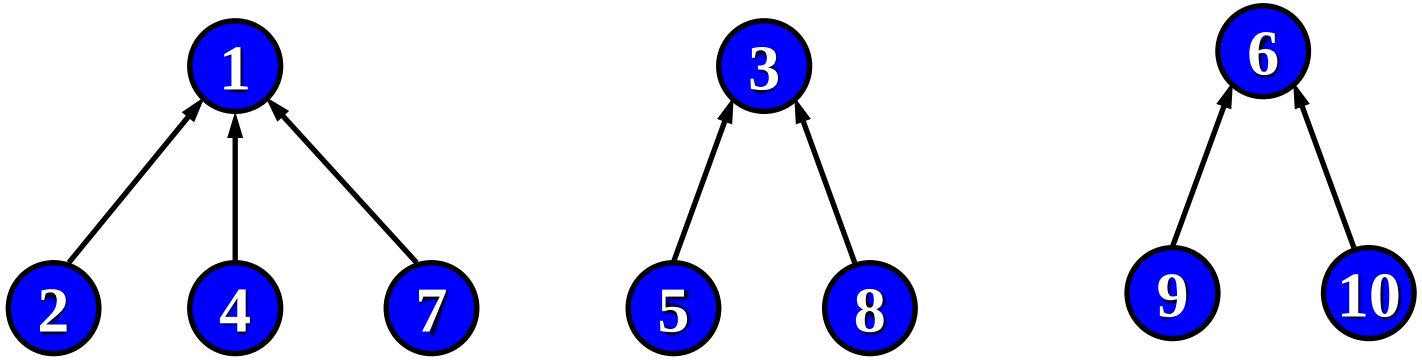
S2={3,5,8}

S3={6,9,10}

同样，也可以将两个集合合并成一个新的集合：

S1∪S2={1，2，3，4，5，7，8}

用一棵树表示一个集合，树中的一个结点表示集合中的一个元素，树结构采用双亲表示法。



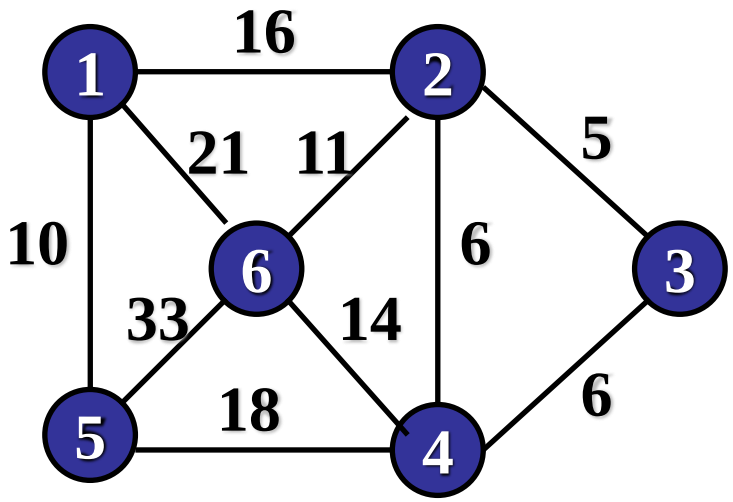
S1={1,2,4,7}　　S2={3,5,8}　　S3={6,9,10}

- 求集合的并集：把一个集合的树根结点作为另一个集合的树根结点的孩子结点。
- 查找某个元素所在的集合，可以沿着该元素的双亲域向上查，当查到某个元素的双亲域值为0时，该元素就是所查元素所属的树根结点。

并查集 (Union-Find Sets)

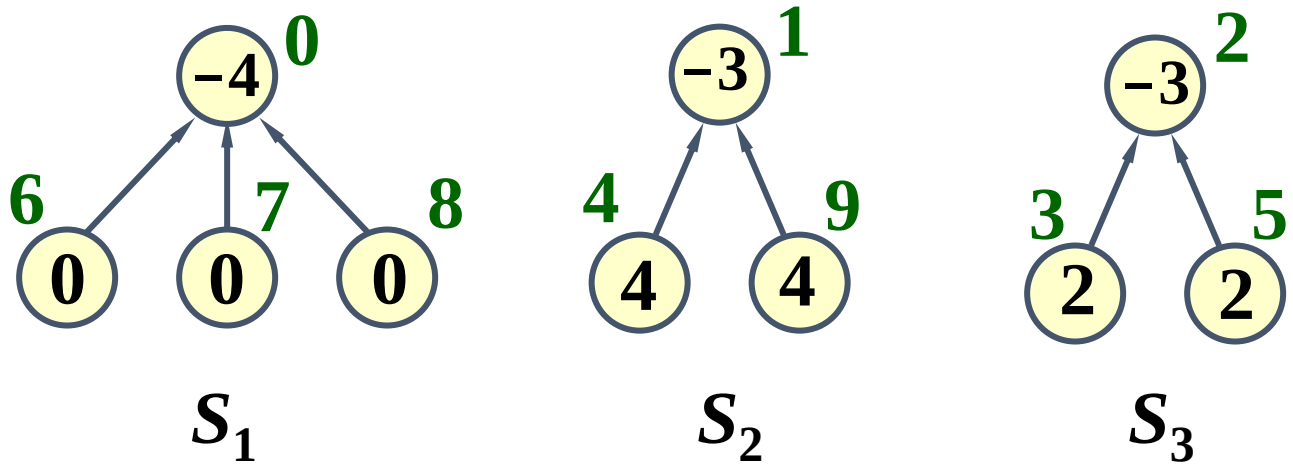
- 并查集支持以下三种操作：
  - **Union** (Root1, Root2) //并操作，把子集合Root2并入Root1
  - **Find** (x) //搜索单元素x所在的集合，并返回该集合的名字
  - **UFSets** (s) //构造函数，将并查集中s个元素初始化为只有一个单元素的子集合。

- 对于并查集来说，**每个集合用一棵树表示**。集合元素的编号从1到 n。其中 n 是最大元素个数。



设 $S1=\{0, 6, 7, 8\}$, $S2=\{1, 4, 9\}$, $S3=\{2, 3, 5\}$

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



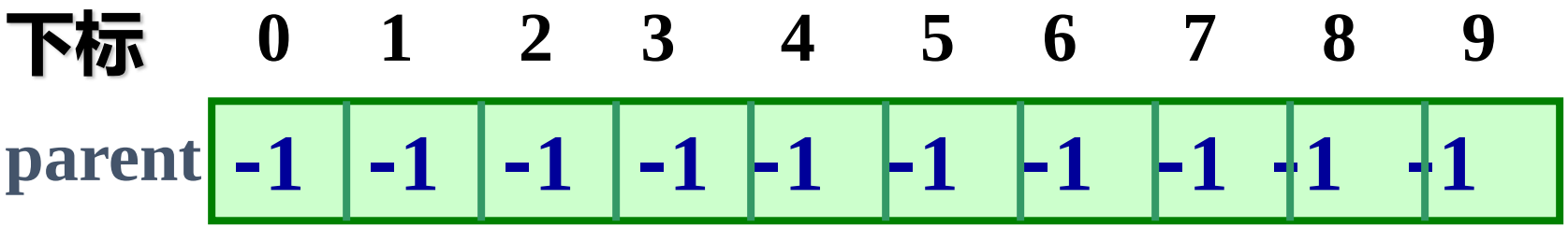
S_1 S_2 S_3

并查集的类定义

```
const int DefaultSize = 10;
class UFSets {  //集合中的各个子集合互不相交
private:
    int *parent;              //集合元素数组(双亲表示)
    int size;                 //集合元素的数目
public:
    UFSets (int sz = DefaultSize);          //构造函数
   ~UFSets() { delete []parent; }    //析构函数
    UFSets& operator = (UFSets& R);   //集合赋值
    void Union (int Root1, int Root2); //子集合并
    int Find (int x);                          //查找x的根
  }
```

构造函数

```
UFSets::UFSets (int sz) {
//构造函数：SZ 是集合元素个数，双亲数组的范围
//为parent[0]～parent[size-1]。
    size = sz;                          //集合元素个数
    parent = new int[size];  //创建双亲数组
    for (int i = 0; i < size; i++) parent[i] = -1;
                                        //每个自成单元素集合
};
```

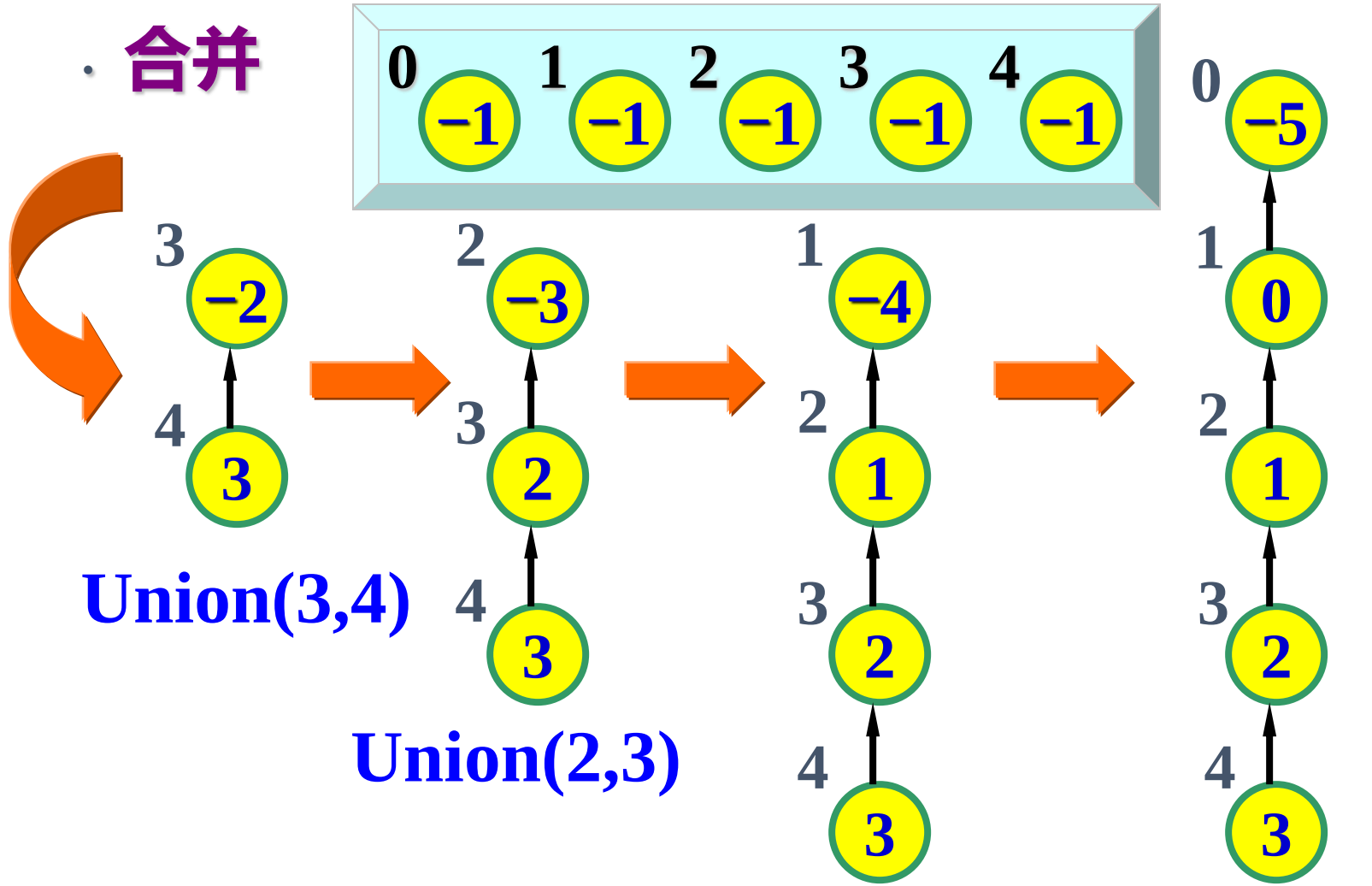


合并操作

```
void UFSets::Union (int Root1, int Root2)  {
   //求两个不相交集合Root1与Root2的并
     parent[Root1] += parent[Root2];
     parent[Root2] = Root1;
                  //将Root2连接到Root1下面
};
```

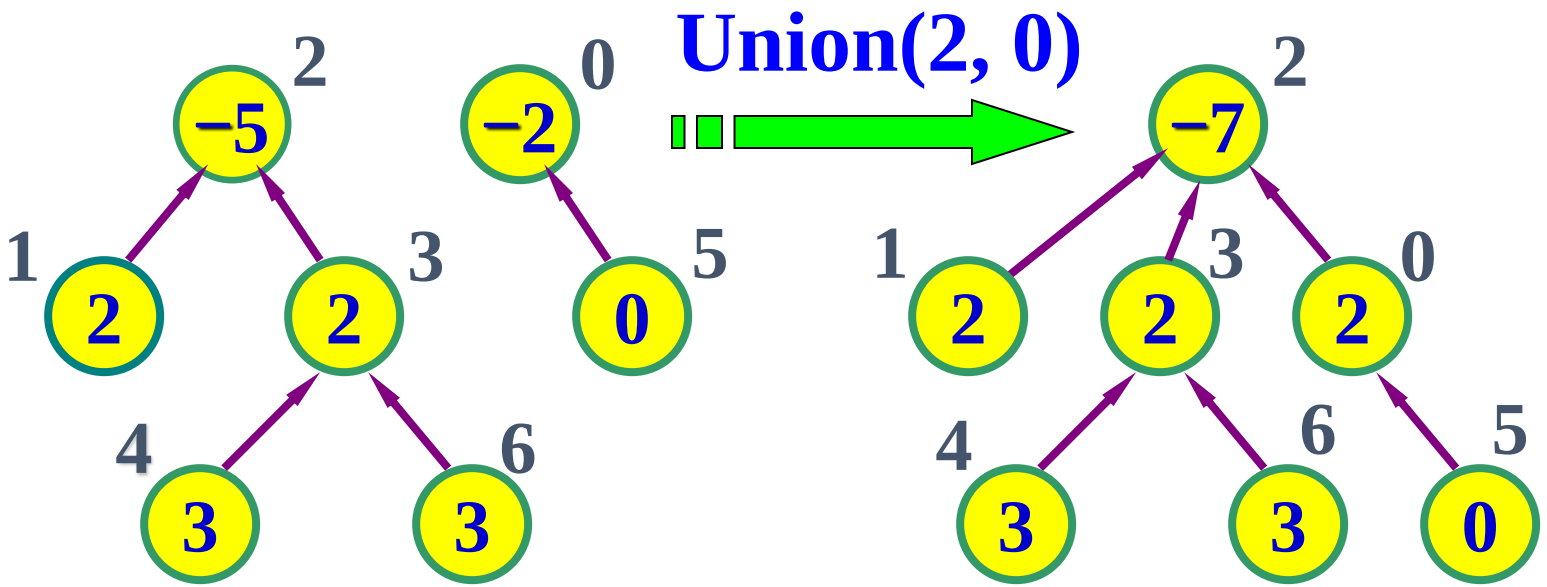
- 执行一次Union操作所需时间是 O(1)，*n*-1次Union操作所需时间是O(*n*)。
- 形成单枝树：性能较低!

· 合并


0 1 2 3 4
−1 −1 −1 −1 −1
3 4
−2 3
Union(3,4)
2 3 4
−3 2 3
Union(2,3)
1 2 3 4
−4 1 2 3
Union(1,2)
0 1 2 3 4
−5 0 1 2 3
Union(0,1)

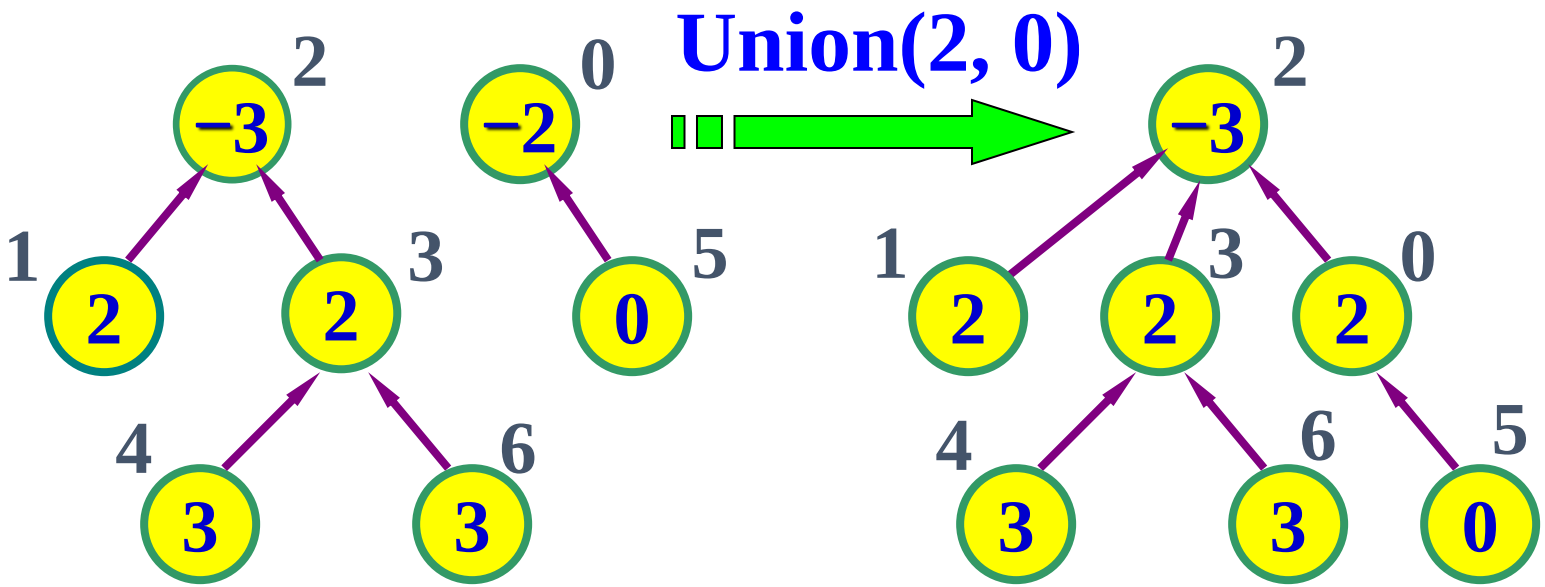
合并操作的改进

- 按树的结点个数合并：重量规则
  - **结点个数多的树的根结点作根**



合并操作的改进（续）

- 按树的高度合并
  - 高度高的树的根结点作根



改进的合并操作：重量规则

```
void UFSets :: WeightedUnion (int Root1, int Root2) {
//按Union的加权规则改进的算法
    int temp = parent[Root1] + parent[Root2];
    if (parent[Root2] < parent[Root1]) {  注意是负数！
        parent[Root1] = Root2;     //Root2中结点数多
        parent[Root2] = temp;      //Root1指向Root2
    }
    else {
        parent[Root2] = Root1;     //Root1中结点数多
        parent[Root1] = temp;      //Root2指向Root1
    }
}
```

查找算法

```
int UFSets::Find (int x) {
//函数搜索并返回包含元素x的树的根。
    if (parent[x] < 0) return x;   //根的parent[]值小于0
    else return (Find (parent[x]));
};
```