Disjoint Set(Union/Find)

Equivalence Relations

复习等价关系（离散数学）

- Symmetric
- Reflexive
- Transitive

a and b are equivalent

R is an equivalence relation, and $(a, b) \in R$

Notation: $a \sim b$

the equivalence class of x

The set of all elements that are related to an element x of A

Notation: $[x]_R$ $[x]$

a representative of the equivalence class $[x]_R$**:** $b \in [x]_R$

The Dynamic Equivalence Problem

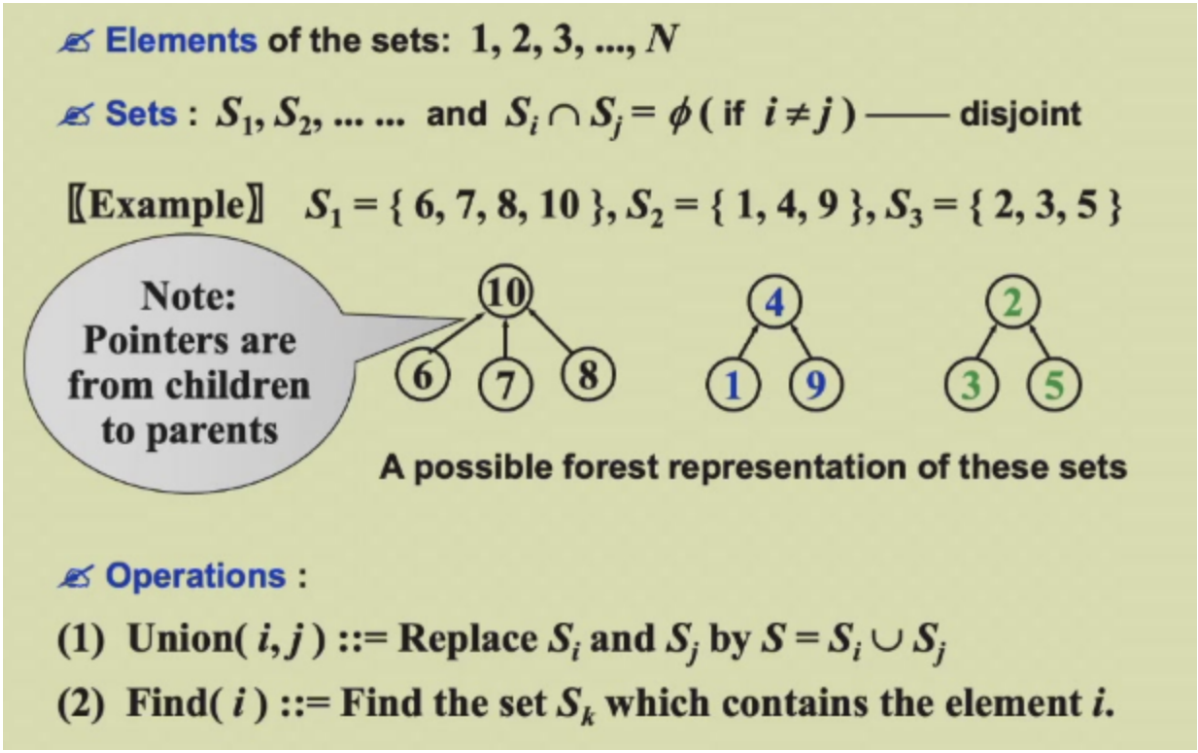
§ 2 The Dynamic Equivalence Problem

Given an equivalence relation ~, decide for any *a* and *b* if *a* ~ *b*.

〖Example〗 Given $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ and 9 relations: 12≡4, 3≡1, 6≡10, 8≡9, 7≡4, 6≡8, 3≡5, 2≡11, 11≡12.

The equivalence classes are { 2, 4, 7, 11, 12 }, { 1, 3, 5 }, { 6, 8, 9, 10 }

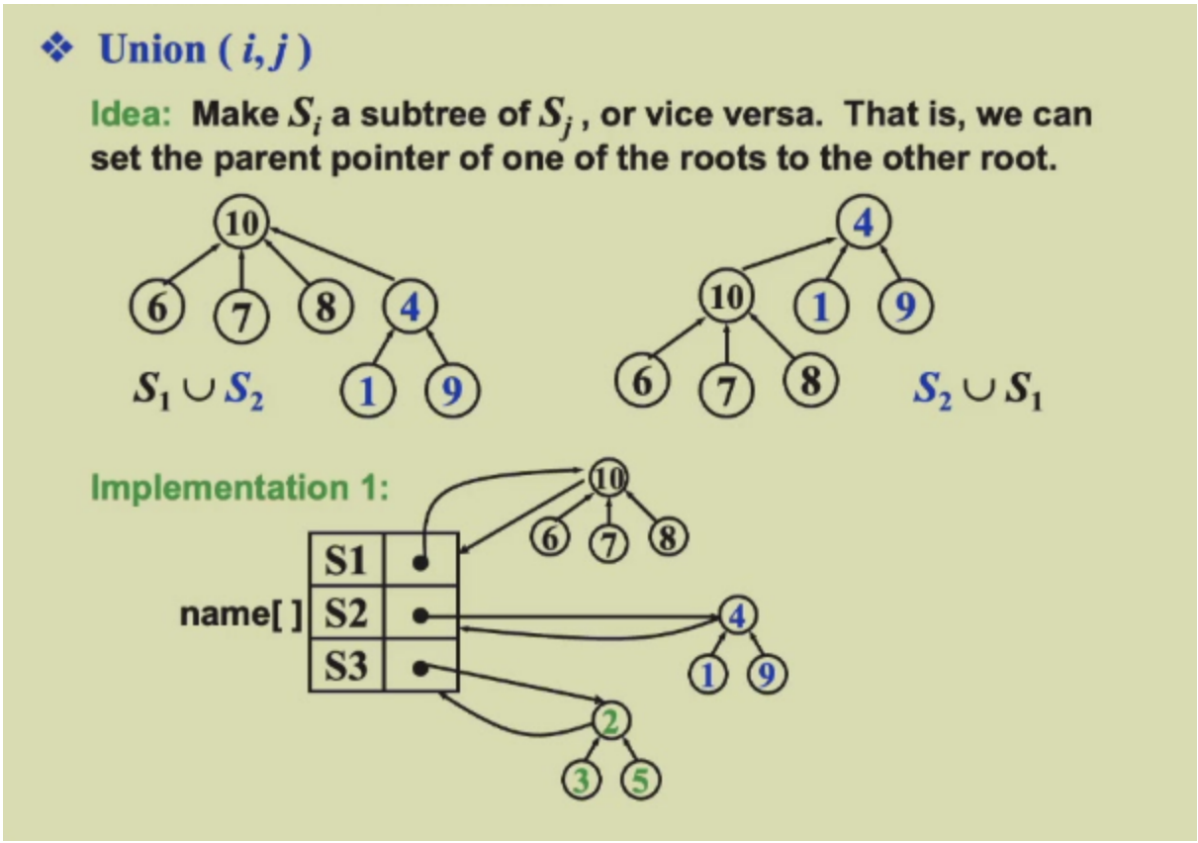
```
Algorithm:
{  /* step 1: read the relations in */
   Initialize N disjoint sets;
   while ( read in a ~ b ) {
      if ( ! (Find(a) == Find(b)) )
           Union the two sets;
   } /* end-while */
   /* step 2: decide if a ~ b */
   while ( read in a and b )
      if ( Find(a) == Find(b) )   output( true );
      else   output( false );
}
```



> 注意pointer的方向是反向的

Basic Data Structure

Union

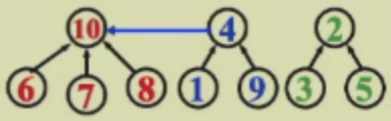


> 链表指针指向父结点

Implementation 2: S [element] = the element's parent.

Note: S [root] = 0 and set name = root index.

〖Example〗 The array representation of the three sets is



S	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	4	0	2	10	2	10	10	10	4	0

$(S_1 \cup S_2 \Rightarrow S_1) \Leftrightarrow$ S [4] = 10

```
void SetUnion ( DisjSet S,
                SetType Rt1,
                SetType Rt2 )
{   S [ Rt2 ] = Rt1 ;   }
```

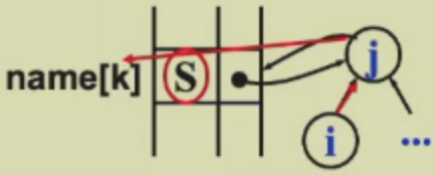
❖ Find (*i*)

> index表示元素，数组内容表示parent's index

Find

❖ Find (*i*)

Implementation 1:



find (i) = 'S'

Implementation 2:

```
SetType Find ( ElementType X,
               DisjSet S )
{  for ( ; S[X] > 0; X = S[X] )  ;
   return X ;
}
```

Analysis

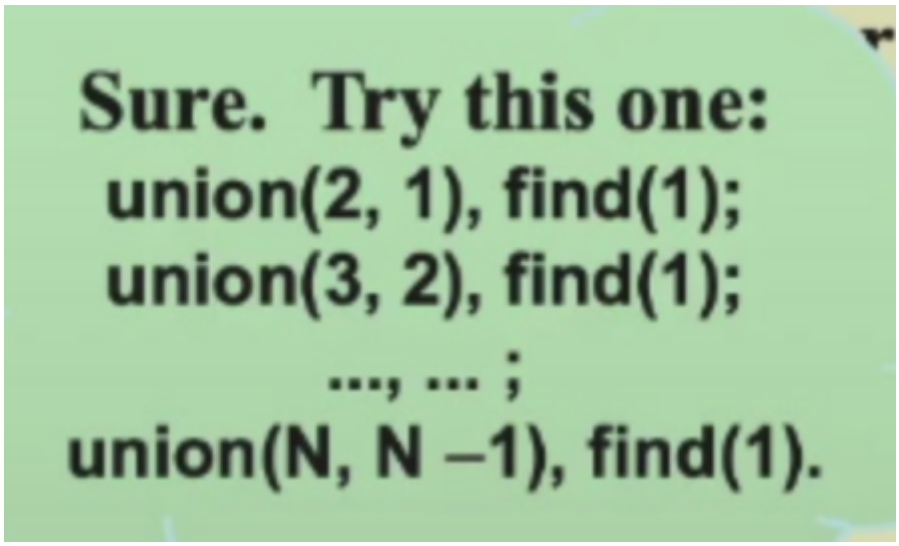
❖ Analysis

Practically speaking, union and find are always paired. Thus we consider the performance of a sequence of union-find operations.

〖Example〗 Given $S = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \}$ and 9 relations: 12≡4, 3≡1, 6≡10, 8≡9, 7≡4, 6≡8, 3≡5, 2≡11, 11≡12. We have 3 equivalence classes { 2, 4, 7, 11, 12 }, { 1, 3, 5 }, and { 6, 8, 9, 10 }

Algorithm using union-find operations

```
{ Initialize S_i = { i } for i = 1, ..., 12 ;
  for ( k = 1; k <= 9; k++ ) { /* for each pair i ≡ j */
    if ( Find( i ) != Find( j ) )
       SetUnion( Find( i ), Find( j ) );
  }
}
```



会导致树的高度太高，时间复杂度太大

Smart Union Algorithm

Union by Size

❖ **Union-by-Size** -- Always change the smaller tree

S [Root] = – size; /* initialized to be –1 */

【Lemma】Let T be a tree created by union-by-size with N nodes, then $height(T) \le \lfloor \log_2 N \rfloor + 1$

Proof: By induction. (Each element can have its set name changed at most $\log_2 N$ times.)

Time complexity of N Union and M Find operations is now $O(N + M \log_2 N)$.

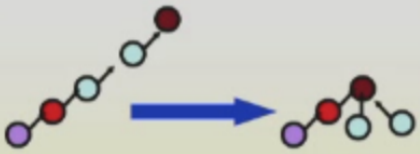
Union by Height

Please read Figure 8.13 on p.273 for detailed implementation.

- 将矮树接在高树上
-

Path Compression

```
SetType  Find ( ElementType  X, DisjSet  S )
{
   if ( S[ X ] <= 0 )   return  X;
   else   return  S[ X ] = Find( S[ X ], S );
}
```



每次经过一个节点，都把这个节点直接指向父结点

```
SetType  Find ( ElementType  X, DisjSet  S )
{  ElementType  root,  trail,  lead;
   for ( root = X; S[ root ] > 0; root = S[ root ] )
      ;  /* find the root */
   for ( trail = X; trail != root; trail = lead ) {
      lead = S[ trail ] ;
      S[ trail ] = root ;
   } /* collapsing */
   return  root ;
}
```

> 可以用union-by-size
>
> 这样会改变树的height，就叫做union-by-rank

Summary

1. Basic Union and Find
2. Union by Size
3. Union by Height
4. Union by Size with Path Compression
5. Union by Rank

Worst Case for Union-by-Rank

§ 6 Worst Case for Union-by-Rank and Path Compression

【Lemma (Tarjan)】 Let $T(M, N)$ be the maximum time required to process an intermixed sequence of $M \geq N$ finds and $N-1$ unions. Then:

$$k_1 M\,\alpha(M,N) \leq T(M,N) \leq k_2 M\,\alpha(M,N)$$

for some positive constants k_1 and k_2.

Ackermann's Function and $\alpha(M,N)$

$$A(i,j)=\begin{cases} 2^j & i=1 \text{ and } j\geq 1 \\ A(i-1,2) & i\geq 2 \text{ and } j=1 \\ A(i-1,A(i,j-1)) & i\geq 2 \text{ and } j\geq 2 \end{cases} \qquad A(2,4)=2^{2^{2^{2^2}}}=2^{65536}$$

http://mathworld.wolfram.com/AckermannFunction.html

$$\alpha(M,N)=\min\{\, i\geq 1 \mid A(i,\lfloor M/N\rfloor) > \log N \,\} \leq O(\log^* N) \leq 4$$

$\log^* N$ (inverse Ackermann function)
= # of times the logarithm is applied to N until the result ≤ 1.