1 Equivalence Relations

【Definition】 A *relation R* is defined on a set *S* if for every pair of elements (a, b), $a, b \in S$, $a\ R\ b$ is either true or false. If $a\ R\ b$ is true, then we say that *a* is related to *b*.

【Definition】 A relation, ~, over a set, *S*, is said to be an *equivalence relation* over *S* iff it is symmetric, reflexive, and transitive over *S*.

【Definition】 Two members *x* and *y* of a set *S* are said to be in the same *equivalence class* iff $x \sim y$.

- symmetric: 可逆性
- reflexive: 自反性
- Transitive: 传递性

2 The Dynamic Equivalence Problem

【Example】 Given $S = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \}$ and 9 relations: $12\equiv4$, $3\equiv1$, $6\equiv10$, $8\equiv9$, $7\equiv4$, $6\equiv8$, $3\equiv5$, $2\equiv11$, $11\equiv12$.

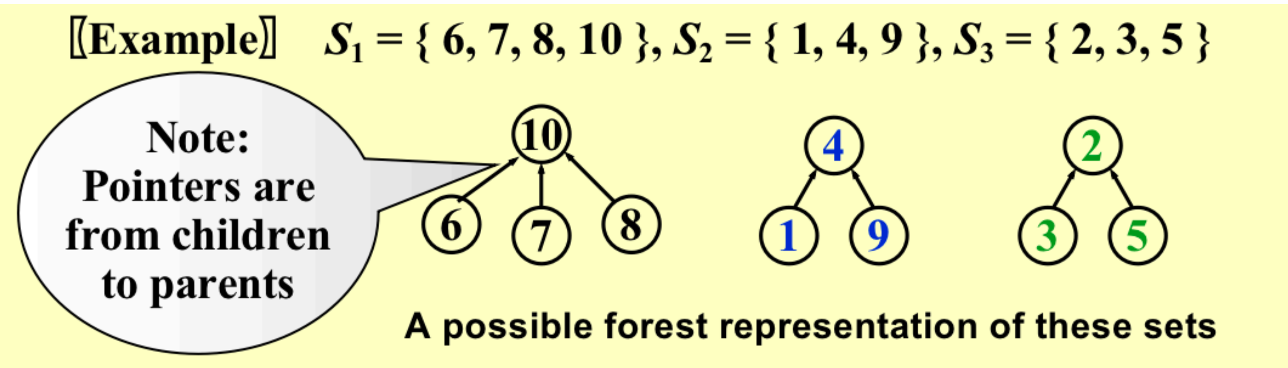
The equivalence classes are $\{ 2, 4, 7, 11, 12 \}$, $\{ 1, 3, 5 \}$, $\{ 6, 8, 9, 10 \}$

```
Algorithm:   (Union / Find)
{  /* 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 );
}
```

Dynamic (on-line)

- 可以把相同类的元素做并运算，把12的label给4
- `Find(a)` 即返回a的label
- `Union` 使label一致

- Elements of the sets：1，2，3，…，N
- Sets: S_1，$S_2 \ldots$ *and* $S_i \cap S_j = \phi \, (i \neq j)$ ——disjoint



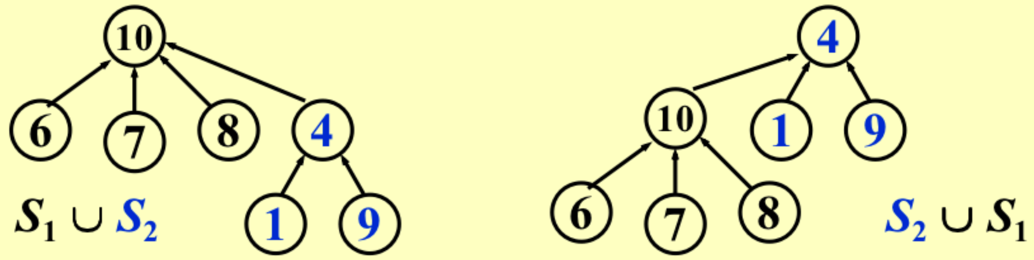
- Operations :
  - (1) Union(i, j) ::= Replace Si and Sj by $S = S_i \cup S_j$
  - (2) Find(i) ::= Find the set Sk which contains the element i.

3 Basic Data Structure

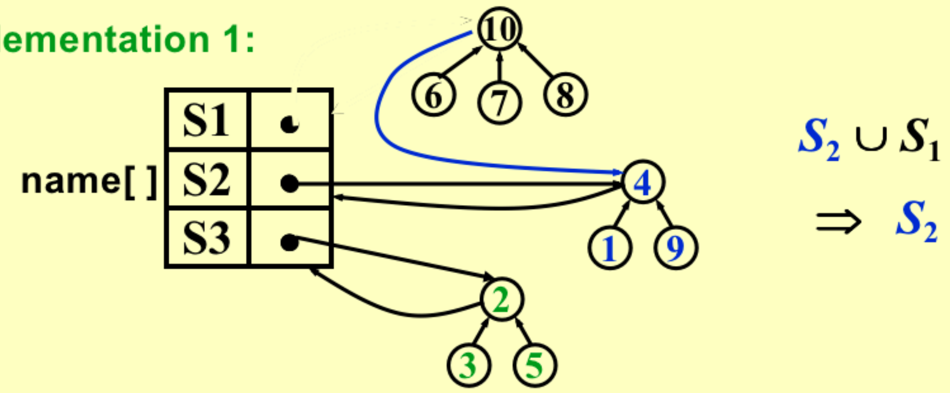
3.1 Union

❖ **Union (i, j)**

Idea: Make S_i a subtree of S_j , or vice versa. That is, we can set the parent pointer of one of the roots to the other root.



Implementation 1:

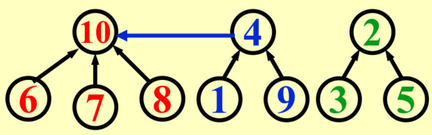


Implementation2:

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

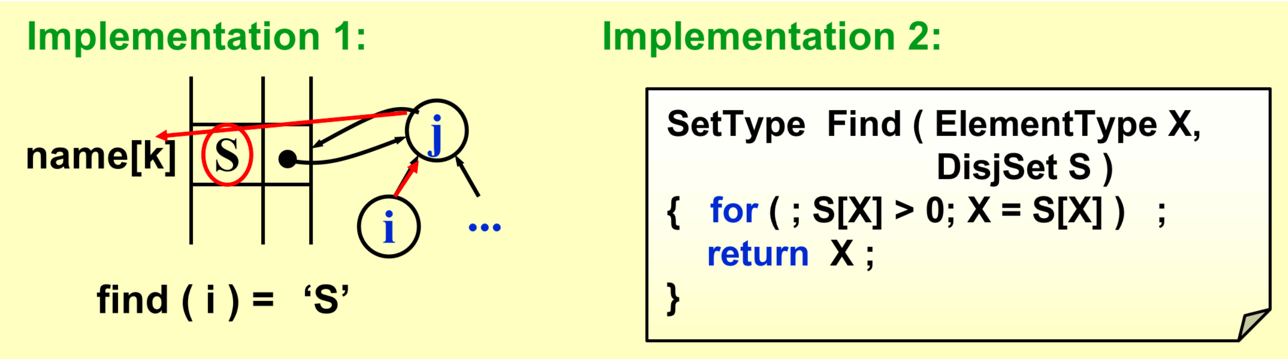


S	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	4	0	2	10	2	10	10	10	4	0

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

```
void  SetUnion ( DisjSet S, SetType Rt1, SetType Rt2 )
{    S [ Rt2 ] = Rt1 ;     }
// 直接改数组内容即可
```

3.2 Find




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

- find就是去找该元素的label
- 第一个就是不停地去指针
- 第二个就是不断访问数组内容，即访问自己的父节点，直到访问到根节点，root=0

3.3 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: $12\equiv4$, $3\equiv1$, $6\equiv10$, $8\equiv9$, $7\equiv4$, $6\equiv8$, $3\equiv5$, $2\equiv11$, $11\equiv12$. 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 ) );
  }
}
```

- 但是最坏时间复杂度会到达$\Theta(N^2)$

4 Smart Union Algorithms

4.1 Union-by-Size-- change the smaller

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 $O(N + M \log_2 N)$.

```
void UnionbySize(Elementtype root1, Elementtype root2, DisjSet S){
  if( S[root1] > S[root2]){
    S[root2] += S[root1];
    S[root1] = root2;
    return;
  }
  else{
    if( S[root1] == S[root2]){
      S[root1] += S[root2];
      S[root2] = root1;
      return;
    }
    S[root1] += S[root2];
    S[root2] = root1;
    return;
  }
}
```

- 就是把最小的那个树归类到主要类别中
- 时间复杂度要记

$$T(N) = O(N + M \log_2 N) \tag{1}$$

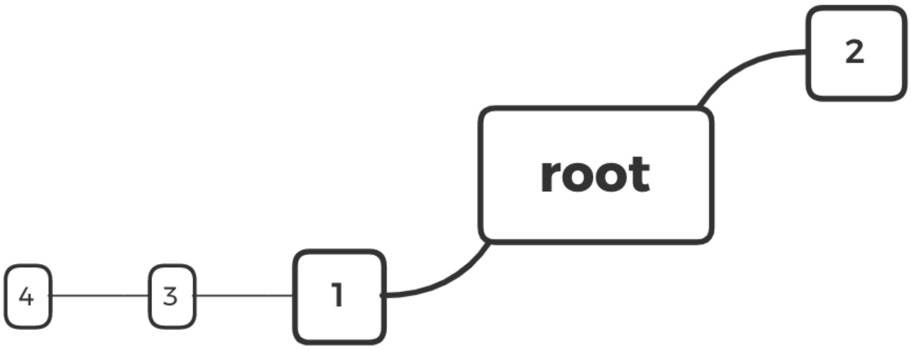
4.2 Union-by-Height(Rank)-- change the shallow

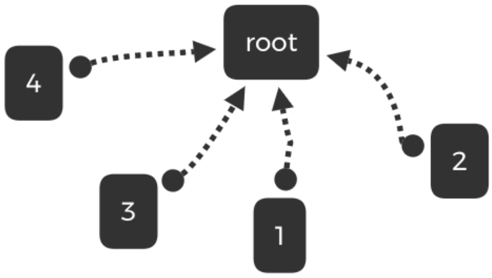
```
// Assume Root1 and Root2 are roots
// union is a C keyword, so this routine is named Setunion

void SetUnion(DisjSet S, SetType Root1, SetType Root2){
  // S[root] = -height;
  if( S[Root2] < S[Root1] ) // Root2 is deeper set
    S[Root1] = Root2; // Make Root2 as Union1's new root
  else{
    if( S[Root1] == S[Root2] )  S[Root1]--; // Same height
    S[Root2] = Root1; // Always choose Root1 as final union
  }
}
```

$$T(N) = O(N + M \log_2 N) \tag{2}$$

5 Path Compression

```
SetType  Find ( ElementType  X, DisjSet  S )
{
    if ( S[ X ] <= 0 )     return  X;
    else return  S[ X ] = Find( S[ X ], S );
}
// 这个递归有点绕
// 可以这么理解，**最后所有的节点都指向根节点**，因此都是S[X] = Find(S[X],S)
// 记住函数目的就是找根节点，希望所有的自己都指向根节点，S[X] = Find
// 并且某个节点不是，那么就向上访问父节点是不是, S[X] = Find(S[X],S)
```





```c
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 ) { // trail是当前节点
       lead = S[ trail ] ;   // 记录父节点
       S[ trail ] = root ;   // 将自己指向根节点
    } /* collapsing */
    return  root ;
}
```

- Note: Not compatible with union-by-height since it changes the heights. Just take “height” as an estimated 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.

$$T(N) = O(N + M \log_2^* N) \tag{3}$$

$$\begin{aligned}\log_2^* 2^{65536} = 5 \\ \log\log\log\log\log 2^{65536} = 1\end{aligned} \tag{4}$$

- 就是取多少次对数能取到1