# Chapter 7.1

## The Disjoint Set ADT

# Equivalence Class

1) **Definition of Equivalence Class:**

Suppose we have a set U={1,2,…,n} of n elements and a set R={($i_1$,$j_1$), ($i_2$,$j_2$)…… ($i_r$,$j_r$)}of r relations. The relation R is an equivalence relation iff the following conditions are true(symbol'≡' represent the equivalence relation on sets, x,y,z are elements in set ) :

- Reflexive x ≡x.
- Symmetric x ≡y,y ≡x
- Transitive x ≡y and y ≡z,then x ≡z

# Equivalence Class

例如：

判别 3 个数 a, b, c 能否构成三角形的三条边?

能构成三角形的等价类：

{ (3,4,5), (4,5,6), (5,6,7), …..}

不能构成三角形的等价类：

{ (1,2,3), (2,3,5), ….. }

# Equivalence Class

2) **Example:**

set s= {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

pairs of equivalence:

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

Initial: {0} , {1} , {2} , {3} , {4} , {5} , {6} , {7} , {8} , {9}, {10} , {11}

$0 \equiv 4$ {0 , 4} , {1} , {2} , {3} , {5} , {6} , {7} , {8} , {9}, {10} , {11}

$3 \equiv 1$ {0 , 4} , {1 , 3} , {2} , {5} , {6} , {7} , {8} , {9} , {10}, {11}

# Equivalence Class

6 ≡10 {0 , 4} , {1 , 3} , {2} , {5} , {6 , 10} , {7} , {8}, {9} , {11}

8 ≡9 {0 , 4} , {1 , 3} , {2} , {5} , {6 , 10} , {7} , {8, 9} , {11}

7 ≡ 4 {0 , 4 , 7} , {1 , 3} , {2} , {5} , {6 , 10} , {8 , 9}, {11}

6 ≡ 8 {0 , 4 , 7} , {1 , 3} , {2} , {5} , {6 , 8 , 9 , 10} , {11}

3 ≡ 5 {0 , 4 , 7} , {1 , 3 , 5} , {2} , {6 , 8 , 9 , 10} , {11}

2 ≡ 11 {0 , 4 , 7} , {1 , 3 , 5} , {2 , 11} , {6 , 8 , 9 , 10}

11 ≡ 0 {0 , 4 , 7 , 2 , 11} , {1 , 3 , 5} , {6 , 8 , 9 , 10}

# Equivalence Class

3) Online equivalence class operation

- Combine(a,b) : combine the equivalence classes that contains elements a and b into a single class
- Find(e) : determine the class that currently contains element e.

Combine(a,b) is equivalent to

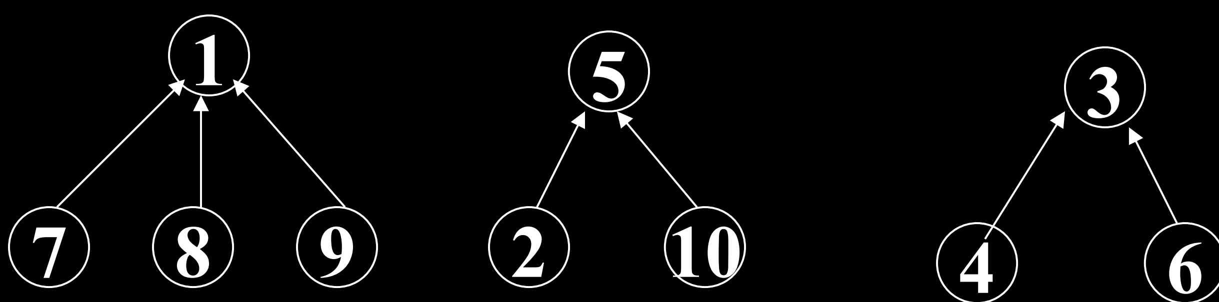
i=Find(a); j=Find(b); if(i!=j) Union(i,j);

# Equivalence Class

## 4) Tree Representation(Union-Find sets)

**Example:**

$S_1=\{1,7,8,9\}$, $S_2=\{5,2,10\}$, $S_3=\{3,4,6\}$,they all belong to $S=\{1,2,3,……,10\}$



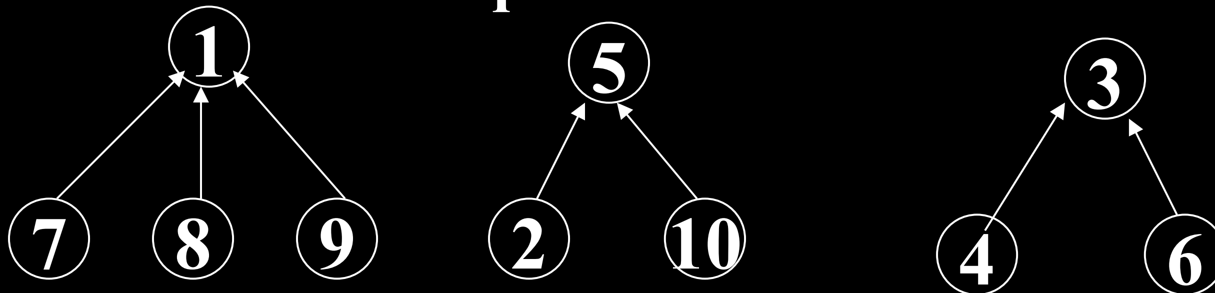

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| parent | | 0 | 5 | 0 | 3 | 0 | 3 | 1 | 1 | 1 | 5 |

# Equivalence Class

**simple tree solution to union-find problem**

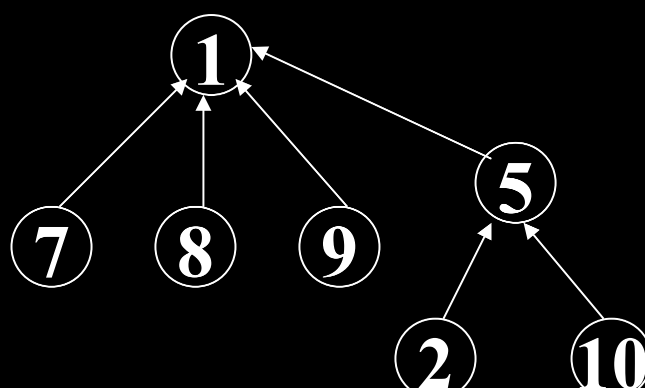
```
  void Initialize(int n)
  {  parent=new int[n+1];
     for(int e=1;e<=n;e++)
       parent[e]=0;
    }

int Find(int e)
 {  while(parent[e])
          e=parent[e];
     return e;
 }
  void Union(int i, int j)
 { parent[j]=i;
 }
```

# Equivalence Class




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| parent | | 0 | 5 | 0 | 3 | 1~~0~~ | 3 | 1 | 1 | 1 | 5 |

Union ( 1,5 )



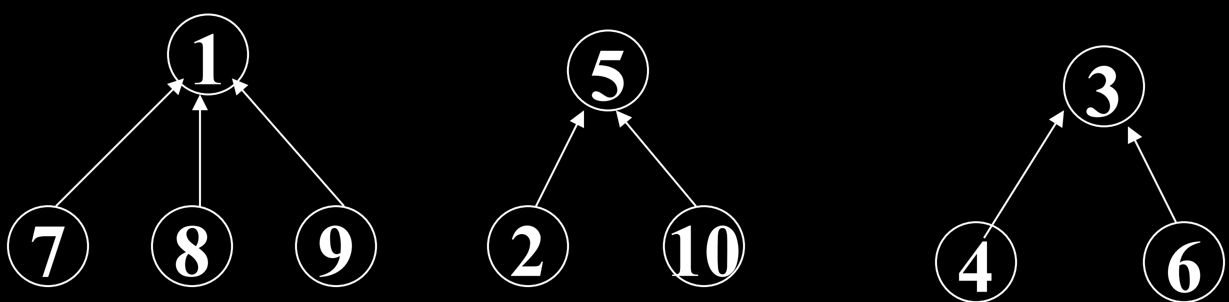
# Equivalence Class

**Java**

```
 public class DisjSets
{  public DisjSets( int numElements )
   public void union( int root1, int root2 )
   public int find( int x )
   private int [ ] s;
}

public DisjSets( int numElements )
{   s = new int [ numElements ];
    for( int i = 0; i < s.length; i++ )
        s[ i ] = -1; // 一个根结点
}
```

# Equivalence Class

```
 public void union( int root1, int root2 )
{   s[ root2 ] = root1;
}
 public int find( int x )
{    if( s[x ] < 0 )
           return x;
     else
           return find( s[ x ] );
}
```




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S: | | -1 | 5 | -1 | 3 | -1 | 3 | 1 | 1 | 1 | 5 |

# Equivalence Class

5) Performance Evaluation

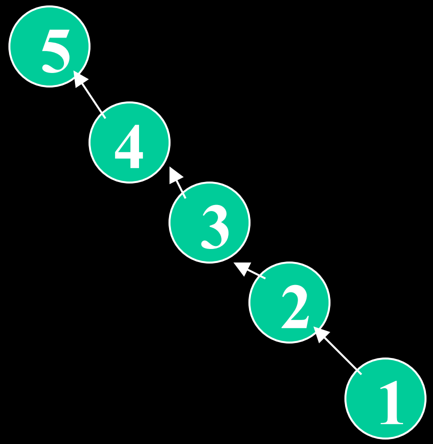
Time complexity: Find-- $O(h)$,

Union-- $\theta(1)$

Assume that u times unions and f times finds are to be performed, $f>u$,

in the worst case a tree with m elements can have a height of m:

Union(2,1),Union(3,2),Union(4,3),Union(5,4)…



# Equivalence Class

①improve Union

two rules:

- Weight rule: if the number of nodes in tree i is less than the number in tree j, then make j the parent of i; otherwise,make i the parent of j.

- Height rule: if the height of tree i is less than that of tree j, then make j the parent of i; otherwise,make i the parent of j.

# Equivalence Class

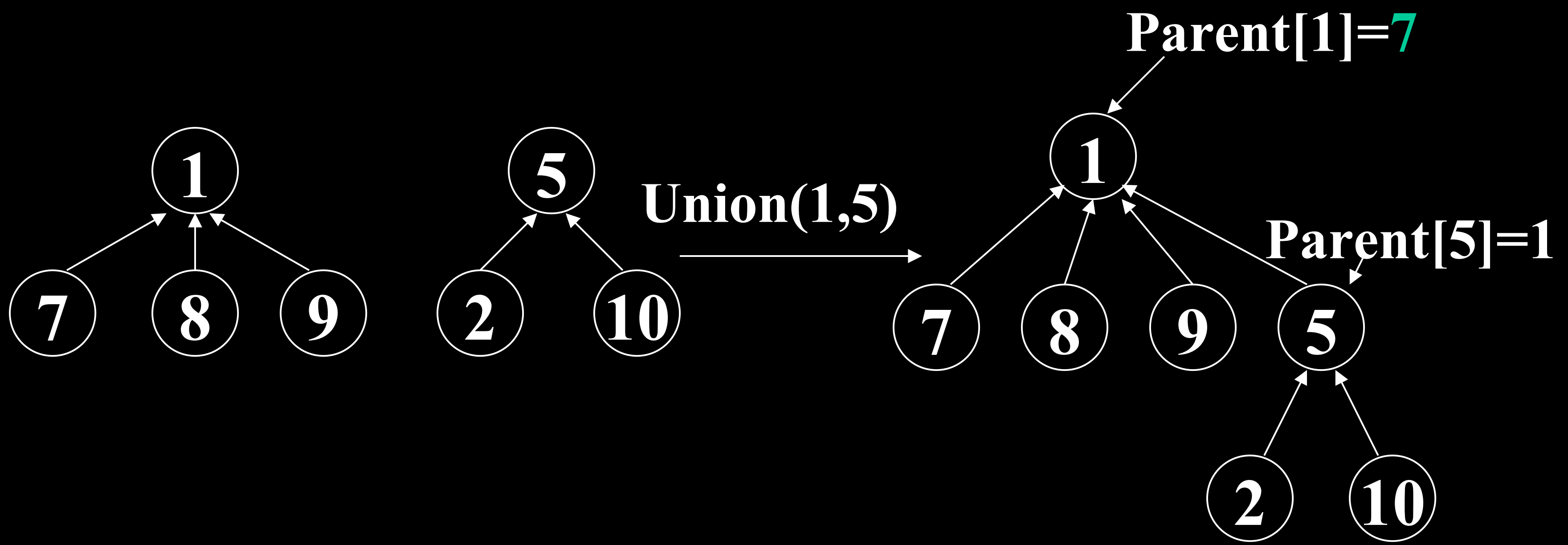
6) **Performance Enhancement**

**①improve Union in order to decrease the time each find take, so that the height of tree will not increase linearly .**

**②.Improvement of *Find* –path compression**

# Equivalence Class

**Let's discuss the weight rule(C++) :**




**Besides the *parent* field, each node has <u>a boolean field *root*</u>.The *root* field is <u>true</u> iff the node is presently a root node.The *parent* field of <u>each root node</u> is used to keep a count of the total number of nodes in the tree.**

# Equivalence Class

**Union with the weight rule**

```
void Initialize(int n)
{ root=new bool[n+1];
  parent=new int[n+1];
  for(int e=1;e<=n;e++)
  { parent[e]=1;
    root[e]=true;
  }
}
int Find(int e)
{ while(!root[e])
      e=parent[e];
  return e;
}
```

# Equivalence Class

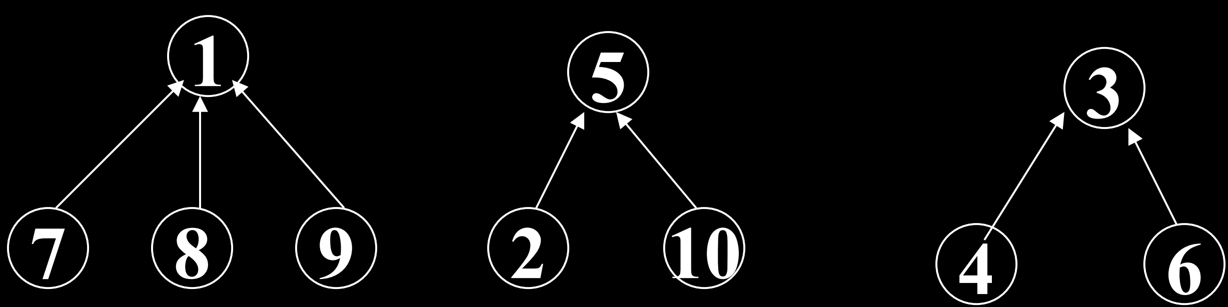
```
void Union(int i, int j)
{  if(parent[i]<parent[j])  //  i becomes subtree of j
      { parent[j]=parent[j]+parent[i];
         root[i]=false;
         parent[i]=j;
       }
    else { parent[i]=parent[i]+parent[j];
             root[j]=false;
             parent[j]=i;
           }
}
```

# Equivalence Class

**Java(高度规则）**

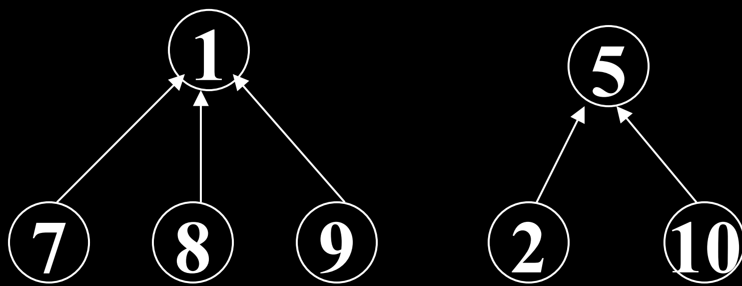
用一个数组来实现，根结点中放负数，而且是代表高度。

```
public void union( int root1, int root2 )
{    if(  s[ root2 ] < s[ root1 ] )
          s[ root1 ] = root2;
     else {   if( s[ root1 ] = = s[ root2 ] )
                 s[ root1 ]--;
              s[ root2 ] = root1;} }
```




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S: | | -2 | 5 | -2 | 3 | -2 | 3 | 1 | 1 | 1 | 5 |

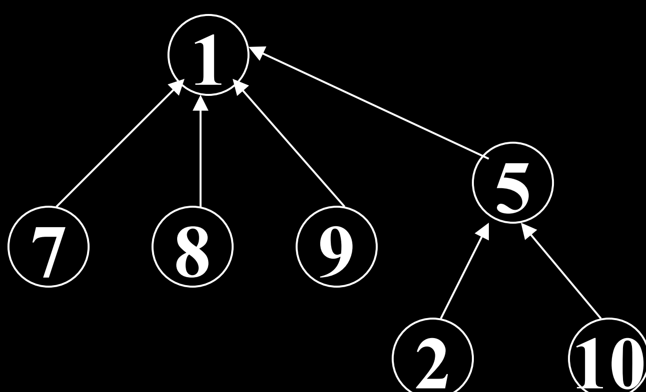
# Equivalence Class




union ( 1, 5 )

s[root1]--

s[root2] = root1

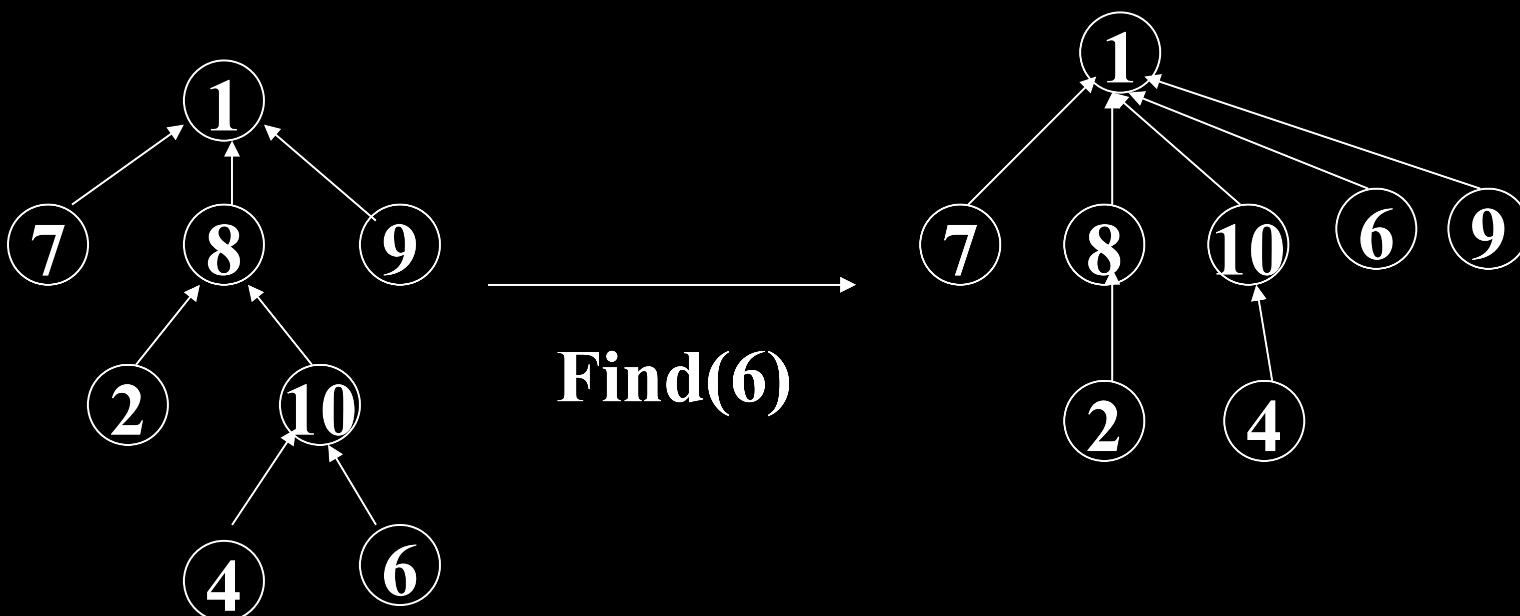


# Equivalence Class

**②.Improvement of *Find* –path compression**

**When processing a equivalence pair, we need to operate *Find* twice, *WeightUnion* once.**

**Example of improvement:**



# Equivalence Class

```
int Find( int e) { /* C++  */ }
{  int j=e;
   while(!root[j])  j=parent[j];
   int f=e;
   while(f!=j)
     { int pf=parent[f];   parent[f]=j;  f=pf; }
}
```

# Equivalence Class

**Java**

```
public int find( int x )
{    if( s[ x ] < 0 )
            return x;
     else
            return s[ x ] = find( s[ x ] );
}
```