# NCKU Programming Contest Training Course
# Disjoint set
**2018/03/07**

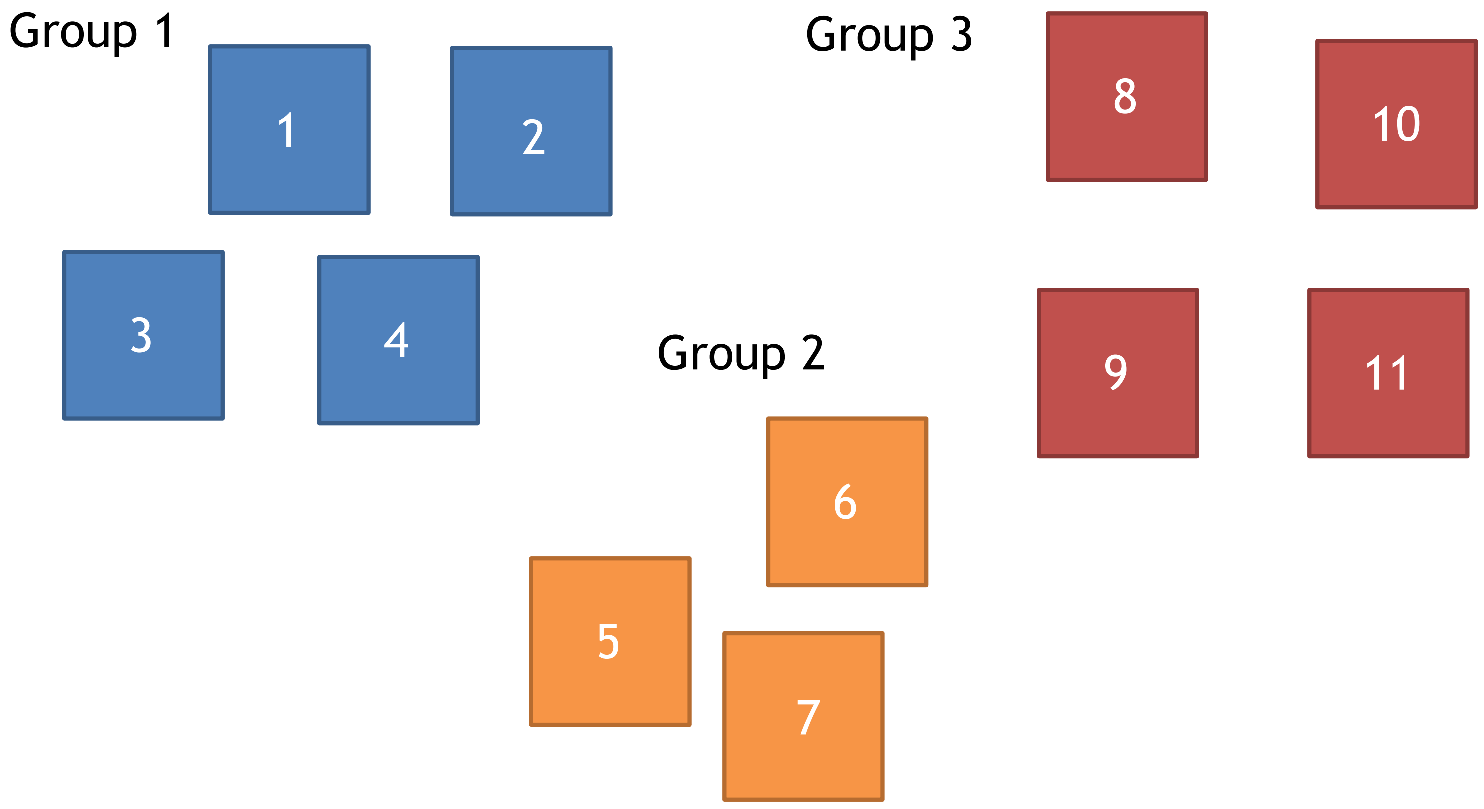
**Chun-Chi, Fang**
*khtp91113@gmail.com*

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

***made by electron & free999***

## Disjoint Set

After classifying elements, we have several disjoint sets.



# Disjoint Set

We want to know which group elements belonged to

Element 1 in group 1
Element 2 in group 1
Element 5 in group 2
Element 10 in group 3
…

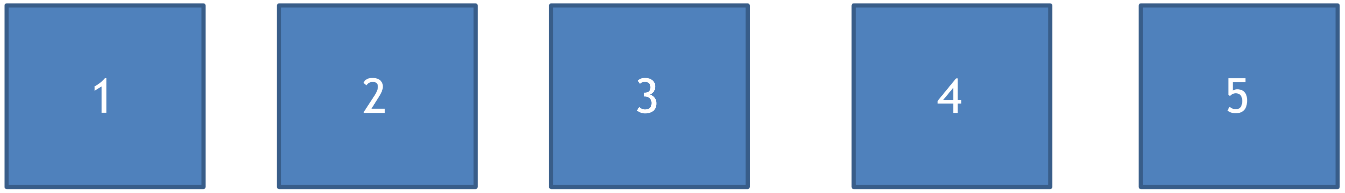
# Disjoint Set

- Main Operation
  - Union
  - Find

*made by electron & free999*

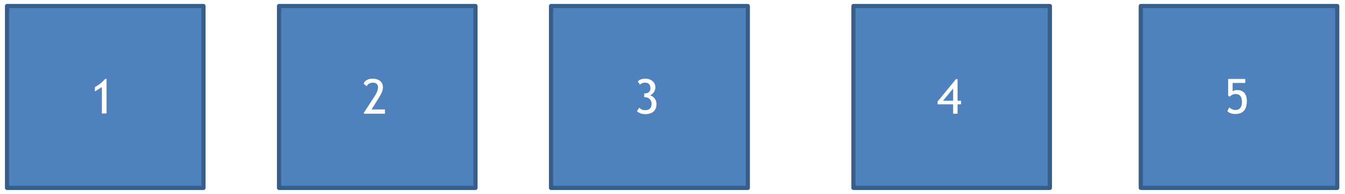
# Disjoint Set

- Initial State



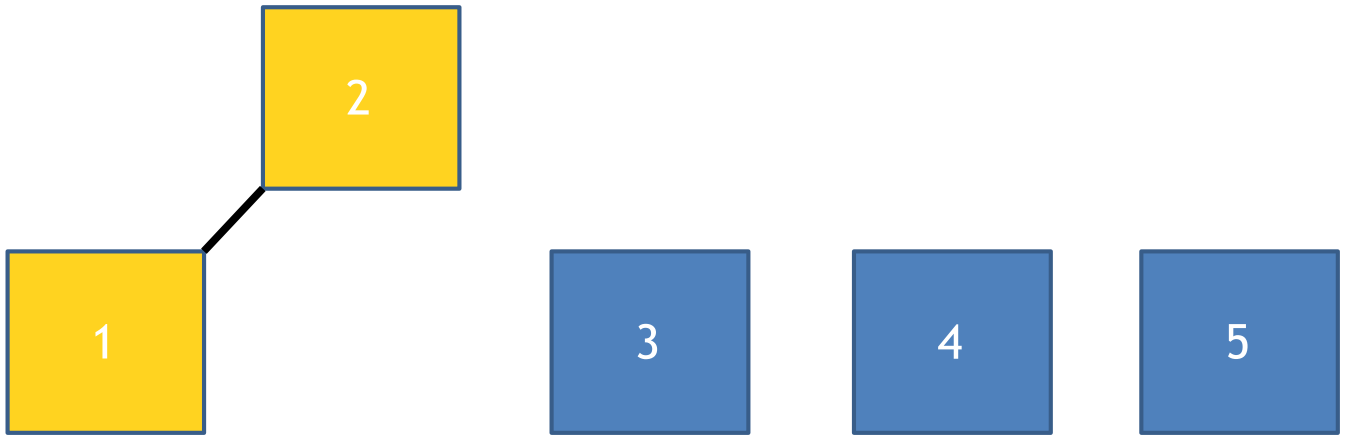
# Disjoint Set

- Find(1) : return 1



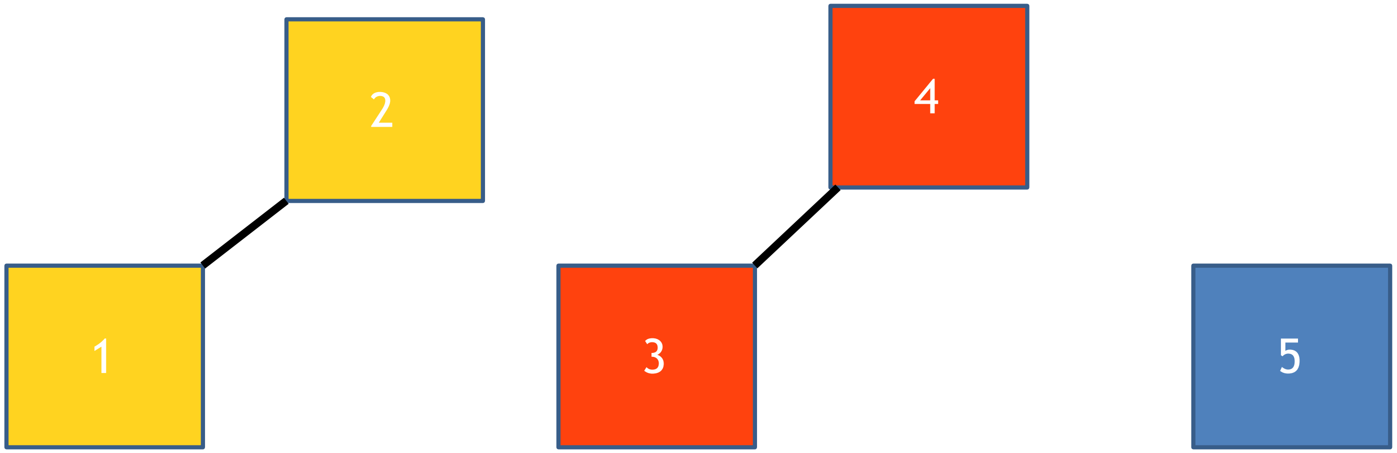
# Disjoint Set

•Union(1, 2)



# Disjoint Set

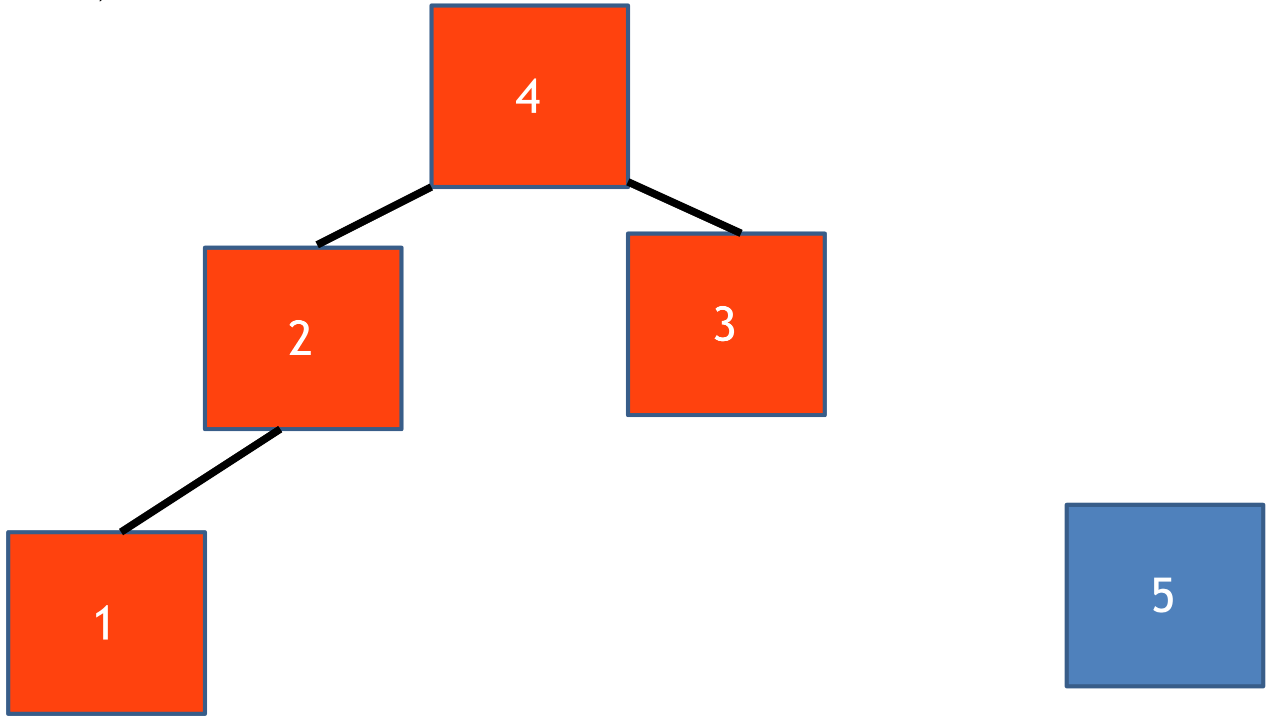
- Union(3, 4)



*made by electron & free999*

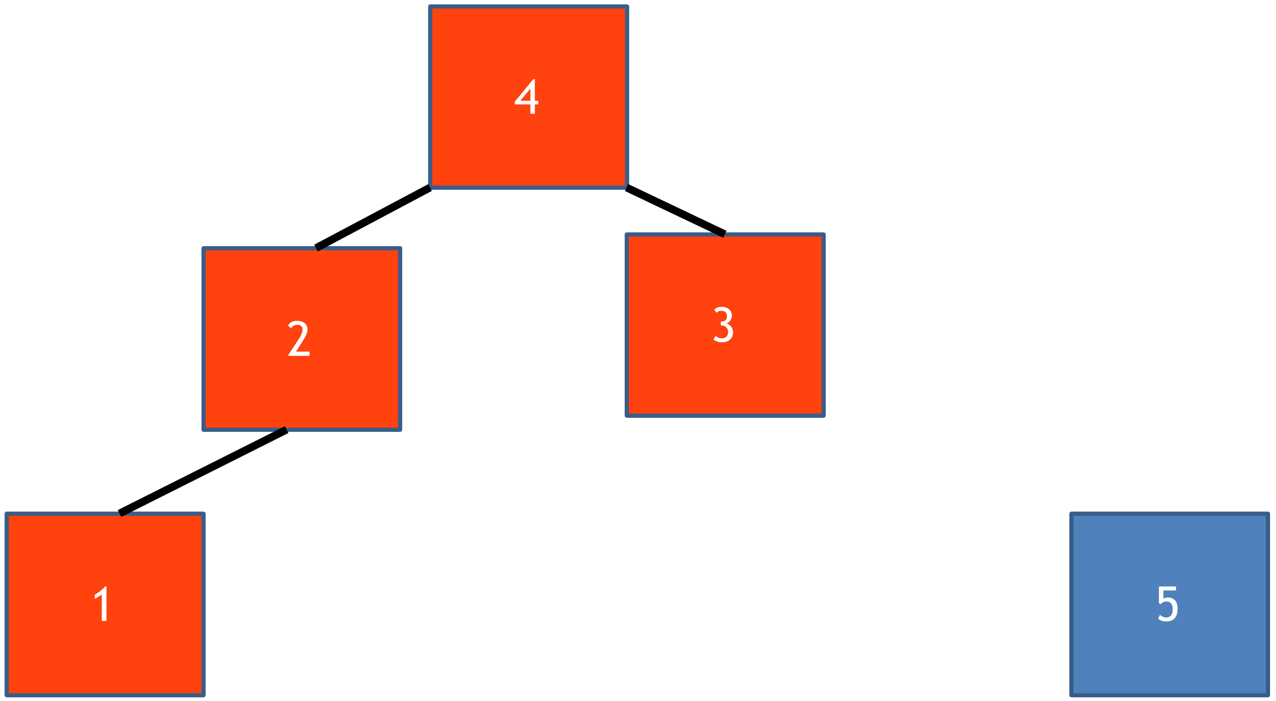
# Disjoint Set

- Union(1, 3)



# Disjoint Set

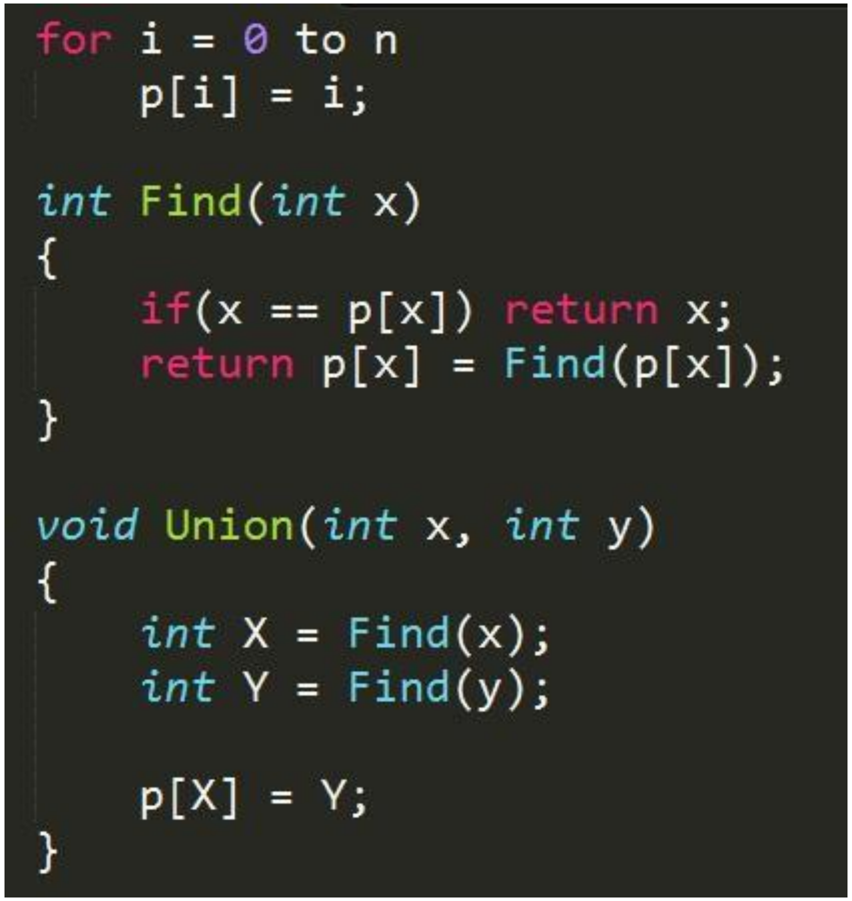
- Find(1) => 4
- Find(5) => 5



# Disjoint Set

- Find
  - –Find the root of each group
  - –Rebuild tree

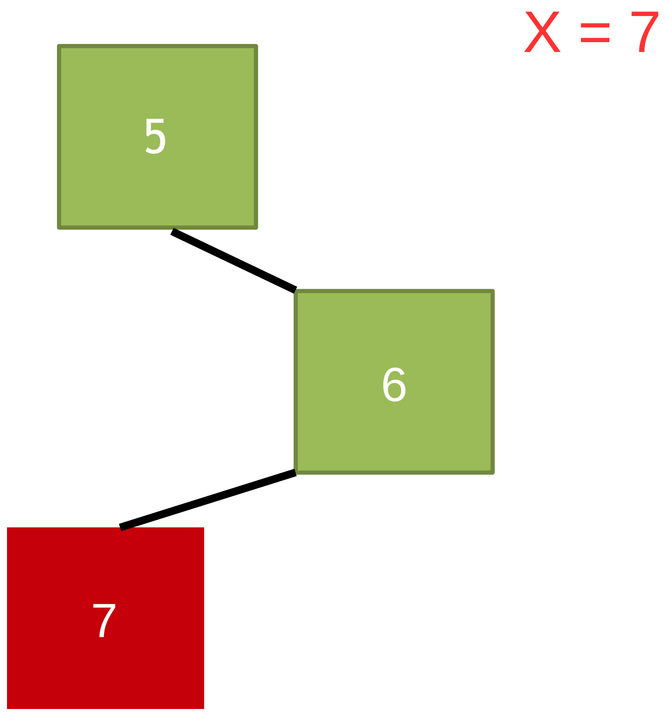
- Union
  - –Combine two group

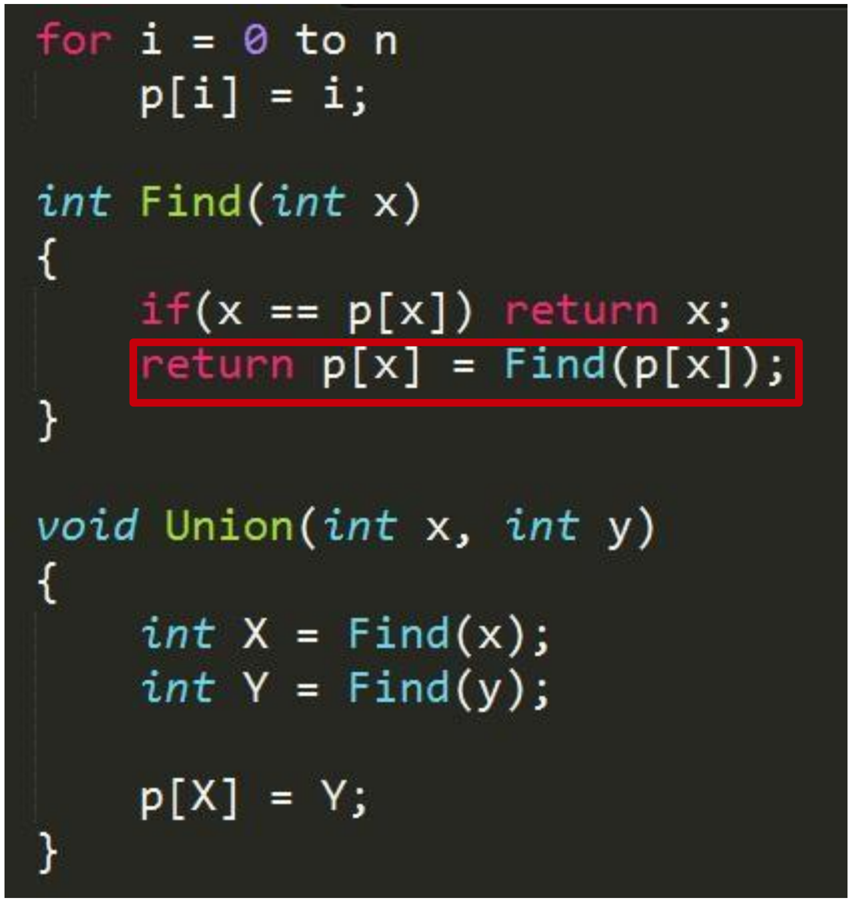


```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

# Disjoint Set

- Find(7) :



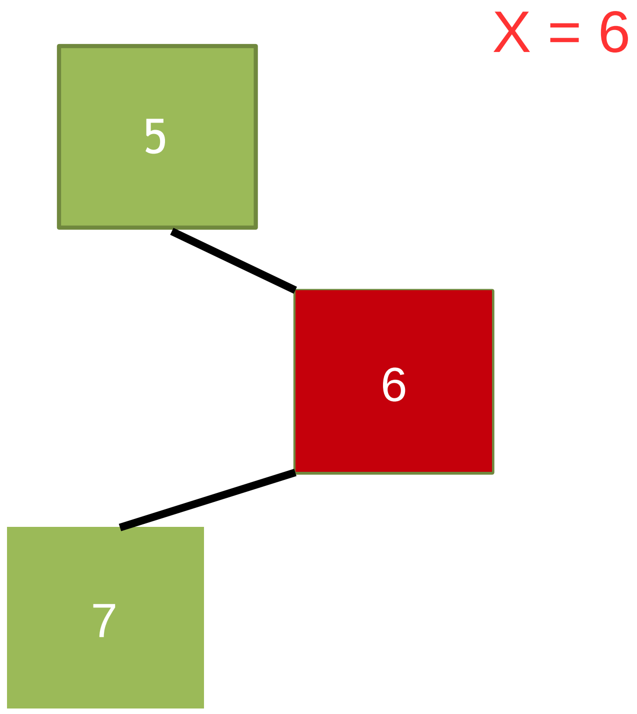


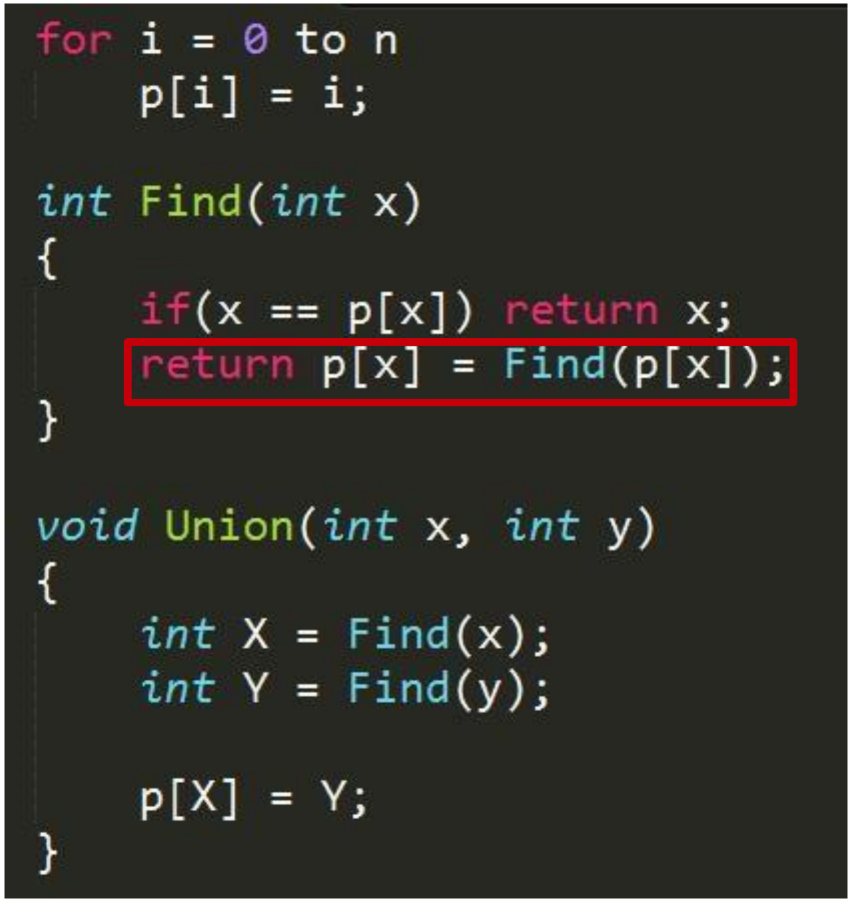
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Find(7) :



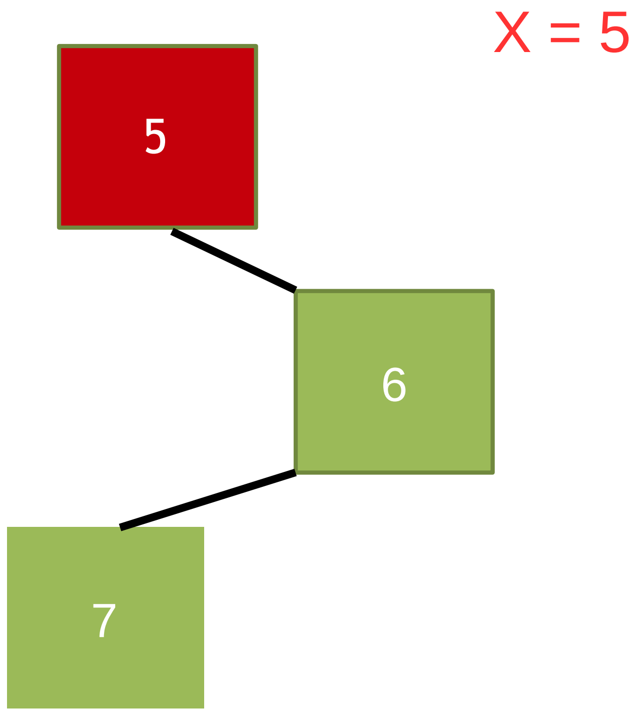


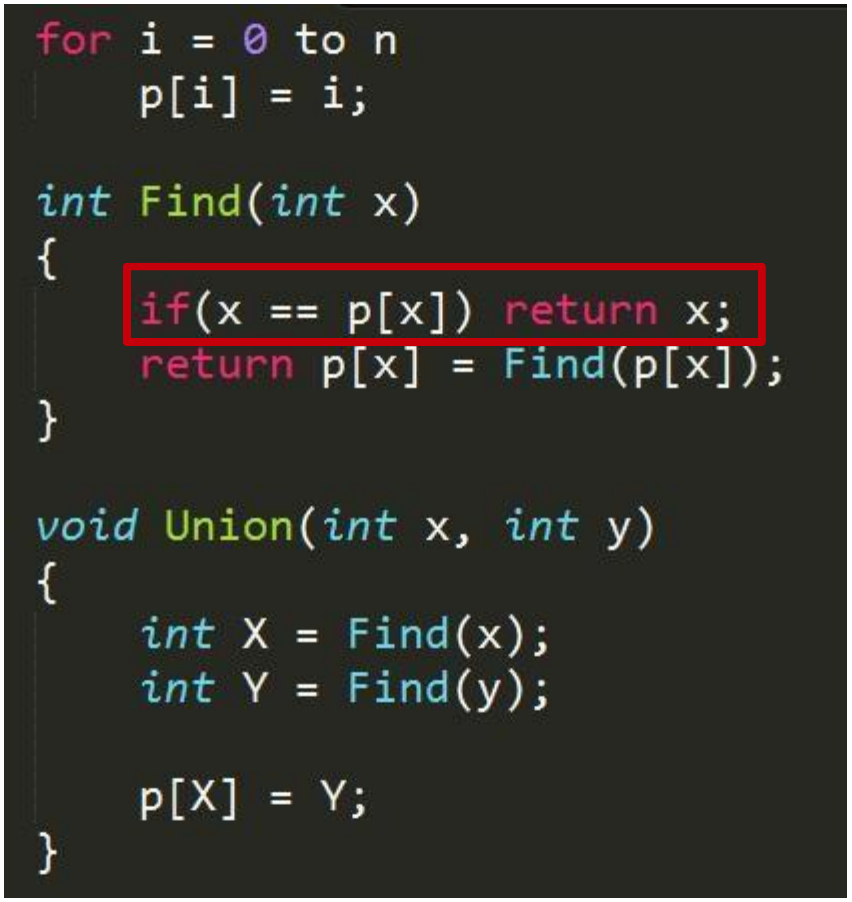
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Find(7) : return 5



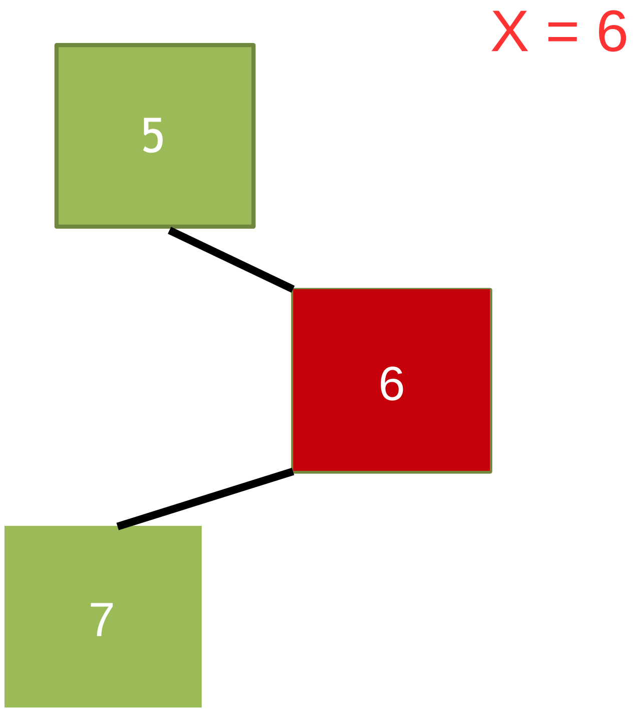


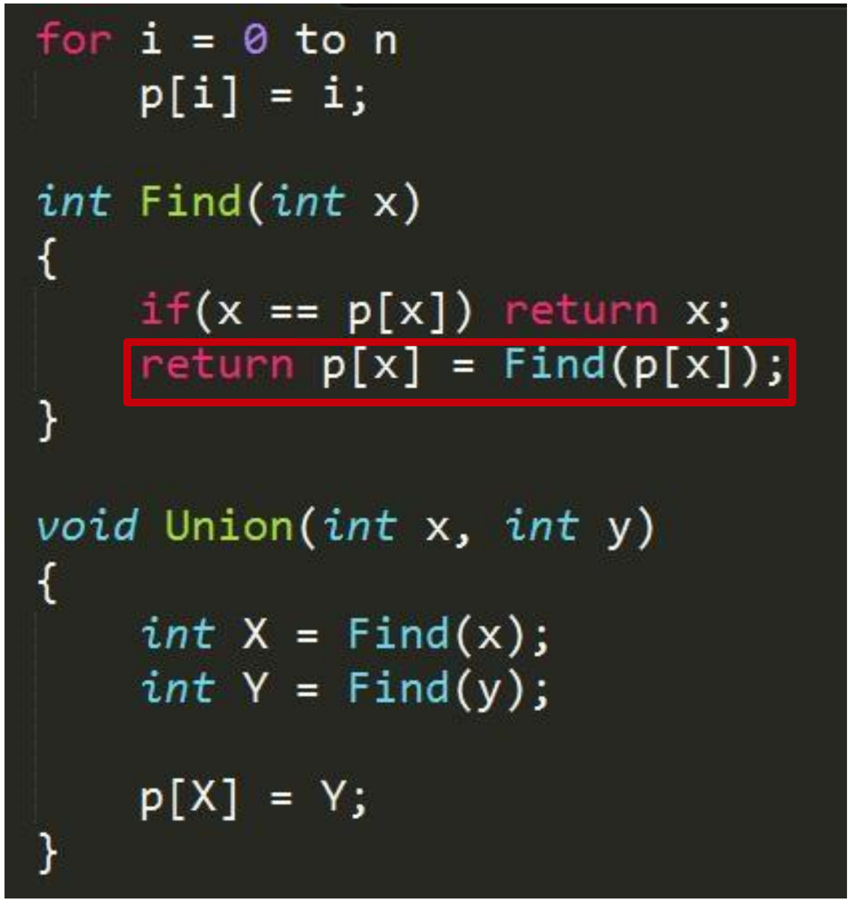
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Find(7) : return 5



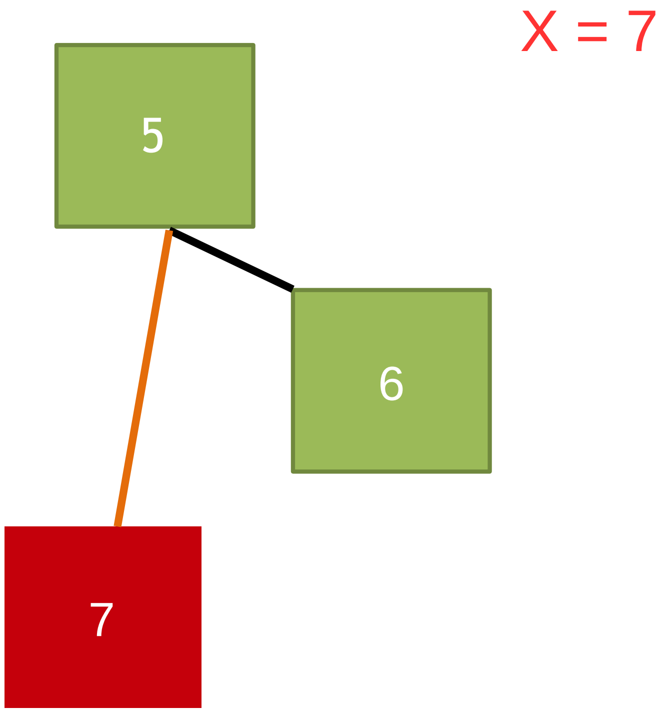


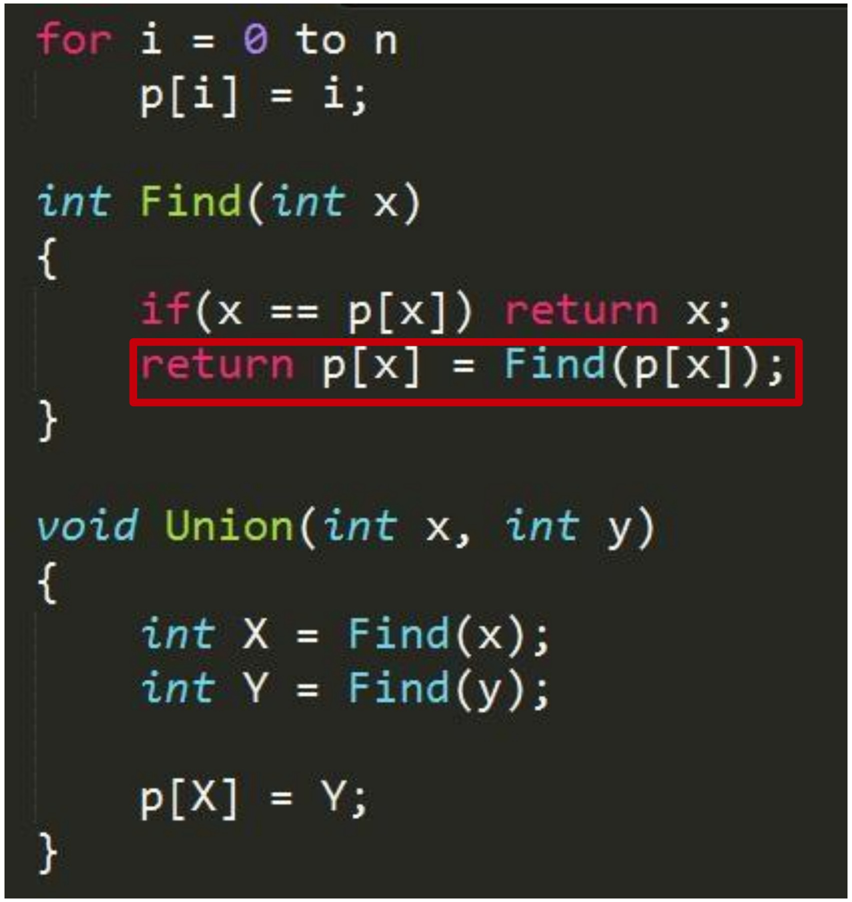
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Find(7) : return 5



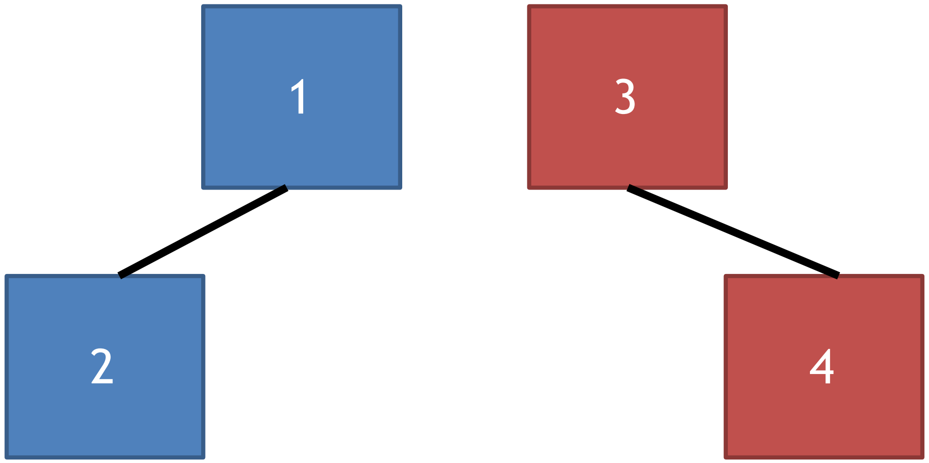


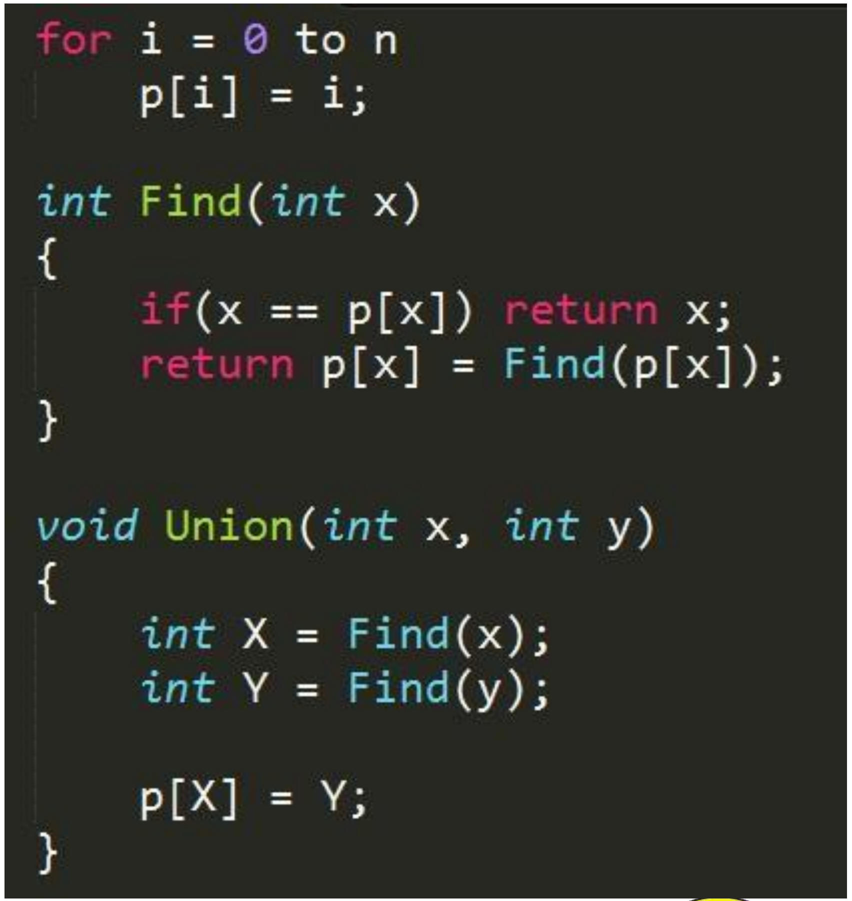
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Union(2, 4)



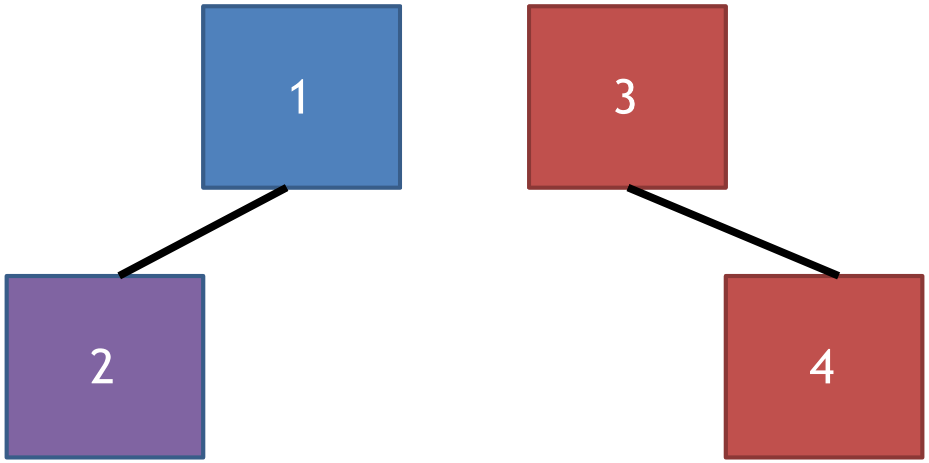


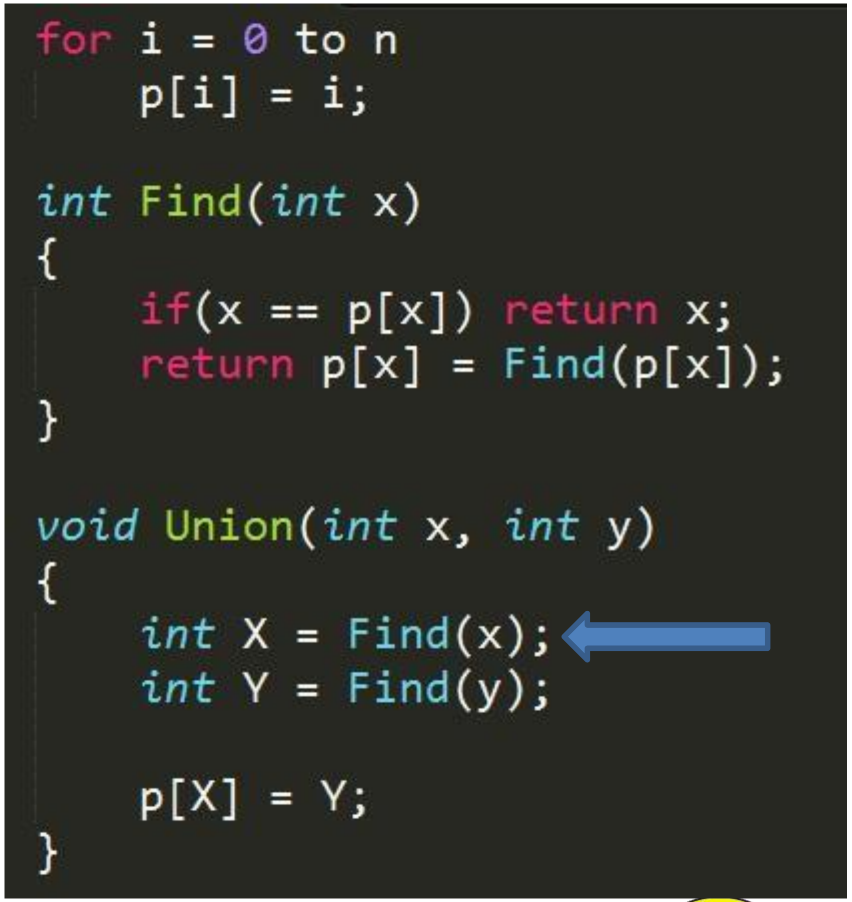
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Union(2, 4)



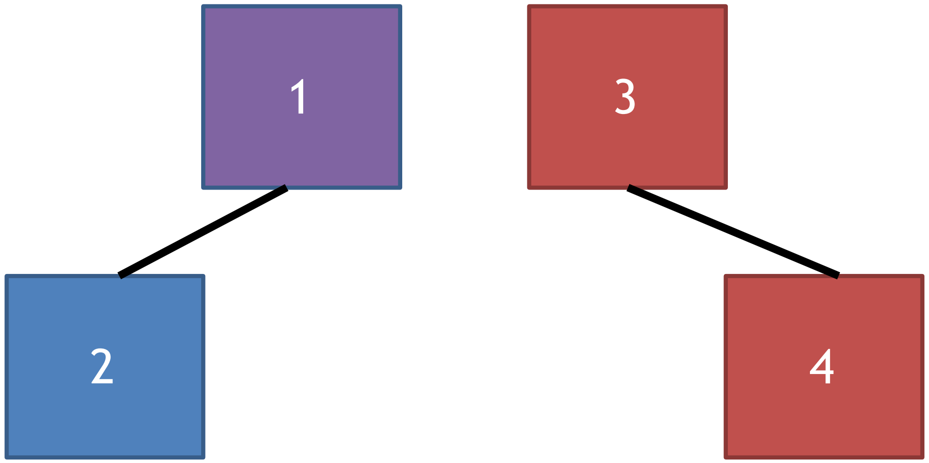


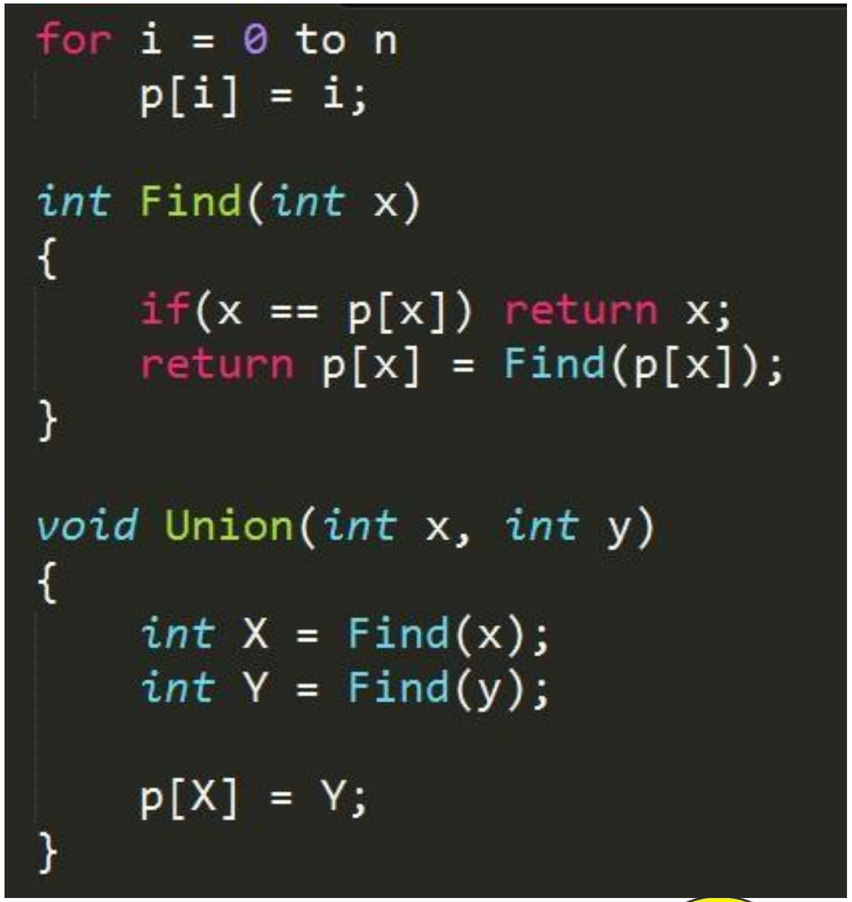
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Union(2, 4)



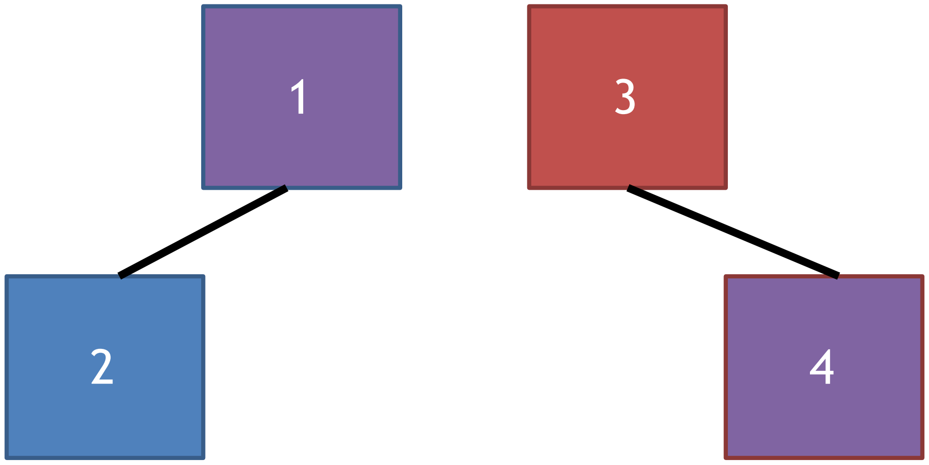


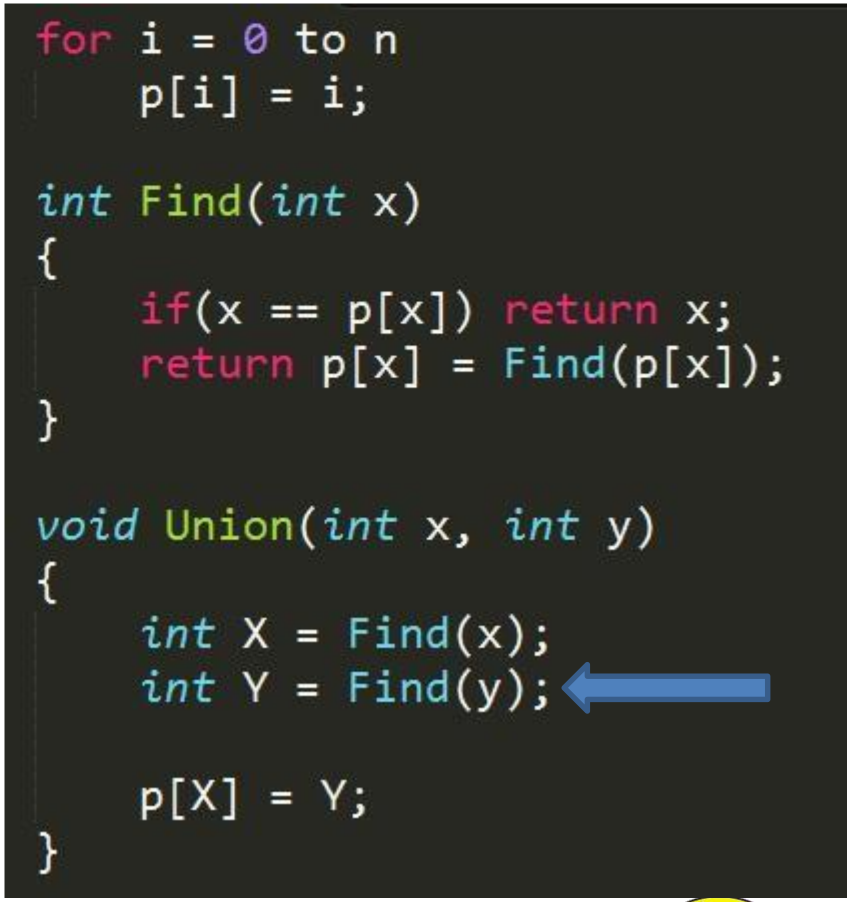
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

•Union(2, 4)



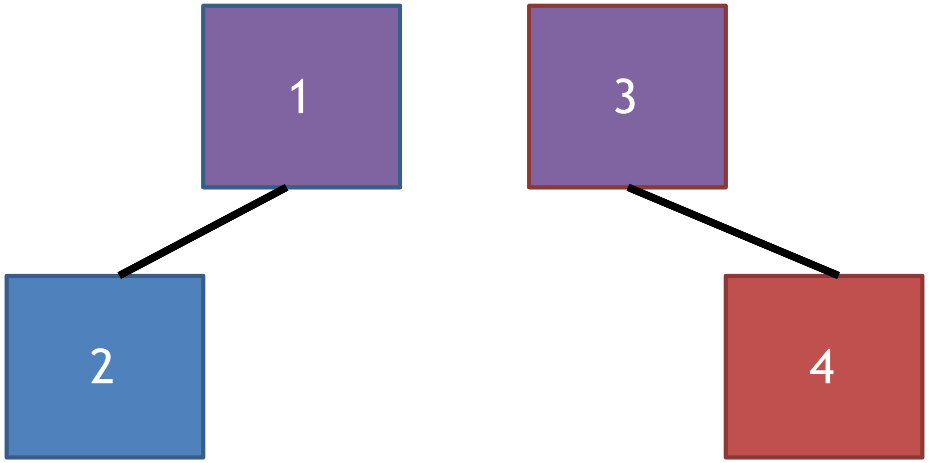


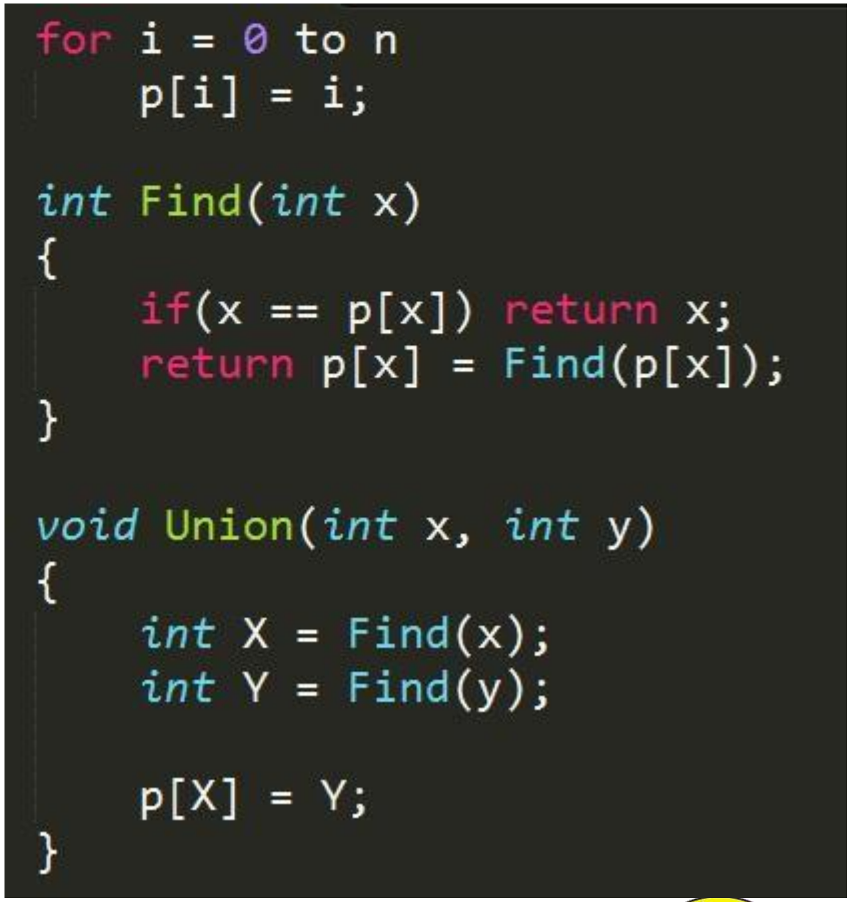
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

- Union(2, 4)



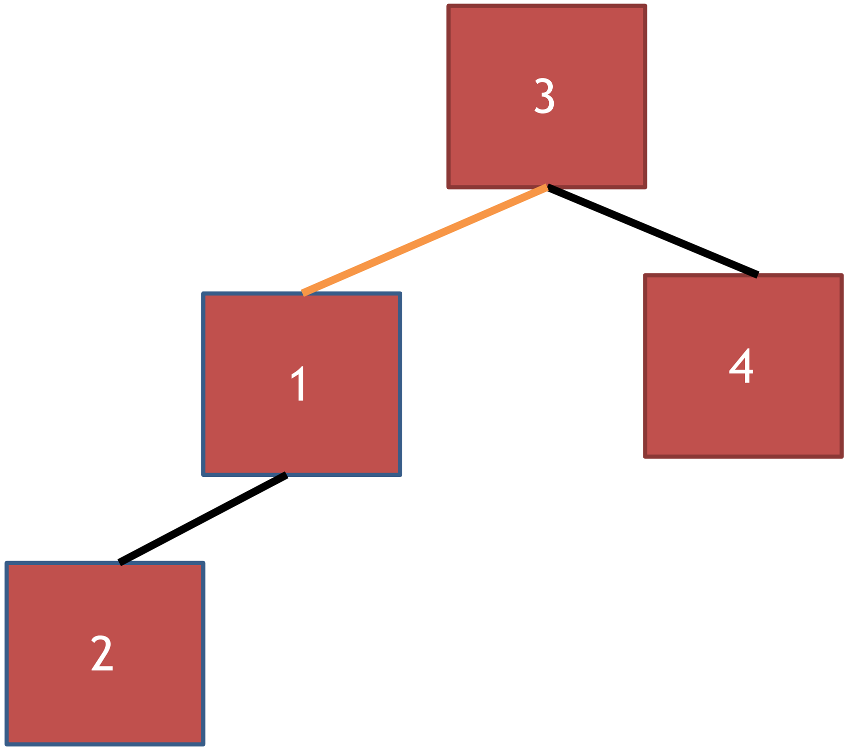


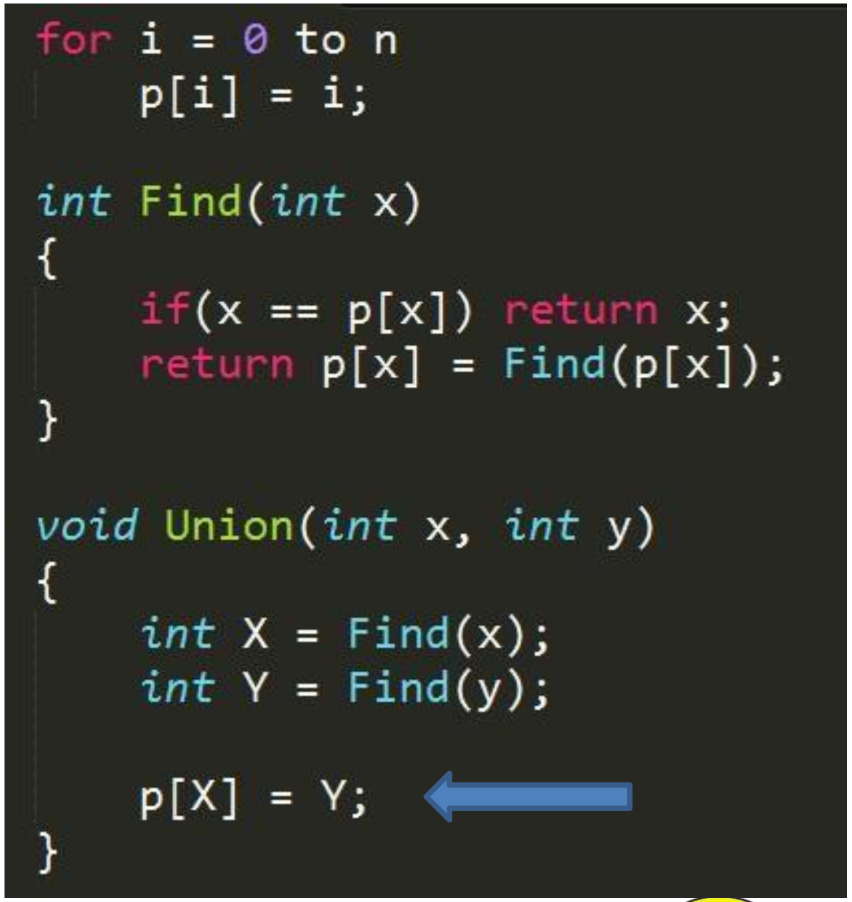
```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Disjoint Set

•Union(2, 4)





```
for i = 0 to n
    p[i] = i;

int Find(int x)
{
    if(x == p[x]) return x;
    return p[x] = Find(p[x]);
}

void Union(int x, int y)
{
    int X = Find(x);
    int Y = Find(y);

    p[X] = Y;
}
```

*made by electron & free999*

# Example 1

**UVa 793 – Network Connections**

**Problem Description**

Bob, who is a network administrator, supervises a network of computers. He is keeping a log connections between the computers in the network. Each connection is bi–directional. Two computers are interconnected if they are directly connected or if they are interconnected with the same computer. Occasionally, Bob has to decide, quickly, whether two given computers are connected, directly or indirectly, according to the log information. Write a program which based on information input from a text file counts the number of successful and the number of unsuccessful answers to the questions of the kind: is computer$_i$ interconnected with computer$_j$ ?

*made by electron & free999*

# Exercise

- Uva(5)
  - 879, 10583, 10685, 10158, 11987
- POJ(1)
  - 1703