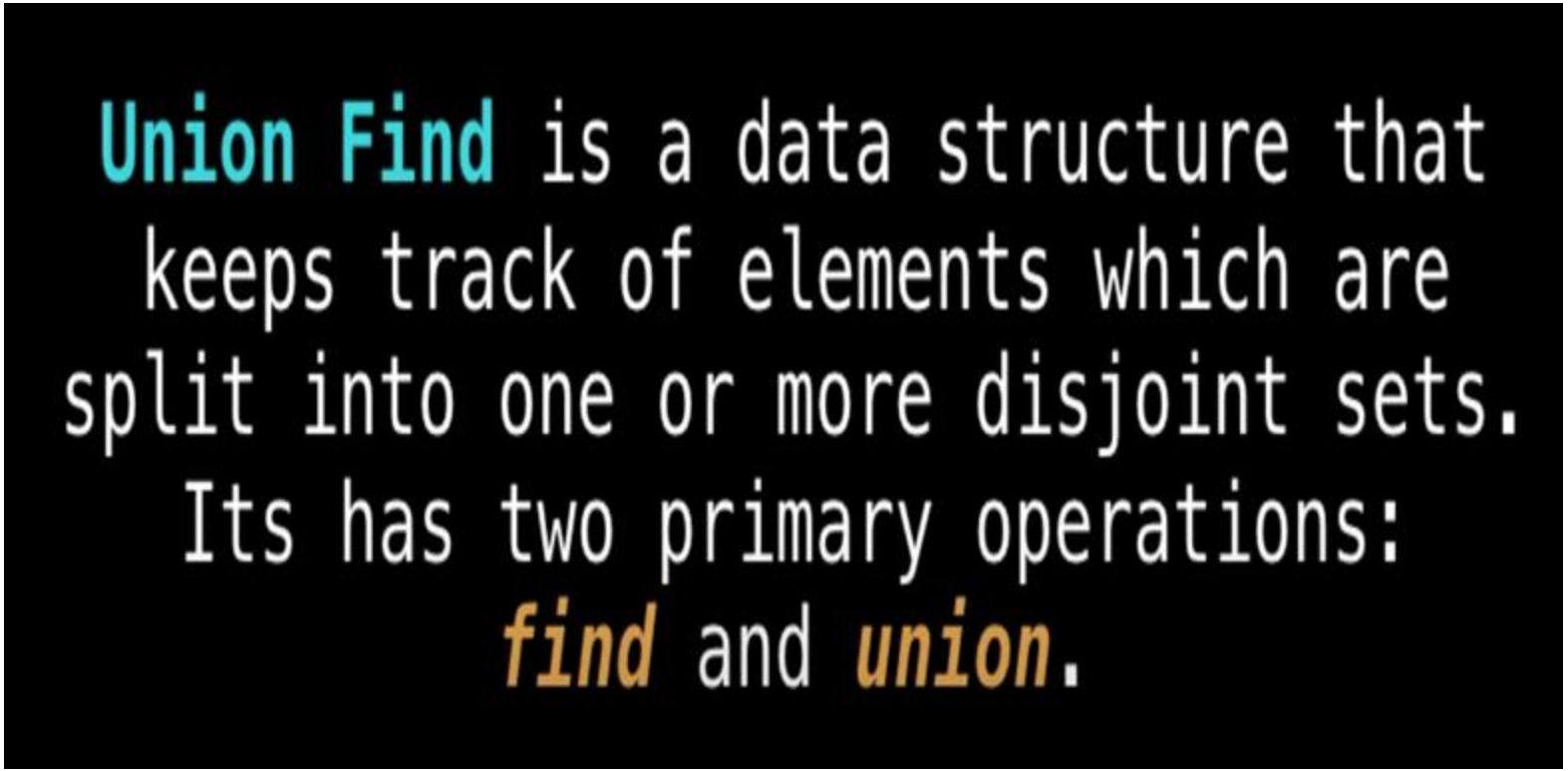
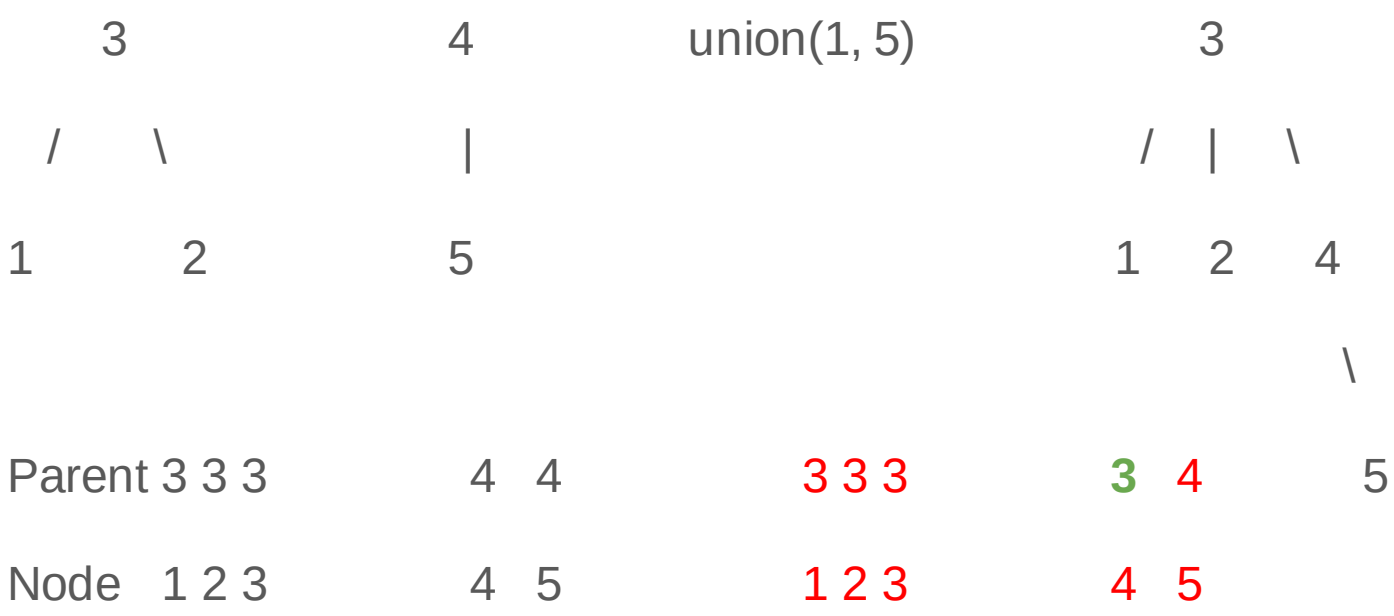
Disjoint Set (Or Union-Find)

Leetcode 547. Friend Circles
Leetcode 305. Number of Islands II
Leetcode 128. Longest Consecutive Sequence

Solve "Dynamic connectivity")　What is Union Find?



example



Basic implementation

```
function MakeSet(x)
    x.parent := x
```

```
function Find(x)
    if x.parent == x
       return x
    else
       return Find(x.parent)
```

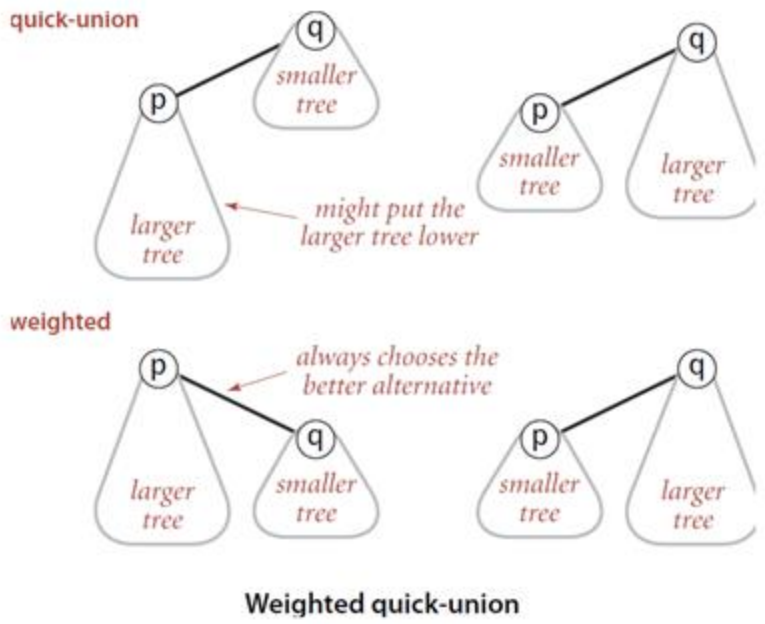
```
function Union(x, y)
    xRoot := Find(x)
    yRoot := Find(y)
    xRoot.parent := yRoot
```

```
39  class DSU {
40      int[] parent;
41      public DSU(int N) {
42          parent = new int[N];
43          for (int i = 0; i < N; i++) parent[i] = i;
44      }
45      public int find(int x) {
46          if (parent[x] != x) parent[x] = find(parent[x]);
47          return parent[x];
48      }
49      public void union(int x, int y) {
50          parent[find(x)] = find(y);
51      }
52  }
```

with path compression

Improved with size (weighted)

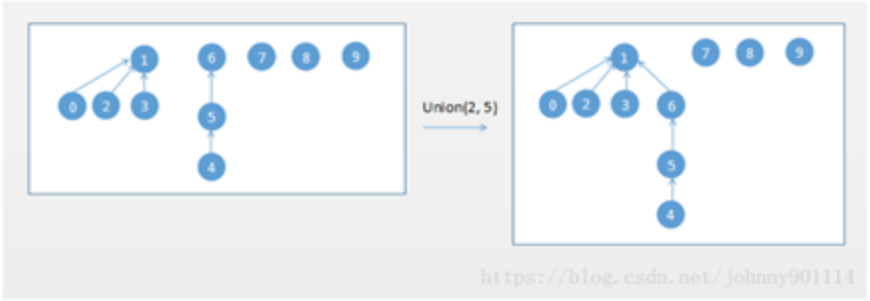
```
63  class DSU {
64      int[] parent;
65      int[] size;
66      public DSU (int N) {
67          parent = new int[N]; size = new int[N];
68          for (int i = 0; i < N; i++) parent[i] = i;
69          Arrays.fill(size, 1);
70      }
71      public int find(int x) {
72          if (parent[x] != x) parent[x] = find(parent[x]);
73          return parent[x];
74      }
75      public void union(int x, int y) {
76          int rootX = find(x), rootY = find(y);
77          if (rootX == rootY) return;
78          if (size[rootX] <= size[rootY]) {
79              parent[rootX] = rootY;
80              size[rootY] += size[rootX];
81          } else if (size[rootX] > size[rootY]) {
82              parent[rootY] = rootX;
83              size[rootX] += size[rootY];
84          }
85      }
86  }
```

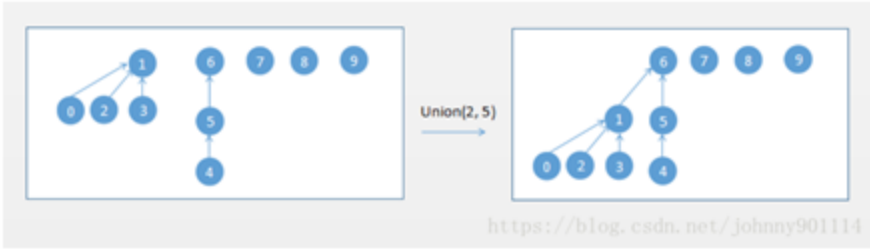


Weighted quick-union

Improved with ranked

```java
class DSU {
    int[] parent;
    int[] rank;
    public DSU (int N) {
        parent = new int[N];
        rank = new int[N];
        for (int i = 0; i < N; i++) parent[i] = i;
        Arrays.fill(rank, 1);
    }
    public int find(int x) {
        if (parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
    }
    public void union(int x, int y) {
        int rootX = find(x), rootY = find(y);
        if (rootX == rootY) return;
        if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        } else if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
        } else {
            parent[rootX] = rootY;
            rank[rootY]++;
        }
    }
}
```





547. Friend Circles

Description | Hints | Submissions | Discuss | Solution

Pick One

There are **N** students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a **direct** friend of B, and B is a **direct** friend of C, then A is an **indirect** friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a **N*N** matrix **M** representing the friend relationship between students in the class. If M[i][j] = 1, then the i_{th} and j_{th} students are **direct** friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

Example 1:

```
Input:
[[1,1,0],
 [1,1,0],
 [0,0,1]]
Output: 2
Explanation:The 0th and 1st students are direct friends, so they are in a friend circle.
The 2nd student himself is in a friend circle. So return 2.
```

Example 2:

```
Input:
[[1,1,0],
 [1,1,1],
 [0,1,1]]
Output: 1
Explanation:The 0th and 1st students are direct friends, the 1st and 2nd students are direct friends,
so the 0th and 2nd students are indirect friends. All of them are in the same friend circle, so retu
```

Just count how many disjoint sets

```java
class Solution {
    public int findCircleNum(int[][] M) {
        DSU dsu = new DSU(M.length);
        for (int i = 0; i < M.length; i++)
            for (int j = 0; j < M[i].length; j++)
                if (M[i][j] == 1) dsu.union(i, j);
        int res = 0;
        for (int i = 0; i < M.length; i++) if (dsu.find(i) == i) res++;
        return res;
    }
}

class DSU {
    int[] parent;
    public DSU(int N) {
        parent = new int[N];
        for (int i = 0; i < parent.length; i++) parent[i] = i;
    }
    public int find(int x) {
        if (parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
    }
    public void union(int x, int y) {
        parent[find(x)] = find(y);
    }
}
```

305. Number of Islands II

Description | Hints | Submissions | Discuss | Solution

Pick One

A 2d grid map of `m` rows and `n` columns is initially filled with water. We may perform an *addLand* operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each *addLand* operation**. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example:

```
Input: m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]
Output: [1,1,2,3]
```

Explanation:

Initially, the 2d grid `grid` is filled with water. (Assume 0 represents water and 1 represents land).

```
0 0 0
0 0 0
0 0 0
```

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

```
1 0 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

```
1 1 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

```
1 1 0
0 0 1   Number of islands = 2
0 0 0
```

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

```
1 1 0
0 0 1   Number of islands = 3
0 1 0
```

```java
    int[][] dirs ={{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    public List<Integer> numIslands2(int m, int n, int[][] positions) {
        DSU dsu = new DSU(m * n);
        boolean[][] island = new boolean[m][n];
        List<Integer> res = new ArrayList<>();
        int count = 0;
        for (int[] cur : positions) {
            if (island[cur[0]][cur[1]]) {
                res.add(count);
                continue;
            }
            island[cur[0]][cur[1]] = true;
            count++;
            for (int[] dir: dirs) {
                int x = cur[0] + dir[0], y = cur[1] + dir[1];
                if (x < 0 || x >= m || y < 0 || y >= n || island[x][y] == false) continue;
                int component1 = dsu.find(cur[0] * n + cur[1]);
                int component2 = dsu.find(x * n + y);
                if (component1 != component2) {
                    dsu.union(component2, component1);
                    count--;
                }
            }
            res.add(count);
        }
        return res;
    }
}

class DSU {
    int[] parent;
    public DSU (int N) {
        parent = new int[N];
        for (int i = 0; i < N; i++) parent[i] = i;
    }
    public int find(int x) {
        if (parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
    }
    public void union(int x, int y) {
        parent[find(x)] = find(y);
    }
}
```

128. Longest Consecutive Sequence

Description | Hints | Submissions | Discuss | Solution

Pick One

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

Your algorithm should run in O(*n*) complexity.

Example:

```
Input: [100, 4, 200, 1, 3, 2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.
```

```java
public class Solution {

    public int longestConsecutive(int[] nums) {
        DSU dsu = new DSU(nums.length);
        Map<Integer,Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(nums[i])) continue;
            map.put(nums[i], i);
            if (map.containsKey(nums[i] + 1)) dsu.union(i,map.get(nums[i] + 1));
            if (map.containsKey(nums[i] - 1)) dsu.union(i,map.get(nums[i] - 1));
        }
        return dsu.findMax();
    }
}
```

```java
class DSU {
    int[] parent;
    int[] size;
    public DSU(int N) {
        parent = new int[N];
        size = new int[N];
        for (int i = 0; i < N; i++) parent[i] = i;
        Arrays.fill(size, 1);
    }
    public int find(int x) {
        if (parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
    }
    public void union(int x, int y) {
        int rootX = find(x), rootY = find(y);
        if (rootX == rootY) return;
        if (size[rootX] <= size[rootY]) {
            parent[rootX] = rootY;
            size[rootY] += size[rootX];
        } else {
            parent[rootY] = rootX;
            size[rootX] += size[rootY];
        }
    }
    public int findMax() {
        int max = 0;
        for (int s : size) max = Math.max(max, s);
        return max;
    }
}
```