

Quick Find

```
1  class UnionFind {
2  public:
3      UnionFind(int n) {
4          size = n;
5          color = new int[n];
6          for (int i = 0; i < n; i++) {
7              color[i] = i;
8          }
9      }
10     // 查看 x 的颜色 -> x 的所属集合
11     int find(int x) {
12         return color[x];
13     }
14     // 将 x 和 y 染色为同一个颜色 -> 合并 x 和 y 的所属集合
15     void merge(int x, int y) {
16         if (color[x] == color[y]) return ;
17         int colorY = color[y];
18         for (int i = 0; i < size; i++) {
19             if (color[i] == colorY) color[i] = color[x];
20         }
21     }
22 public:
23     int *color, size;
24 };;
```

Quick Union

```
1  class UnionFind {
2  public:
3      UnionFind(int n) {
4          size = n;
5          father = new int[n];
6          for (int i = 0; i < n; i++) {
7              father[i] = i;
8          }
9      }
10     // 查看 x 的根结点 -> x 的所属集合
11     int find(int x) {
12         int root = x;
13         while (root != father[root]) {
14             root = father[root];
15         }
16         return root;
```

```

17     }
18     // 将 x 和 y 染色为同一个颜色 -> 合并 x 和 y 的所属集合
19     void merge(int x, int y) {
20         int rootX = find(x);
21         int rootY = find(y);
22         if (rootX == rootY) return ;
23         father[rootX] = rootY;
24     }
25 public:
26     int *father, size;
27 };

```

Weighted Quick Union

```

1  class UnionFind {
2  public:
3      UnionFind(int n) {
4          father = new int[n];
5          treeSize = new int[n];
6          size = n;
7          for (int i = 0; i < n; i++) {
8              father[i] = i;
9              treeSize[i] = 1;
10         }
11     }
12     int find(int x) {
13         int root = x;
14         while (father[root] != root) {
15             root = father[root];
16         }
17         return root;
18     }
19     void merge(int x, int y) {
20         int fx = find(x);
21         int fy = find(y);
22         if (fx == fy) return ;
23         if (treeSize[fx] < treeSize[fy]) {
24             father[fx] = fy;
25             treeSize[fy] += treeSize[fx];
26         }
27         else {
28             father[fy] = fx;
29             treeSize[fx] += treeSize[fy];
30         }
31     }
32 public:

```

```
33     int *father, *treeSize, size;
34 };
```

Weighted Quick Union with Path Compression

```
1  class UnionFind {
2  public:
3      UnionFind(int n) {
4          size = n;
5          father = new int[n];
6          treeSize = new int[n];
7          for (int i = 0; i < n; i++) {
8              father[i] = i;
9              treeSize[i] = 1;
10         }
11     }
12     // 查看 x 的根结点 -> x 的所属集合
13     int find(int x) {
14         int root = x;
15         while (root != father[root]) {
16             root = father[root];
17         }
18         while (x != root) {
19             int fx = father[x];
20             father[x] = root;
21             x = fx;
22         }
23         return root;
24     }
25     // 将 x 和 y 染色为同一个颜色 -> 合并 x 和 y 的所属集合
26     void merge(int x, int y) {
27         int rootX = find(x);
28         int rootY = find(y);
29         if (rootX == rootY) return ;
30         if (treeSize[rootX] < treeSize[rootY]) {
31             father[rootX] = rootY;
32             treeSize[rootY] += treeSize[rootX];
33         }
34         else {
35             father[rootY] = rootX;
36             treeSize[rootX] += treeSize[rootY];
37         }
38     }
39 public:
40     int *father, *treeSize, size;
41 };
```

547. 省份数量

<https://leetcode-cn.com/problems/number-of-provinces/>

```
1  class Solution {
2  public:
3      int findCircleNum(vector<vector<int>>& isConnected) {
4          int n = isConnected.size();
5          UnionFind uf(n);
6          for (int i = 0; i < n; i++) {
7              for (int j = 0; j < n; j++) {
8                  if (isConnected[i][j] == 1) uf.merge(i, j);
9              }
10         }
11         // 集合的数量
12         return uf.setCnt;
13     }
14 }
```

200. 岛屿数量

<https://leetcode-cn.com/problems/number-of-islands/>

```
1  class Solution {
2  public:
3      int numIslands(vector<vector<char>>& grid) {
4          int n = grid.size(), m = grid[0].size();
5          #define id(i, j) ((i) * m + (j))
6          UnionFind uf{n * m};
7          for (int i = 0; i < n; i++) {
8              for (int j = 0; j < m; j++) {
9                  if (grid[i][j] == '0') continue;
10                 // 向下合并
11                 if (i + 1 < n && grid[i + 1][j] == '1') {
12                     uf.merge(id(i, j), id(i + 1, j));
13                 }
14                 // 向右合并
15                 if (j + 1 < m && grid[i][j + 1] == '1') {
16                     uf.merge(id(i, j), id(i, j + 1));
17                 }
18             }
19         }
20     }
```



```

20     int cnt = 0;
21     for (int i = 0; i < n; i++) {
22         for (int j = 0; j < m; j++) {
23             if (grid[i][j] == '0') continue;
24             if (uf.father[id(i, j)] == id(i, j))
25                 cnt++;
26         }
27     }
28     return cnt;
29 }
30 };

```

990. 等式方程的可满足性

<https://leetcode-cn.com/problems/satisfiability-of-equality-equations/>

```

1  class Solution {
2  public:
3      bool equationsPossible(vector<string>& equations) {
4          UnionFind uf(26);
5          for (int i = 0; i < equations.size(); i++) {
6              int x = equations[i][0] - 'a';
7              int y = equations[i][3] - 'a';
8              if (equations[i][1] == '=') uf.merge(x, y);
9          }
10         for (int i = 0; i < equations.size(); i++) {
11             int x = equations[i][0] - 'a';
12             int y = equations[i][3] - 'a';
13             if (equations[i][1] == '!' && uf.find(x) == uf.find(y))
14                 return false;
15         }
16
17         return true;
18     }
19 };

```

1319. 连通网络的操作次数

<https://leetcode-cn.com/problems/number-of-operations-to-make-network-connected/>

```

1  class Solution {
2  public:
3      int makeConnected(int n, vector<vector<int>>& connections) {
4          UnionFind uf(n);

```

```

5     int left = 0;
6     for (int i = 0; i < connections.size(); i++) {
7         int x = connections[i][0];
8         int y = connections[i][1];
9         if (uf.find(x) == uf.find(y)) left++;
10        else uf.merge(x, y);
11    }
12    int cnt = 0;
13    for (int i = 0; i < n; i++) {
14        if (uf.father[i] == i) cnt++;
15    }
16    return left >= cnt - 1 ? cnt - 1 : -1;
17 }
18 };

```

684. 冗余连接

<https://leetcode-cn.com/problems/redundant-connection/>

```

1  class Solution {
2  public:
3      vector<int> findRedundantConnection(vector<vector<int>>& edges) {
4          UnionFind uf{(int)edges.size()};
5          vector<int> ans;
6          for (int i = 0; i < edges.size(); i++) {
7              // -1 ==> 1~n 0~(n-1)
8              int x = edges[i][0] - 1;
9              int y = edges[i][1] - 1;
10             if (uf.find(x) == uf.find(y)) {
11                 ans = edges[i];
12                 break;
13             }
14             uf.merge(x, y);
15         }
16         return ans;
17     }
18 };

```

947. 移除最多的同行或同列石头

<https://leetcode-cn.com/problems/most-stones-removed-with-same-row-or-colum>

```

1  class Solution {
2  public:

```

```

3      int removeStones(vector<vector<int>>& stones) {
4          // Plan A: 将每个石头看成节点  $O(n^2)$ 
5          int n = stones.size();
6          UnionFind uf(n);
7          for (int i = 0; i < n; i++) {
8              for (int j = 0; j < n; j++) {
9                  if (stones[i][0] == stones[j][0] ||
10                     stones[i][1] == stones[j][1]) {
11                      uf.merge(i, j);
12                  }
13              }
14          }
15          int cnt = 0;
16          for (int i = 0; i < n; i++) {
17              if (uf.find(i) == i) cnt++;
18          }
19          return n - cnt;
20          // Plan B: 将每条直线视为结点
21          unordered_set<int> s; // 参与运算的直线  $O(m)$ 
22          int n = stones.size();
23          int m = 10001;
24          UnionFind uf(2 * m);
25          for (int i = 0; i < n; i++) {
26              uf.merge(stones[i][0], stones[i][1] + m);
27              s.insert(stones[i][0]);
28              s.insert(stones[i][1] + m);
29          }
30          int cnt = 0;
31          for (int i = 0; i < 2 * m; i++) {
32              if (uf.find(i) == i && s.count(i)) cnt++;
33          }
34          return n - cnt;
35      }
36  };

```

1202. 交换字符串中的元素

<https://leetcode-cn.com/problems/smallest-string-with-swaps/>

```

1  class Solution {
2  public:
3      string smallestStringWithSwaps(string s, vector<vector<int>>& pairs) {
4          // 将 数组的下标 视为 并查集中的结点
5          int n = s.size();
6          UnionFind uf(n);
7          for (int i = 0; i < pairs.size(); i++) {
8              uf.merge(pairs[i][0], pairs[i][1]);

```

```

9         }
10        // 最小堆
11        priority_queue<char, vector<char>, greater<char>> h[n];
12        for (int i = 0; i < n; i++) {
13            h[uf.find(i)].push(s[i]);
14        }
15        string ans = "";
16        for (int i = 0; i < n; i++) {
17            ans += h[uf.find(i)].top();
18            h[uf.find(i)].pop();
19        }
20        return ans;
21    }
22 };

```

765. 情侣牵手

<https://leetcode-cn.com/problems/couples-holding-hands/>

```

1  class Solution {
2  public:
3      int minSwapsCouples(vector<int>& row) {
4          UnionFind uf((int)row.size() / 2);
5          for (int i = 0; i < row.size(); i += 2) {
6              int x = row[i] / 2;
7              int y = row[i + 1] / 2;
8              uf.merge(x, y);
9          }
10         int cnt = 0;
11         for (int i = 0; i < uf.size; i++) {
12             if (uf.father[i] == i) cnt++;
13         }
14         return row.size() / 2 - cnt;
15     }
16 };

```