Quick Find

```
class UnionFind {
public:
    UnionFind(int n) {
        size = n;
        color = new int[n];
        for (int i = 0; i < n; i++) {
            color[i] = i;
        }
    }
    // 查看 x 的颜色 -> x 的所属集合
    int find(int x) {
        return color[x];
    }
    // 将 x 和 y 染色为同一个颜色 -> 合并 x 和 y 的所属集合
    void merge(int x, int y) {
        if (color[x] == color[y]) return ;
        int colorY = color[y];
        for (int i = 0; i < size; i++) {
            if (color[i] == colorY) color[i] = color[x];
        }
    }
public:
    int *color, size;
};
```

Quick Union

```
class UnionFind {
public:
    UnionFind(int n) {
        size = n;
        father = new int[n];
        for (int i = 0; i < n; i++) {
            father[i] = i;
        }
    }
    // 查看 x 的根结点 -> x 的所属集合
    int find(int x) {
        int root = x;
        while (root != father[root]) {
            root = father[root];
        }
        return root;
    }
    // 将 x 和 y 染色为同一个颜色 -> 合并 x 和 y 的所属集合
    void merge(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX == rootY) return ;
        father[rootX] = rootY;
    }
public:
    int *father, size;
};
```

Weighted Quick Union

```
class UnionFind {
public:
    UnionFind(int n) {
        father = new int[n];
        treeSize = new int[n];
        size = n;
        for (int i = 0; i < n; i++) {
            father[i] = i;
            treeSize[i] = 1;
        }
    }
    int find(int x) {
        int root = x;
        while (father[root] != root) {
            root = father[root];
        }
        return root;
    }
    void merge(int x, int y) {
        int fx = find(x);
        int fy = find(y);
        if (fx == fy) return ;
        if (treeSize[fx] < treeSize[fy]) {
            father[fx] = fy;
            treeSize[fy] += treeSize[fx];
        }
        else {
            father[fy] = fx;
            treeSize[fx] += treeSize[fy];
        }
    }
public:
    int *father, *treeSize, size;
};
```

Weighted Quick Union with Path Compression

```
class UnionFind {
public:
    UnionFind(int n) {
        size = n;
        father = new int[n];
        treeSize = new int[n];
        for (int i = 0; i < n; i++) {
            father[i] = i;
            treeSize[i] = 1;
        }
    }
    // 查看 x 的根结点 -> x 的所属集合
    int find(int x) {
        int root = x;
        while (root != father[root]) {
            root = father[root];
        }
        while (x != root) {
            int fx = father[x];
            father[x] = root;
            x = fx;
        }
        return root;
    }
    // 将 x 和 y 染色为同一个颜色 -> 合并 x 和 y 的所属集合
    void merge(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX == rootY) return ;
        if (treeSize[rootX] < treeSize[rootY]) {
            father[rootX] = rootY;
            treeSize[rootY] += treeSize[rootX];
        }
        else {
            father[rootY] = rootX;
            treeSize[rootX] += treeSize[rootY];
        }
    }
public:
    int *father, *treeSize, size;
};
```

```

```

547. 省份数量

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

```cpp
class Solution {
public:
    int findCircleNum(vector<vector<int>>& isConnected) {
        int n = isConnected.size();
        UnionFind uf(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (isConnected[i][j] == 1) uf.merge(i, j);
            }
        }
        // 集合的数量
        return uf.setCnt;
    }
};
```

200. 岛屿数量

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

```cpp
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        int n = grid.size(), m = grid[0].size();
        #define id(i, j) ((i) * m + (j))
        UnionFind uf{n * m};
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == '0') continue;
                // 向下合并
                if (i + 1 < n && grid[i + 1][j] == '1') {
                    uf.merge(id(i, j), id(i + 1, j));
                }
                // 向右合并
                if (j + 1 < m && grid[i][j + 1] == '1') {
                    uf.merge(id(i, j), id(i, j + 1));
                }
            }
        }
        int cnt = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == '0') continue;
                if (uf.father[id(i, j)] == id(i, j))
                    cnt++;
            }
        }
        return cnt;
    }
};
```

990. 等式方程的可满足性

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

```
class Solution {
public:
    bool equationsPossible(vector<string>& equations) {
        UnionFind uf(26);
        for (int i = 0; i < equations.size(); i++) {
            int x = equations[i][0] - 'a';
            int y = equations[i][3] - 'a';
            if (equations[i][1] == '=') uf.merge(x, y);
        }
        for (int i = 0; i < equations.size(); i++) {
            int x = equations[i][0] - 'a';
            int y = equations[i][3] - 'a';
            if (equations[i][1] == '!' && uf.find(x) == uf.find(y))
                return false;
        }

        return true;
    }
};
```

1319. 连通网络的操作次数

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

```
class Solution {
public:
    int makeConnected(int n, vector<vector<int>>& connections) {
        UnionFind uf(n);
        int left = 0;
        for (int i = 0; i < connections.size(); i++) {
            int x = connections[i][0];
            int y = connections[i][1];
            if (uf.find(x) == uf.find(y)) left++;
            else uf.merge(x, y);
        }
        int cnt = 0;
        for (int i = 0; i < n; i++) {
            if (uf.father[i] == i) cnt++;
        }
        return left >= cnt - 1 ? cnt - 1 : -1;
    }
};
```

684. 冗余连接

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

```
class Solution {
public:
    vector<int> findRedundantConnection(vector<vector<int>>& edges) {
        UnionFind uf{(int)edges.size()};
        vector<int> ans;
        for (int i = 0; i < edges.size(); i++) {
            // -1 ==> 1~n 0~(n-1)
            int x = edges[i][0] - 1;
            int y = edges[i][1] - 1;
            if (uf.find(x) == uf.find(y)) {
                ans = edges[i];
                break;
            }
            uf.merge(x, y);
        }
        return ans;
    }
};
```

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

```
class Solution {
public:
    int removeStones(vector<vector<int>>& stones) {
        // Plan A: 将每个石头看成节点 O(n^2)
        int n = stones.size();
        UnionFind uf(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (stones[i][0] == stones[j][0] ||
                    stones[i][1] == stones[j][1]) {
                    uf.merge(i, j);
                }
            }
        }
        int cnt = 0;
        for (int i = 0; i < n; i++) {
            if (uf.find(i) == i) cnt++;
        }
        return n - cnt;
        // Plan B: 将每条直线视为结点
        unordered_set<int> s; // 参与运算的直线 O(m)
        int n = stones.size();
        int m = 10001;
        UnionFind uf(2 * m);
        for (int i = 0; i < n; i++) {
            uf.merge(stones[i][0], stones[i][1] + m);
            s.insert(stones[i][0]);
            s.insert(stones[i][1] + m);
        }
        int cnt = 0;
        for (int i = 0; i < 2 * m; i++) {
            if (uf.find(i) == i && s.count(i)) cnt++;
        }
        return n - cnt;
    }
};
```

1202. 交换字符串中的元素

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

```
class Solution {
public:
    string smallestStringWithSwaps(string s, vector<vector<int>>& pairs) {
        // 将 数组的下标 视为 并查集中的结点
        int n = s.size();
        UnionFind uf(n);
        for (int i = 0; i < pairs.size(); i++) {
            uf.merge(pairs[i][0], pairs[i][1]);
        }
        // 最小堆
        priority_queue<char, vector<char>, greater<char>> h[n];
        for (int i = 0; i < n; i++) {
            h[uf.find(i)].push(s[i]);
        }
        string ans = "";
        for (int i = 0; i < n; i++) {
            ans += h[uf.find(i)].top();
            h[uf.find(i)].pop();
        }
        return ans;
    }
};
```

765. 情侣牵手

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

```
class Solution {
public:
    int minSwapsCouples(vector<int>& row) {
        UnionFind uf((int)row.size() / 2);
        for (int i = 0; i < row.size(); i += 2) {
            int x = row[i] / 2;
            int y = row[i + 1] / 2;
            uf.merge(x, y);
        }
        int cnt = 0;
        for (int i = 0; i < uf.size; i++) {
            if (uf.father[i] == i) cnt++;
        }
        return row.size() / 2 - cnt;
    }
};
```