In [14]:

```python
### Union-Find 并查集,联通图模板
class UnionFind():
    def __init__(self,n):
        self.count = n 记录连通块
        self.parent = [i for i in range(n)]初始化根为自己
        self.size = [1]*n 大小
    def union(self,p,q):
        rootP = self.find(p)
        rootQ = self.find(q)
        if rootP == rootQ: return
        if self.size[rootP] > self.size[rootQ]:
            self.parent[rootQ] = rootP
            self.size[rootP] += self.size[rootQ]
        else:
            self.parent[rootP] = rootQ
            self.size[rootQ] += self.size[rootP]
        self.count -= 1
    def find(self,x):
        while self.parent[x] != x:
            self.parent[x] = self.parent[self.parent[x]]
            x = self.parent[x]
        return self.parent[x]
    def connected(self,p,q):
        return self.find(q) == self.find(p)
un = UnionFind(5)
un.union(2,3)
un.union(1,2)
un.count,un.parent,un.connected(1,3)
```

Out[14]:

```
(3, [0, 3, 3, 3, 4], True)
```

In [ ]:

```
## 200 运用,岛屿的数量
class UnionFind():
    def __init__(self,n):
        self.count = n
        self.parent = [i for i in range(n)]
        self.size = [1]*n
    def union(self,p,q):
        rootP = self.find(p)
        rootQ = self.find(q)
        if rootP == rootQ: return
        if self.size[rootP] > self.size[rootQ]:
            self.parent[rootQ] = rootP
            self.size[rootP] += self.size[rootQ]
        else:
            self.parent[rootP] = rootQ
            self.size[rootQ] += self.size[rootP]
        self.count -= 1
    def find(self,x):
        while self.parent[x] != x:
            self.parent[x] = self.parent[self.parent[x]]
            x = self.parent[x]
        return self.parent[x]
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        # Union-Find
        n, m = len(grid), len(grid[0])
        ocean = 0
        uf = UnionFind(n*m)
        for i in range(n):
            for j in range(m):
                # 统计水的个数
                if grid[i][j] == "0": 独立连通块
                    ocean += 1
                else:
                    # 只需向右和向下查看,其他的访问过了
                    if i+1 < n and grid[i+1][j]=="1":  # 下方
                        uf.union(i*m+j, (i+1)*m+j)
                    if j+1 < m and grid[i][j+1]=="1":  # 右方
                        uf.union(i*m+j, i*m+(j+1))
                    # 因为遇到是‘0’不会改变parent中的值，所以每个‘0’都是一个独立的联通块
        return uf.count-ocean
```

[i][j] 行 j 列
在 i*m+j
位置上

In [ ]:

```python
# 547 盆友圈
class UnionFind():
    def __init__(self,n):
        self.count = n
        self.parent = [i for i in range(n)]
        self.size = [1]*n
    def union(self,p,q):
        rootP = self.find(p)
        rootQ = self.find(q)
        if rootP == rootQ: return
        if self.size[rootP] > self.size[rootQ]:
            self.parent[rootQ] = rootP
            self.size[rootP] += self.size[rootQ]
        else:
            self.parent[rootP] = rootQ
            self.size[rootQ] += self.size[rootP]
        self.count -= 1
    def find(self,x):
        while self.parent[x] != x:
            self.parent[x] = self.parent[self.parent[x]]
            x = self.parent[x]
        return self.parent[x]

class Solution:
    def findCircleNum(self, isConnected: List[List[int]]) -> int:
        n = len(isConnected)
        uf = UnionFind(n)
        for i in range(n):
            for j in range(i+1,n): # 向下面的城市查，避免重复
                if isConnected[i][j]:  # '1'
                    uf.union(i,j)
        return uf.count
```

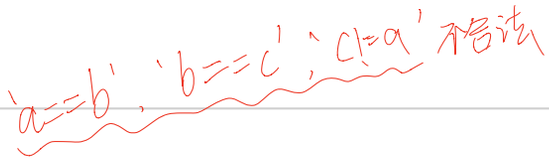
In [ ]:

```python
# 1202 交换排序
class UnionFind():
    def __init__(self,n):
        self.count = n
        self.parent = [i for i in range(n)]
        self.size = [1]*n
    def union(self,p,q):
        rootP = self.find(p)
        rootQ = self.find(q)
        if rootP == rootQ: return
        if self.size[rootP] > self.size[rootQ]:
            self.parent[rootQ] = rootP
            self.size[rootP] += self.size[rootQ]
        else:
            self.parent[rootP] = rootQ
            self.size[rootQ] += self.size[rootP]
        self.count -= 1
    def find(self,x):
        while self.parent[x] != x:
            self.parent[x] = self.parent[self.parent[x]]
            x = self.parent[x]
        return self.parent[x]
class Solution:
    def smallestStringWithSwaps(self, s: str, pairs: List[List[int]]) -> str:
        # 可交换的区域（并查集）进行排序
        uf = UnionFind(len(s))
        for x,y in pairs:
            uf.union(x,y)

        # 获得联通集合
        dic = {}
        for node in range(len(s)):
            # 存成字典
            if uf.find(node) not in dic: dic[uf.find(node)] = [node]
            else: dic[uf.find(node)].append(node)
        res = list(s)
        for nodes in dic.values():
            indices = nodes
            string = sorted(res[node] for node in nodes)
            for i,ch in zip(indices,string):
                # 取出再插入
                res[i] = ch
        return "".join(res)

```

In [ ]:




```python
# 990 判定合法性
class UnionFind():
    def __init__(self,n):
        self.count = n
        self.parent = [i for i in range(n)]
        self.size = [1]*n
    def union(self,p,q):
        rootP = self.find(p)
        rootQ = self.find(q)
        if rootP == rootQ: return
        if self.size[rootP] > self.size[rootQ]:
            self.parent[rootQ] = rootP
            self.size[rootP] += self.size[rootQ]
        else:
            self.parent[rootP] = rootQ
            self.size[rootQ] += self.size[rootP]
        self.count -= 1
    def find(self,x):
        while self.parent[x] != x:
            self.parent[x] = self.parent[self.parent[x]]
            x = self.parent[x]
        return self.parent[x]
    def connected(self,p,q):
        return self.find(q) == self.find(p)
class Solution:
    def equationsPossible(self, equations: List[str]) -> bool:
        uf = UnionFind(26)
        for eq in equations:
            if eq[1] == '=':
                x = ord(eq[0])
                y = ord(eq[-1])
                uf.union(x-ord('a'),y-ord('a'))  # ord字符转ASCII,chr反
        for eq in equations:
            if eq[1] == '!':
                x = ord(eq[0])
                y = ord(eq[-1])
                if uf.connected(x-ord('a'),y-ord('a')):
                    return False
        return True
```

In [ ]:

```python
####  高效求解数学问题
# 204 高效求素数
def countPrimes(n):
    isPrimes = [True]*n
    for i in range(2,n):
        if isPrimes[i]:
            for j in range(i*i,n,i):
                isPrimes[j] = False
    return isPrimes[2:].count(True)  # 不包含0,1,n

# 372 高效求mod
def superPow(self, a: int, b: List[int]) -> int:
    base = 1337
    def mypow(a,k):
        a %= base
        res = 1
        for _ in range(k):
            res *= a
            res %= base
        return res

        # 高效求幂
        if k==0: return 1
        a %= base
        if k % 2 == 0:
            sub = mypow(a,k/2)
            return (sub*sub) % base
        else:
            return (a*mypow(a,k-1))%base

    if b == []: return 1
    part1 = mypow(a,b.pop())
    part2 = mypow(self.superPow(a,b),10)
    return (part1*part2) % base
```