



In [14]:

```
1  ### Union-Find 并查集, 联通图模板
2  class UnionFind():
3      def __init__(self, n):
4          self.count = n 记录连通块
5          self.parent = [i for i in range(n)] 初始化根为自己
6          self.size = [1]*n 大小
7      def union(self, p, q):
8          rootP = self.find(p)
9          rootQ = self.find(q)
10         if rootP == rootQ: return
11         if self.size[rootP] > self.size[rootQ]:
12             self.parent[rootQ] = rootP
13             self.size[rootP] += self.size[rootQ]
14         else:
15             self.parent[rootP] = rootQ
16             self.size[rootQ] += self.size[rootP]
17         self.count -= 1
18     def find(self, x):
19         while self.parent[x] != x:
20             self.parent[x] = self.parent[self.parent[x]]
21             x = self.parent[x]
22         return self.parent[x]
23     def connected(self, p, q):
24         return self.find(q) == self.find(p)
25 un = UnionFind(5)
26 un.union(2, 3)
27 un.union(1, 2)
28 un.count, un.parent, un.connected(1, 3)
```

Out[14]:

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

In []:

```

1  ## 200 运用, 岛屿的数量
2  class UnionFind():
3      def __init__(self, n):
4          self.count = n
5          self.parent = [i for i in range(n)]
6          self.size = [1]*n
7      def union(self, p, q):
8          rootP = self.find(p)
9          rootQ = self.find(q)
10         if rootP == rootQ: return
11         if self.size[rootP] > self.size[rootQ]:
12             self.parent[rootQ] = rootP
13             self.size[rootP] += self.size[rootQ]
14         else:
15             self.parent[rootP] = rootQ
16             self.size[rootQ] += self.size[rootP]
17         self.count -= 1
18     def find(self, x):
19         while self.parent[x] != x:
20             self.parent[x] = self.parent[self.parent[x]]
21             x = self.parent[x]
22         return self.parent[x]
23 class Solution:
24     def numIslands(self, grid: List[List[str]]) -> int:
25         # Union-Find
26         n, m = len(grid), len(grid[0])
27         ocean = 0
28         uf = UnionFind(n*m)
29         for i in range(n):
30             for j in range(m):
31                 # 统计水的个数
32                 if grid[i][j] == "0": 独立连通块
33                     ocean += 1
34                 else:
35                     # 只需向右和向下查看, 其他的访问过了
36                     if i+1 < n and grid[i+1][j] == "1": # 下方
37                         uf.union(i*m+j, (i+1)*m+j)
38                     if j+1 < m and grid[i][j+1] == "1": # 右方
39                         uf.union(i*m+j, i*m+(j+1))
40                     # 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
41         return uf.count-ocean

```

1行j列
在 i*m+j
位置上

In []:

```

1 # 547 盆友圈
2 class UnionFind():
3     def __init__(self, n):
4         self.count = n
5         self.parent = [i for i in range(n)]
6         self.size = [1]*n
7     def union(self, p, q):
8         rootP = self.find(p)
9         rootQ = self.find(q)
10        if rootP == rootQ: return
11        if self.size[rootP] > self.size[rootQ]:
12            self.parent[rootQ] = rootP
13            self.size[rootP] += self.size[rootQ]
14        else:
15            self.parent[rootP] = rootQ
16            self.size[rootQ] += self.size[rootP]
17        self.count -= 1
18    def find(self, x):
19        while self.parent[x] != x:
20            self.parent[x] = self.parent[self.parent[x]]
21            x = self.parent[x]
22        return self.parent[x]
23
24 class Solution:  切分差矩阵
25     def findCircleNum(self, isConnected: List[List[int]]) -> int:
26         n = len(isConnected)
27         uf = UnionFind(n)
28         for i in range(n):
29             for j in range(i+1, n): # 向下面的城市查，避免重复
30                 if isConnected[i][j]: # '1'
31                     uf.union(i, j)
32         return uf.count

```



In []:

```
1 # 1202 交换排序
2 class UnionFind():
3     def __init__(self, n):
4         self.count = n
5         self.parent = [i for i in range(n)]
6         self.size = [1]*n
7     def union(self, p, q):
8         rootP = self.find(p)
9         rootQ = self.find(q)
10        if rootP == rootQ: return
11        if self.size[rootP] > self.size[rootQ]:
12            self.parent[rootQ] = rootP
13            self.size[rootP] += self.size[rootQ]
14        else:
15            self.parent[rootP] = rootQ
16            self.size[rootQ] += self.size[rootP]
17        self.count -= 1
18    def find(self, x):
19        while self.parent[x] != x:
20            self.parent[x] = self.parent[self.parent[x]]
21            x = self.parent[x]
22        return self.parent[x]
23 class Solution:
24     def smallestStringWithSwaps(self, s: str, pairs: List[List[int]]) -> str:
25         # 可交换的区域（并查集）进行排序
26         uf = UnionFind(len(s))
27         for x, y in pairs:
28             uf.union(x, y)
29
30         # 获得联通集合
31         dic = {}
32         for node in range(len(s)):
33             # 存成字典
34             if uf.find(node) not in dic: dic[uf.find(node)] = [node]
35             else: dic[uf.find(node)].append(node)
36         res = list(s)
37         for nodes in dic.values():
38             indices = nodes
39             string = sorted(res[node] for node in nodes)
40             for i, ch in zip(indices, string):
41                 # 取出再插入
42                 res[i] = ch
43         return "".join(res)
44
```

In []:

'a==b', 'b==c', 'c!=a' 合法

```

1 # 990 判定合法性
2 class UnionFind():
3     def __init__(self, n):
4         self.count = n
5         self.parent = [i for i in range(n)]
6         self.size = [1]*n
7     def union(self, p, q):
8         rootP = self.find(p)
9         rootQ = self.find(q)
10        if rootP == rootQ: return
11        if self.size[rootP] > self.size[rootQ]:
12            self.parent[rootQ] = rootP
13            self.size[rootP] += self.size[rootQ]
14        else:
15            self.parent[rootP] = rootQ
16            self.size[rootQ] += self.size[rootP]
17        self.count -= 1
18    def find(self, x):
19        while self.parent[x] != x:
20            self.parent[x] = self.parent[self.parent[x]]
21            x = self.parent[x]
22        return self.parent[x]
23    def connected(self, p, q):
24        return self.find(q) == self.find(p)
25 class Solution:
26     def equationsPossible(self, equations: List[str]) -> bool:
27         uf = UnionFind(26)
28         for eq in equations:
29             if eq[1] == '=':
30                 x = ord(eq[0])
31                 y = ord(eq[-1])
32                 uf.union(x-ord('a'), y-ord('a')) # ord字符转ASCII, chr反
33         for eq in equations:
34             if eq[1] == '!':
35                 x = ord(eq[0])
36                 y = ord(eq[-1])
37                 if uf.connected(x-ord('a'), y-ord('a')):
38                     return False
39         return True

```

In []:

```
1 ##### 高效求解数学问题
2 # 204 高效求素数
3 def countPrimes(n):
4     isPrimes = [True]*n
5     for i in range(2,n):
6         if isPrimes[i]:
7             for j in range(i*i,n,i):
8                 isPrimes[j] = False
9     return isPrimes[2:].count(True) # 不包含0,1,n
10
11 # 372 高效求mod
12 def superPow(self, a: int, b: List[int]) -> int:
13     base = 1337
14     def mypow(a,k):
15         a %= base
16         res = 1
17         for _ in range(k):
18             res *= a
19             res %= base
20         return res
21
22     # 高效求幂
23     if k==0: return 1
24     a %= base
25     if k % 2 == 0:
26         sub = mypow(a,k/2)
27         return (sub*sub) % base
28     else:
29         return (a*mypow(a,k-1))%base
30
31     if b == []: return 1
32     part1 = mypow(a,b.pop())
33     part2 = superPow(self,a,b),10
34     return (part1*part2) % base
```

2, 3, 5, 7
素数 倍数 -> 非素数

开始

$$ab \% k = (a \% k)(b \% k) \% k$$

a