班上有 **N** 名学生。其中有些人是朋友，有些则不是。他们的友谊具有是传递性。如果已知 A 是 B 的朋友，B 是 C 的朋友，那么我们可以认为 A 也是 C 的朋友。所谓的朋友圈，是指所有朋友的集合。

给定一个 **N * N** 的矩阵 **M**，表示班级中学生之间的朋友关系。如果M[i][j] = 1，表示已知第 i 个和 j 个学生**互为**朋友关系，否则为不知道。你必须输出所有学生中的已知的朋友圈总数。

示例 1：

```
输入：
[[1,1,0],
 [1,1,0],
 [0,0,1]]
输出：2
解释：已知学生 0 和学生 1 互为朋友，他们在一个朋友圈。
第2个学生自己在一个朋友圈。所以返回 2 。
```

示例 2：

```
输入：
[[1,1,0],
 [1,1,1],
 [0,1,1]]
输出：1
解释：已知学生 0 和学生 1 互为朋友，学生 1 和学生 2 互为朋友，所以学生 0 和学生 2 也是朋友，所以他们三个在一个朋友圈，返回 1 。
```

解题思路：

BFS，DFS，类似于岛屿数量问题

并查集：（关键在于记住模版）

1 创建并查集

2 M[i][j]=1时，表面，i，j是朋友，将其合并起来

3 返回并查集中parent的数量

```python
class Solution(object):
    def findCircleNum(self, M):
        """
        :type M: List[List[int]]
        :rtype: int
        """
        # 并查集
        # 1 并查集创建起来
        # 2 遍历矩阵，合并i，j
         # 3 看整个n里面有多少个不同的parent
         if not M : return 0

         n = len(M)
         p = [i for i in range(n)]

         for i in range(n):
             for j in range(n):
                 if M[i][j] == 1:
                     self._union(p, i, j)

         return len(set([self._parent(p, i) for i in range(n)]))

    def _union(self, p, i, j):
        p1 = self._parent(p, i)
        p2 = self._parent(p, j)
        p[p1] = p2

```

```
28
29      def _parent(self, p, i):
30          root = i
31          while p[root] != root:
32              root = p[root]
33          while p[i] != i: #
34              x = i; i = p[i]; p[x] = root
35          return root
```