138. Copy List with Random Pointer

```
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        map_node = {}
        map_node[None] = None
        dummy = Node(0, None, None)
        dummy_memo = dummy
        head_memo = head
        while head:
            dummy.next = Node(head.val, None, None)
            map_node[head] = dummy.next
            dummy = dummy.next
            head = head.next
        head = head_memo
        dummy = dummy_memo
        while head:
            dummy.next.random = map_node[head.random]
            dummy = dummy.next
            head = head.next
        return dummy_memo.next
```

133. Clone Graph

```
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        graph_map = {}
        def dfs(node):
          if node.val in graph_map:
            return graph_map[node.val]
          new_node = Node(node.val, [])
          graph_map[node.val] = new_node
          for n in node.neighbors:
            new_node.neighbors.append(dfs(n))
          return new_node
        return dfs(node)
```

200. Number of Islands

```
class Solution:
```

```
    def numIslands(self, grid: List[List[str]]) -> int:
        count = 0
        if not grid or not grid[0]:
          return 0
        rows, cols = len(grid), len(grid[0])

        def visit(i, j):
          grid[i][j] = '0'
          for d_i, d_j in [[1, 0], [-1, 0], [0, -1], [0, 1]]:
            new_i, new_j = i + d_i, j + d_j
            if 0 <= new_i < rows and 0 <= new_j < cols and
grid[new_i][new_j] == '1':
              visit(new_i, new_j)



        for i in range(rows):
          for j in range(cols):
            if grid[i][j] == '1':
                count += 1
                visit(i, j)

        return count
```

547. Friend Circles

```
class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:
        rows = len(M)
        count = rows
        self.parent = {}

        for i in range(rows):
            self.parent[i] = i

        for i in range(rows):
            for j in range(i + 1, rows):
                if M[i][j] == 1:
                    parent_i = self.findParent(i)
                    parent_j = self.findParent(j)
                    if parent_i != parent_j:
                        self.parent[parent_j] = parent_i
                        count -= 1
        return count

    def findParent(self, i):
        while i != self.parent[i]:
            i = self.parent[i]
        return i
```

695. Max Area of Island

```
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        if not grid or not grid[0]:
            return 0
        max_area = 0

        def helper(grid, i, j):
            if 0 <= i < len(grid) and 0 <= j < len(grid[0]) and
grid[i][j]:
                grid[i][j] = 0
                return 1 + helper(grid, i + 1, j) + helper(grid, i - 1,
j) + helper(grid, i, j - 1) + helper(grid, i, j + 1)
            return 0

        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == 1:
                    max_area = max(max_area, helper(grid, i, j))
        return max_area
```