695. Max Area of Island
We will use dfs to solve this question. TC is O(n), SC is O(1)

```
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
      def dfs(cur_i, cur_j, rows, cols):
        cur = 0

        if 0 <= cur_i < rows and 0 <= cur_j < cols and grid[cur_i][cur_j] == 1:
          cur += 1
          grid[cur_i][cur_j] = 0
          for d_i, d_j in [[0, 1], [0, -1], [1, 0], [-1, 0]]:
            new_i, new_j = cur_i + d_i, cur_j + d_j
            cur += dfs(new_i, new_j, rows, cols)
        return cur

      if not grid or not grid[0]:
        return 0
      rows = len(grid)
      cols = len(grid[0])
      max_area = 0
      for i in range(rows):
        for j in range(cols):
          if grid[i][j] == 1:
            max_area = max(max_area, dfs(i, j, rows, cols))
      return max_area
```

207. Course Schedule
We will use topological algo to calculate all courses' number. TC is O(n^2) SC is O(n^2)

```
from collections import defaultdict
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        indegree = defaultdict(set)
        outdegree = defaultdict(set)
        for s, f in prerequisites:
          indegree[s].add(f)
          outdegree[f].add(s)
        begin = set(range(numCourses)) - set(indegree.keys())
        total = len(begin)
        while begin:
          next_ite = set()
          for i in begin:
            if i in outdegree:
              for j in outdegree[i]:
```

```
            indegree[j].remove(i)
            if len(indegree[j]) == 0:
              next_ite.add(j)
              total += 1
        begin = next_ite
      return total == numCourses
```

210. Course Schedule II
The same as the previous one.

```
from collections import defaultdict
class Solution:
    def findOrder(self, numCourses: int, prerequisites: List[List[int]]) -> List[int]:
        indegree = defaultdict(set)
        outdegree = defaultdict(set)
        for s, f in prerequisites:
          indegree[s].add(f)
          outdegree[f].add(s)
        begin = set(range(numCourses)) - set(indegree.keys())
        result = list(begin)
        total = len(begin)
        while begin:
          next_ite = set()
          for i in begin:
            if i in outdegree:
              for j in outdegree[i]:
                indegree[j].remove(i)
                if len(indegree[j]) == 0:
                  next_ite.add(j)
                  total += 1
                  result.append(j)
          begin = next_ite
        return result if total == numCourses else []
```

802. Find Eventual Safe States
The same as the last one.

```
from collections import defaultdict
class Solution:
    def eventualSafeNodes(self, graph: List[List[int]]) -> List[int]:
        indegree = defaultdict(set)
        outdegree = defaultdict(set)
        for i, nodes in enumerate(graph):
          for j in nodes:
```

```
            indegree[j].add(i)
            outdegree[i].add(j)

        begin = set(range(len(graph))) - set(outdegree.keys())
        result = list(begin)
        total = len(begin)
        while begin:
          next_ite = set()
          for i in begin:
            if i in indegree:
              for j in indegree[i]:
                outdegree[j].remove(i)
                if len(outdegree[j]) == 0:
                  next_ite.add(j)
                  total += 1
                  result.append(j)
          begin = next_ite
        return sorted(result)
```

399. Evaluate Division
We use Floyd and TC is O(n^3) SC is O(n^2)

```
from collections import defaultdict
class Solution:
    def calcEquation(self, equations: List[List[str]], values: List[float], queries: List[List[str]]) ->
List[float]:
        memo = defaultdict(lambda: defaultdict(int))
        res = []
        for i, e in enumerate(equations):
          a, b = e
          memo[a][b] = values[i]
          memo[b][a] = 1.0 / values[i]
          memo[a][a] = 1
          memo[b][b] = 1

        for i in memo:
          for j in memo[i]:
            for k in memo[i]:
              memo[j][k] = memo[j][i] * memo[i][k]
              memo[k][j] = 1.0 / memo[j][k]
        for a, b in queries:
          res.append(memo[a][b] if memo[a][b] != 0 else -1.0)
        return res
```