1. class Solution:

    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:

        onPath = [False] \* numCourses

        visited = [False] \* numCourses

        hasCycle = [False]

        graph = self.buildGraph(numCourses, prerequisites)

        def traverse(s):

            if onPath[s]:

                hasCycle[0] = True

            if visited[s] or hasCycle[0]:

                return

            visited[s] = True

            onPath[s] = True

            for t in graph[s]:

                traverse(t)

            onPath[s] = False

        for i in range(numCourses):

            traverse(i)

        return not hasCycle[0]

    def buildGraph(self, numCourses, prerequisites):

        graph = [[] for \_ in range(numCourses)]

        for from\_, to in prerequisites:

            graph[from\_].append(to)

        return graph

2. class Solution:

    def numIslands(self, grid: List[List[str]]) -> int:

        res = 0

        m = len(grid)

        n = len(grid[0])

        for i in range(m):

            for j in range(n):

                if grid[i][j] == '1':

                    res += 1

                    self.dfs(grid, i, j)

        return res

    def dfs(self, grid: List[List[str]], i: int, j: int) -> None:

        m = len(grid)

        n = len(grid[0])

        if i < 0 or j < 0 or i >= m or j >= n:

            return

        if grid[i][j] == '0':

            return

        grid[i][j] = '0'

        self.dfs(grid, i + 1, j)

        self.dfs(grid, i, j + 1)

        self.dfs(grid, i - 1, j)

        self.dfs(grid, i, j - 1)

3.

class Solution:

    def kthSmallest(self, root: TreeNode, k: int) -> int:

        self.res = 0

        self.rank = 0

        self.traverse(root, k)

        return self.res

    def traverse(self, root: TreeNode, k: int) -> None:

        if root is None:

            return

        self.traverse(root.left, k)

        self.rank += 1

        if k == self.rank:

            self.res = root.val

            return

        self.traverse(root.right, k)