113. Path Sum II.

class Solution:

    def pathSum(self, root: Optional[TreeNode], targetSum: int) -> List[List[int]]:

        check=deque()

        check.append((root,targetSum,[]))

        ans=[]

        while check:

            node ,tSum, path=check.popleft()

            if not node:

                continue

            if not node.left and not node.right and node.val==tSum:

                ans.append((path+[node.val]))

            check.append((node.left,tSum-node.val,path+[node.val]))

            check.append((node.right,tSum-node.val,path+[node.val]))

        return ans

class Solution:

    def pathSum(self, root: Optional[TreeNode], targetSum: int) -> List[List[int]]:

        ans=[]

        def find(node,path,ans,tSum):

            if not node:

                return

            path.append(node.val)

            if not node.left and not node.right and node.val==tSum:

                ans.append(path[:])

            find(node.left,path,ans,tSum-node.val)

            find(node.right,path,ans,tSum-node.val)

            path.pop()

        find(root,[],ans,targetSum)

        return ans

2415. Reverse Odd Levels of Binary Tree

class Solution:

    def reverseOddLevels(self, root: Optional[TreeNode]) -> Optional[TreeNode]:

        if not root:

            return

        def find(lnode,rnode,level):

            if not lnode and not rnode :

                return

            if lnode and rnode and level%2==0:

                lnode.val,rnode.val=rnode.val,lnode.val

            find(lnode.left,rnode.right,level+1)

            find(lnode.right,rnode.left,level+1)

        find(root.left,root.right,0)

        return root

class Solution:

    def reverseOddLevels(self, root: Optional[TreeNode]) -> Optional[TreeNode]:

        stack=[root]

        level=0

        while stack:

            level+=1

            newStack=[]

            for node in stack:

                if node.left:

                    newStack.append(node.left)

                if node.right:

                    newStack.append(node.right)

            if level%2!=0:

                for i in range((len(newStack)+1)//2):

                    newStack[i].val,newStack[len(newStack)-1-i].val=newStack[len(newStack)-1-i].val,newStack[i].val

            stack=newStack

        return root

Leetcode 97. Interleaving String

class Solution:

    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:

        if len(s1) + len(s2) !=len(s3):

            return False

        def find(i,j,k):

            if i >= len(s1) and j>=len(s2) and k>=len(s3):

                return True

            if i<len(s1) and s1[i]==s3[k]:

                if find(i+1,j,k+1):

                    return True

            if j<len(s2) and s2[j]==s3[k]:

                if find(i,j+1,k+1):

                    return True

            return False

        return find(0,0,0)

class Solution:

    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:

        if len(s1) + len(s2) != len(s3):

            return False

        check=[[False]\*(len(s2)+1) for \_ in range(len(s1)+1)]

        check[-1][-1]=True

        for i in range(len(s1),-1,-1):

            for j in range(len(s2),-1,-1):

                if i < len (s1) and s1[i] == s3[i+j] and check[i+1][j]:

                    check[i][j]=True

                if j < len(s2) and s2[j] == s3[i+j] and check[i][j+1]:

                    check[i][j]=True

        return check[0][0]

Leetcode 1143. Longest Common Subsequence

class Solution:

    def longestCommonSubsequence(self, text1: str, text2: str) -> int:

        @cache

        def find(i,j):

            if i>=len(text1) or j>=len(text2):

                return 0

            ans=0

            if text1[i]==text2[j]:

                ans=1+find(i+1,j+1)

            else:

                ans=max(find(i+1,j+1), find(i,j+1), find(i+1,j))

            return ans

        return find(0,0)

class Solution:

    def longestCommonSubsequence(self, text1: str, text2: str) -> int:

        check=[[0]\*(len(text2)+1) for \_ in range(len(text1)+1)]

        for i in range(len(text1)-1,-1,-1):

            for j in range(len(text2)-1,-1,-1):

                if text1[i]==text2[j]:

                    check[i][j]=1+check[i+1][j+1]

                else:

                    check[i][j]=max(check[i+1][j+1],check[i][j+1],check[i+1][j])

        return check[0][0]

Leetcode 92. Reverse Linked List II.

class Solution:

    def reverseBetween(self, head: Optional[ListNode], left: int, right: int) -> Optional[ListNode]:

        ans=ListNode()

        cur=ans

        for \_ in range(1,left):

            cur.next=ListNode(head.val)

            cur=cur.next

            head=head.next

        reverse=None

        for \_ in range(left,right+1):

            nextHead=head.next

            head.next=reverse

            reverse=head

            head=nextHead

        while reverse:

            cur.next=ListNode(reverse.val)

            cur=cur.next

            reverse=reverse.next

        cur.next=head

        return ans.next

Leetcode 15. 3Sum

class Solution:

    def threeSum(self, nums: List[int]) -> List[List[int]]:

        ans = set()

        nums.sort()

        for i in range(len(nums)):

            if i>0 and nums[i]==nums[i-1]:

                continue

            l,r=i+1,len(nums)-1

            while l<r:

                total=nums[i]+nums[l]+nums[r]

                if total>0:

                    r-=1

                elif total<0:

                    l+=1

                else:

                    ans.add((nums[i],nums[l],nums[r]))

                    l+=1

                    r-=1

        return ans

class Solution:

    def threeSum(self, nums: List[int]) -> List[List[int]]:

        nums.sort()

        ans=set()

        n=len(nums)

        for i in range(n):

            if i>0 and nums[i]==nums[i-1]:

                continue

            for j in range(i+1,n):

                if j>i+1 and nums[j]==nums[j-1]:

                    continue

                for k in range(j+1,n):

                    if k>j+1 and nums[k]==nums[k-1]:

                        continue

                    if nums[i]+nums[j]+nums[k]==0:

                        ans.add((nums[i],nums[j],nums[k]))

        return ans

Leetcode 16. 3Sum Closest

class Solution:

    def threeSumClosest(self, nums: List[int], target: int) -> int:

        ans=0

        nums.sort()

        check=float("inf")

        for i in range(len(nums)):

            l,r=i+1,len(nums)-1

            while l<r:

                total=nums[i]+nums[l]+nums[r]

                if total==target:

                    return total

                elif total>target:

                    r-=1

                else:

                    l+=1

                diff=abs(total-target)

                if diff<check:

                    check=diff

                    ans=total

        return ans

Leetcode 18. 4Sum.

class Solution:

    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:

        ans=set()

        nums.sort()

        for i in range(len(nums)):

            for j in range(i+1,len(nums)):

                l,r=j+1,len(nums)-1

                while l<r:

                    total=nums[i]+nums[j]+nums[l]+nums[r]

                    if total>target:

                        r-=1

                    elif total<target:

                        l+=1

                    else:

                        ans.add((nums[i],nums[j],nums[l],nums[r]))

                        l+=1

                        r-=1

        return ans

Leetcode 199. Binary Tree Right Side View.

class Solution:

    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:

        ans=[]

        def all(root,level):

            if root:

                if len(ans)==level:

                    ans.append(root.val)

                all(root.right,level+1)

                all(root.left,level+1)

        all(root,0)

        return ans

class Solution:

    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:

        if not root:

            return []

        queue=deque()

        ans=[]

        queue.append(root)

        while queue:

            check=deque()

            val=0

            while queue:

                cur=queue.popleft()

                if cur.left:

                    check.append(cur.left)

                if cur.right:

                    check.append(cur.right)

                val=cur

            ans.append(val.val)

            queue=check

        return ans

Leetcode 11. Container With Most Water

class Solution:

    def maxArea(self, height: List[int]) -> int:

        left,right=0,len(height)-1

        ans=0

        while left<right:

            cur=min(height[left],height[right])\*(right-left)

            ans=max(ans,cur)

            if height[left]<height[right]:

                left+=1

            else:

                right-=1

        return ans

Leetcode 2305. Fair Distribution of Cookies.

class Solution:

    def distributeCookies(self, cookies: List[int], k: int) -> int:

        ans=float("inf")

        check=[0]\*k

        def find(i):

            nonlocal ans,check

            if i>=len(cookies):

                ans=min(ans,max(check))

                return

            if ans<=max(check):

                return

            for j in range(k):

                check[j]+=cookies[i]

                find(i+1)

                check[j]-=cookies[i]

        find(0)

        return ans

class Solution:

    def distributeCookies(self, cookies: List[int], k: int) -> int:

        ans=float("inf")

        check=[(0,[0]\*k)]

        while check:

            i,child=check.pop()

            if i>=len(cookies):

                ans=min(ans,max(child))

                continue

            elif ans<=max(cookies):

                continue

            for j in range(k):

                helper=child[:]

                helper[j]+=cookies[i]

                check.append((i+1,helper))

        return ans

Leetcode 119. Pascal Triangle II

class Solution:

    def getRow(self, rowIndex: int) -> List[int]:

        if rowIndex==0:

            return[1]

        elif rowIndex==1:

            return [1,1]

        def find(check,i):

            if i==rowIndex:

                return check

            helper=[1]\*(len(check)+1)

            for i in range(1,len(check)):

                helper[i]=check[i-1]+check[i]

            return find(helper,i+1)

        return find([1,1],0)

class Solution:

    def getRow(self, rowIndex: int) -> List[int]:

        if rowIndex==0:

            return [1]

        elif rowIndex==1:

            return [1,1]

        check=[1,1]

        while rowIndex>=2:

            rowIndex-=1

            newCheck=[1]\*(len(check)+1)

            for i in range(0,len(check)-1):

                newCheck[i+1]=check[i+1]+check[i]

            check=newCheck

        return check

Leetcode 404. Sum of Left Leaves.

class Solution:

    def sumOfLeftLeaves(self, root: Optional[TreeNode]) -> int:

        if not root:

            return 0

        ans=0

        def find(node,is\_left):

            nonlocal ans

            if not node:

                return

            if not node.left and not node.right:

                if is\_left:

                    ans+=node.val

                return

            find(node.left,True)

            find(node.right,False)

        find(root,False)

        return ans

class Solution:

    def sumOfLeftLeaves(self, root: Optional[TreeNode]) -> int:

        if not root:

            return 0

        ans=0

        check=deque()

        check.append((root,False))

        while check:

            node,is\_left=check.popleft()

            if not node:

                continue

            if not node.left and not node.right:

                if is\_left:

                    ans+=node.val

                continue

            check.append((node.left,True))

            check.append((node.right,False))

        return ans

Leetcode 695. Max Area of Island.

class Solution:

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

        rows , cols=len(grid), len(grid[0])

        ans=0

        visit=set()

        def find(i,j,check):

            if i<0 or i>=rows or j<0 or j>=cols or (i,j) in check or not grid[i][j]:

                return 0

            check.add((i,j))

            return 1+find(i+1,j,check) + find(i,j+1,check) + find(i-1,j,check) + find(i,j-1,check)

        for i in range(rows):

            for j in range(cols):

                if grid[i][j] not in visit:

                    ans=max(ans,find(i,j,visit))

        return ans

Leetcode 1306. Jump Game III

class Solution:

    def canReach(self, arr: List[int], start: int) -> bool:

        visit = set()

        def find(i):

            if i >= len(arr) or i < 0:

                return False

            if i in visit:

                return False

            visit.add(i)

            if arr[i] == 0:

                return True

            return find(i + arr[i]) or find(i - arr[i])

        return find(start)

class Solution:

    def canReach(self, arr: List[int], start: int) -> bool:

        visit=set()

        check=deque()

        check.append(start)

        while check:

            i=check.popleft()

            if i<0 or i>=len(arr) or i in visit:

                continue

            elif arr[i]==0:

                return True

            visit.add(i)

            check.append(arr[i]+i)

            check.append(i-arr[i])

        return False

Leetcode 56. Merge Intervals.

class Solution:

    def merge(self, intervals: List[List[int]]) -> List[List[int]]:

        intervals.sort(key=lambda x:x[0])

        ans=[intervals[0]]

        for l,r in intervals[1:]:

            pl,pr=ans[-1]

            if l<=pr and r<pr:

                continue

            elif l<=pr:

                ans.pop()

                ans.append([pl,r])

            else:

                ans.append([l,r])

        return ans

        intervals=sorted(intervals,key=lambda x:x[0])

        def find(i,ans):

            if i>=len(intervals):

                return ans

            l,r=intervals[i]

            pl,pr=ans[-1]

            if pr<l:

                ans.append([l,r])

            elif pr<=r:

                ans.pop()

                ans.append([pl,r])

            return find(i+1,ans)

        return find(1,[intervals[0]])

Leetcode 516. Longest Palindromic Subsequence.

class Solution:

    def longestPalindromeSubseq(self, s: str) -> int:

        @cache

        def find(i, j):

            if j < i:

                return 0

            elif i == j:

                return 1

            elif s[i] == s[j]:

                return 2 + find(i + 1, j - 1)

            return max(find(i + 1, j - 1), find(i + 1, j), find(i, j - 1))

        return find(0, len(s) - 1)

class Solution:

    def longestPalindromeSubseq(self, s: str) -> int:

        t = s[::-1]

        check = [[0] \* (len(s) + 1) for \_ in range(len(s) + 1)]

        for i in range(len(s), 0, -1):

            for j in range(len(s), 0, -1):

                if s[i - 1] == t[j - 1]:

                    check[i - 1][j - 1] = 1 + check[i][j]

                else:

                    check[i - 1][j - 1] = max(check[i - 1][j], check[i][j - 1], check[i][j])

        return check[0][0]

Leetcode 130. Surrounded Regions.

class Solution:

    def solve(self, board: List[List[str]]) -> None:

        """

        Do not return anything, modify board in-place instead.

        """

        rows, cols = len(board), len(board[0])

        check, visit = set(), set()

        for j in range(cols):

            if board[0][j] == "O":

                check.add((0, j))

            if board[rows - 1][j] == "O":

                check.add((rows - 1, j))

        for i in range(rows):

            if board[i][0] == "O":

                check.add((i, 0))

            if board[i][cols - 1] == "O":

                check.add((i, cols - 1))

        def find(i, j):

            if i < 0 or i >= rows or j < 0 or j >= cols or board[i][j] != "O" or (i, j) in visit:

                return

            visit.add((i, j))

            board[i][j] = "!"

            find(i + 1, j)

            find(i - 1, j)

            find(i, j + 1)

            find(i, j - 1)

        for i, j in check:

            if (i, j) not in visit:

                find(i, j)

        for i in range(rows):

            for j in range(cols):

                if board[i][j] == "O":

                    board[i][j] = "X"

                elif board[i][j] == "!":

                    board[i][j] = "O"

Leetcode 210. Course Schedule II.

class Solution:

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

        check = {cor : [] for cor in range(numCourses)}

        for cor, pre in prerequisites:

            check[cor].append(pre)

        ans, visit, cycle = [], set(), set()

        def find(cor):

            if cor in cycle:

                return False

            elif cor in visit:

                return True

            cycle.add(cor)

            for pre in check[cor]:

                if find(pre) == False:

                    return False

            cycle.remove(cor)

            visit.add(cor)

            ans.append(cor)

        for cor in range(numCourses):

            if find(cor) == False:

                return []

        return ans

class Solution:

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

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

        count = [0 for \_ in range(numCourses)]

        for cor, pre in prerequisites:

            check[pre].append(cor)

            count[cor] += 1

        stack = []

        for cor in range(numCourses):

            if count[cor] == 0:

                stack.append(cor)

        ans = []

        while stack:

            pre = stack.pop()

            ans.append(pre)

            for cor in check[pre]:

                count[cor] -= 1

                if count[cor] == 0:

                    stack.append(cor)

        if len(ans) == numCourses:

            return ans

        return []

Leetcode 63. Unique Paths II.

class Solution:

    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:

        rows, cols = len(obstacleGrid), len(obstacleGrid[0])

        @cache

        def find(i, j):

            if i >= rows or j >= cols or obstacleGrid[i][j]:

                return 0

            elif i == rows - 1 and j == cols - 1:

                return 1

            return find(i + 1, j) + find(i, j + 1)

        return find(0, 0)

        rows,cols=len(obstacleGrid),len(obstacleGrid[0])

        ans=[0]\*cols

        ans[-1]=1

        for r in range(rows-1,-1,-1):

            for c in range(cols-1,-1,-1):

                if obstacleGrid[r][c]:

                    ans[c]=0

                elif c+1<cols:

                    ans[c]+=ans[c+1]

        return ans[0]

Leetcode 213. House Robber II.

class Solution:

    def rob(self, nums: List[int]) -> int:

        if len(nums) == 0: return 0

        elif len(nums) <= 2: return max(nums)

        def find(house):

            check = [0] \* len(house)

            check[0] = house[0]

            check[1] = max(house[0], house[1])

            for n in range(2, len(house)):

                check[n] = max(check[n - 1], check[n - 2] + house[n])

            return check[-1]

        return max(find(nums[1:]), find(nums[:-1]))

class Solution:

    def rob(self, nums: List[int]) -> int:

        if len(nums)==1:

            return nums[0]

        def find(house):

            house1,house2=0,0

            for n in house:

                house1,house2=house2,max(n+house1,house2)

            return house2

        return max(find(nums[1:]),find(nums[:-1]))

Leetcode 1140. Stone Game II.

class Solution:

    def stoneGameII(self, piles: List[int]) -> int:

        def find(alice, i, m):

            if i >= len(piles):

                return 0

            total = 0

            ans = 0 if alice else float("inf")

            for x in range(1, 2 \* m + 1):

                if x + i > len(piles):

                    break

                total += piles[x + i - 1]

                if alice:

                    ans = max(ans, total + find(not alice, x + i, max(m, x)))

                else:

                    ans = min(ans, find(not alice, x + i, max(m, x)))

            return ans

        return find(True, 0, 1)

class Solution:

    def stoneGameII(self, piles: List[int]) -> int:

        check = [[0] \* (len(piles) + 1) for \_ in range(len(piles))]

        totals = [0] \* len(piles)

        totals[-1] = piles[-1]

        for i in range(len(piles) - 2, -1, -1):

            totals[i] = totals[i + 1] + piles[i]

        for i in range(len(piles) - 1, -1, -1):

            for m in range(1, len(piles) + 1):

                if i + 2 \* m >= len(piles):

                    check[i][m] = totals[i]

                else:

                    for x in range(1, 2 \* m + 1):

                        check[i][m] = max(check[i][m], totals[i] - check[i + x][max(x, m)])

        return check[0][1]

Leetcode 309. Best Time to Buy and Sell Stock with Cooldown.

class Solution:

    def maxProfit(self, prices: List[int]) -> int:

        def find(i, buying):

            if i >= len(prices):

                return 0

            ans = 0

            cooldown = find(i + 1, buying)

            if buying:

                buy = find(i + 1, not buying) - prices[i]

                ans = max(buy, cooldown)

            else:

                sell = find(i + 2, not buying) + prices[i]

                ans = max(sell, cooldown)

            return ans

        return find(0, True)

class Solution:

    def maxProfit(self, prices: List[int]) -> int:

        n = len(prices)

        buy = [0] \* n

        sell = [0] \* n

        rest = [0] \* n

        buy[0] = -prices[0]

        for i in range(1, n):

            buy[i] = max(buy[i - 1], rest[i - 1] - prices[i])

            sell[i] = buy[i - 1] + prices[i]

            rest[i] = max(rest[i - 1], sell[i - 1])

        return max(sell[-1], rest[-1])