二分查找

Aggressive cows

```
n,c=map(int,input().split())
list1=[]
for _ in range(n):
    list1.append(int(input()))
list1.sort()
low=0
high=list1[-1]-list1[0]
ans=0
while low<=high:
    mid=(low+high)//2
    count=1
    pre=list1[0]
    for x in list1[1:]:
        if x-pre>=mid:
            count+=1
            pre=x
    if count>=c:
        ans=mid
        low=mid+1
    else:
        high=mid-1
print(ans)
```

河中跳房子

```
l,n,m=map(int,input().split())
list1=[0]for i in range(n):
    list1.append(int(input()))
list1.append(l)def check(x):
    num = 0
    now = 0
    for i in range(1, n + 2):
        if list1[i] - now < x:
            num += 1
        else:
            now = list1[i]
    if num > m:
        return True
    else:
```

```
            return False
low, high = 0, l + 1
ans = -1
while low < high:
    middle = (low + high) // 2
    if check(middle):
        high = middle
    else:
        ans = middle
        low = middle + 1
print(ans)
```

一个查询的最大美丽值

```
class Solution:
    def maximumBeauty(self, items: List[List[int]], queries: List[int]) -> List[int]:
        items.sort(key=lambda x:x[0])
        n=len(items)
        for i in range(1,n):
            items[i][1]=max(items[i][1],items[i-1][1])
        def query(q):
            l=0
            r=n
            while l<r:
                mid=l+(r-l)//2
                if items[mid][0]>q:
                    r=mid
                else:
                    l=mid+1
            if l==0:
                return 0
            else:
                return items[l-1][1]
        res=[query(queries[i]) for i in range(len(queries))]
        return res
```

并查集

宗教信仰

```
k=0
while True:
```

```
    k+=1
    n,m=map(int,input().split())
    if n==m==0:
        break
    fa=list(range(n+1))
    def find(x):
        if fa[x]!=x:
            fa[x]=find(fa[x])
        return fa[x]
    for i in range(m):
        a,b=map(int,input().split())
        fx=find(a)
        fy=find(b)
        if fx==fy:
            continue
        fa[fy]=fx
    inq=set()
    for j in range(1,n+1):
        inq.add(find(j))
    print(f"Case {k}: {len(inq)}")
```

连接所有点的最小费用

```
class Solution:
    def minCostConnectPoints(self, points: List[List[int]]) -> int:
        def dist(x,y):
            return abs(points[x][0]-points[y][0])+abs(points[x][1]-points[y][1])
        n=len(points)
        fa=list(range(n))
        size=[1]*n
        def find(x):
            if fa[x]!=x:
                fa[x]=find(fa[x])
            return fa[x]
        edges=list()
        for i in range(n):
            for j in range(i+1,n):
                edges.append((dist(i,j),i,j))
        edges.sort()
        ans=0
        for length,x,y in edges:
            fx=find(x)
            fy=find(y)
            if fx==fy:
                continue
            ans+=length
            size[fx]+=size[fy]
```

```
                fa[fy]=fx
                if size[fx]==n:
                        break
        return ans
```

最小省份数量

```
class Solution:
    def findCircleNum(self, isConnected: List[List[int]]) -> int:
        n=len(isConnected)
        vis=[0]*n
        def dfs(i):
            if vis[i]:
                return
            vis[i]=1
            for j in range(n):
            if i!=j and isConnected[i][j]:
                dfs(j)
        ans=0
        for i in range(n):
            if not vis[i]:
                dfs(i)
                ans+=1
        return ans
```

Dilistra

走山路

```
import heapq
m,n,p=map(int,input().split())
matrix=[]for _ in range(m):
    matrix.append(input().split())for i in range(p):
    try:
        judge = 0
        a, b, c, d = map(int, input().split())
        q = [(0,a,b)]
        vis = set()
        while q:
            t, cx, cy = heapq.heappop(q)
            vis.add((cx, cy))
            if cx == c and cy == d:
                judge = 1
                break
            for dx, dy in [(0, 1),(0, -1),(1, 0),(-1, 0)]:
```

```
                nx = cx + dx
                ny = cy + dy
                if 0 <= nx < m and 0 <= ny < n and matrix[nx][ny] !=
'#' and (nx, ny) not in vis:
                    heapq.heappush(q, (abs(int(matrix[nx][ny]) -
int(matrix[cx][cy])) + t, nx, ny))
        if judge == 1:
            print(t)
        else:
            print('NO')
    except:
        print('NO')
```

K 站中转内最便宜航班

```
class Solution:
    def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst: int, k: int) -> int:
    g=[[] for _ in range(n)]
    for i,j,w in flights:
        g[i].append((j,w))
    k+=1
    dis=[inf]*n
    dis[src]=0
    h=[(0,0,src)]
    while h:
        time,cost,dx=heappop(h)
        for y,w in g[dx]:
            nd=time+1
            nc=cost+w
            if nc<dis[y] and nd<=k:
            dis[y]=nc
            heappush(h,(nd,nc,y))
    ans=dis[dst]
    return ans if ans<inf else -1
```

到达最后一个房间的最少时间

```
class Solution:
    def minTimeToReach(self, moveTime: List[List[int]]) -> int:        m=len(moveTime[0])
        n=len(moveTime)
```

```
import heapq
q=[(0,0,0)]
time=[[float('inf') for i in range(m)] for j in range(n)]            time[0][0]=0
while q:
    t,cx,cy=heapq.heappop(q)
    if t>time[cx][cy]:
        continue
    for dx,dy in [(0,1),(0,-1),(1,0),(-1,0)]:
        nx=dx+cx
        ny=dy+cy
        if 0<=nx<n and 0<=ny<m:
            if t>=moveTime[nx][ny]:
                nt=t+1
            else:
                nt=moveTime[nx][ny]+1
            if nt<time[nx][ny]:                                    time[nx][ny]=nt
                heapq.heappush(q,(nt,nx,ny))
return time[n-1][m-1]
```

Agri-Net

```
from heapq import heappop,heappushwhile True:
    try:
        n=int(input())
    except:
        break
    mat=[]
    for i in range(n):
        mat.append(list(map(int,input().split())))
    dis=[100000 for i in range(n)]
    vis=set()
    q=[]
    ans=0
    dis[0]=0
    heappush(q,(dis[0],0))
    while q:
        x,y=heappop(q)
        if y in vis:
            continue
        vis.add(y)
        ans+=dis[y]
        for i in range(n):
            if dis[i]>mat[y][i]:
                dis[i]=mat[y][i]
                heappush(q,(dis[i],i))
```

```
    print(ans)
```

道路

```
import heapq
k = int(input())
n = int(input())
r = int(input())
graph = {i:[] for i in range(1, n+1)}
for _ in range(r):
    s, d, dl, dt = map(int, input().split())
    graph[s].append((dl,dt,d))
que = [(0,0,1)]
fee = [10000]*101
def dijkstra(g):
    while que:
        l, t, d = heapq.heappop(que)
        if d == n:
            return l
        if t>fee[d]:
            continue
        fee[d] = t
        for dl, dt, next_d in g[d]:
            if t+dt <= k:
                heapq.heappush(que,(l+dl, t+dt, next_d))
    return -1
print(dijkstra(graph))
```

树形 dp

打家劫舍

```
class Solution:
    def rob(self, root: Optional[TreeNode]) -> int:
        def dfs(node):
            if not node:
                return 0,0
            l,not_l=dfs(node.left)
            r,not_r=dfs(node.right)
            w_node=not_l+not_r+node.val
            not_node=max(l,not_l)+max(r,not_r)
            return w_node,not_node
        return max(dfs(root))
```

宝藏二叉树

```
import sys
sys.setrecursionlimit(10 ** 7)
def max_treasure(N, vals):
    f0 = [0] * (N + 1)
    f1 = [0] * (N + 1)
    for i in range(N, 0, -1):
        l, r = 2 * i, 2 * i + 1
        include = vals[i]
        if l <= N:
            include += f0[l]
        if r <= N:
            include += f0[r]
        f1[i] = include
        not_include = 0
        if l <= N:
            not_include += max(f0[l], f1[l])
        if r <= N:
            not_include += max(f0[r], f1[r])
        f0[i] = not_include
    return max(f0[1], f1[1])
if __name__ == "__main__":
    import sys
    data = sys.stdin.read().split()
    N = int(data[0])
    vals = [0] + list(map(int, data[1:]))
    print(max_treasure(N, vals))
```

滑动窗口

最大质数的子字符串之和

```
class Solution:
    def sumOfLargestPrimes(self, s: str) -> int:
        def jd(x):
            for i in range(2,int(x**(0.5))+1):
                if x%i==0:
                    return False
            return x>=2
```

```
        def search(y):
            primes=set()
            for i in range(len(s)):
                x=0
                for j in range(i,len(s)):
                    x=x*10+int(s[j])
                    if jd(x):
                        primes.add(x)
            return sum(sorted(primes)[-3:])
        return search(s)
```

统计最大元至少出现过 k 次的子数组

```
class Solution:
    def countSubarrays(self, nums: List[int], k: int) -> int:
        m=max(nums)
        ans=cnt_x=left=0
        for x in nums:
            if x==m:
                cnt_x+=1
            while cnt_x==k:
                if nums[left]==m:
                    cnt_x-=1
                left+=1
            ans+=left
        return ans
```

Stack

移除相邻字符

```
class Solution:
    def resultingString(self, s: str) -> str:
        list1=list(s)
        stack=[]
        for i in range(len(list1)):
            stack.append(list1[i])
            if len(stack)>=2:
                if abs(ord(stack[-1])-ord(stack[-2]))==1 or
  abs(ord(stack[-1])-ord(stack[-2]))==25:
                    stack.pop()
                    stack.pop()
        return "".join(str(x) for x in stack)
```

字符串解码

```
class Solution:
    def decodeString(self, s: str) -> str:
```

```
        stack=[]
        res=""
        multi=0
        for x in s:
            if x=='[':
                stack.append([multi,res])
                res,multi='',0
            elif x==']':
                c_multi,last_res=stack.pop()                    res=last_res+c_multi*res
            elif '0'<=x<='9':
                multi=multi*10+int(x)
            else:
                res+=x
        return res
```

差分数组

零数组变换 I

```
class Solution:
    def isZeroArray(self, nums: List[int], queries: List[List[int]]) -> bool:
        d=[0]*(len(nums)+1)
        for left,right in queries:
            d[left]+=1
            d[right+1]-=1
        compare=[]
        co=0
        for x in d:
            co+=x
            compare.append(co)
        for op,tar in zip(compare,nums):
            if op<tar:
                return False
        return True
```

拓扑排序

最小奖金方案

```
from collections import deque
n,m=map(int,input().split())
graph={}
dic={}
for _ in range(n):
    graph[_]=[]
    dic[_]=0for _ in range(m):
```

```
    a,b=map(int,input().split())
    graph[b].append(a)
    dic[a]+=1
ans=0
q=deque()
money=100
for x in dic:
    if dic[x]==0:
        q.append(x)while q:
    l=len(q)
    for i in range(l):
        y=q.popleft()
        ans+=money
        for z in graph[y]:
            dic[z]-=1
            if dic[z]==0:
                q.append(z)
    money+=1
    print(ans)
```

拓扑排序

```
import heapq
v,a=map(int,input().split())
dic1={}
graph={}for b in range(v):
    dic1[b+1]=0
    graph[b+1]=[]for i in range(a):
    x,y=map(int,input().split())
    dic1[y]+=1
    graph[x].append(y)
pq=[]
for c in dic1:
if dic1[c]==0:
        heapq.heappush(pq,c)
res=[]
cnt=0
while pq:
    x=heapq.heappop(pq)
    res.append('v'+str(x))
    for j in graph[x]:
        dic1[j]-=1
```

```
            if dic1[j]==0 :
                heapq.heappush(pq,j)
print(' '.join(str(z) for z in res) )
```

统计被覆盖的建筑

```
class Solution:
    def countCoveredBuildings(self, n: int, buildings: List[List[int]]) ->
int:
        mx=[0]*(n+1)
        my=[0]*(n+1)
        wx=[n+1]*(n+1)
        wy=[n+1]*(n+1)
        for x,y in buildings:
            mx[y]=max(mx[y],x)
            my[x]=max(my[x],y)
            wx[y]=min(wx[y],x)
            wy[x]=min(wy[x],y)
        cnt=0
        for i,j in buildings:
            if wx[j]<i<mx[j] and wy[i]<j<my[i]:
                cnt+=1
        return cnt
```

```
class TreeNode:

    def __init__(self, val):

        self.val = val

        self.left = None

        self.right = None
```

欧拉筛

```
def euler_sieve(n):
    primes = []
    is_prime = [True] * (n + 1)
    is_prime[0] = is_prime[1] = False
    for i in range(2, 10002):
        if is_prime[i]:
            primes.append(i)
```

```
        for p in primes:
            if i * p > 10001:
                break
            is_prime[i * p] = False
            if i % p == 0:
                break
    return primes
```

Merge sort

```
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        mergeSort(L) # Sorting the first half
        mergeSort(R) # Sorting the second half
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] <= R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
        k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
       while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

哈夫曼树

```
import heapq
def huffman(n, weights):
    if n == 1:
        return weights[0]
    heapq.heapify(weights)
    total_cost = 0
    while len(weights) > 1:
        w1 = heapq.heappop(weights)
        w2 = heapq.heappop(weights)
```

```
        combined_weight = w1 + w2
        total_cost += combined_weight
        heapq.heappush(weights, combined_weight)
    return total_cost
```