简化路径

```
stack=list(input().split('/'))while "" in stack:

    stack.remove("")

ans=[]
cnt=0while stack:

    if stack[-1]=='.':

        stack.pop()

    elif stack[-1]=='..':

        cnt+=1
        stack.pop()
    else:
        if cnt>0:
            stack.pop()
            cnt-=1
        else:
            ans.append(stack.pop())

            ans.append('/')

ans.reverse()
if len(ans)==0:

    ans.append('/')print("".join(str(x) for x in ans))
```

没有上司的宴会

```
from collections import defaultdict
n=int(input())
dic1={}
dic2=defaultdict(list)
inq=set()for i in range(n):
    dic1[i+1]=int(input())for j in range(n-1):
    a,b=map(int,input().split())
    dic2[b].append(a)
    inq.add(a)def dfs(i):
    if len(dic2[i])==0:
        return 0,dic1[i]
    w=dic1[i]
    m=0
    for x in dic2[i]:
        a,b=dfs(x)
```

```
        w+=a
        m+=max(a,b)
    return m,wfor i in range(1,n+1):
    if i not in inq:
        print(max(dfs(i)))
        break
```

沉没孤岛

```
from collections import deque
a,b=map(int,input().split())
matrix=[]
vis=set()for _ in range(a):
    matrix.append(list(map(int,input().split())))for i in range(a):
    if matrix[i][0]==1 and (i,0) not in vis:
        q=deque()
        q.append((i,0))
        while q:
            x,y=q.popleft()
            vis.add((x,y))
            for dx,dy in [(0,1),(0,-1),(1,0),(-1,0)]:
                nx=x+dx
                ny=y+dy
                if 0<=nx<a and 0<=ny<b and matrix[nx][ny]==1 and
(nx,ny) not in vis:
                    q.append((nx,ny))for i in range(a):
    if matrix[i][b-1]==1 and (i,b-1) not in vis:
        q=deque()
        q.append((i,b-1))
        while q:
            x,y=q.popleft()
            vis.add((x,y))
            for dx,dy in [(0,1),(0,-1),(1,0),(-1,0)]:
                nx=x+dx
                ny=y+dy
                if 0<=nx<a and 0<=ny<b and matrix[nx][ny]==1 and
(nx,ny) not in vis:
                    q.append((nx,ny))for i in range(b):
    if matrix[0][i]==1 and (0,i) not in vis:
        q=deque()
        q.append((0,i))
        while q:
            x,y=q.popleft()
            vis.add((x,y))
```

```
            for dx,dy in [(0,1),(0,-1),(1,0),(-1,0)]:
                nx=x+dx
                ny=y+dy
                if 0<=nx<a and 0<=ny<b and matrix[nx][ny]==1 and
(nx,ny) not in vis:
                    q.append((nx,ny))for i in range(b):
    if matrix[a-1][i]==1 and (a-1,i) not in vis:
        q=deque()
        q.append((a-1,i))
        while q:
            x,y=q.popleft()
            vis.add((x,y))
            for dx,dy in [(0,1),(0,-1),(1,0),(-1,0)]:
                nx=x+dx
                ny=y+dy
                if 0<=nx<a and 0<=ny<b and matrix[nx][ny]==1 and
(nx,ny) not in vis:
                    q.append((nx,ny))for i in range(a):
    for j in range(b):
        if matrix[i][j]==1 and (i,j) not in vis:
            matrix[i][j]=0for z in matrix:

    print(" ".join(str(x) for x in z))
```

清北学术走廊

```
n,m=map(int,input().split())
q=[]for _ in range(m):
    u,v,w=map(int,input().split())
    q.append((w,u,v))
ans=0
q.sort(key=lambda z:z[0])
size=[1]*(n+1)
fa=list(range(n+1))def find(i):
    if fa[i]!=i:
        fa[i]=find(fa[i])
    return fa[i]for c,x,y in q:
    fx=find(x)
    fy=find(y)
    if fx==fy:
        continue
    ans+=c
    size[fx]+=size[fy]
    fa[fy]=fx
    if size[fx]==n:
```

```
        breakif max(size)==n:
    print(ans)else:

    print("orz")
```

Excel 表列序号

```
s=list(input())
n=len(s)
ans=0
s.reverse()
i=0while i<=n-1:
    ans+=(26**i)*(ord(s[i])-64)
    i+=1print(ans)
```

堆路径

```
from collections import defaultdict
n=int(input())
list1=list(map(int,input().split()))
list2=[0]for i in list1:
    list2.append(i)
dic1=defaultdict(list)
dic2=defaultdict(list)for x in range(1,n+1):
    if 2*x<=n:
        dic1[list2[x]].append(list2[2*x])
    else:
        break
    if (2*x)+1<=n:
        dic2[list2[x]].append(list2[(2*x)+1])
    else:
        break
tmp=[]
ans=[]
tmp.append(list2[1])
judge=0
res=10def dfs(y):
    if len(dic1[y])==0 and len(dic2[y])==0:
        ans.append(tmp[:])
        return
    if len(dic2[y])>0:
        tmp.append(dic2[y][0])
        dfs(dic2[y][0])
        tmp.pop()
```

```
    if len(dic1[y])>0:
        tmp.append(dic1[y][0])
        dfs(dic1[y][0])
        tmp.pop()dfs(list2[1])for z in ans:

    print(" ".join(str(o) for o in z))for t in ans:

    if res==0:
        break
    for v in range(len(t)-1):
        h=t[v+1]-t[v]
        if judge==0:
            judge=h
        else:
            if judge>0 and h<0:
                res=0
                break
            elif judge<0 and h>0:
                res=0
                breakif res==0:

    print("Not Heap")else:

    if judge<0:

        print("Max Heap")

    elif judge>0:

        print("Min Heap")
```

判断等价关系是否成立

```
n=int(input())
fa=list(range(26))
un=[]
judge=0def find(x):
    if fa[x] != x:
        fa[x] = find(fa[x])
    return fa[x]for i in range(n):
    s=input()

    x = ord(s[0]) - ord('a')

    y = ord(s[3]) - ord('a')

    if s[1]=='=':
```

```
        fx=find(x)
        fy=find(y)
        if fx==fy:
            continue
        fa[fy]=fx
    else:
        un.append((x,y))for a,b in un:
    if find(a)==find(b):
        judge=1
        breakif judge==1:
    print(False)else:
    print(True)
```

谣言

```
from collections import defaultdict
n,m=map(int,input().split())
fa=list(range(n+1))
cost=list(map(int,input().split()))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
ans=0
dic=defaultdict(list)for j in range(1,n+1):
    fr=find(j)
    dic[fr].append(cost[j-1])for z in dic:
    if dic[z]:
        ans+=min(dic[z])print(ans)
```

受限条件下可到达节点的数目

```
from collections import defaultdict
n=int(input())
dic=defaultdict(list)
vis=set()for i in range(n-1):
    a,b=map(int,input().split())
    dic[a].append(b)
    dic[b].append(a)
ban=list(map(int,input().split()))
```

```
cnt=0def dfs(x):
    global cnt
    vis.add(x)
    cnt+=1
    for y in dic[x]:
        if y not in vis and y not in ban:
            dfs(y)dfs(0)print(cnt)
```

分糖果

```
from collections import deque

n,m=map(int,input().split())
list1=list(map(int,input().split()))
q=deque()for i in range(n):
    q.append((list1[i],i+1))while len(q)>1:
    x,y=q.popleft()
    if x<=m:
        continue
    else:
        q.append((x-m,y))
ans1,ans2=q.popleft()print(ans2)
```

Okabe and Boxes

```
n=int(input())
stack=[]
cnt=1
ans=0for i in range(2*n):
    s=input().split()

    if s[0]=="add":

        stack.append(int(s[1]))
    else:
        if stack[-1] == cnt:
            stack.pop()
            cnt += 1
        else:
            stack.sort(reverse=True)
            stack.pop()
            cnt += 1
            ans += 1print(ans)
```

扩展二叉树

```
from collections import defaultdict
s=list(input())
stack=[]
list1=[]
inq=set()
ans1=[]
ans2=[]
dic1=defaultdict(list)
dic2=defaultdict(list)for i in range(len(s)):
    stack.append(s[i])

    if s[i]!='.':

        list1.append(s[i])

    while len(stack)>=3 and stack[-1] == stack[-2] == '.':

        stack.pop()
        stack.pop()
        x = stack.pop()
        if stack:

            if stack[-1] != '.':

                dic1[stack[-1]].append(x)
                inq.add(x)
            else:
                dic2[stack[-2]].append(x)
                inq.add(x)

        stack.append('.')def dfs(z):

    if len(dic1[z])==0 and dic2[z]==0:
        ans1.append(z)
        ans2.append(z)
        return
    if len(dic1[z])>0:
        dfs(dic1[z][0])
    ans1.append(z)
    if len(dic2[z])>0:
        dfs(dic2[z][0])
    ans2.append(z)for l in list1:
    if l not in inq:
        dfs(l)
```

```
        breakprint("".join(str(h) for h in ans1))print("".join(str(t)
for t in ans2))
```

遍历树

```
from collections import defaultdict
dic=defaultdict(list)
inq=set()
n=int(input())
list1=[]for _ in range(n):
    x=list(map(int,input().split()))
    for i in x:
        dic[x[0]].append(i)
        list1.append(i)
        if i!=x[0]:
            inq.add(i)
    dic[x[0]].sort()
ans=[]def dfs(z):
    if len(dic[z])==0:
        ans.append(z)
        return
    for y in dic[z]:
        if y != z:
            dfs(y)
        else:
            ans.append(y)for z in list1:
    if z not in inq:
        dfs(z)
        breakfor j in ans:
    print(j)
```

网线主管

```
n,k=map(int,input().split())
list1=[]for i in range(n):
    l=float(input())
    list1.append(l*100)
list1.sort()
low=0
high=list1[-1]
ans=0while low<=high:
    mid=(low+high)//2
    if mid==0:
```

```
        break
    count=0
    for x in list1:
        count+=x//mid
    if count>=k:
        ans=mid
        low=mid+1
    else:

        high=mid-1print(f"{ans/100:.2f}")
```

宗教信仰

```
k=0while 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)}")
```

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=0while 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-1print(ans)
```

求二叉树的高度和叶子数目

```
from collections import defaultdict
dic=defaultdict(list)
n_root=set()
n=int(input())for _ in range(n):
    a,b=map(int,input().split())
    if a!=-1:
        n_root.add(a)
    if b!=-1:
        n_root.add(b)
    dic[_].append(a)
    dic[_].append(b)
root=0for i in range(n):
    if i not in n_root:
        root=i
        break
cnt=0
ans=0def dfs(node):
    global cnt
    if node==-1:
        return 0
    if dic[node][0]==-1 and dic[node][1]==-1:
        cnt+=1
    left=dfs(dic[node][0])
    right=dfs(dic[node][1])
    return max(left,right)+1
ans=dfs(root)print(ans-1,cnt)
```

二叉树的深度

```
from collections import deque
n=int(input())
list1=[]for i in range(n):
    a,b=map(int,input().split())
    list1.append((a,b))
q=deque()
q.append(list1[0])
index=1
depth=0while index<len(list1):
    depth+=1
    for _ in range(len(q)):
        x,y=q.popleft()
        if x!=-1:
            q.append(list1[index])
            index+=1
        if y!=-1:
            q.append(list1[index])
            index+=1print(depth+1)
```

FBI 树

```
n=int(input())
s=input()def j(s):
    l=sum(int(i) for i in list(s))
    if len(s)==l:
        return 'I'
    elif l==0:
        return 'B'
    else:
        return 'F'
ans=[]def dfs(s,cnt):
    if cnt==0:
        return
    dfs(s[:cnt//2],cnt//2)
    dfs(s[cnt//2:],cnt//2)
    ans.append(j(s))dfs(s,2**n)print(''.join(str(x) for x in ans))
```

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')
```

Candies

```
import heapqfrom collections import defaultdict
n,m=map(int,input().split())
graph=defaultdict(list)for _ in range(m):
    a,b,c=map(int,input().split())
    graph[a].append((b,c))
vis=set()
q=[(0,1)]
```

```
dis=[float('inf') for i in range(n+1)]while q:
    x,y=heapq.heappop(q)
    if y==n:
        print(x)
        break
    if y in vis:
        continue
    vis.add(y)
    for z in graph[y]:
        if dis[z[0]]>z[1]+x:
            dis[z[0]]=z[1]+x
            heapq.heappush(q,(z[1]+x,z[0]))
```

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=100for 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+=1print(ans)
```

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)
```

用二次探查法建立散列表

```
import sys
input = sys.stdin.read
data = input().split()
index = 0
n = int(data[index])
index += 1
m = int(data[index])
index += 1
num_list = [int(i) for i in data[index:index+n]]
sign=1
cnt=1
HT=[0.5 for j in range(m)]
list2=[]for x in range(len(num_list)):
    l=num_list[x]%m
    if HT[l]==0.5 or HT[l]==num_list[x]:
        HT[l]=num_list[x]
        list2.append(l)
    else:
```

```
        while HT[l]!=0.5 and HT[l]!=num_list[x]:
            count=sign*(cnt**2)+num_list[x]
            l=count%m
            sign*=-1
            if sign==1:
                cnt+=1
        else:
            HT[l]=num_list[x]

            list2.append(l)print(" ".join(str(z) for z in list2))
```

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))
```

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]*101def 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 -1print(dijkstra(graph))
```

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=0while 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) )
```

用队列对扑克牌排序

```
n=int(input())
list1=list(input().split())
q=[[] for _ in range(9)]
dic={}

dic['C']=3

dic['B']=2

dic['A']=1

dic['D']=4

qu=[[] for _ in range(4)]for x in list1:
    q[int(x[1])-1].append(x)for y in range(9):

    print(f'Queue{y+1}:{" ".join(str(a) for a in q[y])}')

    if len(q[y])>0:
        for b in q[y]:
            qu[dic[b[0]]-1].append(b)for i in range(4):

    print(f'Queue{chr(i+65)}:{" ".join(str(a) for a in qu[i])}')

ans=[]for z in qu:
    for c in z:

        ans.append(c)print(" ".join(str(d) for d in ans))
```

出栈序列统计

```
n=int(input())
stack=[]
list1=[x+1 for x in range(n)]
cnt=0def d(i,x):
    global cnt
    if i==n:
        cnt+=1
        return
    d(i+1,x+1)
    if x>0:
        d(i,x-1)
d(0,0)print(cnt)
```

牛的选举

```
n,k=map(int,input().split())
list1=[]
list2=[]for i in range(n):
    a,b=map(int,input().split())
    list1.append((a,b,i+1))
list1.sort(key=lambda x:x[0])for x in range(k):
    list2.append((list1[-1-x][1],list1[-1-x][2]))
list2.sort(key=lambda x:x[0])print(list2[-1][1])
```

骑士周游

```
import sys
class Graph:
    def __init__(self):
        self.vertices = {}
        self.num_vertices = 0

    def add_vertex(self, key):
        self.num_vertices = self.num_vertices + 1
        new_ertex = Vertex(key)
        self.vertices[key] = new_ertex
        return new_ertex

    def get_vertex(self, n):
        if n in self.vertices:
            return self.vertices[n]
        else:
            return None

    def __len__(self):
        return self.num_vertices

    def __contains__(self, n):
        return n in self.vertices

    def add_edge(self, f, t, cost=0):
        if f not in self.vertices:
            nv = self.add_vertex(f)
        if t not in self.vertices:
            nv = self.add_vertex(t)
        self.vertices[f].add_neighbor(self.vertices[t], cost)
        #self.vertices[t].add_neighbor(self.vertices[f], cost)
```

```
    def getVertices(self):
        return list(self.vertices.keys())

    def __iter__(self):
        return iter(self.vertices.values())

class Vertex:
    def __init__(self, num):
        self.key = num
        self.connectedTo = {}

        self.color = 'white'

        self.distance = sys.maxsize
        self.previous = None
        self.disc = 0
        self.fin = 0

    def __lt__(self,o):
        return self.key < o.key

    def add_neighbor(self, nbr, weight=0):
        self.connectedTo[nbr] = weight


    # def setDiscovery(self, dtime):
    #     self.disc = dtime
    #
    # def setFinish(self, ftime):
    #     self.fin = ftime
    #
    # def getFinish(self):
    #     return self.fin
    #
    # def getDiscovery(self):
    #     return self.disc

    def get_neighbors(self):
        return self.connectedTo.keys()

    # def getWeight(self, nbr):
    #     return self.connectedTo[nbr]

    def __str__(self):
```

```
        return str(self.key) + ":color " + self.color + ":disc " +
str(self.disc) + ":fin " + str(
            self.fin) + ":dist " + str(self.distance) + ":pred \n\t[" +
str(self.previous) + "]\n"


def knight_graph(board_size):
    kt_graph = Graph()
    for row in range(board_size):             #遍历每一行
        for col in range(board_size):         #遍历行上的每一个格子
            node_id = pos_to_node_id(row, col, board_size) #把行、列号转为格子 ID
            new_positions = gen_legal_moves(row, col, board_size) #按照 马走日，返回下一步可能位置
            for row2, col2 in new_positions:
                other_node_id = pos_to_node_id(row2, col2, board_size) #下一步的格子 ID
                kt_graph.add_edge(node_id, other_node_id) #在骑士周游图中为两个格子加一条边
    return kt_graph
def pos_to_node_id(x, y, bdSize):
    return x * bdSize + y
def gen_legal_moves(row, col, board_size):
    new_moves = []
    move_offsets = [                          # 马走日的 8 种走法
        (-1, -2),  # left-down-down
        (-1, 2),  # left-up-up
        (-2, -1),  # left-left-down
        (-2, 1),  # left-left-up
        (1, -2),  # right-down-down
        (1, 2),  # right-up-up
        (2, -1),  # right-right-down
        (2, 1),  # right-right-up
    ]
    for r_off, c_off in move_offsets:
        if (                                  # #检查，不能走出棋盘
            0 <= row + r_off < board_size
            and 0 <= col + c_off < board_size
        ):
            new_moves.append((row + r_off, col + c_off))
```

```
    return new_moves
# def legal_coord(row, col, board_size):#     return 0 <= row < board_size and 0 <= col < board_size

def knight_tour(n, path, u, limit):

    u.color = "gray"

    path.append(u)                #当前顶点涂色并加入路径
    if n < limit:
        neighbors = ordered_by_avail(u) #对所有的合法移动依次深入
        #neighbors = sorted(list(u.get_neighbors()))
        i = 0

        for nbr in neighbors:

            if nbr.color == "white" and \

                knight_tour(n + 1, path, nbr, limit):    #选择"白色"未经深入的点，层次加一，递归深入
                return True
        else:                        #所有的"下一步"都试了走不通
            path.pop()               #回溯，从路径中删除当前顶点

            u.color = "white"        #当前顶点改回白色

            return False
    else:
        return True
def ordered_by_avail(n):
    res_list = []
    for v in n.get_neighbors():

        if v.color == "white":

            c = 0
            for w in v.get_neighbors():

                if w.color == "white":

                    c += 1
            res_list.append((c,v))
    res_list.sort(key = lambda x: x[0])
    return [y[1] for y in res_list]
# class DFSGraph(Graph):#     def __init__(self):#         super().__init__()#         self.time = 0                #不是物理世界，而是算法执行步数# #     def dfs(self):#         for vertex in self:#             vertex.color = "white"      #颜色初始化#             vertex.previous = -1#         for vertex in self:             #从每个顶点开始遍历#
```

```
# if vertex.color == "white":#                 self.dfs_visit(vertex)  # 第一次运行后还有未包括的顶点#                                         # 则建立森林# #     def dfs_visit(self, start_vertex):# start_vertex.color = "gray"#         self.time = self.time + 1       # 记录算法的步骤#         start_vertex.discovery_time = self.time# for next_vertex in start_vertex.get_neighbors():#             if next_vertex.color == "white":#                 next_vertex.previous = start_vertex#                 self.dfs_visit(next_vertex)     #深度优先递归访问#         start_vertex.color = "black"#         self.time = self.time + 1#         start_vertex.closing_time = self.time

def main():
    def NodeToPos(id):
        return ((id//8, id%8))

    bdSize = int(input())  # 棋盘大小
    *start_pos, = map(int, input().split())  # 起始位置
    g = knight_graph(bdSize)
    start_vertex = g.get_vertex(pos_to_node_id(start_pos[0], start_pos[1], bdSize))
    if start_vertex is None:
        print("fail")
        exit(0)

    tour_path = []
    done = knight_tour(0, tour_path, start_vertex, bdSize * bdSize-1)
    if done:
        print("success")
    else:
        print("fail")

    exit(0)

    # 打印路径
    cnt = 0
    for vertex in tour_path:
        cnt += 1
        if cnt % bdSize == 0:
            print()
        else:
            print(vertex.key, end=" ")
```

```
            #print(NodeToPos(vertex.key), end=" ")    # 打印坐标
if __name__ == '__main__':
    main()
```

兔子与樱花

```
import heapqfrom collections import defaultdict

p = int(input())
points = [input().strip() for _ in range(p)]
maps = defaultdict(list)for _ in range(int(input())):
    a, b, d = input().split()
    d = int(d)
    maps[a].append((b, d))
    maps[b].append((a, d))
def dijkstra(src, dst):
    INF = float('inf')
    dist = {point: INF for point in points}
    path = {point: "" for point in points}
    dist[src] = 0
    path[src] = src
    pq = [(0, src)]
    while pq:
        d, u = heapq.heappop(pq)
        if d > dist[u]:
            continue
        if u == dst:
            break
        for v, w in maps[u]:
            nd = d + w
            if nd < dist[v]:
                dist[v] = nd
                path[v] = path[u] + f"->({w})->" + v
                heapq.heappush(pq, (nd, v))
    return path[dst]
for _ in range(int(input())):
    s, t = input().split()
    print(dijkstra(s, t))
```

厚道的调分方法

```
def check(b, scores):
    a = b / 1000000000
    count = 0
    for x in scores:
        adjusted = a*x+1.1**(a*x)
        if adjusted >= 85:
            count += 1
    return count >= len(scores) * 0.6
scores = list(map(float, input().split()))
left, right = 1, 1000000000while left < right:
    mid = (left + right) // 2
    if check(mid, scores):
        right = mid
    else:
        left = mid + 1print(left)
```

献给阿尔吉侬的花束

```
from collections import deque
dx=[0,0,1,-1]
dy=[-1,1,0,0]
n=int(input())for i in range(n):
    judge=0
    matrix=[]
    a,b=map(int,input().split())
    for _ in range(a):
        s=input()
        matrix.append(s)
    for e in range(len(matrix)):
        for f in range(len(matrix[e])):

            if matrix[e][f]=="S":

                h,k=e,f
                break
    q=deque()
    q.append((h,k))
    inq=set()
    inq.add((h,k))
    cnt=0
    while q:
        if judge==1:
            break
```

```
        cnt+=1
        for z in range(len(q)):
            if judge==1:
                break
            x,y=q.popleft()
            for u in range(4):
                nx=x+dx[u]
                ny=y+dy[u]

                if 0<=nx<a and 0<=ny<b and matrix[nx][ny]!="#" and
(nx,ny) not in inq:
                    inq.add((nx,ny))
                    q.append((nx,ny))

                    if matrix[nx][ny]=="E":

                        judge=1
                        break
    if judge==1:
        print(cnt)
    else:

        print("oop!")
```

词梯

```
import sysfrom collections import deque
class Graph:
    def __init__(self):
        self.vertices = {}
        self.num_vertices = 0

    def add_vertex(self, key):
        self.num_vertices = self.num_vertices + 1
        new_vertex = Vertex(key)
        self.vertices[key] = new_vertex
        return new_vertex

    def get_vertex(self, n):
        if n in self.vertices:
            return self.vertices[n]
        else:
            return None

    def __len__(self):
```

```
        return self.num_vertices

    def __contains__(self, n):
        return n in self.vertices

    def add_edge(self, f, t, cost=0):
        if f not in self.vertices:
            nv = self.add_vertex(f)
        if t not in self.vertices:
            nv = self.add_vertex(t)
        self.vertices[f].add_neighbor(self.vertices[t], cost)

    def get_vertices(self):
        return list(self.vertices.keys())

    def __iter__(self):
        return iter(self.vertices.values())

class Vertex:
    def __init__(self, num):
        self.key = num
        self.connectedTo = {}

        self.color = 'white'

        self.distance = sys.maxsize
        self.previous = None
        self.disc = 0
        self.fin = 0

    def add_neighbor(self, nbr, weight=0):
        self.connectedTo[nbr] = weight

    def get_neighbors(self):
        return self.connectedTo.keys()

def build_graph(all_words):
    buckets = {}
    the_graph = Graph()


    for line in all_words:
        word = line.strip()
        for i, _ in enumerate(word):
```

```
            bucket = f"{word[:i]}_{word[i + 1:]}"
            buckets.setdefault(bucket, set()).add(word)


    for similar_words in buckets.values():
        for word1 in similar_words:
            for word2 in similar_words - {word1}:
                the_graph.add_edge(word1, word2)

    return the_graph

def bfs(start, end):
    start.distnce = 0
    start.previous = None
    vert_queue = deque()
    vert_queue.append(start)
    while len(vert_queue) > 0:
        current = vert_queue.popleft()

        if current == end:
            return True

        for neighbor in current.get_neighbors():

            if neighbor.color == "white":

                neighbor.color = "gray"

                neighbor.distance = current.distance + 1
                neighbor.previous = current
                vert_queue.append(neighbor)

        current.color = "black"


    return Falsedef traverse(starting_vertex):
    ans = []
    current = starting_vertex
    while (current.previous):
        ans.append(current.key)
        current = current.previous
    ans.append(current.key)

    return ans
```

```
n = int(input())
all_words = []for _ in range(n):
    all_words.append(input().strip())

g = build_graph(all_words)
s, e = input().split()
start, end = g.get_vertex(s), g.get_vertex(e)if start is None or
 end is None:

    print('NO')

    exit(0)
if bfs(start, end):
    ans = traverse(end)

    print(' '.join(ans[::-1]))else:

    print('NO')
```

图的拉普拉斯矩阵

```
class Vertex:
    def __init__(self,key):
        self.key=key
        self.neighbors={}
    def get_neighbor(self,other):
        return self.neighbors.get(other,None)
    def set_neighbor(self,other,weight=0):
        self.neighbors[other]=weight
    def __repr__(self):

        return f"Vertex({self.key})"

    def __str__(self):
        return (str(self.key)

                + " connected to: "

                + str([x.key for x in self.neighbors]))
    def get_neighbors(self):
        return self.neighbors.keys()
    def get_key(self):
        return self.keyclass Graph:
    def __init__(self):
        self.vertices={}
    def set_vertex(self,key):
```

```
        self.vertices[key]=Vertex(key)
    def get_vertex(self,key):
        return self.vertices.get(key,None)
    def __contains__(self,key):
        return key in self.vertices
    def add_edge(self,from_vert,to_vert,weight=0):
        if from_vert not in self.vertices:
            self.set_vertex(from_vert)
        if to_vert not in self.vertices:
            self.set_vertex(to_vert)

self.vertices[from_vert].set_neighbor(self.vertices[to_vert],wei
ght)

    def get_vertices(self):
        return self.vertices.keys()

    def __iter__(self):
        return iter(self.vertices.values())def
constructLaplacianMatrix(n, edges):
    graph = Graph()
    for i in range(n):
        graph.set_vertex(i)
    for edge in edges:
        a, b = edge
        graph.add_edge(a, b)
        graph.add_edge(b, a)
    laplacianMatrix = []
    for vertex in graph:
        row = [0] * n
        row[vertex.get_key()] = len(vertex.get_neighbors())
        for neighbor in vertex.get_neighbors():
            row[neighbor.get_key()] = -1
        laplacianMatrix.append(row)
    return laplacianMatrix
n, m = map(int, input().split())
edges = []for i in range(m):
    a,b = map(int, input().split())
    edges.append((a, b))
laplacianMatrix = constructLaplacianMatrix(n, edges)for row in
laplacianMatrix:

    print(' '.join(map(str, row)))
```

电话号码

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_number = Falseclass Trie:
    def __init__(self):
        self.root = TrieNode()
    def insert(self, number):
        node = self.root
        for digit in number:
            if digit not in node.children:
                node.children[digit] = TrieNode()
            node = node.children[digit]
            if node.is_end_of_number:
                return False
        node.is_end_of_number = True
        return len(node.children) == 0
    def is_consistent(self, numbers):
        numbers.sort(key=len)
        for number in numbers:
            if not self.insert(number):
                return False
        return Truedef main():
    import sys
    input = sys.stdin.read
    data = input().splitlines()

    t = int(data[0])
    index = 1
    results = []
    for _ in range(t):
        n = int(data[index])
        index += 1
        numbers = data[index:index + n]
        index += n

        trie = Trie()
        if trie.is_consistent(numbers):

            results.append("YES")

        else:

            results.append("NO")
```

```
    print("\n".join(results))if __name__ == "__main__":
    main()
```

哈夫曼编码树

```
import heapq
class Node:
    def __init__(self, weight, char=None):
        self.weight = weight
        self.char = char
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.weight == other.weight:
            return self.char < other.char
        return self.weight < other.weight
def build_huffman_tree(characters):
    heap = []
    for char, weight in characters.items():
        heapq.heappush(heap, Node(weight, char))

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        #merged = Node(left.weight + right.weight) #note: 合并后，char字段默认值是空
        merged = Node(left.weight + right.weight, min(left.char, right.char))
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]
def encode_huffman_tree(root):
    codes = {}

    def traverse(node, code):
        #if node.char:
        if node.left is None and node.right is None:
            codes[node.char] = code
        else:
```

```
            traverse(node.left, code + '0')

            traverse(node.right, code + '1')


    traverse(root, '')
    return codes
def huffman_encoding(codes, string):
    encoded = ''
    for char in string:
        encoded += codes[char]
    return encoded
def huffman_decoding(root, encoded_string):
    decoded = ''
    node = root
    for bit in encoded_string:
        if bit == '0':
            node = node.left
        else:
            node = node.right

        #if node.char:
        if node.left is None and node.right is None:
            decoded += node.char
            node = root
    return decoded
# 读取输入
n = int(input())
characters = {}for _ in range(n):
    char, weight = input().split()
    characters[char] = int(weight)
#string = input().strip()#encoded_string = input().strip()
# 构建哈夫曼编码树
huffman_tree = build_huffman_tree(characters)
# 编码和解码
codes = encode_huffman_tree(huffman_tree)

strings = []while True:
    try:
        line = input()
```

```
        strings.append(line)

    except EOFError:
        break

results = []#print(strings)for string in strings:

    if string[0] in ('0','1'):

        results.append(huffman_decoding(huffman_tree, string))
    else:
        results.append(huffman_encoding(codes, string))
for result in results:
    print(result)
```

实现堆结构

```
class BinaryHeap:
    def __init__(self):
        self._heap = []

    def _perc_up(self, i):
        while (i - 1) // 2 >= 0:
            parent_idx = (i - 1) // 2
            if self._heap[i] < self._heap[parent_idx]:
                self._heap[i], self._heap[parent_idx] = (
                    self._heap[parent_idx],
                    self._heap[i],
                )
            i = parent_idx

    def insert(self, item):
        self._heap.append(item)
        self._perc_up(len(self._heap) - 1)

    def _perc_down(self, i):
        while 2 * i + 1 < len(self._heap):
            sm_child = self._get_min_child(i)
            if self._heap[i] > self._heap[sm_child]:
                self._heap[i], self._heap[sm_child] = (
                    self._heap[sm_child],
                    self._heap[i],
                )
            else:
                break
```

```
            i = sm_child

    def _get_min_child(self, i):
        if 2 * i + 2 > len(self._heap) - 1:
            return 2 * i + 1
        if self._heap[2 * i + 1] < self._heap[2 * i + 2]:
            return 2 * i + 1
        return 2 * i + 2

    def delete(self):
        self._heap[0], self._heap[-1] = self._heap[-1],
self._heap[0]
        result = self._heap.pop()
        self._perc_down(0)
        return result

    def heapify(self, not_a_heap):
        self._heap = not_a_heap[:]
        i = len(self._heap) // 2 - 1    # 超过中点的节点都是叶子节点
        while i >= 0:
            #print(f'i = {i}, {self._heap}')
            self._perc_down(i)
            i = i - 1



n = int(input().strip())
bh = BinaryHeap()for _ in range(n):
    inp = input().strip()

    if inp[0] == '1':

        bh.insert(int(inp.split()[1]))
    else:
        print(bh.delete())
```

二叉搜索树的层次遍历

```
import sysfrom collections import deque

class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```

```
def main():
    nums = list(map(int, input().split()))
    seen = set()
    unique_nums = []
    for num in nums:
        if num not in seen:
            seen.add(num)
            unique_nums.append(num)
    if not unique_nums:
        print()
        return

    root = TreeNode(unique_nums[0])
    for val in unique_nums[1:]:
        current = root
        while True:
            if val < current.val:
                if current.left is None:
                    current.left = TreeNode(val)
                    break
                else:
                    current = current.left
            elif val > current.val:
                if current.right is None:
                    current.right = TreeNode(val)
                    break
                else:
                    current = current.right
            else:
                break
    result = []
    queue = deque()
    queue.append(root)

    while queue:
        node = queue.popleft()
        result.append(str(node.val))
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)

    print(' '.join(result))
```

```
if __name__ == "__main__":
    main()
```

Huffman 编码树

```
import heapq
n=int(input())
list1=list(map(int,input().split()))
heapq.heapify(list1)
cnt=0while len(list1)>1:
    left=heapq.heappop(list1)
    right=heapq.heappop(list1)
    s=left+right
    cnt+=s
    heapq.heappush(list1,s)print(cnt)
```

括号嵌套树

```
def parse_tree(s):
    stack = []
    node = None
    for char in s:
        if char.isalpha():
            node = {'value': char, 'children': []}
            if stack:
                stack[-1]['children'].append(node)
        elif char == '(':
                stack.append(node)
                node = None
        elif char == ')':
            if stack:
                node = stack.pop()
    return node
def preorder(node):
    output = [node['value']]
    for child in node['children']:
```

```
        output.extend(preorder(child))
    return ''.join(output)def postorder(node):
    output = []
    for child in node['children']:
        output.extend(postorder(child))
    output.append(node['value'])

    return ''.join(output)def main():
    s = input().strip()
    s = ''.join(s.split())
    root = parse_tree(s)
    if root:
        print(preorder(root))
        print(postorder(root))
    else:
        print("input tree string error!")if __name__ == "__main__":
    main()
```

根据二叉树前中序序列建树

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
def build_tree(preorder, inorder):
    if not preorder or not inorder:
        return
    root_value = preorder[0]
    root = TreeNode(root_value)
    root_index_inorder = inorder.index(root_value)
    root.left = build_tree(preorder[1:1+root_index_inorder],
inorder[:root_index_inorder])
    root.right = build_tree(preorder[1+root_index_inorder:],
inorder[root_index_inorder+1:])
    return root
def postorder_traversal(root):
    if root is None:
```

```
        return ''
    return postorder_traversal(root.left) +
postorder_traversal(root.right) + root.value
while True:
    try:
        preorder = input().strip()
        inorder = input().strip()
        root = build_tree(preorder, inorder)
        print(postorder_traversal(root))
    except EOFError:
        break
```

遍历树

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []

def traverse_print(root, nodes):
    if root.children == []:
        print(root.value)
        return
    pac = {root.value: root}
    for child in root.children:
        pac[child] = nodes[child]
    for value in sorted(pac.keys()):
        if value in root.children:
            traverse_print(pac[value], nodes)
        else:
            print(root.value)


n = int(input())
nodes = {}
children_list = []for i in range(n):
    info = list(map(int, input().split()))
    nodes[info[0]] = TreeNode(info[0])
    for child_value in info[1:]:
        nodes[info[0]].children.append(child_value)
        children_list.append(child_value)
```

```
root = nodes[[value for value in nodes.keys() if value not in children_list][0]]traverse_print(root, nodes)
```

森林的带度数层次序列存储

```
from collections import dequeimport sysclass Node:
    def __init__(self, value, degree):
        self.value = value
        self.degree = degree
        self.children = []
def build_tree(tokens):

    root = Node(tokens[0], int(tokens[1]))
    queue = deque([root])
    index = 2
    while queue and index < len(tokens):
        current = queue.popleft()
        for _ in range(current.degree):
            child = Node(tokens[index], int(tokens[index+1]))
            current.children.append(child)
            queue.append(child)
            index += 2
    return root
def postorder(node, output):
    for child in node.children:
        postorder(child, output)
    output.append(node.value)
def main():
    input_lines = sys.stdin.read().splitlines()
    if not input_lines:
        return
    n = int(input_lines[0].strip())
    result = []
    for i in range(1, n+1):
        tokens = input_lines[i].split()
        if not tokens:
            continue
        root = build_tree(tokens)
        temp = []
        postorder(root, temp)
        result.extend(temp)

    print(" ".join(result))
```

```
if __name__ == "__main__":
    main()
```

北大夺冠

```
mydict_cnt={}
mydict_ac={}
n=int(input())for i in range(n):
    a,b,c=input().split(",")
    if a in mydict_cnt:
        mydict_cnt[a]+=1
    else:
        mydict_cnt[a]=1
    if a not in mydict_ac:
        mydict_ac[a]=[]
    if c=="yes" and b not in mydict_ac[a]:
        mydict_ac[a].append(b)
list1=[]for x in mydict_cnt:
    list1.append((x,mydict_cnt[x],len(mydict_ac[x])))
list1.sort(key=lambda x:(-x[2],x[1],x[0]))for i in
range(min(len(list1), 12)):
    rank = i + 1
    solved = list1[i][2]
    total_sub = list1[i][1]
    name = list1[i][0]
    print(f"{rank} {name} {solved} {total_sub}")
```

个位为 1 的质数个数

```
n=int(input())
numbers = [True] * (10**4+2)
numbers[0]=numbers[1]=False
primes = []def euler_sieve(numbers):
    for i in range(2, int(1e4)+2):
        if numbers[i]:
            primes.append(i)
        for j in range(len(primes)):
            if i * primes[j] > int(1e4+1):
                break
            numbers[i * primes[j]] = False
```

```
            if i % primes[j] == 0:
                breakeuler_sieve(numbers)for _ in range(n):
    x=int(input())
    list1=[]
    for num in primes:
        if num>=x:
            break
        else:

            if str(num)[-1]=="1":

                list1.append(str(num))

    print(f"Case{_+1}:")

    if list1:

        print(" ".join(str(x) for x in list1))

    else:

        print("NULL")
```

寻找离目标数最近的两数之和

```
n=int(input())
list1=list(map(int,input().split()))
list1.sort()
left=0
right=len(list1)-1
list2=[]while left<right:
    list2.append((list1[left] + list1[right], abs(list1[left] +
list1[right] - n)))
    if list1[left]+list1[right]>n:
        right-=1
    elif list1[left]+list1[right]==n:
        break
    else:
        left+=1
list2.sort(key=lambda x:(x[1],x[0]))print(list2[0][0])
```

木材加工

```
n,k=map(int,input().split())
list1=[]for i in range(n):
    l=int(input())
    list1.append(l)
```

```
sum_all=sum(list1)
left=1
right=max(list1)if sum_all<k:
    print(0)else:
    result=0
    while left <= right:
        mid = (left + right) // 2
        count = 0
        for x in list1:
            count += x // mid
        if count >= k:
            result = mid
            left = mid + 1
        else:
            right = mid - 1
    print(result)
```

最后的最后

```
from collections import deque
n,k=map(int,input().split())
q=deque()for i in range(1,n+1):
    q.append(i)
x=0
list1=[]while len(q)>1:
    x+=1
    y=q.popleft()
    if x%k==0:
        list1.append(y)
    else:

        q.append(y)print(" ".join(str(x) for x in list1))
```

中序表达式转后序表达式

```
n=int(input())

you_xian={"+":1,"-":1,"*":2,"/":2}


all_s="+-*/"for _ in range(n):

    s=input().strip()
    stack1 = []
    stack2 = []
```

```
    number = ""
    for x in s:
        if x.isnumeric() or x==".":
            number+=x
        else:
            if number:
                num=float(number)
                stack2.append(int(num) if num.is_integer() else num)
                number=''
            if x in all_s:
                while stack1 and stack1[-1] in all_s and
you_xian[x]<=you_xian[stack1[-1]]:
                    stack2.append(stack1.pop())
                stack1.append(x)
            elif x=="(":
                stack1.append(x)
            elif x==")":

                while stack1 and stack1[-1]!="(":
                    stack2.append(stack1.pop())
                stack1.pop()
    if number:
        num = float(number)
        stack2.append(int(num) if num.is_integer() else num)
    while stack1:
        stack2.append(stack1.pop())
    print(" ".join(str(x) for x in stack2))
```

蚂蚁王国的越野跑

```
# pylint: skip-filedef merge_sort(arr):
    global cnt
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
```

```
def merge(left, right):
    global cnt
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            cnt += len(right) - j
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])

    return result
cnt=0
n=int(input())
mylist=[]for i in range(n):
    x=int(input())
    mylist.append(x)merge_sort(mylist)print(cnt)
```

约瑟夫问题 No.2

```
from collections import dequewhile True:
    n,p,m=map(int,input().split())
    if n==p==m==0:
        break
    q=deque()
    for i in range(p,n+1):
        q.append(i)
    for j in range(1,p):
        q.append(j)
    mylist=[]
    count=0
    while q:
        count+=1
        x=q.popleft()
        if count%m==0:
            mylist.append(x)
        else:
            q.append(x)
```

```
    print(",".join(str(x) for x in mylist))
```

今日化学论文

```
s=input().strip()
stack=[]

cur_s=""

i=0
n=len(s)while i <n:

    if s[i]=="[":

        stack.append(cur_s)

        cur_s=""

        i+=1

    elif s[i]=="]":

        pre_s=stack.pop()
        j=0

        num=""

        while j<len(cur_s) and cur_s[j].isdigit():
            num+=cur_s[j]
            j+=1
        s_part=cur_s[j:]
        r_int=int(num) if num else 0
        cur_s=pre_s+r_int*s_part
        i+=1
    else:
        cur_s+=s[i]
        i+=1print(cur_s)
```

Sequence

```
import heapqdef merge_sequences(seq1, seq2, n):
    seq1.sort()
    seq2.sort()
    min_heap = [(seq1[i] + seq2[0], i, 0) for i in range(len(seq1))]

    result = []
    while n > 0 and min_heap:
        current_sum, i, j = heapq.heappop(min_heap)
```

```
        result.append(current_sum)
        if j + 1 < len(seq2):
            heapq.heappush(min_heap, (seq1[i] + seq2[j + 1], i, j +
1))
        n -= 1
    return resultdef min_sequence_sums(m, n, sequences):
    for seq in sequences:
        seq.sort()
    current_min_sums = sequences[0]
    for i in range(1, m):
        current_min_sums = merge_sequences(current_min_sums,
sequences[i], n)
    return current_min_sums
t = int(input()) for _ in range(t):
    m, n = map(int, input().split())
    sequences = [list(map(int, input().split())) for _ in range(m)]
    results = min_sequence_sums(m, n, sequences)

    print(' '.join(map(str, results[:n])))
```

邮箱验证

```
while True:
    try:
        n = input()
    except EOFError:
        break

    if n[0] == "." or n[0] == "@" or n[-1] == "@" or n[-1] == ".":

        print("NO")

        continue

    if n.count("@")!=1:

        print("NO")

        continue
    else:

        a,b=n.split("@")

        if "." not in b or a[-1]=="." or b[0]==".":

            print("NO")
```

```
            continue
        else:

            print("YES")
```

A Knight's Journey

```
dx = [-2, -1, 1, 2, -2, -1, 1, 2]
dy = [1, 2, 2, 1, -1, -2, -2, -1]

mydict={0:"A",1:"B",2:"C",3:"D",4:"E",5:"F",6:"G",7:"H",8:"I",9:"J",10:

"K",11:"L",12:"M",13:"N",14:"O",15:"P",16:"Q",17:"R",18:"S",19:"T",20:

"U",21:"V",22:"W",23:"X",24:"Y",25:"Z"}def

dfs(x,y,a,b,visited,s,mylist):
    if len(s)==2*a*b:
        if mylist and s>min(mylist):
            return
        else:
            mylist.append(s)
            return
    for z in range(8):
        nx = x + dx[z]
        ny = y + dy[z]
        if 0 <= nx < a and 0 <= ny < b and not visited[nx][ny]:
            visited[nx][ny] = True
            s += str(mydict[ny])
            s += str(nx + 1)
            if mylist and s>min(mylist):
                s=s[:-2]
                visited[nx][ny]=False
                continue
            dfs(nx, ny, a, b, visited, s, mylist)
            visited[nx][ny] = False
            s = s[:-2]
n=int(input())for i in range(n):
    mylist=[]

    s=""

    a,b=map(int,input().split())
    visited =[[False]*b for _ in range(a)]
    for y in range(b):
        for x in range(a):
```

```
            s += str(mydict[y])
            s += str(x + 1)
            visited[x][y]=True
            dfs(x, y, a, b, visited, s, mylist)
            visited[x][y]=False
            s=s[:-2]
    mylist.sort()

    print(f"Scenario #{i+1}:")

    if mylist:
        print(mylist[0])
    else:

        print("impossible")

    print()
```

Grandpa is Famous

```
while True:
    n,m=map(int,input().split())
    if n==m==0:
        break
    mydict={}
    for i in range(n):
        nums=list(input().split())
        for x in nums:
            if x in mydict:
                mydict[x]+=1
            else:
                mydict[x]=1
    list1=[]
    for y in mydict:
        list1.append((y,mydict[y]))
    list1.sort(key=lambda x:x[1],reverse=True)
    list2=[]
    for z in list1:
        if z[1]==list1[1][1]:
            list2.append(int(z[0]))
    list2.sort()

    print(" ".join(str(x) for x in list2))
```

垃圾炸弹

```
d=int(input())
n=int(input())
mylist=[[0]*1025 for _ in range(1025)]for i in range(n):
    x,y,j=map(int,input().split())
    l1=x-d
    l2=y-d
    r1=x+d+1
    r2=y+d+1
    if l1<0:
        l1=0
    if l2<0:
        l2=0
    if r1>1025:
        r1=1025
    if r2>1025:
        r2=1025
    for a in range(l1,r1):
        for b in range(l2,r2):
            mylist[a][b] += j
num=max(max(_) for _ in mylist)
count=0for i in range(1025):
    for j in range(1025):
        if mylist[i][j]==num:
            count+=1print(count,num)
```

反反复复

```
n=int(input())
s=input()
mylist=[]

result=""for x in range(int(len(s)/n)):

    if x%2==0:
        mylist.append(s[n*x:x*n+n])
    else:
        mylist.append(s[n * x+n-1:n * x-1 :-1])for i in
range(len(mylist[0])):
    for j in range(len(mylist)):
        result+=mylist[j][i]
print(result)
```

宗教信仰

```
k=0while 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)}")
```

方程求解

```
def f(x):
    return x**3-5*x**2+10*x-80def df(x):
    return 3*x**2-10*x+10def newton(x,times):
    a=x
    for i in range(times):
        da=-f(a)/df(a)
        a=a+da
        if abs(da)<=1e-10:
            break
    return a

x=0print(f"{newton(x,50):.9f}")
```

倒排索引查询

```
n=int(input())
set_list=[]for i in range(n):
    nums=list(map(int,input().split()))
    set_list.append(set(nums[1:]))
m=int(input())for y in range(m):
```

```
    judge=list(map(int,input().split()))
    must=[i for i in range(n) if judge[i]==1]
    must_not=[i for i in range(n) if judge[i]==-1]
    same=set_list[must[0]].copy()
    for x in must[1:]:
        same={i for i in same if i in set_list[x]}
        if not same:
            break
    if not same:

        print("NOT FOUND")

        continue
    ban=set()
    for _ in must_not:
        for x in set_list[_]:
            ban.add(x)
    result=[i for i in same if i not in ban ]
    result.sort()
    if not result:

        print("NOT FOUND")

    else:

        print(" ".join(str(x) for x in result))
```

二维矩阵上的卷积运算

```
m,n,p,q=map(int,input().split())
matrix=[]for _ in range(m):
    matrix.append(list(map(int,input().split())))
core=[]for i in range(p):
    core.append(list(map(int,input().split())))
result=[[0]*(n-q+1) for x in range(m-p+1)]for i in range(m-p+1):
    for j in range(n-q+1):
        for x in range(p):
            for y in range(q):
                result[i][j]+=matrix[i+x][j+y]*core[x][y]for lst
in result:

    print(" ".join(str(x) for x in lst))
```

月度开销

```
n,m=map(int,input().split())
```

```
num_list=[]for i in range(n):
    num_list.append(int(input()))
left=max(num_list)
right=sum(num_list)while left<right:
    sumall=0
    count=1
    mid=(left+right)//2
    for x in num_list:
        if sumall+x>mid:
            sumall=0
            count+=1
        sumall+=x
    if count>m:
        left=mid+1
    else:
        right=midprint(right)
```

滑雪

```
r,c=map(int,input().split())
list1=[]
list1.append([10001 for i in range(c+2)])for i in range(r):
    list1.append([10001]+list(map(int,input().split()))+[10001])
list1.append([10001 for i in range(c+2)])
list2=[]for i in range(1,r+1):
    for j in range(1,c+1):
        list2.append((list1[i][j],i,j))
list3=[[1 for i in range(c+2)] for j in range(r+2)]
list2.sort(key=lambda x:x[0],reverse=True)for x in list2:
    if list1[x[1]+1][x[2]]<list1[x[1]][x[2]]:

list3[x[1]+1][x[2]]=max(list3[x[1]][x[2]]+1,list3[x[1]+1][x[2]]
)
    if list1[x[1]][x[2]+1] < list1[x[1]][x[2]]:
        list3[x[1]][x[2]+1] = max(list3[x[1]][x[2]] + 1,
list3[x[1]][x[2]+1])
    if list1[x[1] - 1][x[2]] < list1[x[1]][x[2]]:
        list3[x[1]- 1][x[2]] =max(list3[x[1]][x[2]] + 1, list3[x[1]
- 1][x[2]])
    if list1[x[1]][x[2]-1] < list1[x[1]][x[2]]:
        list3[x[1]][x[2]-1] = max(list3[x[1]][x[2]] + 1,
list3[x[1]][x[2]-1])print(max(max(x) for x in list3))
```

矩阵运算

```
import sys
matrix1=[]
matrix2=[]
matrix3=[]
a, b = map(int, input().split())for i in range(a):
    matrix1.append(list(map(int, input().split())))
c,d=map(int,input().split())for i in range(c):
    matrix2.append(list(map(int, input().split())))
e,f=map(int,input().split())for i in range(e):
    matrix3.append(list(map(int, input().split())))
list1=[[0]*d for i in range(a)]if b!=c or a!=e or d!=f:

    print("Error!")

    sys.exit()for i in range(a):
    for j in range(d):
        for x in range(b):
            list1[i][j]+=matrix1[i][x]*matrix2[x][j]
list2=[[0]*f for i in range(e)]for x in range(e):
    for y in range(f):
        list2[x][y]=list1[x][y]+matrix3[x][y]for row in list2:

    print(" ".join(str(x) for x in row))
```

Radar Installation

```
import math
num=0while True:
    num+=1
    count = 1
    n, d = map(int, input().split())
    if n==d==0:
        break
    list1 = []
    for i in range(n):
        a, b = map(int, input().split())
        if b>d or b<0:
            count=-1
        else:
            delta=math.sqrt(d**2-b**2)
            list1.append((a-delta,a+delta))
    if count!=-1:
        list1.sort(key=lambda x:x[1])
```

```
        right=list1[0][1]
        for x in range(len(list1)):
            if list1[x][0]<=right:
                continue
            else:
                count+=1
                right=list1[x][1]

    print(f"Case {num}: {count}")

    input()
```

倒排索引

```
n=int(input())
mydict={}for i in range(n):
    x=list(input().split())
    for y in range(1,len(x)):
        if x[y] not in mydict:
            mydict[x[y]] = [i + 1]
        else:
            if i+1 not in mydict[x[y]]:
                mydict[x[y]].append(i+1)
results=[]
m=int(input())for i in range(m):
    z=input().strip()
    if z in mydict:

        results.append((" ".join(str(x) for x in sorted(mydict[z]))))

    else:

        results.append("NOT FOUND")for x in results:

    print(x)
```

马走日

```
# pylint: skip-file
dx=[1,1,2,2,-1,-1,-2,-2]
dy=[2,-2,1,-1,2,-2,1,-1]def dfs(x,y,cnt):
    global count
    used[x][y]=True
    if cnt==n*m-1:
        count+=1
    for i in range(8):
        nx=x+dx[i]
        ny=y+dy[i]
```

```
        if 0<=nx<n and 0<=ny<m:
            if not used[nx][ny]:
                used[nx][ny]=True
                dfs(nx,ny,cnt+1)
                used[nx][ny]=False
t=int(input())for i in range(t):
    n,m,x,y=map(int,input().split())
    used=[[False for i in range(m)] for _ in range(n)]
    count=0
    dfs(x,y,0)
    print(count)
```

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

模型整理

```
n=int(input())

mydict={"M":1e6,"B":1e9}

mydict1={}for i in range(n):
```

```
    a,b=input().split("-")
    if a in mydict1:
        mydict1[a].append(b)
    else:
        mydict1[a]=[]
        mydict1[a].append(b)
sorted_keys=sorted(mydict1.keys())for x in sorted_keys:
    mydict1[x].sort(key=lambda x:float(x[:-1])*mydict[x[-1]])for
y in sorted_keys:
    mystr=", ".join(str(x) for x in mydict1[y])

    print(f"{y}: {mystr}")
```

双端队列

```
t=int(input())for i in range(t):
    stack=[]
    n=int(input())
    for i in range(n):
        a,b=map(int,input().split())
        if a==1:
            stack.append(b)
        elif a==2:
            if len(stack)==0:
                continue
            if b==0:
                stack.pop(0)
            elif b==1:
                stack.pop()
    if len(stack)==0:

        print("NULL")

    else:

        print(" ".join(str(x) for x in stack))
```

约瑟夫问题

```
def myfunc(n,m):
    list1=list(range(1,n+1))
    index=0
    while list1:
```

```
        if len(list1)==1:
            return list1[0]
        temp=list1.pop(0)
        index+=1
        if index==m:
            index=0
            continue
        list1.append(temp)while True:
    n,m=map(int,input().split())
    if n==m==0:
        break
    else:
        print(myfunc(n,m))
```

后序表达式求值

```
n=int(input())for i in range(n):
    stack=[]
    s=list(input().split())
    for x in range(len(s)):

        if s[x]!="+" and s[x]!="-" and s[x]!="*" and s[x]!="/":

            stack.append(float(s[x]))
        else:
            op1=stack.pop()
            op2=stack.pop()

            if s[x]=="+":

                stack.append(op1+op2)

            elif s[x]=="-":

                stack.append(op2-op1)

            elif s[x]=="*":

                stack.append(op1*op2)

            elif s[x]=="/":

                stack.append(op2/op1)

    num=f"{stack[-1]:.2f}"

    print(num)
```

十进制到八进制

```
def myfunc(x):
    if int(x)<8:
        return str(x)
    else:
        return myfunc(str(int(x)//8))+str(int(x)%8)
x=input()print(myfunc(x))
```

Fraction 类

```
a,b,c,d=map(int,input().split())
top1=a*d+c*b
low1=b*d
top=a*d+c*b
low=b*dwhile low1:


top1,low1=low1,top1%low1print(str(int(top/top1))+"/"+str(int(low
/top1)))
```

位查询

```
n,m=map(int,input().split())
all_nums=list(map(int,input().split()))for i in range(m):
    s1,s2=input().split( )

    if s1=="Q":

        num=0
        for j in all_nums:

            if int(s2)<=len(bin(j))-1 and bin(j)[-(1+int(s2))]=="1":

                num+=1
        print(num)

    elif s1=="C":

        for x in range(len(all_nums)):
            all_nums[x]+=1
```

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
```
每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
```
H 指数
```
class Solution:    def hIndex(self, citations: List[int]) -> int:
s=sorted(citations,reverse=True)        h=0        i=0
n=len(citations)        while i<n and s[i]>h:           h+=1
i+=1        return h
```
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
```
蛇梯棋
```
class Solution:    def snakesAndLadders(self, board: List[List[int]]) ->
int:        n=len(board)        dic={}        cnt=1        for i in
range(n-1,-1,-1):           if (n-i)%2==1:               for j in range(n):
dic[cnt]=(i,j)                   cnt+=1           else:               for
j in range(n-1,-1,-1):                   dic[cnt]=(i,j)
cnt+=1              q=deque()        q.append((1,0))        vis=set()
while q:           m,step=q.popleft()           for i in range(1,7):
nm=m+i              if nm>n**2:                  break              if
board[dic[nm][0]][dic[nm][1]]!=-1:
nm=board[dic[nm][0]][dic[nm][1]]               if nm==n**2:
return step+1              if nm not in vis:                   vis.add(nm)
q.append((nm,step+1))        return -1
```
森林中的兔子
```
class Solution:    def numRabbits(self, answers: List[int]) -> int:
dic=defaultdict(int)        res=0        for x in answers:
dic[x]+=1        for y in dic:           if dic[y]%(y+1)==0:
res+=(y+1)*(dic[y]//(y+1))           else:
res+=(y+1)*(dic[y]//(y+1))               res+=y+1        return res
```
统计完全数组的数量

```
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
```

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
```

课程表 II

```
class Solution:    def findOrder(self, num: int, pre: List[List[int]]) ->
List[int]:        import heapq        graph={}        dic={}        for i in
range(num):            dic[i]=0            graph[i]=[]        for x,y in pre:
graph[y].append(x)            dic[x]+=1        ans=[]        cnt=0
q=[]        for z in dic:            if dic[z]==0:
heapq.heappush(q,z)        while q:            x=heapq.heappop(q)
ans.append(x)            cnt+=1            for j in graph[x]:
dic[j]-=1                if dic[j]==0:
heapq.heappush(q,j)        return ans if cnt==num else []
```

零数组转换

```
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
```

统计最大元素最少出现 k 次的数组

```
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
```

统计坏数对的次数

```
class Solution:    def countBadPairs(self, nums: List[int]) -> int:
mp=defaultdict(int)        res=0        for i,x in enumerate(nums):
res+=i-mp[x-i]            mp[x-i]+=1        return res
```

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
```

课程表

```
class Solution:    def canFinish(self, numCourses: int, prerequisites:
List[List[int]]) -> bool:        vis=set()        graph={}        dic={}
q=[]        for i in range(numCourses):            graph[i]=[]
dic[i]=0        for x in prerequisites:            graph[x[0]].append(x[1])
dic[x[1]]+=1        for y in dic:            if dic[y]==0:
heapq.heappush(q,y)        while q:            p=heapq.heappop(q)
vis.add(p)          for j in graph[p]:                dic[j]-=1
if dic[j]==0:                    heapq.heappush(q,j)        return
len(vis)==numCourses
```

网络延迟时间

```
class Solution:    def networkDelayTime(self, times: List[List[int]], n:
int, k: int) -> int:        q=[]        heapq.heappush(q,(0,k))
time=[float('inf') for _ in range(n+1)]        graph=defaultdict(list)
for x in times:            graph[x[0]].append((x[1],x[2]))        time[0]=0
time[k]=0        while q:            t,ck=heapq.heappop(q)            if
t>time[ck]:                continue            for j in graph[ck]:
nt=t+j[1]                if nt<time[j[0]]:                    time[j[0]]=nt
heapq.heappush(q,(nt,j[0]))        for y in time :            if
y==float('inf'):                return -1        return max(time)
```

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]
```

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)
```

最大质数子字符串之和

```
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)
```

k 站中转内的最大航班

```
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
```

网络传送门旅游

```
class Solution:    def minMoves(self, matrix: List[str]) -> int:        if
matrix[-1][-1]=="#":            return -1        from collections import
deque        m=len(matrix)        n=len(matrix[0])        q=deque()
dic=defaultdict(list)        cnt=0        for i in range(m):
matrix[i]=list(matrix[i])            for j in range(n):                if
matrix[i][j].isalpha():                    dic[matrix[i][j]].append((i,j))
if matrix[0][0].isalpha():            for t in dic[matrix[0][0]]:
q.append(t)                matrix[t[0]][t[1]]='#'        else:
q.append((0,0))            matrix[0][0]="#"        while q:
cnt+=1            for _ in range(len(q)):                cx,cy=q.popleft()
if cx==m-1 and cy==n-1:                    return cnt-1                for
dx,dy in [(0,1),(0,-1),(1,0),(-1,0)]:                    nx=cx+dx
ny=cy+dy                    if 0<=nx<=m-1 and 0<=ny<=n-1 and
matrix[nx][ny]!='#':                        if matrix[nx][ny]=='.':
q.append((nx,ny))                            matrix[nx][ny]='#'
else:                            for c in dic[matrix[nx][ny]]:
q.append(c)                                matrix[c[0]][c[1]]='#'
return -1
```

最长同值路径

```
class Solution:    def longestUnivaluePath(self, root: Optional[TreeNode])
-> int:        if not root:            return 0        ans=0        def
```

```
dfs(node):            nonlocal ans           if not node:
return -1           l=dfs(node.left)+1          r=dfs(node.right)+1
if node.left and node.val!=node.left.val:l=0          if node.right and
node.val !=node.right.val:r=0          ans=max(ans,l+r)
return max(l,r)       dfs(root)       return ans
```

修建二叉搜索树

```
class Solution:    def trimBST(self, root: Optional[TreeNode], low: int,
high: int) -> Optional[TreeNode]:       if not root:          return
if root.val<low:          return self.trimBST(root.right,low,high)
if root.val>high:          return self.trimBST(root.left,low,high)
root.left=self.trimBST(root.left,low,high)
root.right=self.trimBST(root.right,low,high)       return root
```

打家劫舍 III

```
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))
```

二叉树最大宽度

```
class Solution:    def widthOfBinaryTree(self, root: Optional[TreeNode]) ->
int:       if not root:          return 0       from collections import
deque       q=deque()       q.append((root,1))       res=[]
while q:          list1=[]          for _ in range(len(q)):
x,i=q.popleft()              list1.append(i)              if x.left:
q.append((x.left,2*i-1))              if x.right:
q.append((x.right,2*i))          res.append(list1[:])       ans=[]
for x in res:          if x:              ans.append(x[-1]-x[0])
return max(ans)+1
```

最大二叉树

在二叉树中增加一行

根据二叉树创建字符串

把二叉搜索树转换成累加树

在每个树行中找最大值

找树左下角的值

出现次数最多的子树元素和

删除二叉搜索树中的节点

路径总和 III

N 叉树的层序遍历

验证二叉树的前序序列化

二叉树的最近公共祖先

二叉搜索树的最近公共祖先

二叉搜索树中第 K 小的元素

二叉树的右视图

有序链表转换二叉搜索树

二叉树展开为链表
填充每个节点的下一个右侧节点指针 II
填充每个节点的下一个右侧节点指针
二叉树中的最大路径和
二叉树的最大深度
从中序与后序遍历序列构造二叉树
从前序与中序遍历序列构造二叉树
二叉树的层序遍历 II
不同的二叉搜索树

彩灯装饰记录 II

开幕式焰火
判断根结点是否等于子结点之和
找出克隆二叉树中的相同节点
从根到叶的二进制数之和
二叉树的堂兄弟节点
单值二叉树
二叉搜索树的范围和
递增顺序搜索树
叶子相似的树
二叉搜索树节点最小距离

数据流中的第 K 大元素

二叉搜索树中的搜索
二叉树中第二小的节点

两数之和 IV - 输入二叉搜索树

二叉树的层平均值
合并二叉树

N 叉树的后序遍历

N 叉树的前序遍历

另一棵树的子树
二叉树的坡度

N 叉树的最大深度

二叉树的直径
二叉搜索树的最小绝对差
二叉搜索树中的众数
左叶子之和
二叉树的所有路径
翻转二叉树
二叉树的后序遍历

二叉树的前序遍历

路径总和 II

路径总和
二叉树的最小深度
平衡二叉树
对称二叉树
相同的树
验证二叉搜索树

不同的二叉搜索树 II

针对图的路径存在性查询 I

N 皇后

电话号码的字母组合
子集
二叉树的锯齿形层序遍历
完全二叉树的节点个数

移除最小数对使数组有序 II

求根节点到叶节点数字之和
将有序数组转换为二叉搜索树

LRU 缓存

分割回文串
二叉树的层序遍历
二叉树的中序遍历
单词搜索
设计浏览器历史记录
回文链表
合并两个有序链表

选出和最大的 K 个元素

反转链表
相交链表
只出现一次的数字

搜索二维矩阵 II

颜色分类
全排列

01 矩阵

袋子里最少数目的球

腐烂的橘子
旋转图像
合并区间
构造乘积矩阵

Bigram 分词

买卖股票的最佳时机
盛最多水的容器
将每个元素替换为右侧最大元素
最小栈

删除链表的倒数第 N 个结点

杨辉三角
搜索插入位置
有效的括号