# dp

核电站

```python

# 有最长连续长度限制的放置问题

dp=[0]\*(n+1)

dp[0]=1

for i in range(1,n+1):

if i<m:

dp[i]=dp[i-1]\*2

elif i==m:

dp[i]=dp[i-1]\*2-1

else:

dp[i]=dp[i-1]\*2-dp[i-m-1]

print(dp[n])

```

flowers(474D)

```python

# 成组放置问题

dp=[0]\*100001

dp[0]=1

for i in range(1,100001):

dp[i]=dp[i-1]

if i>=k:

dp[i]=(dp[i]+dp[i-k])%MOD

for i in range(1,100001):

dp[i]=(dp[i]+dp[i-1])%MOD

print((dp[b]-dp[a-1]+MOD)%MOD)

```

k-tree（CF431C）

```python

# 有序的整数划分问题

A = [1] + [0] \* n

B = [1] + [0] \* n

for i in range(1, n + 1):

for j in range(1, min(i,k)+1):

A[i] = (A[i] + A[i - j]) % mod

for j in range(1, min(d, i + 1)):

B[i] = (B[i] + B[i - j]) % mod

print((A[n] - B[n]) % mod)

```

公共子序列

```python

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

for i in range(1, m + 1):

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

if x[i - 1] == y[j - 1]:

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

else:

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

print(dp[m][n])

```

宠物小精灵之收服

```python

# 双限制背包问题

# dp[i][j]为捕获i个小精灵，皮卡丘剩余j体力时，剩余的最大精灵球数量

dp=[[-1]\*(m+1) for \_ in range(k+1)]

dp[0][m]=n

for i in range(k):

cost,harm=map(int,input().split())

for blood in range(m):

for catch in range(i+1):

pre\_blood=blood+harm

if pre\_blood<=m and dp[catch][pre\_blood]!=-1:

dp[catch+1][blood]=max(dp[catch+1][blood],dp[catch][pre\_blood]-cost)

for i in range(k,-1,-1):

for j in range(m,-1,-1):

if dp[i][j]!=-1:

print(i,j)

exit()

```

最佳凑单

```python

# 稀疏桶

a,b=map(int,input().split());c={0}

for i in map(int,input().split()):

for j in c.copy():

if j<b:c.add(i+j)

for i in sorted(c):

if i>=b:print(i);exit()

print(0)

```

piggy-bank

```python

# 完全背包中的最优解问题

dp = [float('inf')] \* (n + 1)

dp[0] = 0

for i in range(m):

for j in range(w[i], n + 1):

dp[j] = min(dp[j], dp[j - w[i]] + v[i])

if dp[n] == float('inf'):

print('This is impossible.')

else:

print(f'The minimum amount of money in the piggy-bank is {dp[n]}.')

```

coins

```python

# 多重背包中的方案数问题

dp = [0] \* (m + 1)

dp[0] = 1

for i in range(n):

coin, count = value[i], counts[i]

for j in range(count):

for v in range(m, coin - 1, -1):

dp[v] += dp[v - coin]

print(sum(1 for x in dp[1:] if x > 0))

```

```python

# 二进制&位运算优化

dp = 1 # 初始化为只有第0位为1的整数，表示初始时只有0这个和是可以达到的

mask = (1 << (m + 1)) - 1 # 创建一个掩码，用于限制dp的长度

for value, count in zip(values, counts):

while count:

k = 1

while k <= count: # 找到不超过count的最大2的幂

dp = (dp | (dp << (value \* k))) & mask

count -= k

k <<= 1

print(bin(dp).count('1') - 1)

```

NBA门票

```python

# 多重背包中的最优解问题

dp=[float('inf')]\*(n+1)

dp[0]=0

for i in range(6,-1,-1):

cur=price[i]

for k in range(n,cur-1,-1):

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

if k>=cur\*j:

dp[k]=min(dp[k],dp[k-cur\*j]+j)

else:

break

if dp[-1]==float('inf'):

print('Fail')

else:

print(dp[-1])

```

```python

# 二进制优化

dp = [float('inf')] \* (n + 1)

dp[0] = 0

for i in range(7):

cur\_price = price[i]

cur\_num = nums[i]

k = 1

while cur\_num > 0:

use\_num = min(cur\_num, k)

cur\_num -= use\_num

for j in range(n, cur\_price \* use\_num - 1, -1):

dp[j] = min(dp[j], dp[j - cur\_price \* use\_num] + use\_num)

k \*= 2

if dp[-1] == float('inf'):

print('Fail')

else:

print(dp[-1])

```

神奇的口袋

```python

# 0-1背包中的方案数问题

dp=[0]\*(41)

dp[0]=1

for item in items:

for volume in range(40,item-1,-1):

dp[volume]+=dp[volume-item]

print(dp[40])

```

采药

```python

# 0-1背包中的最优解问题

dp=[-1]\*(T+1)

dp[0]=0

for \_ in range(m):

t,v=map(int,input().split())

for i in range(T,t-1,-1):

if dp[i-t]!=-1:

dp[i]=max(dp[i],dp[i-t]+v)

print(max(dp))

```

最长上升子序列

```python

dp=[1]\*n

for i in range(n):

for j in range(i):

if numbers[j]<numbers[i]:

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

print(max(dp))

```

复杂的整数划分问题

```python

# N划分成K个正整数之和

def divide\_k(n,k):

dp=[[0]\*(k+1) for \_ in range(n+1)]

for i in range(n+1):

dp[i][1]=1

for i in range(1,n+1):

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

if i>=j:

# dp[i-1][j-1]为包含1的划分的数量

# 若不包含1，我们对每个数-1仍为正整数，划分数量为dp[i-j][j]

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

return dp[n][k]

# N划分成若干个不同正整数之和

def divide\_dif(n):

# dp[i][j]表示将数字 i 划分，其中最大的数字不大于 j 的方法数量

dp = [[0] \* (n + 1) for \_ in range(n + 1)]

for i in range(1, n + 1):

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

if i < j:

dp[i][j] = dp[i][i]

elif i == j:

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

# 用/不用j

else:

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

return dp[n][n]

```

世界杯只因

```python

# 区间覆盖问题

dp=[1<<30]\*(n+1)

dp[0]=0

for i in range(n):

l=max(1,i+1-a[i])

r=min(n,i+1+a[i])

if dp[r]>dp[l-1]+1:

for j in range(l,r+1):

dp[j]=min(dp[j],dp[l-1]+1)

print(dp[-1])

```

幸福的寒假生活

```python

# 不重叠区间的最优解问题

dp=[0]\*46

for i in range(1,46):

dp[i]=dp[i-1]

for start,end,happiness in data:

if end==i:

dp[i]=max(dp[i],dp[start-1]+happiness)

print(dp[-1])

```

# greedy

holiday hotel

```python

# 筛选问题

hotels=[tuple(map(int,input().split())) for \_ in range(n)]

hotels.sort(key=lambda x:(x[0],x[1]))

candidates=1

max\_cost\_so\_far=hotels[0][1]

for i in range(n):

if hotels[i][1]<max\_cost\_so\_far:

candidates+=1

max\_cost\_so\_far=hotels[i][1]

print(candidates)

```

expedition

```python

# 加油问题

stations.sort() # 按距离升序

stations.append((L, 0) # 添加起点

pq = [] # 最大堆

stops, prev, fuel = 0, 0, P

for location, capacity in stations:

fuel -= location - prev

while pq and fuel < 0: # 当前燃料不够到达下一加油站

fuel += -heapq.heappop(pq) # 选择提供最多燃料的加油站加油

stops += 1

if fuel < 0: print(-1) ;exit() # 无法到达下一个加油站或终点

heapq.heappush(pq, -capacity)

prev = location

print(stops)

```

畜栏保留问题

```python

# 时间调度问题

# cows元素：（index，start, end）

cows.sort(key=lambda x:x[1])

space=[]

space\_num=[0]\*n

max\_num=1

for cow in cows:

if space:

if space[0][0]<cow[1]:

space\_num[cow[0]]=space\_num[space[0][1]]

heappop(space)

else:

space\_num[cow[0]]=max\_num+1

max\_num+=1

else:

space\_num[cow[0]]=1

heappush(space,(cow[2],cow[0],cow[1]))

print(len(space))

for num in space\_num:

print(num)

```

建筑修建

```python

# 区间调度问题

def generate(x,w):

for i in range(max(0,x-w+1),min(x,m-w)+1):

a.append((i,i+w))

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

a=[]

for \_ in range(n):

x,w=map(int,input().split())

generate(x,w)

a.sort(key=lambda x:x[1])

ans=0;end=0

for i in a:

if i[0]>=end:

ans+=1;end=i[1]

print(ans)

```

月度开销

```python

# 二分贪心

def reachable(expenses,target,m):

......

l,r=max(expenses),sum(expenses)

while l<=r:

mid=(l+r)//2

if reachable(expenses,mid,m):

r=mid-1

else:

l=mid+1

print(l)

```

wooden sticks

```python

# 单调子列的最小拆分数

data=list(zip(data[0::2],data[1::2]))

data.sort(key=lambda x:(x[0],x[1]))

flag=[False]\*n

cnt=0

for i in range(n):

if flag[i]:

continue

cur=data[i][1]

cnt+=1

for j in range(i,n):

if flag[j]==False and data[j][1]>=cur:

flag[j]=True

cur=data[j][1]

print(cnt)

```

potions(CF1526C1)

```python

# 对前缀和限制的最多选择问题

total\_health = 0

min\_heap = []

for health\_change in health\_changes:

heapq.heappush(min\_heap, health\_change)

total\_health += health\_change

if total\_health < 0:

total\_health -= heapq.heappop(min\_heap)

potions\_drank = len(min\_heap)

print(potions\_drank)

```

# implementation

质数筛

```python

n=10\*\*4

prime=[True for \_ in range(n+1)]

p=2

while p\*p<=n:

if prime[p]:

for i in range(p\*p,n+1,p):

prime[i]=False

p+=1

primes=set([p for p in range(2,n+1) if prime[p]])

```

排列

```python

# 排列问题

def next\_permutation():

i=n-2

while a[i]>a[i+1]:

i-=1

j=n-1

while a[j]<a[i]:

j-=1

a[i],a[j]=a[j],a[i]

a[i+1:]=a[i+1:][::-1]

```

分解因数

```python

def decompositions(n,minfactor):

if n==1:

return 1

count=0

for i in range(minfactor,n+1):

if n%i==0:

count+=decompositions(n//i,i)

return count

print(decompositions(x,2))

```

最短前缀

```python

# 字典树

def insert(root, word):

# 将单词插入字典树

node = root

for char in word:

if char not in node:

node[char] = {} # 如果字符不存在，则在当前节点下创建一个新节点

node = node[char] # 移动到下一个节点

node['count'] = node.get('count', 0) + 1 # 更新节点上的计数

def find\_prefix(root, word):

# 在字典树中为单词找到独特的最短前缀

node = root

prefix = ""

for char in word:

if node[char].get('count', 1) == 1:

return prefix + char # 如果该节点的计数为1，则返回当前前缀加上该字符

prefix += char # 否则，将字符添加到前缀中

node = node[char] # 继续遍历下一个字符

return prefix

root = {}

for word in words:

insert(root, word)

for word in words:

prefix = find\_prefix(root, word)

print(f"{word} {prefix}")

```

假币问题

```python

# 集合运算

for \_ in range(3):

left,right,judge=input().split()

left,right=set(left),set(right)

if judge=='even':

even.append(left|right) # 并

elif judge=='up':

if len(heavy)!=0:heavy&=left;light&=right # 交

else:heavy=left;light=right

else:

if len(heavy)!=0:heavy&=right;light&=left

else:heavy=right;light=left

for i in even:

light-=i;heavy-=i # 差

```

最小新整数

```python

# 单调栈

def removeKDigits(num, k):

stack = []

for digit in num:

while k and stack and stack[-1] > digit:

stack.pop()

k -= 1

stack.append(digit)

# 如果还未删除k位，从尾部继续删除

while k:

stack.pop()

k -= 1

return int(''.join(stack))

```

护林员盖房子

```python

# 寻找最大全0子矩阵

# 单调栈

for row in ma:

stack=[]

for i in range(n):

h[i]=h[i]+1 if row[i]==0 else 0

while stack and h[stack[-1]]>h[i]:

y=h[stack.pop()]

w=i if not stack else i-stack[-1]-1

ans=max(ans,y\*w)

stack.append(i)

while stack:

y=h[stack.pop()]

w=n if not stack else n-stack[-1]-1

ans=max(ans,y\*w)

print(ans)

```

接雨水

```python

# 单调栈

stack=[]

water=0

for i,h0 in enumerate(h):

while stack and h0>h[stack[-1]]:

top=stack.pop()

if not stack:

break

width=i-stack[-1]-1

depth=min(h[stack[-1]],h0)-h[top]

water+=width\*depth

stack.append(i)

print(water)

```

滑动窗口最大值

```python

q=[]

for i in range(k):

heappush(q,(-a[i],i))

ans=[-q[0][0]]

for i in range(k,n):

heappush(q,(-a[i],i))

while q[0][1]<=i-k:

heappop(q)

ans.append(-q[0][0])

print(\*ans)

```

股票买卖

```python

buy1=buy2=float('inf')

sell1=sell2=0

for price in prices:

buy1=min(buy1,price)

sell1=max(sell1,price-buy1)

buy2=min(buy2,price-sell1)

sell2=max(sell2,price-buy2)

print(sell2)

```

number of ways

```python

n=int(input())

nums=list(map(int,input().split()))

s=sum(nums)

if s%3!=0:

print(0)

exit()

s=s//3

ans,cnt,pre\_sum=0,0,0

for i in range(n-1):

pre\_sum+=nums[i]

if pre\_sum==s\*2:

ans+=cnt

if pre\_sum==s:

cnt+=1

print(ans)

```

查找最接近的元素

```python

# bisect库

from bisect import bisect\_left

def find\_closest(arr, target):

n = len(arr)

if target <= arr[0]:

return arr[0]

if target >= arr[n-1]:

return arr[n-1]

pos = bisect\_left(arr, target)

if (arr[pos] - target) < (target - arr[pos - 1]):

return arr[pos]

else:

return arr[pos - 1]

```

consecutive subsequence(CF977F)

```python

# seq\_len中value为以key结尾的consecutive subsequence最大长度

seq\_len = {}

max\_len = 0

for num in nums:

seq\_len[num] = seq\_len.get(num - 1, 0) + 1

if seq\_len[num]>max\_len:

max\_len=seq\_len[num]

end=num

indices = []

# 从end开始回溯路径

for i in range(n,0,-1):

if nums[i - 1] == end:

indices.append(i)

end -= 1

if end == 0:

break

print(max\_len)

print(\*indices[::-1])

```

最大子矩阵

```python

# kadane算法：计算最大子段和

def kadane(arr):

max\_sum = float('-inf')

current\_sum = 0

for num in arr:

current\_sum = max(num, current\_sum + num)

max\_sum = max(max\_sum, current\_sum)

return max\_sum

# 计算前缀和

for i in range(1, n):

for j in range(n):

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

ans = float('-inf')

for top in range(n):

for bottom in range(top, n):

temp = [0] \* n

for i in range(n):

sum\_col = matrix[bottom][i]

if top > 0:

sum\_col -= matrix[top - 1][i]

temp[i] = sum\_col

ans = max(ans, kadane(temp))

print(ans)

```

完美的爱

```python

from collections import defaultdict

dic=defaultdict(list)

data=[0]+data

ans=0

for i in range(1,n+1):

# 构造一个减去了520i的前缀和

# 使得有相同前缀和的位置间即为满足题意的区间

data[i]+=data[i-1]-520

for i in range(n+1):

dic[data[i]].append(i)

for i in dic:

ans=max(ans,max(dic[i])-min(dic[i]))

print(ans\*520)

```

Kefa and company(CF580B)

```python

# 对滑动窗口内某变量极差限制的最优解问题

friends.sort()

ans=cur=left=0

for right in range(n):

cur+=friends[right][1]

while friends[right][0]-friends[left][0]>=d:

cur-=friends[left][1]

left+=1

ans=max(ans,cur)

print(ans)

```

in love

```python

# dict+heapq实现最值更新的效率提升

from heapq import heappop,heappush

from collections import defaultdict

q = int(input())

ldict, rdict = defaultdict(int), defaultdict(int)

pq\_l, pq\_r = [], []

for \_ in range(q):

op, l, r = map(str, input().split())

l, r = int(l), int(r)

if op == "+":

ldict[l] += 1; rdict[r] += 1

heappush(pq\_l, -l); heappush(pq\_r, r)

if op == "-":

ldict[l] -= 1; rdict[r] -= 1

while len(pq\_l) > 0 >= ldict[-pq\_l[0]]:

heappop(pq\_l)

while len(pq\_r) > 0 >= rdict[pq\_r[0]]:

heappop(pq\_r)

if len(pq\_l) > 0 and pq\_r[0] < -pq\_l[0]:

print("Yes")

else: print("No")

```

cat party

```python

# 桶套桶实现时间复杂度的降低

from collections import defaultdict

n = int(input())

a = list(map(int, input().split()))

cc = defaultdict(int) # color count

fc = defaultdict(int) # frequency count

ans = 0

for i in range(n):

c = a[i]

if cc[c] in fc:

fc[cc[c]] -= 1

if fc[cc[c]] == 0:

del fc[cc[c]]

cc[c] += 1

fc[cc[c]] += 1

if len(fc) == 1 and (1 in fc or list(fc.values())[0] == 1):

ans = i + 1

elif len(fc) == 2:

k = sorted(fc.keys())

if k[0] + 1 == k[1] and fc[k[1]] == 1 or k[0] == 1 and fc[k[0]] == 1:

ans = i + 1

print(ans)

```

最大点数（外太空2048）

```python

# 矩阵的翻转与转置应用

from copy import deepcopy

def slide(ma,dir,step):

if step==p:return max([max(i) for i in ma])

cur=deepcopy(ma)

if dir[1]:cur=list(zip(\*cur))

if dir[0]:cur=[i[::-1] for i in cur]

for j in range(len(cur)):

line=cur[j]

k=len(line)

line=[i for i in line if i!=0]

i=len(line)-2

while i>=0:

if line[i+1]==line[i]:

line[i]\*=2

del line[i+1]

i-=1

cur[j]=line+[0]\*(k-len(line))

if dir[0]:cur=[i[::-1] for i in cur]

if dir[1]:cur=list(zip(\*cur))

ans=0

for i in [0,1]:

for j in [0,1]:

ans=max(ans,slide(cur,(i,j),step+1))

return ans

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

ma=[list(map(int,input().split())) for \_ in range(m)]

res=0

for i in [0,1]:

for j in [0,1]:

res=max(res,slide(ma,(i,j),0))

print(res)

```

# bfs,dfs

棋盘问题

```python

# 回溯法

def dfs(row, k):

if k == 0:

return 1

if row == n:

return 0

count = 0

for col in range(n):

if board[row][col] == '#' and not col\_occupied[col]:

col\_occupied[col] = True

count += dfs(row + 1, k - 1)

col\_occupied[col] = False

count += dfs(row + 1, k)

return count

col\_occupied = [False] \* n

print(dfs(0, k))

```

迷宫问题

```python

# 要求输出路径的问题

queue = deque([((0, 0), [])])

while queue:

(x, y), path = queue.popleft()

if (x, y) == (n - 1, m - 1):

return path + [(x, y)]

for dx, dy in directions:

nx, ny = x + dx, y + dy

if 0 <= nx < n and 0 <= ny < m and not visited[nx][ny] and maze[nx][ny] == 0:

visited[nx][ny] = True

queue.append(((nx, ny), path + [(x, y)]))

return []

```

滑雪

```python

# 记忆化搜索

from functools import lru\_cache

@lru\_cache(maxsize=None)

def dfs(x,y):

ans=0

for dx,dy in dir:

nx,ny=x+dx,y+dy

if 0<=nx<m and 0<=ny<n and h[nx][ny]<h[x][y]:

ans=max(ans,dfs(nx,ny)+1)

return ans

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

h=[list(map(int,input().split())) for \_ in range(m)]

dir=[(0,1),(1,0),(-1,0),(0,-1)]

res=0

for i in range(m):

for j in range(n):

res=max(res,dfs(i,j))

print(res+1)

```

八皇后

```python

def is\_valid(board,row,col):

for i in range(row):

if board[i]==col or board[i]-i==col-row or board[i]+i==col+row:

return False

return True

def dfs(board):

if len(board)==8:

solutions.append(''.join(str(x+1) for x in board))

return

for col in range(8):

if is\_valid(board,len(board),col):

dfs(board+[col])

solutions=[]

dfs([])

```

小游戏

```python

# heap+bfs

from heapq import heappop,heappush

from copy import deepcopy

def bfs(i,j):

vis=set();q=[(0,-1,i,j)]

while q:

cnt,d,x,y=heappop(q)

vis.add((x,y))

if x==x2 and y==y2:return cnt

for i in range(4):

dx,dy=dir[i];nx,ny=x+dx,y+dy

if 0<=nx<h+2 and 0<=ny<w+2 and\

temp[nx][ny]==' ' and (nx,ny) not in vis:

if i==d:heappush(q,(cnt,i,nx,ny))

else:heappush(q,(cnt+1,i,nx,ny))

return -1

dir=[(-1,0),(1,0),(0,1),(0,-1)]

board=0

while True:

w,h=map(int,input().split())

if w==0:break

board+=1

print(f'Board #{board}:')

pair=0

ma=[[' ']\*(w+2)]

for \_ in range(h):

ma.append([' ']+list(input())+[' '])

ma.append([' ']\*(w+2))

while True:

y1,x1,y2,x2=map(int,input().split())

if y1==0:break

pair+=1

temp=deepcopy(ma)

temp[x2][y2]=' '

ans=bfs(x1,y1)

if ans==-1:print(f'Pair {pair}: impossible.')

else:print(f'Pair {pair}: {ans} segments.')

print()

```

城堡问题

```python

# 连通分量的数量和最大面积

def dfs(x,y):

if visited[x][y]:

return 0

visited[x][y]=True

size=1

for k in range(4):

# walls从后往前第(k+1)位为0

if not walls[x][y] & (1 << k):

nx,ny=x+dx[k],y+dy[k]

if 0<=nx<m and 0<=ny<n:

size+=dfs(nx,ny)

return size

m=int(input())

n=int(input())

walls=[list(map(int,input().split())) for \_ in range(m)]

visited=[[False]\*n for \_ in range(m)]

dx=[0,-1,0,1]

dy=[-1,0,1,0]

room\_num=0

max\_size=0

for i in range(m):

for j in range(n):

if not visited[i][j]:

size=dfs(i,j)

if size>0:

room\_num+=1

max\_size=max(max\_size,size)

print(room\_num)

print(max\_size)

```

变换的迷宫

```python

# 三维visited

from collections import deque

def bfs(x,y):

visited={(0,x,y)}

dx=[0,0,1,-1]

dy=[1,-1,0,0]

queue=deque([(0,x,y)])

while queue:

time,x,y=queue.popleft()

for i in range(4):

nx,ny=x+dx[i],y+dy[i]

temp=(time+1)%k

if 0<=nx<r and 0<=ny<c and (temp,nx,ny) not in visited:

cur=maze[nx][ny]

if cur=='E':

return time+1

elif cur!='#' or temp==0:

queue.append((time+1,nx,ny))

visited.add((temp,nx,ny))

return 'Oop!'

t=int(input())

for \_ in range(t):

r,c,k=map(int,input().split())

maze=[list(input()) for \_ in range(r)]

for i in range(r):

for j in range(c):

if maze[i][j]=='S':

print(bfs(i,j))

```

a knight's journey

```python

# 骑士巡逻问题

# 回溯法

def dfs(x,y,s,vis):

if len(vis)==m\*n:

return ''.join(s)

for dx,dy in dir:

nx,ny=x+dx,y+dy

if 0<=nx<m and 0<=ny<n and (nx,ny) not in vis:

vis.add((nx,ny))

res=dfs(nx,ny,s+[chr(65+ny)+str(nx+1)],vis)

if res:return res

vis.remove((nx,ny))

def solve():

for j in range(n):

for i in range(m):

vis=set()

vis.add((i,j))

t=dfs(i,j,[chr(65+j)+str(i+1)],vis)

if t:return t

dir=[(-1,-2),(1,-2),(-2,-1),(2,-1),(-2,1),(2,1),(-1,2),(1,2)]

for k in range(int(input())):

print(f'Scenario #{k+1}:')

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

s=solve()

if s:print(s)

else:print('impossible')

print()

```

海贼王之伟大航路

```python

# 旅行商问题

def tsp(n,cost):

# dp[mask][i]为从起始岛屿开始，经过mask表示的岛屿，最后停在岛屿i的最短时间

dp=[[float('inf')]\*(n+1) for \_ in range(1<<n)]

dp[1][1]=0

# 使用二进制来表示每个岛屿是否访问

for mask in range(1,1<<n):

for u in range(1,n+1):

if not (mask&(1<<(u-1))):

continue

# 尝试从所有其他岛屿v转移到岛屿u

for v in range(1,n+1):

if mask&(1<<(v-1)) and u!=v:

# 对于每一个mask，我们可以从v岛前往u岛，并更新dp[mask][u]的值。

dp[mask][u]=min(dp[mask][u],dp[mask^(1<<(u-1))][v]+cost[v][u])

return dp[(1<<n)-1][n]

n=int(input())

cost=[[0]\*(n+1)]

for \_ in range(n):

row=[0]+list((map(int,input().split())))

cost.append(row)

print(tsp(n,cost))

```

roads

```python

# 有限制的加权图最短路径问题

# Dijkstra算法

import heapq

def dijkstra(k,n,roads):

graph=[[] for \_ in range(n+1)]

for road in roads:

s,d,l,t=road

graph[s].append((d,l,t))

# dist[i][j]表示到达城市i且支付了j个硬币时的最短距离

dist=[[float('inf')]\*(k+1) for \_ in range(n+1)]

dist[1][0]=0

# 优先队列

queue=[(0,1,0)]

while queue:

# 选择最短距离的未访问节点

distance,node,toll=heapq.heappop(queue)

if node==n:

return distance

for next\_node,length,cost in graph[node]:

new\_toll=toll+cost

new\_distance=distance+length

# 更新邻居节点的距离

if new\_toll<=k and new\_distance<dist[next\_node][new\_toll]:

dist[next\_node][new\_toll]=new\_distance

heapq.heappush(queue,(new\_distance,next\_node,new\_toll))

return -1

k=int(input())

n=int(input())

r=int(input())

roads=[tuple(map(int,input().split())) for \_ in range(r)]

print(dijkstra(k,n,roads))

```

# 工具

int(str,n)

for key,value in dict.items()

for index,value in enumerate(list)

dict.get(key,default)

list(zip(a,b))

math.pow(m,n)

math.log(m,n)

lrucache

```python

from functools import lru\_cache

@lru\_cache(maxsize=None)

```

1. `str.lstrip() / str.rstrip()`: 移除字符串左侧/右侧的空白字符。

2. `str.find(sub)`: 返回子字符串`sub`在字符串中首次出现的索引，如果未找到，则返回-1。

3. `str.replace(old, new)`: 将字符串中的`old`子字符串替换为`new`。

4. `str.startswith(prefix) / str.endswith(suffix)`: 检查字符串是否以`prefix`开头或以`suffix`结尾。

5. `str.isalpha() / str.isdigit() / str.isalnum()`: 检查字符串是否全部由字母/数字/字母和数字组成。

calendar

1. `calendar.month(年, 月)`: 返回一个月份的日历字符串。它接受年份和月份作为参数，并以多行字符串的形式返回该月份的日历。

2. `calendar.calendar(年)`: 返回一个年份的日历字符串。这个函数生成整个年份的日历，格式化为多行字符串。

3. `calendar.monthrange(年, 月)`: 返回两个整数，第一个是该月第一天是周几（0-6表示周一到周日），第二个是该月的天数。

4. `calendar.weekday(年, 月, 日)`: 返回给定日期是星期几。0-6的返回值分别代表星期一到星期日。

5. `calendar.isleap(年)`: 返回一个布尔值，指示指定的年份是否是闰年。

6. `calendar.leapdays(年1, 年2)`: 返回在指定范围内的闰年数量，不包括第二个年份。

7. `calendar.monthcalendar(年, 月)`: 返回一个整数矩阵，表示指定月份的日历。每个子列表表示一个星期；天数为0表示该月份此天不在该星期内。

8. `calendar.setfirstweekday(星期)`: 设置日历每周的起始日。默认情况下，第一天是星期一，但可以通过这个函数更改。

9. `calendar.firstweekday()`: 返回当前设置的每周起始日。

counter：计数

```python

from collections import Counter

a=['red', 'blue', 'red', 'green', 'blue', 'blue']

a=Counter(a)

```

permutations：全排列

```python

from itertools import permutations as per

elements = [1, 2, 3]

permutations = list(per(elements))

```

combinations：组合

```python

from itertools import combinations as com

elements = ['A', 'B', 'C', 'D']

# 生成所有长度为2的组合

combinations = list(com(elements, 2))

```

bisect

```python

import bisect

# 创建一个已排序的列表

sorted\_list = [1, 3, 3, 6, 7, 9]

# 使用 bisect\_left 查找元素应插入的位置

insert\_index = bisect.bisect\_left(sorted\_list, 4)

print("Insert at index:", insert\_index)

# 使用 insort\_left 插入元素并保持有序

bisect.insort\_left(sorted\_list, 4)

print("Updated list:", sorted\_list)

```

reduce：累积

```python

import functools

numbers = [1, 2, 3, 4, 5]

# 使用 reduce 计算累积乘积

product = functools.reduce(lambda x, y: x \* y, numbers)

```

product：笛卡尔积

```python

from itertools import product

# 创建两个可迭代对象

colors = ['red', 'blue']

numbers = [1, 2]

# 生成它们的笛卡尔积

cartesian\_product = list(product(colors, numbers))

# 创建一个可迭代对象

colors = ['red', 'blue']

# 生成它们的重复笛卡尔积

repeat\_cartesian\_product = list(product(colors, repeat=3))

```

# 注意

1.读题仔细

\*wrong answer先看看有没有弱智错误

2.边界情况：例如空数组、数组中只有一个元素、最大值和最小值等

3.循环条件和边界： 检查循环的条件和边界是否正确。确保不会死循环

4.变量名千万别重(复杂题不要单字母命名大法)

5.逻辑错误：尝试分析代码中的逻辑错误，可以手动模拟算法的执行步骤，或者使用示例输入数据来验证。

6.标志性打印语句：在代码中插入标志性的打印语句，以追踪代码的执行流程，找到问题所在。

（但是提交前别忘了注释掉）（ctrl+/）

7.正难则反：乌鸦坐飞机，剪绳子

8.确保输入和输出格式正确