**enumerate快速获取索引和值**

for index, value in enumerate(list, start)

**列表排序**

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

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

**堆**

import heapq

heappush,heappop,heapify

**队列**

from queue import Queue

popleft

**位运算**

1. 按位与（&）按位与运算符 & 将两个整数的对应二进制位进行与运算，只有两个位都为1时，结果才为1。

a = 5 # 5的二进制表示是 101

b = 3 # 3的二进制表示是 011

result = a & b # 结果是 001，即 1

print(result) # 输出 1

2. 按位或（|）按位或运算符 | 将两个整数的对应二进制位进行或运算，只要两个位中有一个为1，结果就是1。

a = 5 # 101

b = 3 # 011

result = a | b # 结果是 111，即 7

print(result) # 输出 7

3. 按位异或（^）按位异或运算符 ^ 将两个整数的对应二进制位进行异或运算，当两个位不相同时，结果为1。

a = 5 # 101

b = 3 # 011

result = a ^ b # 结果是 110，即 6

print(result) # 输出 6

4. 按位取反（~）按位取反运算符 ~ 将整数的每一位取反，0变1，1变0。

a = 5 # 101

result = ~a # 结果是 -6，因为按位取反后的二进制补码表示

print(result) # 输出 -6

5. 左移（<<）左移运算符 << 将一个数的二进制位向左移动指定的位数，左移相当于乘以 2 的 n 次方。

a = 5 # 101

result = a << 1 # 结果是 1010，即 10

print(result) # 输出 10

6. 右移（>>）右移运算符 >> 将一个数的二进制位向右移动指定的位数，右移相当于除以 2 的 n 次方。

a = 5 # 101

result = a >> 1 # 结果是 10，即 2

print(result) # 输出 2

**字典**

Get（）访问不存在的值，del dict[key]

**埃氏筛**

def sieve\_of\_eratosthenes(n):

is\_prime = [True] \* (n + 1)

is\_prime[0] = is\_prime[1] = False # 0 和 1 不是素数

for p in range(2, int(n\*\*0.5) + 1):

if is\_prime[p]:

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

is\_prime[i] = False

return [p for p in range(2, n + 1) if is\_prime[p]]

**欧拉筛**

def sieve\_of\_euler(n):

is\_prime = [True] \* (n + 1)

is\_prime[0] = is\_prime[1] = False # 0 和 1 不是素数

primes = []

for i in range(2, n + 1):

if is\_prime[i]:

primes.append(i)

for p in primes:

if i \* p > n:

break

is\_prime[i \* p] = False

if i % p == 0:

break

return primes

**merge sort 分治**

mininum=0

def mergesort(arr):

global mininum

if len(arr) > 1:

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

mergesort(left)

mergesort(right)

Lptr = Rptr = ptr = 0

while len(left) > Lptr and len(right) > Rptr:

if left[Lptr] <= right[Rptr]:

arr[ptr] = left[Lptr]

Lptr += 1

else:

arr[ptr] = right[Rptr]

Rptr += 1

mininum += len(left) - Lptr

ptr += 1

while len(left) > Lptr:

arr[ptr] = left[Lptr]

ptr += 1

Lptr += 1

while len(right) > Rptr:

arr[ptr] = right[Rptr]

ptr += 1

Rptr += 1

n = int(input())

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

mergesort(arr)

print(mininum)

**binary search**

def binary\_search(arr, target):

low = 0

high = len(arr) - 1

while low <= high:

mid = (low + high) // 2 # 计算中间索引

if arr[mid] == target:

return mid # 找到目标值，返回索引

elif arr[mid] < target:

low = mid + 1 # 目标值在右半部分

else:

high = mid - 1 # 目标值在左半部分

return -1 # 如果未找到目标值，返回 -1、

**dfs**

**马走日**

k = 10

dirx = [-2,-1,1,2, 2, 1,-1,-2]

diry = [ 1, 2,2,1,-1,-2,-2,-1]

ans = 0

def dfs(num, x, y):

if n\*m == num:

global ans

ans += 1

return

for r in range(8):

s = x + dirx[r]

t = y + diry[r]

if matrix[s][t]==False and 0<=s<n and 0<=t<m :

matrix[s][t]=True

dfs(num+1, s, t)

matrix[s][t] = False

for \_ in range(int(input())):

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

matrix = [[False]\*k for \_ in range(k)]

ans = 0

matrix[x][y] = True

dfs(1, x, y)

print(ans)

**bfs**

**寻宝**

from collections import deque

def bfs(maze):

    queue=deque()

    visited=set()

    queue.append((0,0,0))

    visited.add((0,0))

    while queue:

        x,y,step=queue.popleft()

        if maze[x][y]==1:

            return step

        for i in range(4):

            nx,ny=x+dx[i],y+dy[i]

            if 0<=nx<m and 0<=ny<n:

                if maze[nx][ny]!=2 and (nx,ny) not in visited:

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

                    visited.add((nx,ny))

    return 'NO'

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

maze=[]

for i in range(m):

    maze.append(list(map(int,input().split())))

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

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

print(bfs(maze))

**Dijkstra**

**走山路**

import heapq

def dijkstra(start,end,hill):

    consume=[[float('inf')]\*n for \_ in range(m)]

    x0,y0=start

    xe,ye=end

    if hill[x0][y0]=='#' or hill[xe][ye]=='#':

        return 'NO'

    consume[x0][y0]=0

    pq=[(0,x0,y0)]

    while pq:

        con,x,y=heapq.heappop(pq)

        if (x,y)==end:

            return con

        for i in range(4):

            x1=x+dx[i]

            y1=y+dy[i]

            if 0<=x1<m and 0<=y1<n and hill[x1][y1]!='#':

                con1=con+abs(int(hill[x][y])-int(hill[x1][y1]))

                if con1<consume[x1][y1]:

                    consume[x1][y1]=con1

                    heapq.heappush(pq,(con1,x1,y1))

    return 'NO'

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

hill=[]

for i in range(m):

    hill.append(list(map(str,input().split())))

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

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

out=[]

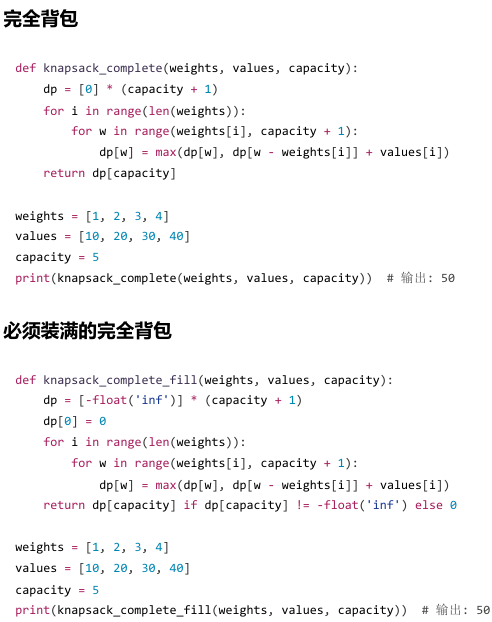
for i in range(p):

    a,b,c,d=map(int,input().split())

    print(dijkstra((a,b),(c,d),hill))

**dp 动态规划**

****

****

**核电站**

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

dp=[0 for i in range(n+1)]

dp[0]=1

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

if i <m:

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

elif i==m:

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

else:

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

print(dp[-1])

**无聊**

n=int(input())

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

m=100001

lst=[0]\*m

for i in a:

lst[i]+=i

dp=[[0,0] for \_ in range(m)]

for i in range(1,m):

dp[i][0]=max(dp[i-1][0],dp[i-1][1])

dp[i][1]=dp[i-1][0]+lst[i]

print(max(dp[-1][0],dp[-1][1]))

**八皇后**

answer=[]

def queen(s):

for i in range(1,9):

for j in range(len(s)):

if str(i)==s[j] or abs(i-int(s[j]))==abs(len(s)-j):

break

else:

if len(s)==7:

answer.append(s+str(i))

else:

queen(s+str(i))

queen('')

n=int(input())

for i in range(n):

b=int(input())

print(answer[b-1])

**幸福的寒假生活**

def date\_to\_index(date):

month,day=map(int,date.split('.'))

return (month-1)\*31+day-6

data=[]

dp=[0]\*46

n=int(input())

for \_ in range(n):

start,end,happiness=input().split()

start,end,happiness=date\_to\_index(start),date\_to\_index(end),int(happiness)

data.append((start,end,happiness))

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])

**最大整数**

def f(string):

if string=='':

return 0

else:

return int(string)

m=int(input())

n=int(input())

number=input().split()

for i in range(n):

for j in range(n-1-i):

if number[j] + number[j+1] > number[j+1] + number[j]:

number[j],number[j+1] = number[j+1],number[j]

length=[]

for num in number:

length.append(len(num))

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

for k in range(m+1):

dp[0][k]=''

for l in range(n+1):

dp[l][0]=''

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

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

if length[i-1]>j:

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

else:

dp[i][j]=str(max(f(dp[i-1][j]),int(number[i-1]+dp[i-1][j-length[i-1]])))

print(dp[n][m])

**Bisect**

bisect\_left(a, x, lo=0, hi=len(a))

在列表 a 中查找 x 的插入位置，保证插入后列表依然是升序排列。如果 x 已经存在于列表中，返回 x 的第一个位置。

lo 和 hi 参数可以限制查找的范围（默认是整个列表）。

bisect\_right(a, x, lo=0, hi=len(a)) 或 bisect(a, x, lo=0, hi=len(a))

与 bisect\_left 类似，但如果 x 已经存在于列表中，它返回 x 的下一个位置（即将 x 插入到列表中所有相等的元素之后）。

insort\_left(a, x, lo=0, hi=len(a))

在列表 a 中插入元素 x，并保持列表的升序排序。如果列表中已经有与 x 相等的元素，它会将 x 插入到相同元素的最前面。

insort\_right(a, x, lo=0, hi=len(a)) 或 insort(a, x, lo=0, hi=len(a))

类似于 insort\_left，但是如果列表中已经有与 x 相等的元素，它会将 x 插入到相同元素的最后面。

示例

1. bisect\_left 和 bisect\_right

import bisect

# 已排序的列表

data = [1, 2, 4, 4, 5, 6]

# 查找元素 4 应该插入的位置（返回插入位置的索引）

index\_left = bisect.bisect\_left(data, 4)

index\_right = bisect.bisect\_right(data, 4)

print(f"bisect\_left(4): {index\_left}") # 返回 2

print(f"bisect\_right(4): {index\_right}") # 返回 4

输出：

bisect\_left(4): 2

bisect\_right(4): 4

**calendar，datetime**

**判断闰年**

import calendar

# 判断 2024 是否是闰年

print(calendar.isleap(2024)) # 输出 True

# 判断 2023 是否是闰年

print(calendar.isleap(2023)) # 输出 False

**星期**

import datetime

# 获取当前日期

today = datetime.date.today()

# 获取星期几

weekday = today.weekday()

print(weekday) # 输出 0（表示星期一）

**zip( )**

a = [1, 2, 3]

b = ['a', 'b', 'c']

zipped = list(zip(a, b))

print(zipped) # 输出: [(1, 'a'), (2, 'b'), (3, 'c')]

**defaultdict**

from collections import defaultdict

from collections import defaultdict

# 创建一个默认值为 list 的 defaultdict

d = defaultdict(list)

d['fruits'].append('apple')

d['fruits'].append('banana')

d['vegetables'].append('carrot')

print(d) # 输出: defaultdict(<class 'list'>, {'fruits': ['apple', 'banana'], 'vegetables': ['carrot']})

**完美的爱**

from collections import defaultdict

n=int(input())

lst=[int(i)-520 for i in input().split()]

dicta=defaultdict(list)

dicta[0].append(-1)

num=0

tot=0

for i in range(n):

tot+=lst[i]

if tot in dicta:

num=max(num,i-dicta[tot][0])

dicta[tot].append(i)

print(num\*520)

print(f"{num:.2f}")

**跳房子**

def search(s,e,m,rock):

left,right=s,e

ans=0

while left<right:

mid=(left+right)//2

s0=0

num=0

for i in range(1,n+2):

if rock[i]-s0<mid:

num+=1

else:

s0=rock[i]

if num>m:

right=mid

else:

left=mid+1

ans=mid

return ans

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

rock=[0]

for i in range(n):

rock.append(int(input()))

rock.append(l)

print(search(0,l+1,m,rock))

**replace**

string.replace(old, new, count)

参数：

old：要被替换的子字符串。

new：替换成的新子字符串。

count（可选）：指定替换的次数。如果省略或设置为 -1，则会替换所有出现的子字符串。