计概

1.def is\_prime(n): 5.矩阵构建：mx = [(m+2)\*[0] for i in range(n+2)] (关注是否要加保护层)

    if n <= 1: 输出：for i in num:

        return False print(' '.join(map(str,i)))

    for i in range(2, int(n\*\*0.5) + 1): // print(\*i)

        if n % i == 0: matrix=[[0]\*(m+2)]+[[0]+list(map(int,input().split()))+[0] for i in range(n)]+[[0]\*(m+2)]

            return False （带有保护层的矩阵）

    return True

**2.埃氏筛：**

def is\_prime(x):

numbers=[True]\*(x + 1)

numbers[0]=numbers[1]=False 6.优化算法：sum可以用累加代替，max、

for i in range(2,int(x\*\*0.5)+1) : min可以用比较然后赋值代替

if numbers[i]: count有时可用索引代替

j=i\*i 例如：if a[i]!=a[i-1]:

while j <= x: edge1+=i

numbers[j]=False

j+=i

return numbers[-1]

3.字典：

phone\_dict = {'A': '2', 'B': '2', 'C': '2'} 7.sort中lambda的应用：

phone\_dict['A'] c.sort(key=lambda y:(y[0],-y[1]))

4.冒泡排序（最大最小数字） 先按y[0]排序 再按-y[1]排序

n=int(input()) 8.def is\_even(x):

s=input().split() return x % 2 == 0

M="" numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

m="" result = filter(is\_even, numbers)

s.sort(reverse=True) filter→筛选出满足特定条件的元素

for i in range(1,len(s)): 9.比较 map和filter

    tag=i numbers = [1, 2, 3, 4, 5, 6, 7, 8]

    while s[tag][0]==s[tag-1][0]: even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

        if s[tag-1]+s[tag]<s[tag]+s[tag-1]: print(even\_numbers)  # 输出：[2, 4, 6, 8]

s[tag-1],s[tag]=s[tag],s[tag-1] 用map代替filter则会输出[False, True, False, True, False, True, False, True]

     tag-=1

       if tag==0: 10.list切片可以超过index！取到没有为止！！！！

            break

for j in range(len(s)): 11.print(f'Cube = {str(a)} , Triple = ({str(b)},{str(c)},{str(d)})')

    M+=s[j] 在字符串间插入变量

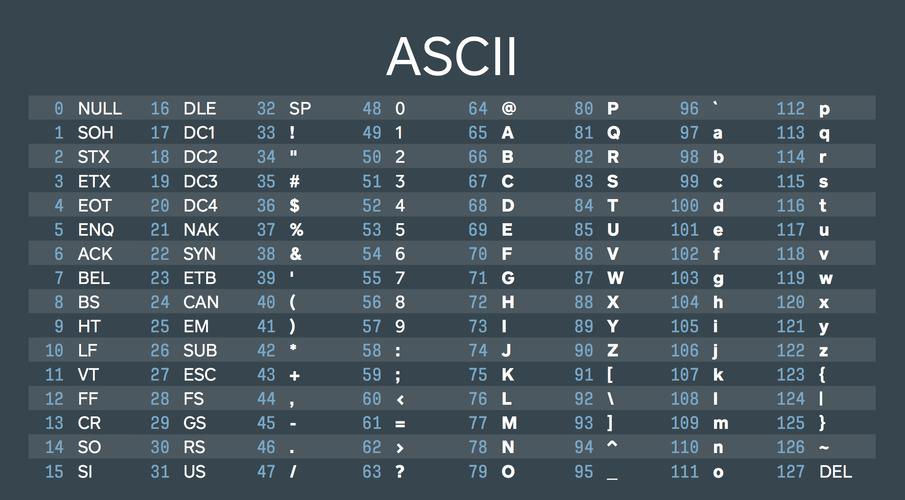
    m+=s[-(j+1)] 12.保留小数

print(int(M),int(m)) **print('%.2f'% b)**

**print(f"{num:.2f}")可以输出保留两位小数的num**

13.ASCII码表

chr→转为字符 ord→转回ASCII码



**14.剪绳子：**

n=int(input())

import heapq

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

heapq.heapify(a)

money=0

for i in range(n-1):

    A=heapq.heappop(a)

    B=heapq.heappop(a)

    C=A+B

    heapq.heappush(a,C)

    money+=C

print(money)

heapq模块中的主要操作函数的时间复杂度如下：

* heapify: 将列表转换为堆的时间复杂度为O(N)，其中N是列表的长度。
* heappush: 向堆中插入元素的时间复杂度为O(logN)，其中N是堆的大小。
* heappop: 从堆中弹出最小元素的时间复杂度为O(logN)，其中N是堆的大小。
* heappushpop: 向堆中插入元素并弹出最小元素的时间复杂度为O(logN)，其中N是堆的大小。
* heapreplace: 弹出最小元素并插入新元素的时间复杂度为O(logN)，其中N是堆的大小。

import heapq

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

heapq.heapify(a)

ans = 0

for i in range(n-1):

x = heapq.heappop(a)

y = heapq.heappop(a)

z = x + y

heapq.heappush(a, z)

ans += z

print(ans)

15.双端队列：

from collections import deque

# 创建一个空的双端队列

d = deque()

# 向队列中添加元素

d.append(1)

d.append(2)

d.appendleft(3)

print(d) # 输出： deque([3, 1, 2])

# 从队列中删除元素

d.pop()

d.popleft()

print(d) # 输出： deque([1])

16.处理整除的例子（只要多1，总车数就算+1）

input()

a,b,c,d=map(input().count,('1','2','3','4'))

print(d+c+(b\*2+max(0,a-c)+3)//4)

1. 列表、集合与字典

创建：

X=[]

Y=set()

Z={}

添加：

列表用list.append()

集合用set.add()

字典用dict[x]=y

set1 = {1, 2, 3, 4} # 直接使用大括号创建集合

set2 = set([4, 5, 6, 7]) # 使用 set() 函数从列表创建集合

****注意：**创建一个空集合必须用 set() 而不是 { }，因为 { } 是用来创建一个空字典。**

1. **## bfs模版：**

寻宝

import heapq

def bfs(x,y):

    d=[[-1,0],[1,0],[0,1],[0,-1]]

    queue=[]

    heapq.heappush(queue,[0,x,y])

    check=set()

    check.add((x,y))

    while queue:

        step,x,y=map(int,heapq.heappop(queue))

        if martix[x][y]==1:

            return step

        for i in range(4):

            dx,dy=x+d[i][0],y+d[i][1]

            if martix[dx][dy]!=2 and (dx,dy) not in check:

                heapq.heappush(queue,[step+1,dx,dy])

                check.add((dx,dy))

    return "NO"

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

martix=[[2]\*(n+2)]+[[2]+list(map(int,input().split()))+[2] for i in range(m)]+[[2]\*(n+2)]

print(bfs(1,1))

1. **dfs模版：**

**最大连通域面积**

dire = [[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-1],[1,0],[1,1]]  # 8个方向的偏移量

area = 0  # 当前连续白色区域的大小

def dfs(x, y):

    global area  # 声明使用全局变量area

    if matrix[x][y] == '.':  # 如果当前位置为黑色，直接返回

        return

    matrix[x][y] = '.'  # 将当前位置标记为已访问

    area += 1  # 增加连续白色区域的大小

    for i in range(len(dire)):  # 遍历所有方向

        dfs(x + dire[i][0], y + dire[i][1])  # 递归调用dfs函数

for \_ in range(int(input())):  # 读取测试用例数量

    n, m = map(int, input().split())  # 读取矩阵的行数和列数

    matrix = [['.' for \_ in range(m + 2)] for \_ in range(n + 2)]  # 创建矩阵并初始化边界元素

    for i in range(1, n + 1):

        matrix[i][1:-1] = input()  # 读取矩阵内容

    sur = 0  # 初始化最大连续白色区域的大小

    for i in range(1, n + 1):

        for j in range(1, m + 1):

            if matrix[i][j] == 'W':  # 如果当前位置为白色

                area = 0  # 重置连续白色区域的大小

                dfs(i, j)  # 调用dfs函数探索连续白色区域

                sur = max(sur, area)  # 更新最大连续白色区域的大小

print(sur)  # 打印结果

**dfs模版题2：螃蟹采蘑菇（用于判断有无实现的可能，优点：代码比较简洁）**

dire= [[-1,0],[0,-1],[0,1],[1,0]]

visited=set()

def dfs(x,y,a,b):

    if matrix[x][y] == 0 and matrix[a][b] == 0:

        if(x,y,a,b) not in visited:

            visited.add((x,y,a,b))

            for i in range(4):  # 遍历所有方向

                if dfs(x + dire[i][0], y + dire[i][1],a + dire[i][0],b + dire[i][1]):# 递归调用函数

                    return True

    if matrix[x][y]==9 or matrix[a][b] == 9:

        if matrix[x][y] != 1 and matrix[a][b] != 1:

            return True

    return False

n=int(input())

matrix=[[1]\*(n+2)]+[[1]+list(map(int,input().split()))+[1] for i in range(n)]+[[1]\*(n+2)]

count=0

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

    for j in range(1, n + 1):

        if count==0 and matrix[i][j] == 5:

            x=i

            y=j

            matrix[i][j]=0

            count+=1

        if count==1 and matrix[i][j]==5:

            a=i

            b=j

            matrix[i][j]=0

            break

if dfs(x,y,a,b):

    print('yes')

else:

    print('no')

**模版三 寻宝（dfs版本）（适用于统计路径，或者求最大最小路径）**

dire= [[-1,0],[0,-1],[0,1],[1,0]]

visited=set()

ku=[]

step=0

def dfs(matrix,x,y,step):

global ku

if matrix[x][y]==1:

ku.append(0)

for i in range(4): # 遍历所有方向

nx,ny,nstep=x + dire[i][0], y + dire[i][1],step+1

if matrix[nx][ny] == 1:

ku.append(nstep)

continue

if matrix[nx][ny]==0:

matrix[x][y]=2

dfs(matrix,nx,ny,nstep)

matrix[x][y]=0

return

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

matrix = [[2] \* (m + 2)] + [[2] + list(map(int, input().split())) + [2] for i in range(n)] + [[2] \* (m + 2)]

dfs(matrix,1,1,0)

if ku:

print(min(ku))

else:

print('NO')

**模版四 马走日（适用性更广）**

ans=0

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

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

def dfs(sum\_,x,y):

    global ans

    if sum\_==n\*m:

        ans+=1

        return

    for i in range(8) :

        nx=x+dx[i]

        ny=y+dy[i]

        if mx[nx][ny]==0 and 0<=nx<n and 0<=ny<m:

            mx[nx][ny]=1

            dfs(sum\_+1, nx,ny)

            mx[nx][ny]=0

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

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

    mx=[[0]\*10 for j in range(10)]

    ans=0

    mx[x][y]=1

    dfs(1,x,y)

    print(ans)

**20.双指针（三数之和）：**

def threeSum(nums):

nums.sort()

res = set()

for i in range(len(nums) - 2):

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

continue

d = set() ***//d={}***

for x in nums[i + 1:]:

if x not in d:

d.add(-nums[i] - x) ***//d[-nums[i] - x]=1***

else:

res.add((nums[i], -nums[i] - x, x))

return len(res)

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

print(threeSum(n))

**21.dp的一些模版**

**背包采药（一维，逆排）**

T,M=map(int,input().split())

herb=[]

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

for i in range(M):

    a,b=map(int,input().split())

    herb.append((a,b))

for i in range(M):

    for j in range(T,herb[i][0]-1,-1):

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

print(dp[-1])

**幸福的寒假生活（二维，顺排）**

def day(x):

    a,b=x.split('.')

    index=(int(a)-1)\*31-6+int(b)

    return index

n=int(input())

data=[]

dp=[0]\*46

for i in range(n):

    x,y,z=input().split()

    data.append((day(x),day(y),int(z)))

for i in range(1,46):

    dp[i]=dp[i-1]

    for j in range(n):

        if data[j][1]==i:

            dp[i]=max(dp[i],dp[data[j][0]-1]+data[j][2])

print(dp[-1])

**最佳凑单 （0-1背包经典）**

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

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

dp=[0]\*(sum(a)+1)

dp[0]=1

for i in a:

    for j in range(sum(a),i-1,-1):

        dp[j]+=dp[j-i]

        print(dp)

if t>sum(a):

    print(0)

else:

    for i in range(t,sum(a)+1):

        if dp[i]>0:

            print(i)

            Break

**健身房（“恰好型”dp）**

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

group=[]

for i in range(n):

    t,w=map(int,input().split())

    group.append((t,w))

dp=[-999999]\*(T+1) ***//除了dp[0],均设为负无穷***

dp[0]=0

for i in range(n):

    for j in range(T,group[i][0]-1,-1):

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

if dp[-1]<=0:

    print(-1)

else:

    print(dp[-1])

**Sumsets 完全背包**

import math

n=int(input())

ln=int(math.log(n,2))

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

MOD=10\*\*9

for i in range(ln+1):

    ii=2\*\*i

    for j in range(ii,n+1):

        dp[j]=(dp[j]+dp[j-ii])

ans=dp[-1]%MOD

print(ans)

**完全背包（个数无穷）**

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

val=[]

for i in range(n):

    a,b=map(int,input().split())

    val.append((a,b))

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

for k in val:

    if k[0]<=m:

        for i in range(k[0],m+1):

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

print(dp)

1. **螺旋矩阵**

n=int(input())

a=[[0]\*n for i in range(n)]

j=1

t=0

k=n

while j<=n\*n:

    for i in range(t,k): →

        a[t][i]=j

        j+=1

    for i in range(t+1,k): ↓

        a[i][k-1]=j

        j+=1

    for i in range(k-2,t-1,-1): ←

        a[k-1][i]=j

        j+=1

    for i in range(k-2,t,-1): ↑

        a[i][t]=j

        j+=1

    k-=1

    t+=1

for i in a:

    print(\*i)

字符串

swapcase(): 此方法用于将原字符串中的大写字母转换为小写字母，并将小写字母转换为大写字母。返回转换后的字符串。

title(): 该方法用于将原字符串中的每个单词的首字母大写，并将其余部分转换为小写字母。如果原字符串为空或只包含空白字符，则返回原字符串。

capitalize(): 此方法用于将原字符串的首字母大写，并将其余部分转换为小写字母。如果原字符串为空或只包含空白字符，则返回原字符串。

串。以下是一些常用的字符串函数：

lower(): 此函数用于将原字符串中的所有大写字母转换为小写字母，并返回转换后的字符串。

upper(): 该函数用于将原字符串中的所有小写字母转换为大写字母，并返回转换后的字符串。

find(sub): 此函数用于查找子字符串sub在原字符串中的最低索引位置，如果找不到则返回-1。

join(iterable): 该函数用于将一个可迭代对象（如列表、元组等）中的元素以指定的字符连接生成一个新的字符串。

replace(old, new): 用于将原字符串中的旧子字符串替换为新子字符串，并返回替换后的字符串。

split(separator): 此函数根据指定的分隔符将原字符串分割成多个子字符串，并以列表的形式返回这些子字符串。