递归算法 Recursion Questions

一、递归的基本定义

1.递归的概念

递归是一种解决问题的方法，其通过将问题分解为越来越小的子问题，直到分解为可以简单解决的足够小的问题。递归涉及到调用函数自身的算法，能够帮助我们为那些可能很难直接进行代码编写的问题提供一个优雅的解决方案。

2.递归三定律

所有递归算法都必须遵循三个重要定律：

（1）递归算法必须有一个基本情况；

基本情况是允许算法停止递归的条件，通常是一个小到可以直接解决的问题。

（2）递归算法必须改变其状态并向基本情况移动；

在递归算法中，我们必须安排状态的改变，使算法朝着基本情况移动。状态的改变意味着算法正在使用的一些数据被修改。通常，代表我们问题的数据在某种程度上会随着递归的深入而变小。

（3）递归算法必须递归地调用自己

递归的核心定义是算法必须调用自己。递归函数通过调用自身来解决问题，其逻辑在于将问题分解为更小、更容易的问题来进行解决

二、例题1：将整数转换为任意基数的字符串

对于将一个整数转换为某一进制下的字符串表达形式这一问题，递归算法可以较为优雅的解决。

其具体的实现方式是通过不断做整除法的形式。在数字小于基数的情况下，达到基本情况直接返回；否则进行整除，将整除所得的数字再调用自身，同时尾部加上余数

```
def to_str(n,base):
    mystring = "0123456789ABCDEF"
    if n<base:
        return mystring[n]
    return to_str(n//base) + mystring[n%base]
```

当然，该程序也可以使用栈进行解决：

```
def to_str(n,base):
    stack = []
    mystring = "0123456789ABCDEF"
    while n>0:
        if n < base:
            stack.append(mystring[n])
        else:
            stack.append(mystring[n % base])
            n = n // base
    res = ""
    while stack:
        res += str(stack.pop())
    return res
```

三、例题2：汉诺塔

汉诺塔（Tower of Hanoi）是法国数学家Edouard Lucas基于一个传说的启发所发明的。具体而言，有三个杆子和若干个圆盘，每个圆盘都比下面的小一点，任务是将所有的圆盘从三个杆子中的一个转移到另一个。但是，有两个重要的限制：他们一次只能移动一个磁盘，而且永远无法将较大的磁盘放在较小的磁盘上。

从递归的角度解释这一问题，我们应该进行如下操作：

（1）对高度为n的塔而言，将高度为n-1的子塔移动到中间支柱上。自然，这里目标支柱是中间支柱，而中间支柱是总问题的目标支柱。

（2）将剩余的最大圆盘从原支柱移到目标支柱。

（3）将中间支柱上高度为n-1的子塔移动到目标支柱上。其中，中间支柱为原始支柱，目标支柱即为最终支柱。

而对于高度为1的塔而言，其恰为递归的基本情况：我们只需直接将该塔从原始支柱移动到目标支柱即可。

```
def moveTower(height,fromPole, toPole, withPole):
    if height >= 1:
        moveTower(height-1,fromPole,withPole,toPole) #Recursive call
        moveDisk(fromPole,toPole)
        moveTower(height-1,withPole,toPole,fromPole) #Recursive call

def moveDisk(fp,tp):
    print("moving disk from",fp,"to",tp)
```

注：对于高度为n的汉诺塔，移动所需的最短次数是2^n-1

四、Openjudge例题

1.01161:Help Jimmy

```
'''
描述
"Help Jimmy" 是在下图所示的场景上完成的游戏。
场景中包括多个长度和高度各不相同的平台。地面是最低的平台，高度为零，长度无限。
Jimmy老鼠在时刻0从高于所有平台的某处开始下落，它的下落速度始终为1米/秒。当Jimmy落到某个平台上时，游戏者选择让它
设计一个程序，计算Jimmy到底地面时可能的最早时间。
输入
第一行是测试数据的组数t（0 <= t <= 20）。每组测试数据的第一行是四个整数N，X，Y，MAX，用空格分隔。N是平台的数目
Jimmy的大小和平台的厚度均忽略不计。如果Jimmy恰好落在某个平台的边缘，被视为落在平台上。所有的平台均不重叠或相连。
输出
对输入的每组测试数据，输出一个整数，Jimmy到底地面时可能的最早时间。
'''

#2200015507 王一粟
from functools import lru_cache
@lru_cache
def dfs(x,y,z):
    for i in range(z+1,N+1):
        if y-MaxVal > p[i][2]:
            return 1<<30
        elif p[i][0] <= x <= p[i][1]:
            left = x-p[i][0] + dfs(p[i][0],p[i][2],i)
            right = p[i][1]-x+dfs(p[i][1],p[i][2],i)
            return min(left,right)
    if y<=MaxVal:
        return 0
    else:
        return 1<<30
for _ in range(int(input())):
    N,ini_x,ini_y,MaxVal = map(int,input().split())
    p = []
    p.append([0,0,1<<30])
    for k in range(N):
        p.append([int(x) for x in input().split()])
    p.sort(key = lambda x:-x[2])
    print(ini_y+dfs(ini_x,ini_y,0))
```

2.02386:Lake Counting

```
'''
描述
Due to recent rains, water has pooled in various places in Farmer John's field, which is repre
Given a diagram of Farmer John's field, determine how many ponds he has.
输入
* Line 1: Two space-separated integers: N and M
* Lines 2..N+1: M characters per line representing one row of Farmer John's field. Each charac
输出
* Line 1: The number of ponds in Farmer John's field.
'''
```

```
import sys
sys.setrecursionlimit(20000)
def dfs(x,y):
        #标记，避免再次访问
    field[x][y]='.'
    for k in range(8):
        nx,ny=x+dx[k],y+dy[k]
        #范围内且未访问的lake
        if 0<=nx<n and 0<=ny<m\
                and field[nx][ny]=='W':
            #继续搜索
            dfs(nx,ny)
n,m=map(int,input().split())
field=[list(input()) for _ in range(n)]
cnt=0
dx=[-1,-1,-1,0,0,1,1,1]
dy=[-1,0,1,-1,1,-1,0,1]
for i in range(n):
    for j in range(m):
        if field[i][j]=='W':
            dfs(i,j)
            cnt+=1
print(cnt)
```

3.05585：晶矿的个数

```
'''
描述
在某个区域发现了一些晶矿，已经探明这些晶矿总共有分为两类，为红晶矿和黑晶矿。现在要统计该区域内红晶矿和黑晶矿的个数
输入
第一行为k，表示有k组测试输入。
每组第一行为n，表示该区域由n*n个地点组成，3 <= n<= 30
接下来n行，每行n个字符，表示该地点的类型。
输出
对每组测试数据输出一行，每行两个数字分别是红晶矿和黑晶矿的个数，一个空格隔开。
'''
dire = [[-1,0], [1,0], [0,-1], [0,1]]

def dfs(x, y, c):
    m[x][y] = '#'
    for i in range(len(dire)):
        tx = x + dire[i][0]
        ty = y + dire[i][1]
        if m[tx][ty] == c:
            dfs(tx, ty, c)

for _ in range(int(input())):
    n = int(input())
    m = [[0 for _ in range(n+2)] for _ in range(n+2)]

    for i in range(1, n+1):
        m[i][1:-1] = input()
```

```
r = 0 ; b=0
for i in range(1, n+1):
    for j in range(1, n+1):
        if m[i][j] == 'r':
            dfs(i, j, 'r')
            r += 1
        if m[i][j] == 'b':
            dfs(i,j,'b')
            b += 1
print(r, b)
```