Python算法之旅（第16期）

上期回顾：

描述：使用"贝格尔"编排法设置单循环赛制，根据参赛队伍的数量生成对战表。

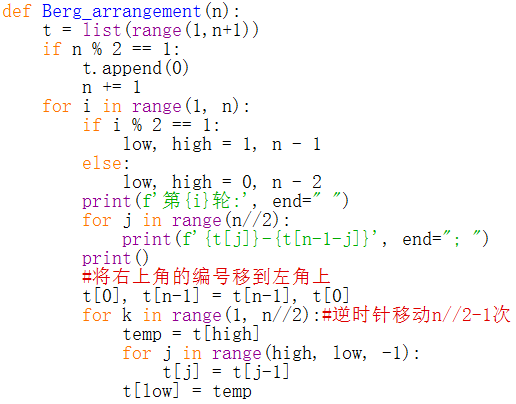
函数名：Berg\_arrangement(n)

参数表：n，参加比赛的队伍数量。

返回值：无，函数直接输出编排结果。

算法分析：

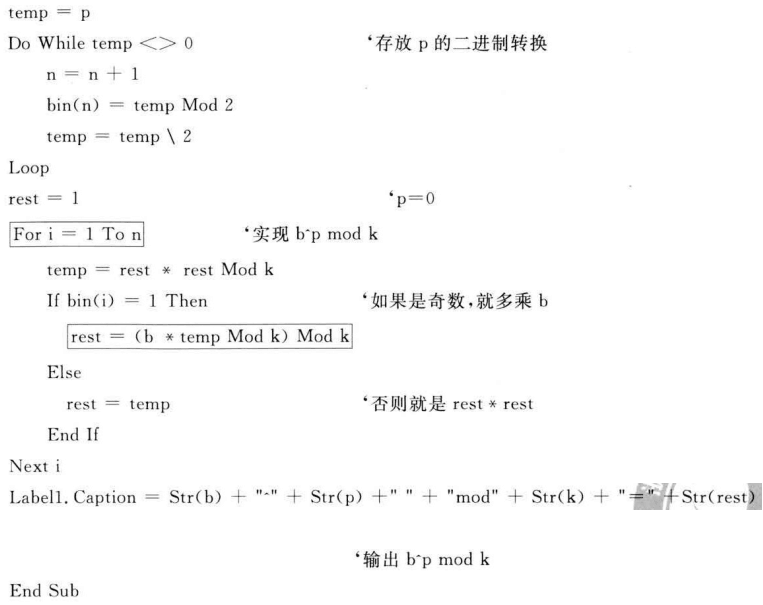
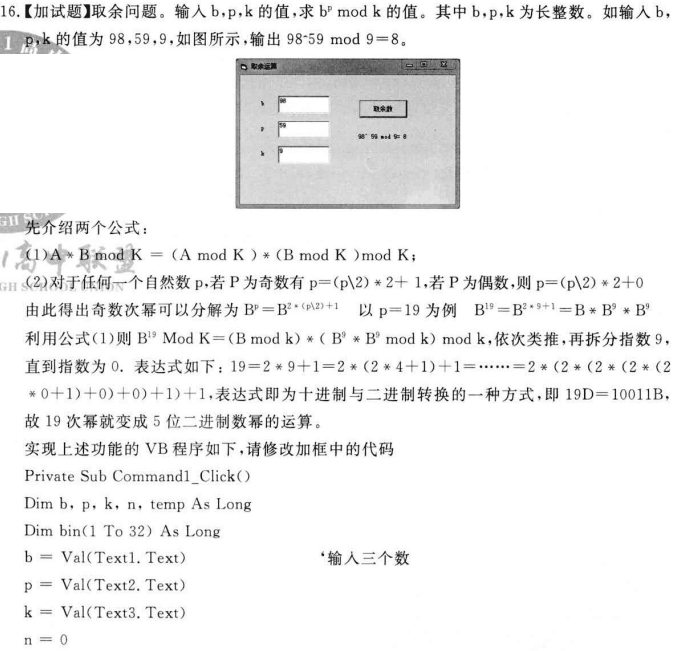
"贝格尔"编排法的特点是，单数轮次时"0"或最大的一个代号在右上角，双数轮次时则在左上角，其他元素逆时针循环移动n//2-1次。如此循环反复n-1次。代码如下：



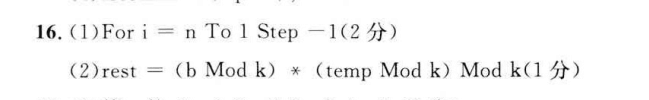
题目：快速幂取模（浙江省9+1高中联盟16题）

难度：3星 有趣：3星 有用：3星

分类：递归分治， 迭代， 同余定理



答案：



算法分析：

本题是经典的“快速幂取模”问题。

要理解本问题，我们先解决它的一个孪生问题“快速幂”。顾名思义，快速幂就是快速算底数的n次幂。其时间复杂度为 O(log₂N)， 与朴素的O(N)相比效率有了极大的提高。

例如要求x^19，可以拆解成x^19 = x^(2\*9+1) = x\*x^9\*x^9，依次类推,再拆分指数9，直到指数为0。即19=2\*9+1=2\*(2\*4+1)+......=2\*(2\*(2\*(2\*(2\*0+1)+0)+0)+1)+1。

解决快速幂问题的基本思路有两种，一是自顶向下的递归算法，二是自底向上的迭代算法。

我们先来看递归算法：

描述：求快速幂，即快速求x^n

函数名：fun(x, n)

参数表：x -- 底数； n -- 指数。

返回值：返回x^n。

def fun(x, n): #递归法求x^n

if n == 1:

return x

t = fun(x, n//2)

if n % 2 == 1:

return x \* t \* t

else:

return t \* t

此法思路简明清晰，但是由于用到了递归，效率总归有些不高，我们可以使用效率更高的迭代算法。

关于从递归算法到迭代算法的转化，有很多经典的例子，斐波那契数列是经典中的经典，我们先来看看它。

描述：求斐波那契数列第n项

函数名：Fibonacci(n)

参数表：n -- 第n项。

返回值：返回斐波那契数列第n项。

def Fibonacci(n):#递归法求斐波那契数列

if n == 1 or n == 2:

return 1

else:

return Fibonacci(n-1) + Fibonacci(n-2)

def Fibonacci2(n):#迭代法求斐波那契数列

f1, f2, f3 = 1, 1, 1

for i in range(3, n+1):

f3 = f1 + f2

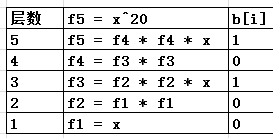
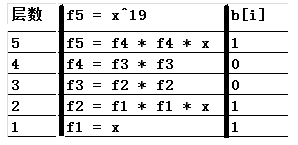
f1 = f2

f2 = f3

return f3

在斐波那契数列问题中，从递归算法到迭代算法的转化是很好理解的，迭代时递推的规律也很简单，就是利用第(n-1)和第(n-2)来推出第n项。“快速幂”问题要稍微复杂些，因为它不仅仅是利用第(n-1)项来求第n项，还要考虑第(n-1)项对应的指数是奇数还是偶数，所以我们需要先将拆分后的指数的奇偶性记录下来，刚好对应一个二进制数。

例如要求x^19，则指数19拆分后可以写成二进制数（10011）B；同理20 =（10100）B，作与递归算法相对应的示意图如下：



我们发现由于规定了x^n中n>0，所以无论n是奇数还是偶数，最底层f1的值都是x，之后向上的每一层，根据其对应的二进制数的奇偶性来确定是fn = f(n-1) \* f(n-1)还是fn = f(n-1) \* f(n-1) \* x。我们把n对应的二进制数的每一位存储在数组b中，注意迭代时要逆序获取b的元素。

根据上述分析，我们得到Python代码如下：

def fun2(x, n): #迭代法求x^n

b = []

while n > 0:

b.append(n % 2)

n //= 2

r = 1

for i in b[::-1]:

r \*= r

if i == 1:

r \*= x

return r

搞清楚了“快速幂”问题， “快速幂取模”问题就简单了，因为它仅仅是比“快速幂”问题多用了一个“同余定理”而已。

所谓同余定理就是指：(A \* B) mod k = (A mod K) \* (B mod K) mod K

仍以n=19为例，x^19 mod k = (x mod k) \* ( x^9 \* x^9 mod k) mod k，依次类推，再拆分指数9，直到指数为0。

同样的，我们先给出递归算法的代码：

描述：“快速幂取模”问题，即快速求x^n % k

函数名：fmod(x, n, k)

参数表：x -- 底数； n -- 指数； k – 模。

返回值：返回x^n % k。

def fmod(x, n, k): #递归法求x^n % k

if n == 1:

return x % k

t = fmod(x, n//2, k)

if n % 2 == 1:

return (x % k) \* (t \* t % k) % k

else:

return t \* t % k

是不是和“快速幂”问题几乎一模一样呢？

课后思考：

前面我们依次分析了斐波那契数列，快速幂和快速幂取模问题的递归算法，也给出了前两者的迭代算法，可是却没有把最重要的“快速幂取模”问题的迭代算法代码给出来。

相信聪明的你通过上述分析，再结合前面的联考卷题目，应该已经知道该怎么写出“快速幂取模”问题的迭代算法了吧，那就赶快试试吧！

另外，如果你有更 Pythonic（优雅的、地道的、整洁的）代码，或者与本文不同的算法思路和代码实现，请你一定留言或联系我，让我们一起讨论，共同进步。