Python算法之旅（第15期）

上期回顾：

描述：上一期我们分析了查找循环递增数组中最小值的方法，那么，如何来查找最大值呢？

算法分析：这里有两种思路：

一是利用最大值和最小值的位置关系，先找到最小值，则其左侧就是最大值；

二是模仿查找最小值的算法，修改部分代码，直接找到最大值。代码如下：

#对分查找循环递增数组的最大值下标（无重复元素）

def max\_val\_pos(a):

if a[0] < a[-1]: #纯递增数组

return len(a) - 1

low, high = 0, len(a) - 1

while low <= high:

mid = (low + high + 1) // 2

if a[mid] > a[low]:

low = mid

else:

high = mid - 1

return low

#对分查找循环递增数组的最大值下标（有重复元素）

def max\_val\_pos2(a):

if a[0] < a[-1]: #纯递增数组

return len(a) - 1

low, high = 0, len(a) - 1

while low <= high:

mid = (low + high + 1) // 2

#三者相等时，让mid左移

if a[low] == a[mid] == a[high]:

high -= 1

elif a[mid] >= a[low]:

low = mid

else:

high = mid - 1

return low

#对分查找循环递增数组的最大值（有重复元素）

def max\_val(a):

def fun(low, high):

mid = (low + high) // 2

if low == high:

return a[low]

elif low == high - 1:

return max(a[low], a[high])

elif a[low] == a[mid] == a[high]:

return max(fun(low, mid), fun(mid + 1, high))

elif a[mid] >= a[low]:

return fun(mid, high)

else:

return fun(low, mid - 1)

return fun(0, len(a) - 1)

算法设计好编程的目的是为了解决实际问题，很多VB算法题都是来自于实际问题的简单模拟和抽象，下面就让我们来看一道经典的好题。

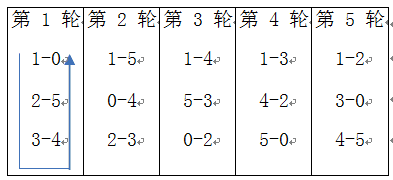
题目：单循环赛制

难度：3星 有趣：4星 有用：4星

分类：模拟，循环移位

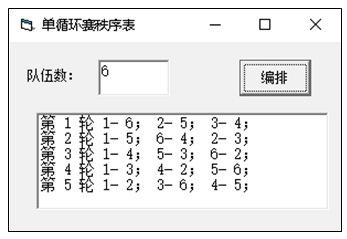
描述：单循环赛制是一种较为公平合理的比赛制度，比赛过程中所有参赛队伍均能相遇--次。其秩序编排可采用“逆时针轮转方法”:

数字1~ n依次作为队伍编号，把编号按U型走向分成均等两边(若n为奇数，则在末尾增加编号0，使总数为偶数)，即可得到第一轮的比赛秩序，例如,有7个队参加比赛,比赛秩序编排如下所示：

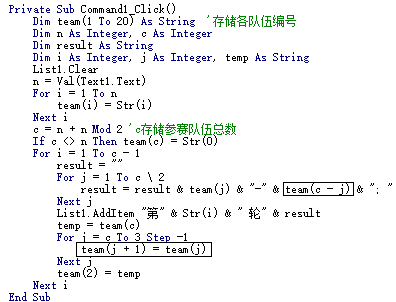


第二轮，固定编号1，其余编号均按逆时针方向移动-一个位置，即为该轮比赛秩序;以后各轮比赛秩序以此类推，与编号0对阵的表示本轮轮空。

小赵这此编写了VB 程序，功能如下:在文本框Text1中输入参赛队伍数n,单击“编排”按钮Command1,在列表框List1中输出每轮比赛秩序。程序运行效果如图b所示。



实现上述功能的V代码如下，但加框处代码有错，请改正。



答案：

(1) team(c + 1 - j)

(2) team(j) = team(j - 1)

算法分析：

这是典型的模拟算法，我们只要模拟手工设置单循环赛制的流程就可以了。

队伍1对战队伍c，队伍2对战队伍c-1，以此类推，队伍j对战队伍c+1-j。

固定编号1，其余队伍逆时针循环移动时，程序采用了类似插入排序的算法，先把team(c)保存的辅助变量temp中，再把[2, c-1]的元素依次后移，最后把temp值存储到team(2)，完成循环移位的操作。

使用Python语言可以写出更简洁的代码。

描述：单循环赛制是一种较为公平合理的比赛制度，比赛过程中所有参赛队伍均能相遇--次。其秩序编排可采用“逆时针轮转方法”：

数字1~ n依次作为队伍编号，把编号按U型走向分成均等两边(若n为奇数，则在末尾增加编号0，使总数为偶数)，即可得到第一轮的比赛秩序，例如,有7个队参加比赛,比赛秩序编排如下所示：

第一轮 第二轮 第三轮 第四轮 第五轮 第六轮 第七轮

1-0 1-7 1-6 1-5 1-4 1-3 1-2

2-7 0-6 7-5 6-4 5-3 4-2 3-0

3-6 2-5 0-4 7-3 6-2 5-0 4-7

4-5 3-4 2-3 0-2 7-0 6-7 5-6

第二轮，固定编号1，其余编号均按逆时针方向移动-一个位置，即为该轮比赛秩序;以后各轮比赛秩序以此类推，与编号0对阵的表示本轮轮空。

函数名：single\_loop\_system(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。

例1，当n=6时，输出：

第1轮: 1-6; 2-5; 3-4;

第2轮: 1-5; 6-4; 2-3;

第3轮: 1-4; 5-3; 6-2;

第4轮: 1-3; 4-2; 5-6;

第5轮: 1-2; 3-6; 4-5;

例2，当n=7时，输出：

第1轮: 1-0; 2-7; 3-6; 4-5;

第2轮: 1-7; 0-6; 2-5; 3-4;

第3轮: 1-6; 7-5; 0-4; 2-3;

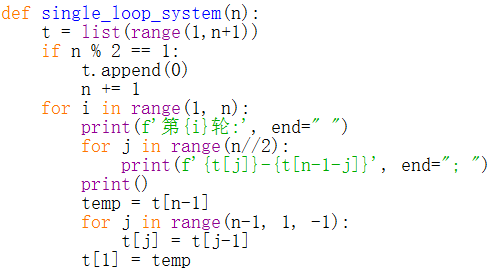
第4轮: 1-5; 6-4; 7-3; 0-2;

第5轮: 1-4; 5-3; 6-2; 7-0;

第6轮: 1-3; 4-2; 5-0; 6-7;

第7轮: 1-2; 3-0; 4-7; 5-6;

代码如下：



课后思考：

上述的单循环赛制算法虽然能保证比赛过程中所有参赛队伍均能相遇--次，但是当队伍数量为单数时，第二轮的轮空队从第四轮起每场都与前一轮的轮空队比赛，对其非常不利。例如，当n=7时，6号队伍第二轮轮空，从第四轮开始，他每场都与前一轮的轮空队比赛，非常不利。

为了避免出现这种不合理现象，从1985年起，世界性比赛多采用"贝格""编排法。

采用"贝格尔"编排法，编排时如果参赛队为双数时，把参赛队数分一半(参赛队为单数时，最后以"0"表示形成双数)，前一半由1号开始，自上而下写在左边;后一半的数自下而上写在右边，然后用横线把相对的号数连接起来。这即是第一轮的比赛。

第二轮将第一轮右上角的编号("0"或最大的一个代号数)移到左角上，三轮又移到右角上，以此类推。

即单数轮次时"0"或最大的一个代号在右上角，双数轮次时则在左上角。

如下表示:

7 个队比赛的编排方法

第一轮 第二轮 第三轮 第四轮 第五轮 第六轮 第七轮

1-0 0-5 2-0 0-6 3-0 0-7 4-0

2-7 6-4 3-1 7-5 4-2 1-6 5-3

3-6 7-3 4-7 1-4 5-1 2-5 6-2

4-5 1-2 5-6 2-3 6-7 3-4 7-1

无论比赛队是单数还是双数，最后一轮时，必定是"0"或最大的一个代号在右上角，"1"在右下角。

聪明的你是否明白了"贝格尔"编排法的原理呢？如果搞懂了，不妨用程序来实现它。

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。

例1，当n=6时，输出：

第1轮: 1-6; 2-5; 3-4;

第2轮: 6-4; 5-3; 1-2;

第3轮: 2-6; 3-1; 4-5;

第4轮: 6-5; 1-4; 2-3;

第5轮: 3-6; 4-2; 5-1;

例2，当n=7时，输出：

第1轮: 1-0; 2-7; 3-6; 4-5;

第2轮: 0-5; 6-4; 7-3; 1-2;

第3轮: 2-0; 3-1; 4-7; 5-6;

第4轮: 0-6; 7-5; 1-4; 2-3;

第5轮: 3-0; 4-2; 5-1; 6-7;

第6轮: 0-7; 1-6; 2-5; 3-4;

第7轮: 4-0; 5-3; 6-2; 7-1;

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。