## 约瑟夫环问题(Josephus loop problem)

据说著名犹太历史学家Josephus有过以下的故事：在罗马人占领乔塔帕特后，39 个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式，41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。然而Josephus 和他的朋友并不想遵从。首先从一个人开始，越过k-2个人（因为第一个人已经被越过），并杀掉第k个人。接着，再越过k-1个人，并杀掉第k个人。这个过程沿着圆圈一直进行，直到最终只剩下一个人留下，这个人就可以继续活着。问题是，给定了和，一开始要站在什么地方才能避免被处决。Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏。

17世纪的法国数学家加斯帕在《数目的游戏问题》中讲了这样一个故事：15个教徒和15 个非教徒在深海上遇险，必须将一半的人投入海中，其余的人才能幸免于难，于是想了一个办法：30个人围成一圆圈，从第一个人开始依次报数，每数到第九个人就将他扔入大海，如此循环进行直到仅余15个人为止。问怎样排法，才能使每次投入大海的都是非教徒。

我们可以把问题简单描述为：总共n个人，从1开始顺序编号，排成一个圆形队列。从1号开始报数，报数到k的人出圈。然后从出圈位置的下一个人继续从1报数，到k的人出圈。已经出圈的人不再参与报数，继续上面的做法，直到环形队列中最后只留下m个人。

即：n为初始总人数；k为出圈的报数；m为最后留下的人数。

这个问题有多种解法，下面列出了五种解法。其中基于递推公式的循环和递归解法，最后只能留下1个人，其它解法最后可以留下m个人。

1. 数组模拟

用数组A[]标记当前下标的人是否出圈，初始化0，表示都未出圈。当报数到k，将数组相应下标元素置为1，标记已经出圈。经过n-m轮报数出圈，数组中元素值为0的即为剩下的m个人，其对应下标+1（数组下标从0开始）即为剩下人的标号。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* Jos(int A[],int n,int k,int m) \*//

//\* --A[]，标记数组，初始化0，i出圈时，A[i]=1 \*//

//\* --n 总人数 \*//

//\* --k 报数k的人出圈 \*//

//\* --m 最后留下的人数 \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Jos(int A[],int n,int k,int m)

{

int i,num=0,len=n;

i=-1;

while(len>m) //最后保留m个人。len保存当前剩下人数。循环n-m次。

{

num=0; //报数计数

while(num!=k)

{

i=(i+1)%n; //循环数组，用模运算实现逻辑环

if(A[i]==0) //报数计数，跳过已经出圈的元素

num++;

}

A[i]=1; //i出圈

len--; //总人数减1

}

}

输出时，循环控制输出A[]中值为0的元素的下标加1即为剩下m个人的编号。

//打印结果

for(i=0;i<n;i++)

{

if(A[i]==0)

cout<<i+1<<" "; //编号为数组下标+1

}

cout<<endl;

这种方法的时间复杂度为O(n\*k)。

2. 数组循环左移模拟

把n个人的编号1--n存入数组A[0]到A[n-1]，相当于把编号减1，这样用模运算做循环左移k比较方便。循环左移k，相当于把报数k出圈的人，循环移到了数组的最后。循环左移k到数组最后的元素，下一轮不再参与移动（已出圈）。下一轮任然这样循环左移k，出圈的人又被移到数组的后面，继续这样处理，数组A[]中下标0到m-1的元素即为剩下人的编号。下面以n=11，k=3，m=1为例来演示循环移动和出圈的过程，为了都是数字看着混乱，用abc...对人进行编号。11人经过10轮循环左移，每轮左移3，最后剩下1人。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 原始排列 | a | b | c | d | e | f | g | h | i | j | k |
| 1轮 | d | e | f | g | h | i | j | k | a | b | c |
| 2轮 | g | h | i | j | k | a | b | d | e | f | c |
| 3轮 | j | k | a | b | d | e | g | h | i | f | c |
| 4轮 | b | d | e | g | h | j | k | a | i | f | c |
| 5轮 | g | h | j | k | b | d | e | a | i | f | c |
| 6轮 | k | b | d | g | h | j | e | a | i | f | c |
| 7轮 | g | h | k | b | d | j | e | a | i | f | c |
| 8轮 | b | g | h | k | d | j | e | a | i | f | c |
| 9轮 | b | g | h | k | d | j | e | a | i | f | c |
| 10轮 | g | b | h | k | d | j | e | a | i | f | c |

第一轮，c出圈，不再参与移动；第二轮，f出圈，不再参与移动；每轮一个人出圈，参与循环左移的人减1，直到剩下最后1人。本例经过10轮循环左移，最后剩下g，存放在数组A[0]。g如果对应到编号为7，数组下标为6。上图，有背景部分即为出圈的顺序，即第十轮结束，数组A[]从高到底输出即为出圈顺序。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* Jos3(int A[],int n,int k,int m)--数组循环左移k \*//

//\* --n 总人数 \*//

//\* --k 报数k的人出圈 \*//

//\* --m 最后留下人数 \*//

//\* --Jos3()数组循环左移k实现 \*//

//\* --每循环左移k位，出队人在当前数组最后 \*//

//\* --最后数组A[]从n-1到0下标输出，即出队次序 \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Jos3(int A[],int n,int k,int m)

{

int t; //循环左移缓存变量

while(n>m) //最后留下m人，循环左移n-m轮

{

for(int i=0;i<k;i++) //循环左移k次，最后1人出队

{

t=A[0];

for(int j=0;j<n-1;j++)

A[j]=A[j+1];

A[j]=t;

}

n--; //总人数减1，参与移动的人数减1

}

}

算法时间复杂度O(n\*k)。

3. 约瑟夫环递推公式求解

我们从上面数组循环左移的过程来推导约瑟夫环问题的递推公式，这样便于理解。在上面的例子中，最后剩下的人是g，存在数组A[0]中，即最后下标为0。假定我们用变量p来保存g在数组中的下标，则p=0。那么g在上一轮（倒数第一轮，9轮）的下标是什么呢？从上图可以看出下标是1。我们怎么还原出上一轮g的下标呢？这是关键！

我们知道g的下标是上一轮循环左移k（3）的结果，那么我们把出圈的b算上，执行一个逆操作，即循环右移k（3），即可还原出g和b在上一轮（9轮）的下标。总共2个元素，执行循环右移k，所以上一轮g的下标为p=(p+k)%2=(0+3)%2=1。也可以算出已出圈的b 在第9轮的下标为p=(p+k)%2=(1+3)%2=0。

第8轮下标，根据第9轮g、b以及出圈h下标0、1、2，倒推它们在第8轮的下标，此时共3元素，循环右移k(3)，所以第8轮下标分别为：g的下标p=(p+k)%3=(1+3)%3=1；b下标p=(p+k)%3=(0+3)%3=0；h下标p=(p+k)%3=(2+3)%3=2。出圈的元素，也可倒推处上一轮的下标，事实上任何一个元素，根据当前的下标，通过循环右移k，还原出上一轮的下标。下面我们只跟踪g在上一轮的下标。

第7轮，共4个元素循环右移k(3)，则g的下标p=(p+k)%4=(1+3)%4=0。

第6轮，共5个元素循环右移k(3)，则g的下标p=(p+k)%5=(0+3)%5=3。

第5轮，共6个元素循环右移k(3)，则g的下标p=(p+k)%6=(3+3)%6=0。

第4轮，共7个元素循环右移k(3)，则g的下标p=(p+k)%7=(0+3)%7=3。

第3轮，共8个元素循环右移k(3)，则g的下标p=(p+k)%8=(3+3)%8=6。

第2轮，共9个元素循环右移k(3)，则g的下标p=(p+k)%9=(6+3)%9=0。

第1轮，共10个元素循环右移k(3)，则g的下标p=(p+k)%10=(0+3)%10=3。

原始序列，共11个元素循环右移k(3)，则g的下标p=(p+k)%11=(3+3)%11=6。

由上面的例子，可以看出，如果知道某个人（元素），在当前序列中的下标p，不管有没有出圈，通过循环右移k，即可还原出上一轮的下标，参与循环的人数要包含左移出圈的那个人。比如，某元素在当前序列的下标p，上一轮的人数是x，则此元素在上一轮的下标为p=(p+k)%x，即x个人（包含出圈的那个），循环右移k，还原出元素上一轮的下标。

当前序列下标还原上一轮下标的递推公式：**p=(p+k)%x**。

利用这个递推公式，最后留下人的下标一定为p=0，通过n-1轮循环右移k，即可倒推出剩下人在原始序列中的下标。循环右移时，人数从2开始，直到n，每次循环右移，人数增1，是循环左移的逆过程。

为方便写递归函数实现，我们有时把递推公式写成如下形式。假定当前序列有n-1个人（元素），则上一轮就有n个人。某个人（元素）在当前序列的下标为f(n-1)，上一轮序列的下标为f(n)，是同一个元素在上下2个序列的下标，变量n只是记录当前总人数，则递推公式可以写为：

**f(n)=(f(n-1)+k)%n**

按照这种记法，则最后留下元素（人）的下标为：f(1)=0。

使用递推公式还原下标计算，省略了数组的真实移动，算法效率提高很多。

4. 递推公式循环求解

利用递推公式：p=(p+k)%x，最后留下人的下标p=0，经过n-轮循环右移k的下标还原即可计算出留下来的人在原始序列中的下标。循环右移k的总人数x从2开始，直到n。下标+1为对应的编号。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* int Jos1(int n,int k) --只能留下1人 \*//

//\* --n 总人数 \*//

//\* --k 报数k的人出圈 \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int Jos1(int n,int k)

{

int p=0; //最后留下人的下标为0

for(int x=2;x<=n;x++) //人数从2开始的n-1轮循环右移k

p=(p+k)%x;

return p+1; //下标加1为标号。

}

算法时间复杂度O(n)。因为省略了数组的真正移动，实际时间效率要好得多。

5. 递推公式递归求解

利用f(n)=(f(n-1)+k)%n形式递推公式，f(1)=0，很容易用递归函数实现下标的递推计算。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* int f(int n,int k) --只能留下1人 \*//

//\* --n 总人数 \*//

//\* --k 报数k的人出圈 \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int f(int n,int k)

{

if(n==1)

return 0; //递归出口

else

return (f(n-1,k)+k)%n;

}

6.链表模拟求解

约瑟夫环问题也可以用链表模拟求解，各种线性链表均可模拟求解。下面以不带头结点的单循环链表，解释链表模拟求解过程。首先构造一个单循环链表，头指针L指示的首元素结点数据域元素值写1，接下来2、3、…、n。从头指针结点开始计数，移动头指针到计数k-1的结点，则L->next指示的结点即为要出圈的结点（报数k），删除此结点。L后移一个结点，即L=L->next，从此结点开始继续报数，到k-1个结点，重复上面的工作，直到最后剩下m个结点。这种链表模拟，头指针指向是在不停变化的。算法描述如下：

typedef int elementType;

typedef struct LNode //单链表结点定义

{

elementType data;

struct LNode \*next;

} node, \*linkedList;

void Jos(node \*&L,int n,int k,int m)

{

node \*u;

int c;

while(n>m) //最后保留m人

{

c=1;

while(c!=k-1) //找到报数为k-1结点，L指向此结点

{

L=L->next;

c++;

}

//删除报数为k的结点

u=L->next;

L->next=u->next;

delete u;

L=L->next; //从此结点重新开始报数k

n--; //结点数减1

}

}

int main(int argc, char\* argv[])

{

int n,k,m,i;

node \*L,\*p,\*R;

cout<<"输入总人数 n=";

cin>>n;

cout<<"输入出圈报数 k=";

cin>>k;

cout<<"输入留下的人数 m=";

cin>>m;

L=new node;

L->data=1;

L->next=L;

R=L; //设置尾指针，采用尾插法创建单循环链表

for(i=2;i<=n;i++) //尾插法插入剩下n-1个结点

{

p=new node;

p->data=i;

R->next=p;

p->next=L; //形成循环

R=p; //移动尾指针

}

Jos(L,n,k,m); //调用约瑟夫环处理函数

cout<<L->data; //打印留下的第一个人

p=L->next;

while(p!=L) //打印留下的其他人

{

cout<<" "<<p->data;

p=p->next;

}

cout<<endl;

//释放剩下结点

p=L->next;

L->next=NULL;

while(p)

{

R=p->next;

delete p;

p=R;

}

return 0;

}

7.队列模拟求解

把编号1—n的所有人入队，出队每次从0计数，每出队一个人计数加1，计数到k的人直接出队，其他人出队后再入队，同时记录队列中剩下人数，当剩下人数为m时，输出队列的人。算法描述如下：

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* Jos(queue<int>Q,int n,int k,int m) \*//

//\* --queue<int>Q，队列 \*//

//\* --n 总人数 \*//

//\* --k 报数k的人出圈 \*//

//\* --m 最后留下的人数 \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Jos(queue<int> &Q,int n,int k,int m)

{

int num=1; //报数计数初值

while(Q.size()!=m)

{

int x=Q.front();

Q.pop();

if(num!=k){

num++; //报数+1

Q.push(x); //报数未到k，重新入队

}

else num=1; //报数k的人直接出队，报数计数初始化1

}

}

8. 测试用例

输入：9 2 1 //总人数n； 报数数值k； 剩下人数m

输出：3 //剩下人的编号

输入：6 5 1

输出：1 //剩下人的编号

输入：11 3 2

输出：2 7 //剩下人的编号

输入：41 3 2

输出：16 31 //剩下人的编号

输入：100 5 2

输出：47 79 //剩下人的编号