Josephus 约瑟夫 问题

假设n个竞赛者排成一个环形，依次顺序编号1，2，…，n。从某个指定的第1号开始，沿环计数，每数到第m个人就让其出列，且从下一个人开始重新计数，继续进行下去。这个过程一直进行到所有的人都出列为止。最后出列者为优胜者。

无论是用链表实现还是用数组实现来解约瑟夫问题都有一个共同点：要模拟整个游戏过程，不仅程序写起来比较麻烦，而且时间复杂度高达O(nm)，当n，m非常大(例如上百万，上千万)的时候，几乎是没有办法在短时间内出结果的。注意到原问题仅仅是要求出最后的胜利者的序号，而不是要模拟整个过程。因此如果要追求效率，就要打破常规，实施一点数学策略。

为了讨论方便，先把问题稍微改变一下，并不影响原意：

问题描述：n个人（编号0~(n-1))，从0开始报数，报到(m-1)的退出，剩下的人继续从0开始报数。求胜利者的编号。

我们知道第一个人(编号一定是m%n-1) 出列之后，剩下的n-1个人组成了一个新的约瑟夫环（以编号为k=m%n的人开始）:   k   k+1   k+2   ... n-2, n-1, 0, 1, 2, ... k-2并且从k开始报0。

现在我们把他们的编号做一下转换： k      --> 0 k+1    --> 1 k+2    --> 2 ... ...

k-2    --> n-2

变换后就完完全全成为了(n-1)个人报数的子问题，假如我们知道这个子问题的解：例如x是最终的胜利者，那么根据上面这个表把这个x变回去不刚好就是n个人情况的解吗？变回去的公式很简单：x'=(x+k)%n

如何知道(n-1)个人报数的问题的解？显然，只要知道(n-2)个人的解就行了。(n-2)个人的解呢？当然是先求(n-3)的情况 ---- 这显然就是一个倒推问题！

递推公式：

令f[i]表示i个人玩游戏报m退出最后胜利者的编号，最后的结果自然是f[n]

递推公式 f[1]=0;

f[i]=(f[i-1]+m)%i;   (i>1)

有了这个公式，我们要做的就是从1-n顺序算出f[i]的数值，最后结果是f[n]。因为实际生活中编号总是从1开始，我们输出f[n]+1由于是逐级递推，不需要保存每个f[i]，程序也是很简单：

#include <stdio.h> main() {

   int n, m, i, s=0;    printf ("N M = ");

   scanf("%d%d", &n, &m);

   for (i=2; i<=n; i++) s=(s+m)%i;    printf ("The winner is %d\n", s+1); }

这个算法的时间复杂度为O(n)，相对于模拟算法已经有了很大的提高。算n，m等于一百万，一千万的情况不是问题了。可见，适当地运用数学策略，不仅可以让编程变得简单，而且往往会成倍地提高算法执行效率。

**Wikipedia**: 这个问题是以[弗拉维奥·约瑟夫斯](http://zh.wikipedia.org/wiki/%E5%BC%97%E6%8B%89%E7%B6%AD%E5%A5%A7%C2%B7%E7%B4%84%E7%91%9F%E5%A4%AB%E6%96%AF" \o "弗拉維奧·約瑟夫斯)命名的，它是[1世纪](http://zh.wikipedia.org/wiki/1%E4%B8%96%E7%BA%AA)的一名犹太历史学家。

他在自己的日记中写道，他和他的40个战友被罗马军队包围在洞中。

他们讨论是自杀还是被俘，最终决定自杀，并以抽签的方式决定谁杀掉谁。约瑟夫斯和另外一个人是最后两个留下的人。

约瑟夫斯说服了那个人，他们将向罗马军队投降，不再自杀。约瑟夫斯把他的存活归因于运气或天意，他不知道是哪一个.

**问题分析：**解决该问题有两种思路，第一种**：通过建立循环链表来模拟这个过程**

第二种：**通过递归方式（数学归纳将问题转化为数学问题）**

由于递归方式，代码更简洁，下面首先以递归方式来解决问题

**——————递归实现：**

**例如**对 m= 10,k=3

**0　　1　　2　　3　　4　　5　　6　　7　　8　　9　　(\*)**

**0　　1　　3　　4　　5　　6　　7　　8　　9 　  (\* 循环下去)**

**转化为：　　　　　　　 3　　4　　5　　6　　7　　8　　9　　0　　1　　(\* 循环下去）**

**0　　1　　2　　3　　4　　5　　6　　7　　8**

**m=10,k=3 去掉一个元素之后，变成了一个m=9,k=3的约瑟夫环问题**

**并且有如下关系**

**3 = (0+3)%10   4 = （1+3）%10 　　... 0 = (3+7)%10**

**即 3 = (0+k)% m    4 =  (1+k) % m       ... 0  = (3+k) % m**

**m =10,k =3 设约瑟夫环最后一个出列的人为 Joseph(10,3),那么存在如下关系**

**Joseph(10,3) = (Joseph(9,3)+k) %m;**

**...**

**Joseph(n,k) = (Joseph(n-1,k)+k) % n （n>1）;**

**C++实现如下：**

**递归方法一：**

[复制代码](javascript:void(0);)

1 int Joseph(int m,int k)

2 {

3 if(m<=0||k<=0)

4 {

5 cout<<"error!"<<endl;

6 return -1;

7 }else

8 {

9 if(m==1)

10 {

11 return 0;

12 }else

13 {

14 return ((Joseph(m-1,k)+k)%m);

15 }

16 }

17 }

[复制代码](javascript:void(0);)

**递归方法二：如果输出整个出队的顺序**

[复制代码](javascript:void(0);)

int Joseph(int m,int k,int i)

{

if(m<=0||k<=0||m<i)

{

cout<<"error"<<endl;

return -1;

}else

{

if(i==1)

{

return (m+k-1)%m;

}else

{

return ((Joseph(m-1,k,i-1)+k)%m);

}

}

}

[复制代码](javascript:void(0);)

**程序运行结果如下：**

[复制代码](javascript:void(0);)

int main()

{

cout<<"递归方法一"<<endl;

cout << Joseph(6,3) << endl;

cout<<"递归方法二"<<endl;

for(int i=1;i<=6;i++)

{

cout<<Joseph(6,3,i)<<endl;

}

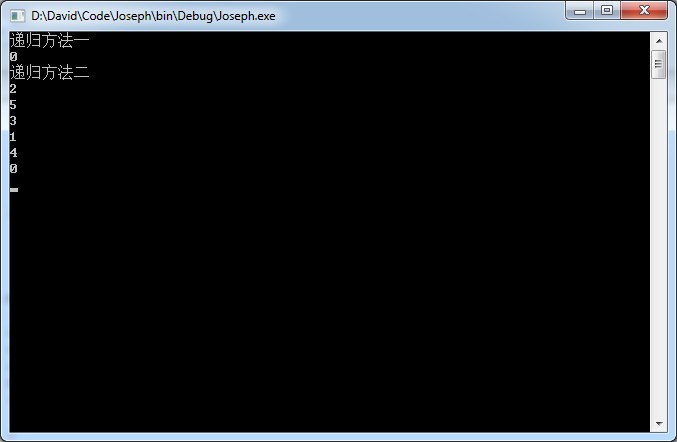
getchar();

return 0;

}

[复制代码](javascript:void(0);)

 结果：



**——————循环链表实现：**

**建立节点数据结构：循环链表**

[复制代码](javascript:void(0);)

struct Node

{

int data;

Node \* next;

};

struct LinkedList

{

Node \*pHead;

Node \*pTail;

int len;

};

[复制代码](javascript:void(0);)

**建立循环链表**

[复制代码](javascript:void(0);)

//建立个节点

Node \* GetNode(int i)

{

Node \* p = (Node \*)malloc(sizeof(Node));

if(p!=NULL&&i>=0)

{

p->data = i;

p->next = NULL;

return p;

}else

{

cout<<"error"<<endl;

exit(-1);

return NULL;

}

}

//建立链表

LinkedList\* CreateLinkedList(int i)

{

Node\* node = GetNode(0);

LinkedList \*head = (LinkedList\*)malloc(sizeof(LinkedList));

if(head == NULL)

{

cout<<"CreateLinkedList:memory error"<<endl;

exit(-1);

return NULL;

}

if(i<=0)

{

cout<<"CreateLinkedList: error"<<endl;

exit(-1);

return NULL;

}

head->pHead = node;

head->pTail = node;

head->len = 1;

if(i==1)

{

node->next = node;

}else

{

Node \*p = head->pHead;

for(int j=1;j<=i-1;j++)

{

Node\* node = GetNode(j);

node->data = j;

p->next = node;

p=p->next;

head->len++;

}

head->pTail = p;

p->next = head->pHead;

}

return head;

}

[复制代码](javascript:void(0);)

**删除节点：**

[复制代码](javascript:void(0);)

//删除节点

void RemoveNode(LinkedList\*head,Node \*deleNode)

{

Node\* p = head->pHead;

cout<<deleNode->data<<endl;

if(p!=NULL){

if(head->len>1){

do

{

if(p->data==deleNode->data)

{

if(p==head->pHead)

{

head->pHead = p->next;

}

if(p==head->pTail)

{

head->pTail = p->next;

}

p->data = p->next->data;

p->next = p->next->next;

head->len--;

return;

}else{

p=p->next;

}

}while(p!=head->pHead);

}else

{

cout<<"error";

exit(-1);

return;

}

}else

{

return;

}

}

[复制代码](javascript:void(0);)

**约瑟夫模拟：**

[复制代码](javascript:void(0);)

int Joseph(int m,int k)

{

if(m<=0||k<=0)

{

cout<<"error:input"<<endl;

return -1;

}

LinkedList\* list = CreateLinkedList(m);

//Print\_List(list);

Node \* p = list->pHead;

for(int i=1;i<=k;i++)

{

if(list->len ==1)

{

return p->data;

}

if(i==k)

{

RemoveNode(list,p);

i = 1;

}

p=p->next;

}

return 0;

}

[复制代码](javascript:void(0);)

**程序运行结果如下：**

[复制代码](javascript:void(0);)

int \_tmain(int argc, \_TCHAR\* argv[])

{

cout<<"循环列表"<<endl;

cout<<Joseph(6,3)<<endl;

getchar();

return 0;

}

[复制代码](javascript:void(0);)

