实验 1　约瑟夫（Josephu）问题

王天一 320200931301 2022 年 4 月 10 日

注：该次实验的所有代码和文档均放入了压缩包以及我在 GitHub 的仓库：

wtyqqq/Data_Structure_Lab (github.com)

1 题目

设编号为 1，2，… m 的 m 个人围坐一圈，从 1 开始报数，数到 n 的那个人出列，它的下一位又从 1 开始报数，数到 n 的那个人又出列，依次类推，直到所有人出列为止，由此产生一个出队编号的序列。

2 测试样例

(1) 总人数 m,出列序号 n。

(2) 测试数据

m	n	出列顺序
8	6	64358721
7	5	5324716
6	1	123456
6	8	254163
1	1	1
1	0	error

下面是运行测试：

```
PS C:\Users\wty02\Desktop\数据结构实验\Data_Structure_Lab\lab1> python3 .\lab1.py
8
6
Normal Solution 1:
Normal Solution 2:
Quick Solution:
1
PS C:\Users\wty02\Desktop\数据结构实验\Data_Structure_Lab\lab1> python3 .\lab1.py
7
5
Normal Solution 1:
5 3 2 4 7 1 6
Normal Solution 2:
5 3 2 4 7 1 6
Quick Solution:
6
PS C:\Users\wty02\Desktop\数据结构实验\Data_Structure_Lab\lab1> python3 .\lab1.py
6
1
Normal Solution 1:
1 2 3 4 5 6
Normal Solution 2:
2 4 6 3 1 5
Quick Solution:
6
```

```
PS C:\Users\wty02\Desktop\数据结构实验\Data_Structure_Lab\lab1> python3 .\lab1.py
6
8
Normal Solution 1:
2 5 4 1 6 3
Normal Solution 2:
2 5 4 1 6 3
Quick Solution:
3
PS C:\Users\wty02\Desktop\数据结构实验\Data_Structure_Lab\lab1> python3 .\lab1.py
1
1
Normal Solution 1:
1
Normal Solution 2:
1
Quick Solution:
1
PS C:\Users\wty02\Desktop\数据结构实验\Data_Structure_Lab\lab1> python3 .\lab1.py
1
0
The input is invalid!
```

3 算法描述

（1）顺序存储算法

(a)将 m 个人的位置用数组 a 的元素下标表示，数组元素的初始值为 1,表示在圆圈内，出列后数值为 0。设当前已出列人数 total 为 0。

(b)从 a 的下标 0 位置开始递增报数，当报数达到 n 时，对 a 的该元素赋值为 0，出列人数 total 加 1。

(c)再从 a 的下一个下标位置开始，从 1 开始对 a 未出列元素继续报数，当报数达到 n 时，对 a 该位置的元素赋值为 0，出列人数 total 加 1。

(d)当出列人数 total 为 m 时，程序结束；否则，执行(c)。

注： python 语言 list 列表是动态数组，可以删除元素。

(2) 链表存储算法

常用方法：

（a）用循环链表存储 m 个人的位置信息。

（b）当报数到需要出列的结点时，删除该结点。

（c）当只有一个结点时，循环结束。

其它方法：

（1） 大循环内（小于 m 时一直循环），结点存储位置信息，单向链表遍历，元素出列，删除结点。

（2） 大循环内（小于 m 时一直循环），结点元素含位置信息，是否出列标志（圈内时为 1），单向链表遍历，出列元素对应结点的标志置为 0，出列元素个数加 1。

4. 测试数据模拟算法执行过程

m=8　n= 6

1　2　3　4　5　 6　7　8

*

第 1 次出列　6　a[5]=0　total=1

1　2　3　4　5　　7　 8

*

第 2 次出列 4　　a[3]=0　total=2

1　2　3　　5　　7　 8

*

第 3 次出列 4　　a[2]=0　total=3

1　2　　5　　7　 8

*

第 4 次出列 5　　a[4]=0　total=4

1　2　　7　 8

*

第 5 次出列 8　　a[7]=0　total=5

1　2　　7

*

第 6 次出列 7　　a[6]=0　total=6

1　2

*

第 7 次出列 2　　a[1]=0　total=7

1

第 8 次出列 1　　a[0]=0　total=8

5. 扩展思考

（1）　如果只关心最后出列的人员，不考虑数据结构要求，是否还有更快的算法？

有，使用递推公式法。

$$f(m,n)=(f(m-1,n)+n-1)\bmod m+1$$

$$f(m,n)=0\ (m=0)$$

使用此法能够大大减小算法的时间复杂度。

（2）　是否可以将出列顺序保存起来，需要的时候输出？

可以，使用一个列表存储起来即可

6. 代码实现

所有的代码实现在 lab1.py 内，可供查阅。

lab1.py

对于此次实验，我共使用了三种方法，一种是用实验指导上顺序存储的方法实现的，一种是用实验指导上的循环链表实现的，最后一种是使用的数学的递推方法实现的。（具体代码在一起打包的源文件内，后面也有截图）

首先，我们规定两个输入：

m：人（猴）数

n：第 n 个人（猴）出列

对这两个输入，必须均为正数。

```
m = int(input())
n = int(input())
if m <= 0 or n <= 0:
    print("The input is invalid!")
    exit(0)
```

对于输入的判断

对于第一种方法，我先建立了一个 List（因为 List 很像 C++中 std::vector，长度是可以变化的，故也可以叫这种方法为向量的实现），存储每个猴子的编号，并设置了一个计数变量 counter，索引变量 index。利用 index，遍历整个列表，每遍历一个编号，counter 也加一，当 counter 等于 n 时，我们使用 list 的 pop 功能，将其删除出列表并打印出来，然后将 counter 恢复为 0。到了 list 的最后一个元素时，我们直接将其重新变为 0。如此反复，直到只剩下一个元素。

```
def normalSolution1(m, n):
    '''
    @description: 这个函数实现了实验报告上的顺序存储方法来解决该问题。
    This function implements the sequential storage method on the lab report to
    solve this problem.
    '''
    List = [x for x in range(1, m+1)]
    index = 0
    while len(List) > 1:
        counter = 0
        while counter < n-1:
            index = (index + 1) % len(List)
            counter += 1
        print(List[index], end=" ")
        List.pop(index)
        counter = 0
    print(List[0])
```

第一种实现方案

对于第二种方法，我先写出了 node 的类和循环链表的类。然后建立一个有 n 个结点的循环链表，并输入数据。跟第一种方法类似，我也设计了一个计数变量 counter。每过一个节点加一，直到 counter=n，将这个节点删除并输出，然后恢复 counter，继续循环，直到只剩一个节点，即为最后剩下的猴子（人）。

```python
class Node:
    def __init__(self, value, next=None):
        self.value = value
        self.next = next


class ringLinkedList:
    '''
    循环链表的实现
    '''

    def __init__(self, value):
        self.head = Node(value)
        self.head.next = self.head

    def insert(self, value):
        node = Node(value)
        node.next = self.head.next
        self.head.next = node
        self.head = node

    def delete(self):
        if self.head.next == self.head:
            print("The list is empty!")
            return
        else:
            self.head.next = self.head.next.next
```

节点和循环链表类的实现

```python
def normalSolution2(m, n):
    '''
    @description: 这个函数实现了实验报告上的循环链表方法来解决该问题。
    This function implements the loop linked list method on the lab report to
    solve this problem.
    '''
    remainNumber = m
    List = ringLinkedList(1)
    for i in range(2, m+1):
        List.insert(i)
    List.head = List.head.next
    while remainNumber > 1:
        counter = 1
        while counter < n-1:
            List.head = List.head.next
            counter += 1
        print(List.head.next.value, end=" ")
        List.delete()
        remainNumber -= 1
        List.head = List.head.next
    print(List.head.next.value)
```

第二种算法的实现

最后一中方法，即为使用数学方法，使用递推公式实现求解，直接求出最后剩下的猴子（人）。递推式为：

$$f(m,n)=(f(m-1,n)+n-1)\bmod m+1$$

$$f(m,n)=0\ (m=0)$$

```python
def quickSolution(m, n):
    '''
    Q: 如果只关心最后出列的人员，不考虑数据结构要求，是否还有更快的算法？
    A: 有，使用数学方法.
    '''
    if m == 0:
        return 0
    else:
        return (quickSolution(m-1, n) + n-1) % m+1
```

递推法的实现