

数据结构：加里森的任务 实验报告

毛子恒 李臻 张梓靖

2020 年 10 月 10 日

小组成员

班级：2019211309	姓名：毛子恒	学号：2019211397	分工：代码 文档
班级：2019211310	姓名：李臻	学号：2019211458	分工：测试 文档
班级：2019211308	姓名：张梓靖	学号：2019211379	分工：可视化 文档

1 需求分析

1.1 题目描述

在由序号为 1 至 n 的 n 个元素依次排列并且首尾相接而组成的环中，规定初始时从序号 1 开始依次经过 2, 3, ... 元素走到第 n 个元素的方向为正方向。

初始时以第 x 个元素为起点 st ，重复以下过程 $n-1$ 次：以 st 为第 1 个元素，沿正方向找到第 y 个元素 del ，从环中删除 del 元素，再将原 del 的下一个元素作为新的 st 。

求经过 $n-1$ 次操作之后，环中仅剩的一个元素的序号是否是 1。

1.2 输入描述

程序从标准输入中读入数据。输入一行三个整数，用空格分隔，分别表示 n, x, y 。

其中各个值的范围需要满足 $1 < n \leq 10^9$ $0 < x \leq n$ $0 < y \leq 10^9$ 。

由于程序时间复杂度较大，建议 $n \leq 10^4$ 。

1.3 输出描述

输出分为三种情况：

1. 输入合法，程序正常运行结束。此时输出两行，第一行一个字符串“Yes”或者“No”（不带引号），分别表示最后一个元素是/不是 1，第二行一个数字，表示最后一个元素的序号。
2. 输入不合法。此时输出一行一个字符串>Please check your input.”（不带引号）。
3. 程序发生运行时错误，比如内存分配失败。此时程序没有输出。

1.4 样例输入输出

1.4.1 样例输入输出 1

【输入】

10 1 3

【输出】

No
4

1.4.2 样例输入输出 2

【输入】

10 3 7

【输出】

Yes
1

1.4.3 样例输入输出 3

【输入】

100 87 305

【输出】

No
50

1.4.4 样例输入输出 4

【输入】

1000 725 801

【输出】

No
798

1.4.5 样例输入输出 5

【输入】

1 1 3

【输出】

Please check your input.

1.4.6 样例输入输出 6

【输入】

5 6 3

【输出】

Please check your input.

1.5 程序功能

程序通过给定的 n, x, y 计算出最后环中仅剩的元素序号，并且与 1 比较。

2 概要设计

2.1 问题解决的思路

使用单循环链表维护此约瑟夫环，首先在链表中依次插入 n 个结点表示 n 名队员，以 now 指针模拟计数过程。从头结点找到第 x 个结点，此后执行以下操作 $n - 1$ 次：找到当前结点之后的第 $y - 1$ 个结点，删除这个结点。此题中单循环链表实现了初始化、判空、在指定位置增加节点、删除指定位置的节点、释放空间这五种操作。由于链表的删除操作实现是删除给定结点的后继，所以 now 指针始终指向当前正在计数元素的前驱。由于单循环链表中存在一个特殊的头结点，所以另实现一个函数，返回某个结点的后继（跳过头结点）。更多细节在调试分析报告部分中讨论。

2.2 链表的定义

```
1 // 数据对象
2 typedef struct node
3 {
4     int item;
5     struct Node * next;
6 } Node;
7
8 typedef Node * List;
9
10 // 基本操作
11 /*
12 * 操作：初始化链表
13 * 后件：plist指向一个循环链表的头结点
14 */
15 void initList(List * plist);
16
17 /*
18 * 操作：判断链表是否为空
19 * 前件：list是循环链表的头结点
20 * 后件：如果该链表为空，返回true，否则返回false
21 */
```

```
22 bool isEmpty(const List list);
23
24 /*
25  * 操作：向链表的某个节点后插入一个节点
26  * 前件：pnode是链表中的某一个节点
27  * 后件：如果成功，pnode之后添加一个新节点，item属性为传入的第二个参数
28  */
29 void addNode(List pnode, int item);
30
31 /*
32  * 操作：删除链表中指定的节点
33  * 前件：list是该链表的头结点，pnode是需要删除的节点
34  * 后件：删除链表中的pnode节点
35  */
36 void delNode(List list, List pnode);
37
38 /*
39  * 操作：找到链表中某一节点的后继
40  * 前件：pnode指向链表中的某一个节点
41  * 后件：函数返回pnode的后继，并且跳过头结点
42  */
43 List nextNode(const List pnode);
44
45 /*
46  * 操作：释放链表空间
47  * 前件：plist指向需要释放空间的链表的头结点
48  * 后件：释放plist指向链表的空间，plist重置为空指针
49  */
50 void destroyList(List * plist);
```

2.3 主程序的流程

1. 输入
2. 初始化链表
3. 在链表中依次插入 n 个结点
4. 找到第 x 个结点
5. 循环 $n - 1$ 次：找到当前节点之后的第 $y - 1$ 个结点，删除这个结点
6. 输出
7. 释放空间

2.4 各程序模块之间的层次关系

3 详细设计

3.1 链表的实现

3.2 函数的调用关系图

4 调试分析报告

4.1 调试过程中遇到的问题和解决方案

4.2 设计实现的回顾讨论

4.3 算法复杂度分析

4.4 改进设想的经验和体会

4.4.1 改进 1

在主程序的这一部分：

```
1 for (int i = 1; i <= n; ++i) // 逐个添加元素
2 {
3     addNode(now, i);
4     now = nextNode(now);
5 }
6 now = list;
7 for (int i = 1; i < x; ++i) // 找到第x个元素的前驱
8     now = nextNode(now);
```

可以另用一个指针变量在向链表逐个添加元素的同时记录第 $x - 1$ 个元素的位置，以省去第二个循环。优化后的实现如下：

```
1 List temp = NULL;
2 for (int i = 1; i <= n; ++i)
3 {
4     addNode(now, i);
5     now = nextNode(now);
6     if (i == x - 1) temp = now;
7 }
8 now = temp;
```

4.4.2 改进 2

在主程序的这一部分：

```
1 for (int i = 1; i < n; ++i)
2 {
3     for (int j = 1; j < y; ++j)
4         now = nextNode(now);
```

```
5     delNode(list, now);  
6 }
```

对于有 $n-i+1$ 个元素的环，找到当前元素之后的第 $y-1$ 个元素和找到当前元素之后的第 $(y-1) \bmod (n-i+1)$ 个元素并无区别。优化后的实现如下：

```
1 for (int i = 1; i < n; ++i)  
2 {  
3     for (int j = 1; j <= (y - 1) % (n - i + 1); ++j)  
4         now = nextNode(now);  
5     delNode(list, now);  
6 }
```

当 y 比 n 大的时候对时间复杂度有很可观的优化。

5 用户使用说明

6 测试结果

6.1 测试实例 1