组号: 5

图片包含 游戏机, 画

描述已自动生成

上海大学计算机工程与科学学院

**实 验 报 告**

（数据结构1）

学 期：2024-2025年冬季

组 长： 冯俊佳

学 号： 23122721

指导教师： 朱能军

成绩评定： （教师填写）

二〇二四年十一月二十九日

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **小组信息** | | | | |
| 登记序号 | 姓名 | 学号 | 贡献比 | 签名 |
| 1 | 冯俊佳 | 23122721 | 1/3 | 冯俊佳 |
| 2 | 钱傲栋 | 23122803 | 1/3 | 钱傲栋 |
| 3 | 邱添 | 23122720 | 1/3 | 邱添 |
|  |  |  |  |  |

|  |  |
| --- | --- |
| **实验概述** | |
| 实验零 | （熟悉上机环境、进度安排、评分制度；确定小组成员） |
| 实验一 | 抽取简历 |
| 实验二 |  |
| 实验三 |  |
| 实验四 |  |

实验一

# 1 实验题目

## 1.1抽取简历（1）

## 1.1.1问题描述

某IT公司招聘新员工，已收到N份简历，人力资源部X和Y负责挑选简历安排面试。他们把N份简历排成一个圆圈，按逆时针方向编号为1～N。开始时X站在1号简历前，按逆时针方向数到第K份简历，选中；Y站在N号简历前，按顺时针方向数到第M份简历，选中。两人同时取走所选简历后，分别按逆时针和顺时针走到下一份简历前，然后X和Y再重复上述方法取简历，直到取走全部简历，如果两人选中同一份简历，则只输出一个编号。

## 1.1.2基本要求

要求输入3个数N、K和M，按取走简历的顺序（先甲后已）输出简历编号。

## 1.1.3测试数据

输入样例：

10 4 3

输出样例：

4, 8; 9, 5; 3, 1; 2, 6; 10; 7

## 1.2 抽取简历（2）

## 1.2.1问题描述

某IT公司招聘新员工，已收到N份简历，人力资源部X和Y负责挑选简历安排面试，Z负责补充新的简历。他们把N份简历排成一个圆圈，按逆时针方向编号为1～N。开始时X站在1号简历前，按逆时针方向数到第K份简历，选中；Y站在N号简历前，按顺时针方向数到第M份简历，选中。两人同时取走所选简历后，分别按逆时针和顺时针走到下一份简历前，此时（除非所有简历都已取完或者刚刚X和Y取走的是同一份简历），Z会拿来1份新简历（编号从N+1开始递加），插到X前面，然后X和Y再重复上述方法取简历，直到取走全部简历，如果两人选中同一份简历，则只输出一个编号。

## 1.2.2基本要求

要求输入3个数N、K和M，按取走简历的顺序（先甲后已）输出简历编

## 1.2.3测试数据

输入样例：

5 1 1

输出样例：

1, 5; 6, 4; 7, 3; 8, 2

# 2 实验内容

利用数组或双向循环链表来实现以上问题，以达到熟练使用数组和双向循环链表数据结构的目的，总结规律，深刻理解数组和双向循环链表的特点。

# 3 解决方案

## 3.1算法设计

## 3.1.1 数据结构

本实验要实现的目的是结点的循环，因此首先考虑的是使用数组或者顺序。由于第二问需要实现对该循环的增减，而数组缺少对成员增删的功能，因此考虑使用具有此功能的数据结构——链表。

在对链表的选择中，有带头结点的双向链表和不带头结点的双向链表两种选择，由于题目要求的是实现循环的功能，而如果选择带头结点的话在寻找结点前驱时的处理比较复杂，因此本实验采取不带头结点的双向循环链表作为数据结构。

此外，本实验本着研讨钻研的理念，对第一问也设计了用顺序表来实现。（在3.4中会单独加以说明，后续内容仅讨论用链表实现）

## 3.1.2 算法思想

本代码的思想为：在初始化时，设置两根指针指向链表的头结点和尾部结点（即头结点的前驱），分别代表题目中x与y的初始站位。之后将指针分别顺时针与逆时针移动k与m个结点，此时需要做判断：如果两根指针指向同一个结点，即题目中的x和y走到了同一份简历的面前，输出当前简历的序号并删除该简历；若指向不同的结点，则分别输出指向的结点的序号并删除这两个结点。

## 3.1.3 主要操作

在代码中，主要需要实现的功能有：新增结点（在传入1~N的简历中使用），删除结点（在x与y分别拿走简历时使用），插入结点（在第二问中新放入的简历Z）。

## 3.2 源程序代码

以下简介几个该程序的核心代码，其中包括：新增结点、删除结点以及主函数等。

## 3.2.1 尾插法新增结点

TailInsert函数实现的是传入数据、新增结点，是对于链表的尾插功能，具体应用在初始化创建链表的时候，如下所示。

1. **void** DblLinkList::TailInsert(**int** d)
2. {
3. Node \*p;
4. **if**(head==NULL)  //如果头指针为空，代表链表为空，在头指针的位置新增结点
5. {
6. head=**new** Node(d);
7. head->next=head;
8. head->prior=head;
9. length++;
10. **return**;
11. }
12. p=**new** Node(d);      //否则在尾部新增一个结点
13. length++;
14. head->prior->next=p;
15. p->prior=head->prior;
16. head->prior=p;
17. p->next=head;
18. }
20. cin >> n;         //输入简历的份效
21. **for**(**int** i=1;i<=n;i++) DLL.TailInsert(i); //用简历的编码创建链表

## 3.2.2 在某个位置新增结点

Insert函数实现的是传入数据以及链表中的一个结点，并在该结点之前新增结点的插入功能，具体应用在问题2需要在x前插入新增简历（结点）时使用。具体代码如下所示。

1. **void** DblLinkList::Insert(**int** d,Node \*p)
2. {
3. Node \*q;
4. q=**new** Node(d,p->prior);      //创建新结点，改变指针指向使新结点增加到传入结点的前面
5. p->prior->next=q;
6. q->next=p;
7. p->prior=q;
8. length++;                   //链表长度+1
9. }

## 3.2.3 删除结点

Delete函数实现的是传入链表中的一个结点并删除该结点，同时返回该结点的数据域的功能，在取走简历（消除结点）时使用。具体代码如下所示。

1. **int** DblLinkList::Delete(Node \*p)  //传入结点p，删除该结点
2. {
3. **int** d=p->data;     //将d作为返回值，记录p结点的数据域
4. p->prior->next=p->next;
5. p->next->prior=p->prior;
6. **delete** p;
7. length--;
8. **return** d;
9. }

## 3.2.4 主函数

主函数分为初始化部分和运行部分。

初始化部分如下所示。

1. DblLinkList DLL;
2. cin >> n;         //输入简历的份效
3. **for**(**int** i=1;i<=n;i++) DLL.TailInsert(i);//用简历的编码创建链表
4. Node \*x=DLL.head;           //x指向链表的头指针，即第一份简历1
5. Node \*y=DLL.head->prior;  //y指向链表的尾指针，即最后一份简历N

运行部分如下图代码所示。

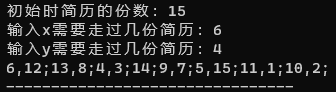
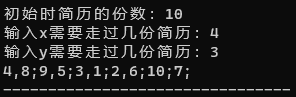
1. **while**(DLL.IsEmpty()==**false**)   //特判:如果链表非空，则继续循环
2. {
3. **for**(**int** i=1;i<k;i++) x=x->next;  //数了k份简历，即移动k个结点，改变x的指向
4. q1=x;
5. x=x->next;
6. **for**(**int** i=1;i<m;i++) y=y->prior; //数了m份简历，即移动m个结点，改变y的指向
7. q2=y;
8. y=y->prior;

由于需要删除结点同时保留x与y的指向，因此本代码使用了两根指针q1和q2来指向需要删除的结点。如果两根指针所指向的结点为同一个结点，那么只需删除该结点并且返回该结点的数据；如果分别指向两个结点，先用q1和q2暂存两个结点，再将x和y分别顺时针、逆时针移动一位，最后做q1和q2指向的结点的删除。需要注意的是，有以下一种特殊情况：如果要x拿走的简历恰好是y移动后面对的简历，那其实y所面对的没有简历，即y应该再逆时针走一步。用代码理解的话，就是y所指向的结点为空，因此y移动到它的前驱结点。X指针同理。代码如下所示。

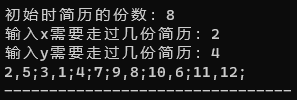
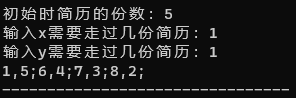
1. **while**(DLL.IsEmpty()==**false**)   //特判:如果链表非空，则继续循环
2. {
3. **for**(**int** i=1;i<k;i++) x=x->next;  //数了k份简历，即移动k个结点，改变x的指向
4. q1=x;
5. x=x->next;
6. **for**(**int** i=1;i<m;i++) y=y->prior; //数了m份简历，即移动m个结点，改变y的指向
7. q2=y;
8. y=y->prior;
10. **if**(q1==q2) cout << DLL.Delete(q1) << " "; //如果指向相同，删除该结点
11. **else**
12. {
13. **if**(y==q1) y=y->prior;
14. **if**(x==q2) x=x->next;
15. cout << DLL.Delete(q1) << " " << DLL.Delete(q2) << " ";
17. **if** (DLL.IsEmpty()) **return** 0;
18. DLL.Insert(++n,x);            //在x前新增结点
19. x=x->prior;                   //x位置移动
20. }
21. }
22. **return** 0;

3.3 运行结果

## 3.3.1 面试安排1的运行结果：



## 3.3.2 面试安排2的运行结果：



经过验算，本程序可以实现预期的运行结果。

3.4 用顺序表实现

在本实验中，我们也尝试设计用顺序表实现相关问题，解决了第一问。但是经过研究后发现用顺序表实现的效果并不理想，具体原因如下：

·顺序表删除元素时会导致顺序表的长度和下标元素发生改变。在本题中体现在抽取简历时有先后顺序，在抽取序号较小的简历后，会对抽取另一份简历产生影响。

·在代码实现时，需要通过取模运算来实现顺序表的循环，即当下标超出顺序表的容量范围时，需要用取模运算使其回到顺序表下一次取简历的正确位置上。

而取模运算需要使用到顺序表当前的长度，而先前删除元素的同时又会改变顺序表的长度。因此，使得顺序表的循环容易出错。

·顺序表的优势未能体现。顺序表在随机读取元素时的时间复杂度为O（1），但是在本题中几乎未使用到随机读取，因此没能凸显出顺序表相比于链表的优势。

具体核心代码如下：

1. // 简历选择逻辑
2. **template** <**class** ElemType>
3. **void** SeqList<ElemType>::Select(**int** n, **int** k, **int** m) {
4. **int** cur\_IndexX = 0;        // X的起始位置（数组索引从0开始）
5. **int** cur\_IndexY = n - 1;    // Y的起始位置
7. **while** (length > 0) {
8. // 计算X的选择位置（逆时针走K步）
9. **if** (cur\_IndexY < 0){
10. cur\_IndexY += length;
11. }
12. cur\_IndexX = (cur\_IndexX + k - 1) % length;
13. ElemType x\_selected = elems[cur\_IndexX];
15. // 计算Y的选择位置（顺时针走M步）
16. cur\_IndexY = (cur\_IndexY - m + 1 + length) % length;
17. ElemType y\_selected = elems[cur\_IndexY];
19. // 输出X和Y的选择结果
20. **if** (cur\_IndexX == cur\_IndexY) {
21. // 如果X和Y选择的是同一份简历
22. cout << x\_selected;
23. DeleteElem(cur\_IndexX + 1, x\_selected); //删除该简历
24. }
25. **else** {
26. // 如果X和Y选择不同简历
27. cout << x\_selected << ", " << y\_selected;
28. **if** (cur\_IndexX < cur\_IndexY) {
29. DeleteElem(cur\_IndexY + 1, y\_selected); //先删除Y的位置
30. DeleteElem(cur\_IndexX + 1, x\_selected); //再删除X的位置
31. }
32. **else** {
33. DeleteElem(cur\_IndexX + 1, x\_selected); //先删除X的位置
34. DeleteElem(cur\_IndexY + 1, y\_selected); //再删除Y的位置
35. }
36. }
38. **if** (length > 0) {
39. cout << "; ";  // 输出分隔符
40. }
42. // 删除后需要调整X和Y的索引位置，对应题目中的“走到下一份简历前”
43. // 若X的当前索引等于Y的当前索引，则将Y的索引往前一位。
44. **if** (cur\_IndexX == cur\_IndexY) {
45. cur\_IndexY--;
46. **goto** flag;
47. }
48. // 若X的当前索引小于Y的当前索引，则将Y的索引往前两位。
49. **if** (cur\_IndexX < cur\_IndexY) {
50. cur\_IndexY-=2;
51. **goto** flag;
52. }
53. // 若X的当前索引大于Y的当前索引，则将X和Y的索引各往前一位。
54. **if** (cur\_IndexX > cur\_IndexY) {
55. cur\_IndexX--;
56. cur\_IndexY--;
57. **goto** flag;
58. }
59. flag:;
60. }
61. cout << endl;
62. }

# 4 算法分析

## 4.1 算法时间复杂度分析

主循环中的操作：主函数的循环体首先执行若干次x=x->next和y=y->prior，这些操作每次的时间复杂度是O(1)；然后，检查q1和q2是否相等，如果相等调用Delete(q1)，否则调用Delete(q1)和Delete(q2)两次，每次删除的时间复杂度是O(n)（因为Delete(int n)最坏情况下需要遍历n个节点）；如果q1!= q2，还需要调用Insert(++n, x)插入新节点，时间复杂度是O(1)。

因此，总时间复杂度是：最坏情况下，每次删除操作的复杂度为 O(n)，循环最多执行n次，因此总时间复杂度是O(n2)。

## 4.2算法空间复杂度的分析

每个Node结点包含了三个字段：prior、next和data，每个节点占用O(1)的空间；DblLinkList类的空间消耗：head指针占用O(1)的空间，length变量占用O(1)的空间；主程序中的空间消耗：在主函数中，创建了一个DblLinkList对象，空间消耗为O(n)，其中n是链表的长度，同时使用了两个额外的指针x和y，占用O(1)的空间。

因此，总空间复杂度是 O(n)，主要由链表中存储的节点数量决定。

5 总结与心得

本实验在写代码时，由于在部分位置没有考虑特殊情况导致了结果出错例如运行超时、循环未退出等等。因此，需要考虑到特殊情况，以增加代码的鲁棒性。

在设计顺序表的实现是，对于如果进行取模运算，以及索引如何改变进行了不断地研究、尝试和改进，最终完善了顺序表的代码。但在本题中，用顺序表实现第二问颇具难度，在尝试后仍然无法实现，因此只设计了第一问的解决方案。