同濟大學
TONGJI UNIVERSITY

项目说明文档

两个有序链表序列的交集

指导老师：张颖

1851009 沈益立

目录

目录
1.分析
1.1 背景分析
1.2 功能分析
2.设计
2.1 数据结构设计
2.2 类结构设计
2.3 成员和操作设计
2.4 系统设计
3.实现
3.1 查找交集功能的实现
3.1.1 查找交集功能流程图
3.1.2查找交集功能核心代码实现
4.测试
4.1 功能测试
4.1.1 一般情况测试
4.1.2 交集为空的情况
4.1.3 完全相交的情况
4.1.4 其中一个序列完全属于交集的情况
4.1.5 其中一个序列为空的情况
4.1.6 两个序列都为空的情况
4.2 错误测试
4.2.1 输入降序排列的情况
4.2.2 输入负数作为节点的情况

1.分析

1.1 背景分析

链表是一种较为基础的数据结构，能够正确有效的创造、遍历、比较链表是课程的基本要求，为此 需要解决本次两有序链表的合成、交集问题。

1.2 功能分析

本次功能设计的输入分为两行，以-1作为每行链表截止的标志，并且规定输入为非降序排列，用户仅需逐个地输入链表的节点值便可完成输入操作。对于输出，要求空格揭开并且结尾不能有多余的空格，此外，应考虑到链表为空的特殊情况。

2.设计

2.1 数据结构设计

以上功能题目要求使用链表来实现，因此定义存储结构为链表。按照经验，在链表之前添加一个头结点（哨兵节点）能便于处理链表的CRUD操作。

2.2 类结构设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（LNode）与链表类 （LinkedList），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。本程序较为简单，没有采用嵌套、继承等方法，而是开辟了两个类：链表节点类LNode和链表类LinkedList，并且将

LinkedList类中的头结点、尾结点设为LNode类。

由题意，本程序需要比较两数是否相等等操作，蕴含了该程序比较的是整型链表，故默认类成员变量为 int 类型而没有使用template。

2.3 成员和操作设计

链表节点类 LNode：

公有操作：

```
LNode() {};           //默认的构造函数
LNode(int val) :      //含一个赋值参数的构造函数
        m_val(val) {};
```

公有成员：

```
int m_val;            //存储的数字
LNode* m_next = nullptr;//下一个指针域
```

链表类 LinkedList：

公有操作：

```
LinkedList();//默认初始化构造函数
void pushBackNode(int val);//从尾部插入一个节点
bool createList();//通过IO来创建链表，若创建失败则返回假
void findIntersectionWith(LinkedList B);//寻找this与链表B的
交集
```

私有成员

```
LNode* m_head;//头指针节点
LNode* m_tail;//尾指针节点
```

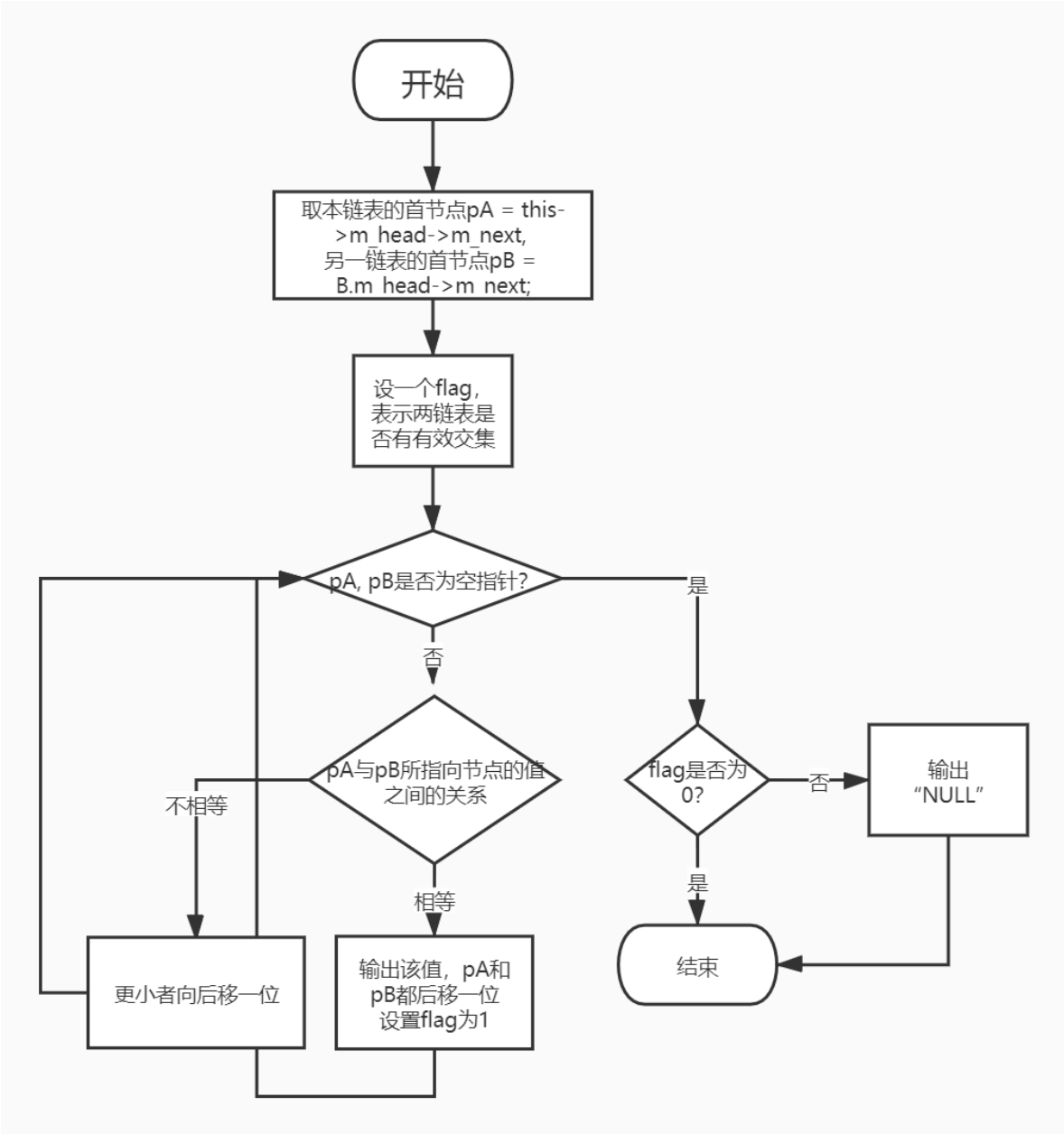
2.4 系统设计

程序运行后，系统会自动创建两个链表A, B。之后分别调用两个链表的`createList()`函数来对其进行初始化赋值，之后通过`A.findIntersectionWith(B)`函数来找到对应相同的值并且输出到屏幕上。

3.实现

3.1 查找交集功能的实现

3.1.1 查找交集功能流程图



3.1.2查找交集功能核心代码实现

```
void LinkedList::findIntersectionWith(LinkedList B)
```

```
{
    LNode* pA = this->m_head->m_next, *pB = B.m_head-
>m_next;
    bool flag = 0;
    while (pA && pB)
    {
        if (pA->m_val == pB->m_val)
        {
            if (flag == 1) {
                cout << ' ';
            }
            cout << pA->m_val;
            flag = 1;
            pA = pA->m_next, pB = pB->m_next;
        }
        else if (pA->m_val < pB->m_val)
        {
            pA = pA->m_next;
        }
        else
        {
            pB = pB->m_next;
        }
    }
    if (!flag)  //空集检验
    {
        cout << "NULL" ;
    }
    cout << endl;
}
```

4.测试

4.1 功能测试

4.1.1 一般情况测试

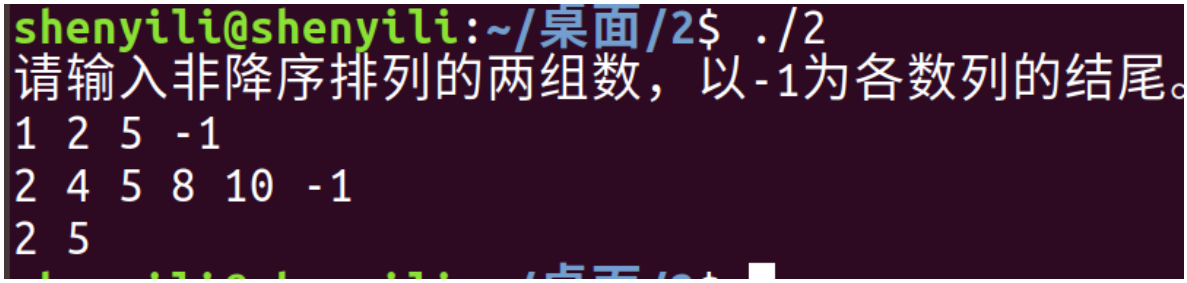
测试用例：

```
1 2 5 -1
2 4 5 8 10 -1
```

预期结果：

```
2 5
```

实验结果：



4.1.2 交集为空的情况

测试用例：

```
1 3 5 -1

2 4 6 8 10 -1
```

预期结果：

```
NULL（无交集）
```

实验结果：

```
shenyili@shenyili:~/桌面/2$ ./2
请输入非降序排列的两组数，以-1为各数列的结尾。
1 3 5 -1

2 4 6 8 10 -1
NULL
```

4.1.3 完全相交的情况

测试用例：

```
1 2 3 4 5 -1

1 2 3 4 5 -1
```

预期结果：

```
1 2 3 4 5
```

实验结果：

```
shenyili@shenyili:~/桌面/2$ ./2
请输入非降序排列的两组数，以-1为各数列的结尾。
1 2 3 4 5 -1

1 2 3 4 5 -1
1 2 3 4 5
```

4.1.4 其中一个序列完全属于交集的情况

测试用例：

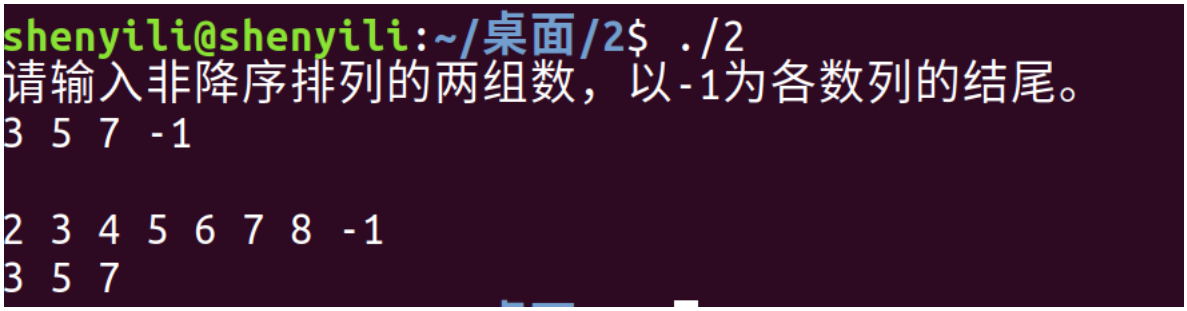
```
3 5 7 -1

2 3 4 5 6 7 8 -1
```

预期结果：

```
3 5 7
```

实验结果：



4.1.5 其中一个序列为空的情况

测试用例：

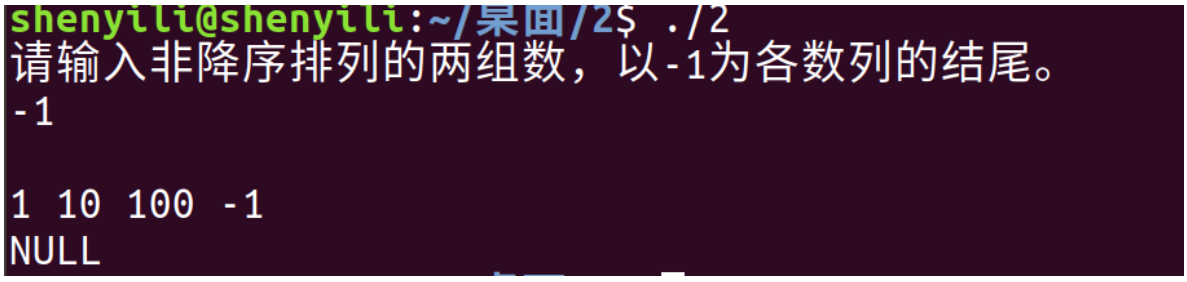
```
-1

1 10 100 -1
```

预期结果：

```
NULL（无交集）
```

实验结果：



4.1.6 两个序列都为空的情况

测试用例：

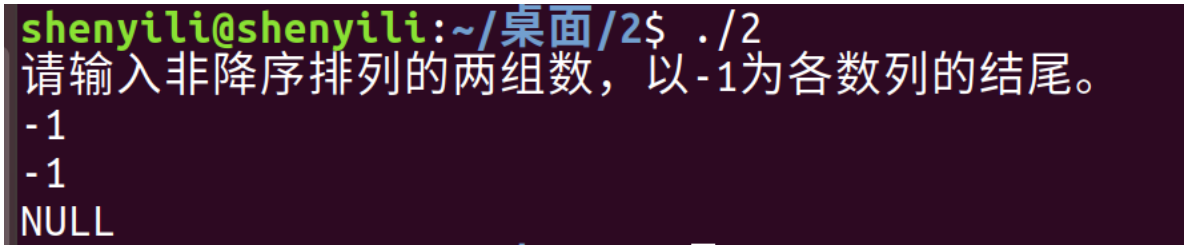
```
-1

-1
```

预期结果：

```
NULL（无交集）
```

实验结果：



4.2 错误测试

4.2.1 输入降序排列的情况

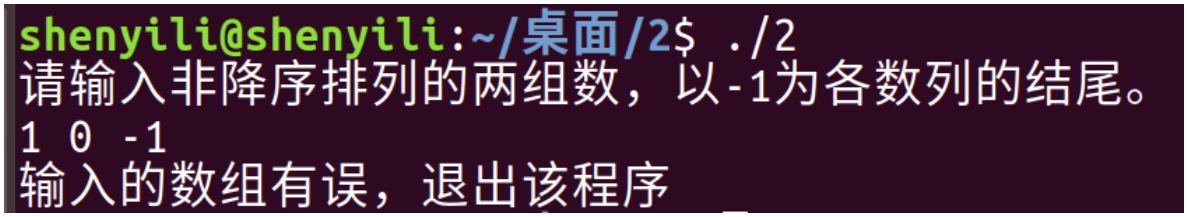
测试用例：

```
1 0 -1
```

预期结果：

```
提示数组输入有误
```

实验结果：



4.2.2 输入负数作为节点的情况

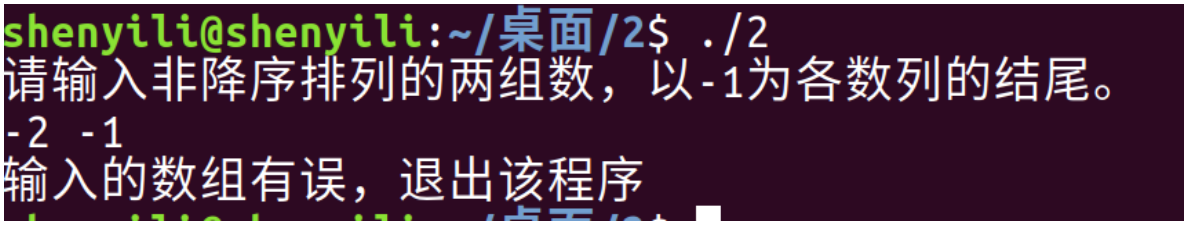
测试用例：

```
-2 -1
```

预期结果：

```
提示数组输入有误
```

实验结果：


shenyili@shenyili:~/桌面/2$ ./2
请输入非降序排列的两组数，以-1为各数列的结尾。
-2 -1
输入的数组有误，退出该程序