数据结构课程设计项目说明文档

——两个有序链表序列的交集

作 者 姓 名： 马威

学 号： 2151294

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

****

目 录

[1 项目分析 1](#_Toc495668153)

[1.1 项目要求 1](#_Toc495668154)

[1.2 项目需求分析 1](#_Toc495668155)

[2 项目设计 2](#_Toc495668156)

[2.1 数据结构设计 2](#_Toc495668157)

[2.2 类设计 2](#_Toc495668158)

2.2.1 结点类（SetNode） 3

2.2.2 链式集合类（LinkedNode） 3

2.3 算法设计 4

2.3.1 算法思路 4

2.3.2 性能评估 5

2.3.3 流程图表示 5

2.3.4 代码实现 5

[3 项目测试 7](#_Toc495668161)

[3.1 一般情况 7](#_Toc495668162)

[3.2 交集为空的情况 7](#_Toc495668166)

[3.3 完全相交的情况 7](#_Toc495668170)

[3.4 其中一个序列完全属于交集的情况 8](#_Toc495668174)

[3.5 其中一个序列为空的情况 8](#_Toc495668178)

[3.6 两个序列均为空的情况 8](#_Toc495668182)

[3.7 其中一个序列输入不为升序的情况 9](#_Toc495668182)

[3.8 两个序列输入均不为升序的情况 9](#_Toc495668182)

**1.项目分析**

1.1 项目要求

已知两个非降序链表序列S1和S2，设计函数构造出S1和S2的交集新链表S3。

输入要求：输入分2行，分别在每行给出由若干个正整数构成的非降序序列，用-1表示序列的结尾（-1不属于这个序列）。数字用空格间隔。

输出说明：在一行中输出两个输入序列的交集序列，数字间用空格分开，结尾不能有多余空格；若新链表为空，输出NULL。

测试用例：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入 | 输出 | 说明 |
| 1 | 1 2 5 -1  2 4 5 8 10 -1 | 2 5 | 一般情况 |
| 2 | 1 3 5 -1  2 4 6 8 10 -1 | NULL | 交集为空的情况 |
| 3 | 1 2 3 4 5 -1  1 2 3 4 5 -1 | 1 2 3 4 5 | 完全相交的情况 |
| 4 | 3 5 7 -1  2 3 4 5 6 7 8 -1 | 3 5 7 | 其中一个序列完全属于交集的情况 |
| 5 | -1  10 100 1000 -1 | NULL | 其中一个序列为空的情况 |

1.2 项目需求分析

对于求两个有序链表序列的交集的程序，需考虑以下需求：

**·正确性**

程序应当能够应付各类序列情况，如S1、S2存在空列，S1、S2长度不相同，互相包含等情况

**·健壮性**

程序应当对序列元素输入过程中的错误做出相应处理

**·功能提升**

程序可以处理输入序列不为升序的两个序列并求出交集

**2.项目设计**

2.1 数据结构设计

由项目分析可以得出，该项目需要实现一个链表的数据结构，并在此基础上实现求解两序列交集的算法。由于两序列存在有交集、无交集等各类情况，非常类似数学上的集合，故在本程序中以单链表的形式封装了一个集合类，每个对象表示一个序列（即一个集合），完成所需的功能。

2.2 类设计

集合类本质上仍是一种链表，其包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

为了实现代码的复用性，本系统实现了一个链表为形式的集合。采用struct描述链表结点类（SetNode），这样使得链式集合类（LinkedSet）可以直接访问链表结点而不需要定义友元关系。本系统实现的链表结构各种操作的时间复杂度如下：

插入操作：O(n)

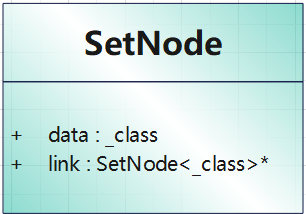
删除操作：O(n)

查询操作：O(n)

遍历操作：O(n)

**2.2.1 结点类（SetNode）**

链表结点存储了结点数据、后继节点位置，其UML图如下：



**2.2.2链式集合类（LinkedSet）**

链式集合类本质上仍是单链表，在保存了链表的基本结构的基础上，对于插入等操作进行了微调，使得集合内的元素以一定的顺序进行排列（一般是逻辑上的升序），以期让求解集合的交、并、差等操作的时间复杂度更低，代码实现更有逻辑性。当然，我们在使用集合时并不关心某个元素在集合中的具体位置，所以插入和删除操作并不像一般链表操作那样提供位置，而是仅仅需要元素关键码即可自动找到对应的位置并进行插入或删除。

为了使得链表中各种操作的时间复杂度都尽可能低，因此这里选择了带头结点和尾结点的单链表来实现链表中的各种操作。也即：选择了牺牲空间来达到降低时间复杂度的效果

其UML图如下：



其中，主要函数如下：

//将集合置空

void makeEmpty();

//判断集合是否为空集

bool IsEmpty()const;

//判断集合是否已满

bool IsFull()const;

//用输入方式给当前集合赋值

void Input(const \_class endTag);

//输出当前集合中的所有元素

void Output()const;

//把新元素x加入到集合之中。若集合中已有此元素，则加入失败，返回false，否则返回true

bool addMember(const \_class x);

//把集合中成员x删去。若集合不空且元素x在集合中，删除成功，返回true，否则返回false

bool delMember(const \_class x);

//求当前集合与LS的交

LinkedSet<\_class> operator\*(const LinkedSet<\_class>& LS);

2.3 算法设计

**2.3.1 算法思路**

初始化结果链表为空，分别定义两个链表上的指针1和2，分别指向附加头结点的后继地址。

当两个指针均不为空时进行循环：比较指针1和指针2所指向的元素，有如下五种情况：

（1）若两个元素相等，则将该元素插入到新链表末端，两个指针指向后继地址；

（2）指针1指向元素较大，则指针2指向当前结点的后继地址。

（3）指针2指向元素较大，则指针1指向当前结点的后继地址。

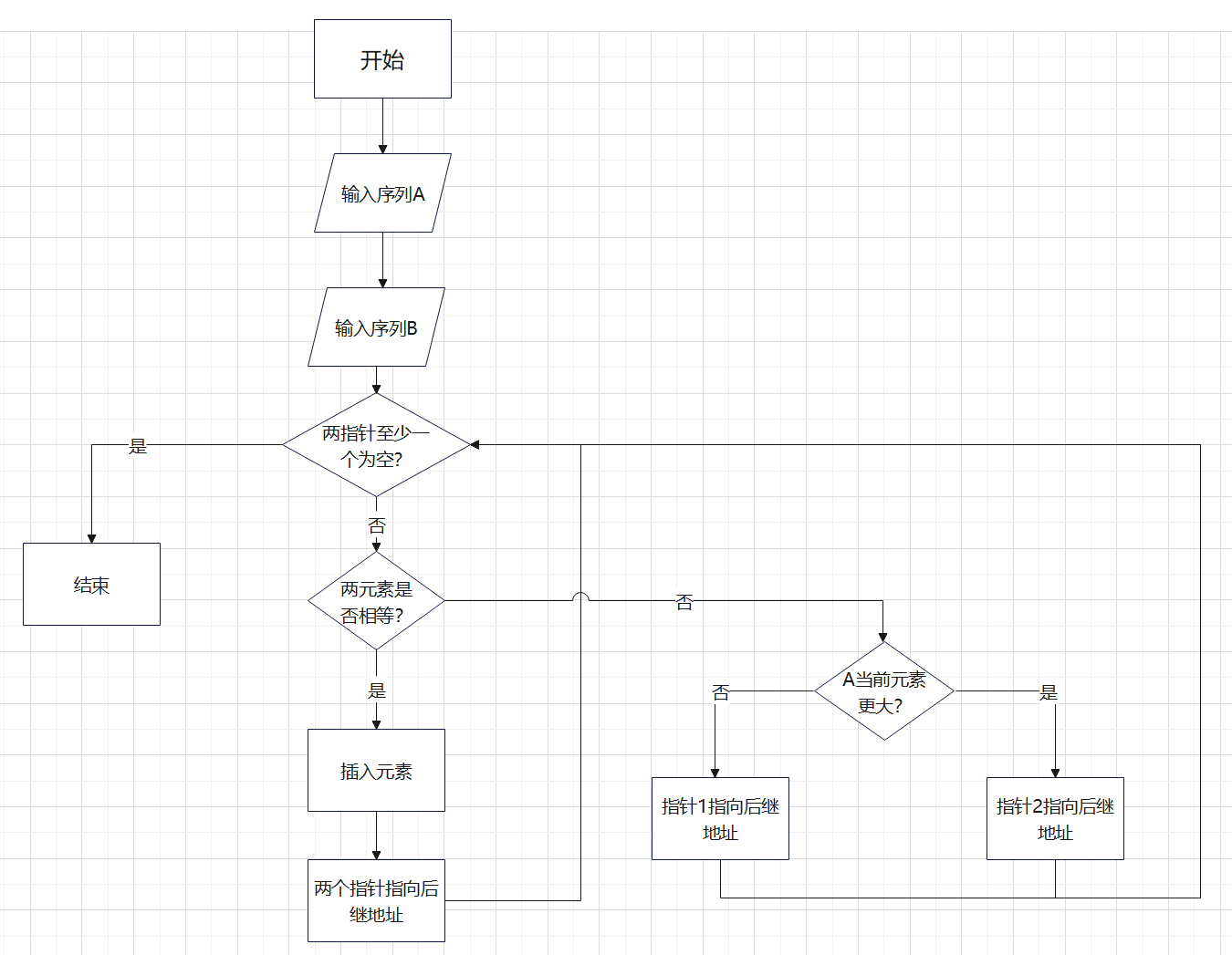
（4）指针1为空，则说明S2剩余部分一定与S1无交集，求解结束

（5）指针2为空，则说明S1剩余部分一定与S2无交集，求解结束

**2.3.2 性能评估**

设两个链表长度为m、n，两个链表在最坏情况下都要扫描一遍，所需时间为O(m+n)；元素插入结果链表需要O(min{m,n})，综上，总时间复杂度为O(m+n)。

**2.3.3 流程图表示**

****

**2.3.4 代码实现**

template<class \_class> LinkedSet<\_class> LinkedSet<\_class>::operator\*(const LinkedSet<\_class>& LS)

{

    SetNode<\_class>\* pa = first->link, \* pb = LS.first->link;

    LinkedSet<\_class> temp;

    SetNode<\_class>\* pc = temp.first;

    while (pa != NULL && pb != NULL) {

        if (pa->data == pb->data) {

            SetNode<\_class>\* newNode = new SetNode<\_class>(pa->data);

            if (newNode == NULL) {

                cerr << "存储分配失败！" << endl;

                exit(1);

            }

            pc->link = newNode;

            pc = pc->link;

            pa = pa->link;

            pb = pb->link;

        }

        else if (pa->data < pb->data)

            pa = pa->link;

        else

            pb = pb->link;

    }

    pc->link = NULL;

    temp.last = pc;

    return temp;

}

**3.项目测试**

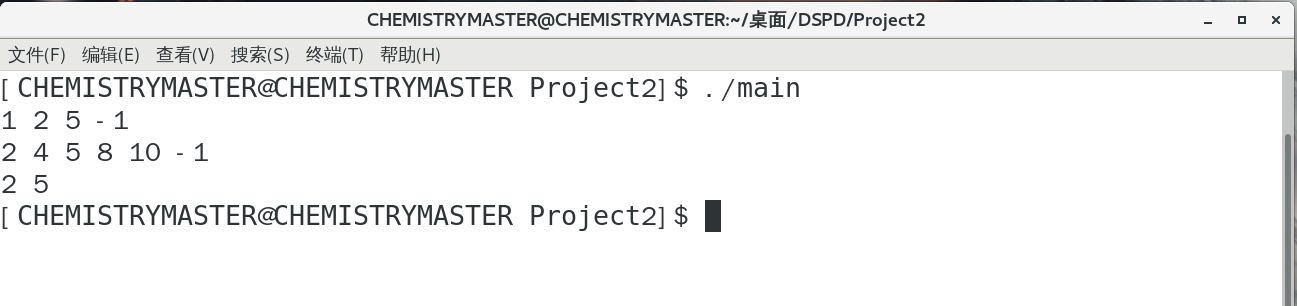
3.1 一般情况

输入内容：1 2 5 -1

2 4 5 8 10 -1

预期结果：2 5

测试结果：



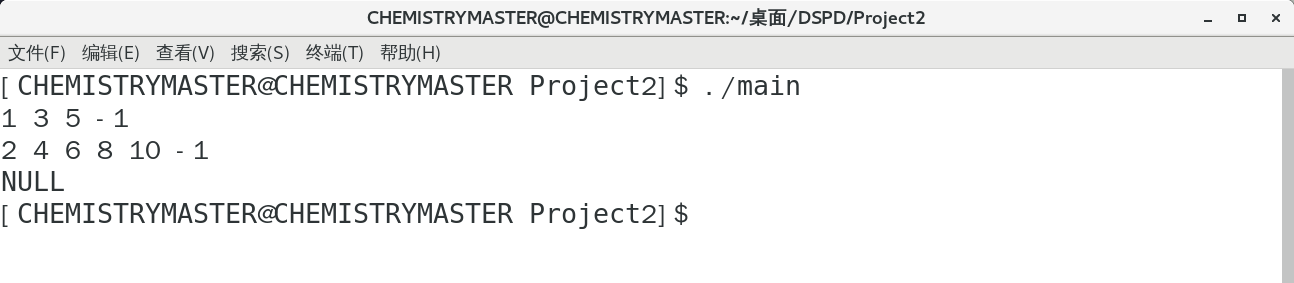
3.2交集为空的情况

输入内容：1 3 5 -1

2 4 6 8 10 -1

预期结果：NULL

测试结果：



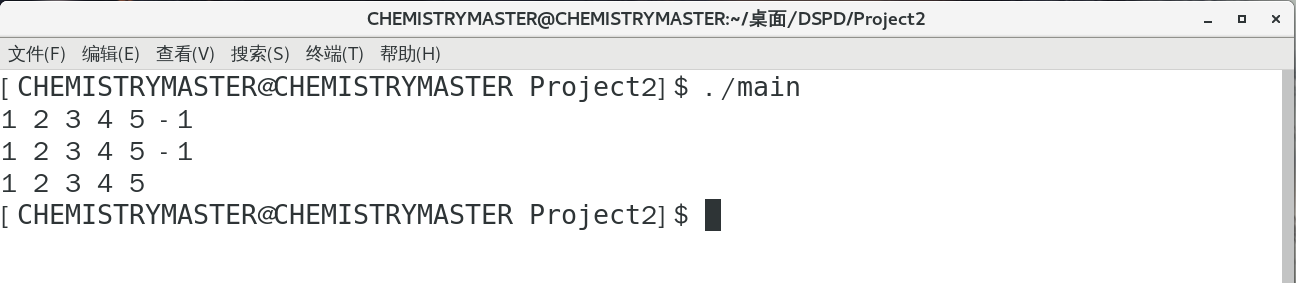
3.3 完全相交的情况

输入内容：1 2 3 4 5 -1

1 2 3 4 5 -1

预期结果：1 2 3 4 5

测试结果：



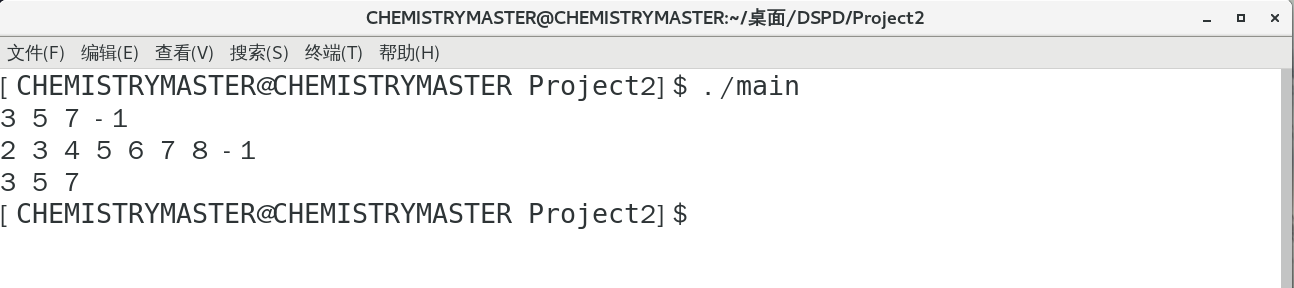
3.4 其中一个序列完全属于交集的情况

输入内容：3 5 7 -1

2 3 4 5 6 7 8 -1

预期结果：3 5 7

测试结果：



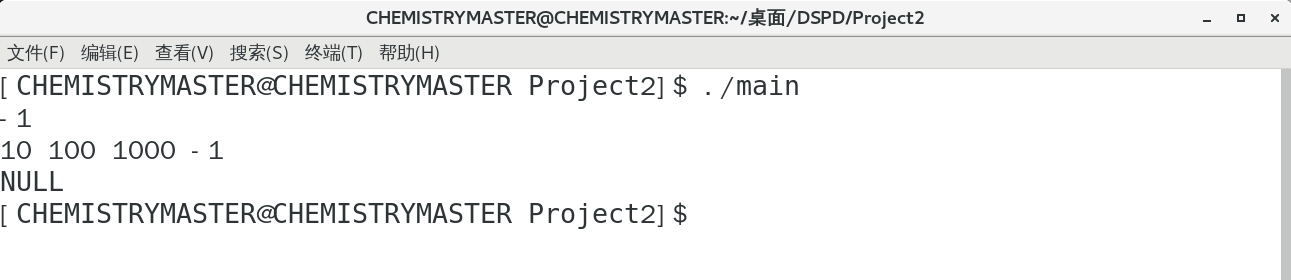
3.5 其中一个序列为空的情况

输入内容：-1

10 100 1000 -1

预期结果：NULL

测试结果：



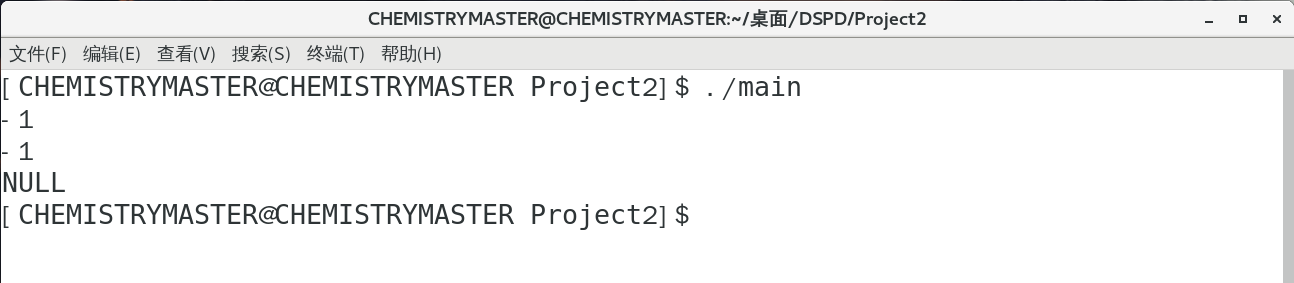
3.6 两个序列均为空的情况

输入内容：-1

-1

预期结果：NULL

测试结果：



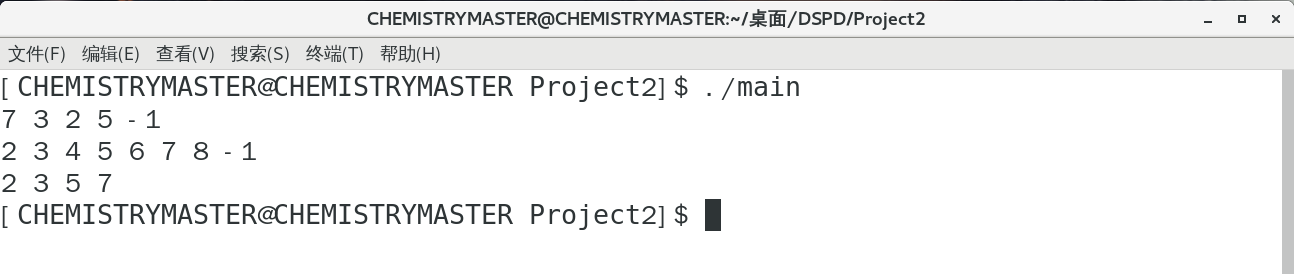
3.7 其中一个序列输入不为升序的情况

输入内容：7 3 2 5 -1

2 3 4 5 6 7 8 -1

预期结果：2 3 5 7

测试结果：



3.8 两个序列输入均不为升序的情况

输入内容：9 2 6 5 4 -1

1 7 4 5 2 3 -1

预期结果：2 4 5

测试结果：

