项目说明文档

数据结构课程设计

——两个有序链表的交集

作 者 姓 名： 郑柯凡

学 号： 1950072

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目录

[1 分析 2](#_Toc5983)

[1.1背景分析 2](#_Toc1208)

[1.2功能分析 2](#_Toc20054)

[2 设计 3](#_Toc28398)

[2.1思路设计 3](#_Toc15742)

[2.2数据结构的选择 3](#_Toc1717)

[2.3类的设计 3](#_Toc31374)

[3 功能实现 4](#_Toc9719)

[3.1 输入链表元素功能的实现 4](#_Toc28303)

[3.1.1输入链表元素功能流程图 4](#_Toc21821)

[3.1.2输入功能核心代码 5](#_Toc15617)

[3.1.3输入功能截屏示例 5](#_Toc23224)

[3.2 删除链表结点功能的实现 5](#_Toc18749)

[3.2.1删除链表结点功能流程图 5](#_Toc4767)

[3.2.2删除链表结点功能核心代码 5](#_Toc29653)

[3.2.3删除链表结点功能截屏示例 6](#_Toc25487)

[3.3 输出链表元素功能的实现 6](#_Toc14334)

[3.3.1输出链表元素功能流程图 6](#_Toc30410)

[3.3.2输出链表元素功能核心代码 6](#_Toc20812)

[3.3.3输出链表元素功能截屏示例 7](#_Toc23430)

[3.4 求交集函数功能的实现 7](#_Toc5190)

[3.4.1求交集函数功能流程图 7](#_Toc13745)

[3.4.2求交集函数功能核心代码 7](#_Toc14489)

[3.4.3求交集函数功能截屏示例 8](#_Toc23508)

[4测试 8](#_Toc2555)

[4.1功能测试 8](#_Toc9460)

[4.1.1输入功能测试 8](#_Toc23795)

[4.1.2删除功能测试 9](#_Toc28327)

[4.1.3输出功能测试 9](#_Toc20264)

[4.1.4求交功能测试 9](#_Toc25788)

[4.2不同数据类型测试 9](#_Toc22247)

[4.2.1整型数据求交测试 9](#_Toc7968)

[4.2.2浮点型数据求交测试 10](#_Toc1013)

[4.2.3字符型数据求交测试 10](#_Toc4070)

[4.3边界测试 10](#_Toc9202)

[4.3.1两个链表都为空或其中一个为空 10](#_Toc19103)

[4.3.2交集为空 10](#_Toc20183)

[4.3.3某个链表全是交集 10](#_Toc14181)

**1 分析**

**1.1背景分析**

生活中，我们经常会遇到求两个事物共同点的问题。其实，此类问题都可以抽象成为求两个集合交集元素的模型。对于元素数量众多的集合而言，显然人眼观察法已不再适用，因此我们可以借助计算机的计算能力来帮助我们求解交集。

**1.2功能分析**

我们需要对集合中的元素进行比较，删除，添加等功能。

**2 设计**

**2.1思路设计**

首先确定两个已有的有序集合A﹑B，我们需要求它们的交集，并输出一个新的集合。此题没有对程序运行结束后原有的两个集合做要求，因此我们可以借助其中一个元素数量较少的集合作为基准，假设是A（因为新生成的集合元素数量必定小于等于两个原集合大小的最小值），避免不必要的空间浪费。

由于集合是有序的，我们可以对元素进行顺序处理：若A的当前元素小于B的当前元素，说明B中无该元素，将该元素从A中删除并处理A下一个元素；若A当前元素等于B当前元素，说明是交集中的元素，保留并分别指向A和B的下一个元素；若A当前元素大于B当前元素，A不做处理，B指向其下一个元素。若A先处理完元素，而B未处理完，显然B中剩余不是交集元素，A不做处理直接输出；若B先处理完元素，而A有剩余，显然A中剩余不是交集元素，将其删除再输出；若AB同时处理完，直接输出A。

**2.2数据结构的选择**

由于需要频繁地对集合进行删除操作，因此采用删除较为方便的双向链表来实现。并为其添加一个表头结点，便于删除时的代码统一。

同时各种数据类型的链表求交集方法都大同小异，因此采用类模板方式来设计链表，减少重复劳动，提高效率。

**2.3类的设计**

此项目中只需要使用链表，因此使用传统的链表结构，一个结点类和一个链表类。

结点类的私有成员包括一个未定数据类型的数据成员，用于存储有序链表中的元素。以及两个指针，一个指向前一个结点的地址，一个指向后一个结点的地址，便于删除结点时链接其两边的结点。同时在结点类中也声明了链表类是它的友元类，便于在链表类中直接操作结点的私有成员。

1. **template**<**class** Type>
2. //存放数字和前后两个节点地址的节点类
3. **class** Node {
4. **friend** **class** Linklist<Type>;
5. **private**:
6. Type num;
7. Node<Type>\* next;
8. Node<Type>\* last;
9. **public**:
10. Node<Type>() {  next = NULL; last = NULL; };
11. //析构函数，自动释放空间
12. ~Node() { next = NULL;last = NULL;};
13. };

然后是链表类，用于以两个有序链表的形式存储两个集合。私有成员包括链表长度（即链表中元素的个数），以及指向链表头结点的指针。公有成员包括构造析构函数，以及其他一些功能函数：

Inport()输入函数，用于输入链表中的元素，完成有序链表的初始化；

Display()打印函数，用于将链表中的的元素从头到尾输出一遍；

Remove()删除函数，用于删除链表中的某个结点；

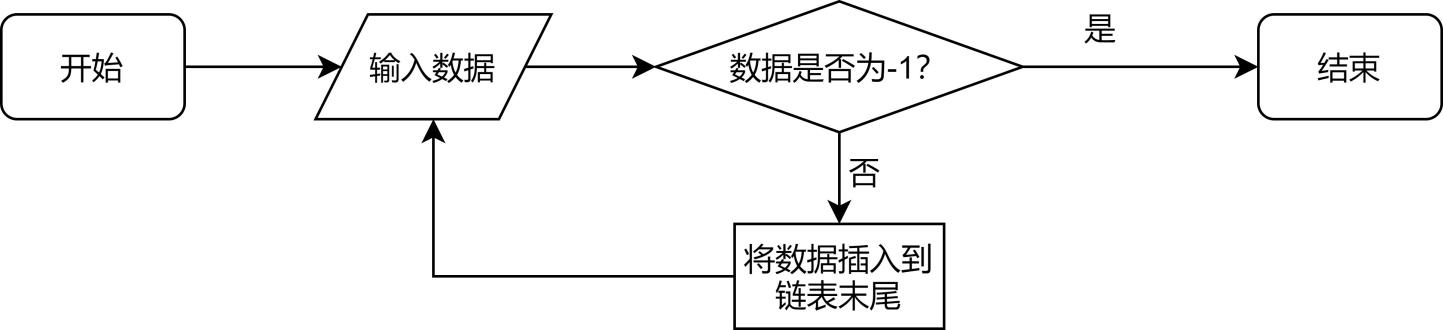
SeekIntersection()求交函数，用于构造含S1和S2交集元素的新链表。

1. **template**<**class** Type>
2. //链表类
3. **class** Linklist {
4. **private**:
5. **int** size;//链表长度
6. Node<Type>\* head;//链表头地址
7. **public**:
8. //构造函数(存在表头结点)
9. Linklist();
10. //析构函数，自动释放空间
11. ~Linklist();
12. //输入函数
13. **void** Inport();
14. //打印函数。输出链表的各个元素
15. **void** Display();
16. //删除函数，删除链表中的某个节点
17. **void** Remove(Node<Type>\*Iterator);
18. //求交集函数，寻找两链表相同的元素
19. **void** SeekIntersection(Linklist<Type>& S2);
20. };

**3 功能实现**

**3.1 输入链表元素功能的实现**

**3.1.1输入链表元素功能流程图**



**3.1.2输入功能核心代码**

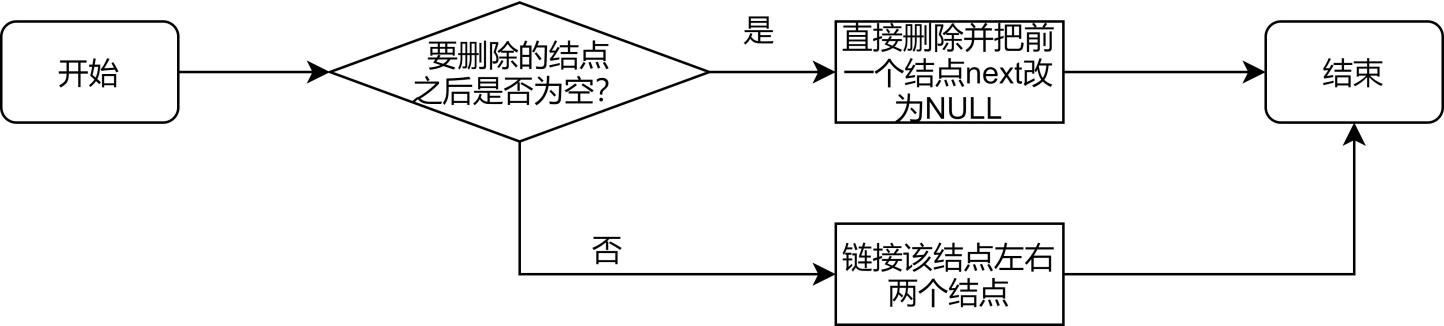
1. **template**<**class** Type>
2. **void** Linklist<Type>::Inport()
3. {
4. cout << "请输入序列元素，以-1结束:";
5. Node<Type>\* current = **this**->head;
6. Type innum;
7. **while** (cin>>innum)
8. {
9. **if** (innum == -1|| innum=='-')
10. {
11. **break**;
12. }
13. //开辟空间
14. current->next = **new** Node<Type>;
15. current->next->num = innum;
16. current->next->last = current;
17. current->next->next = NULL;
18. current = current->next;
19. **this**->size++;
20. }
21. }

**3.1.3输入功能截屏示例**



**3.2** 删除链表结点功能的实现

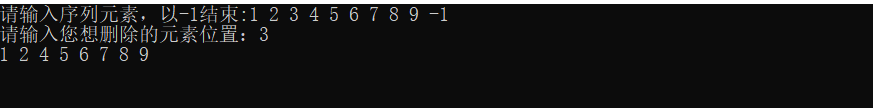
**3.2.**1删除链表结点功能流程图



**3.2.2删除链表结点功能核心代码**

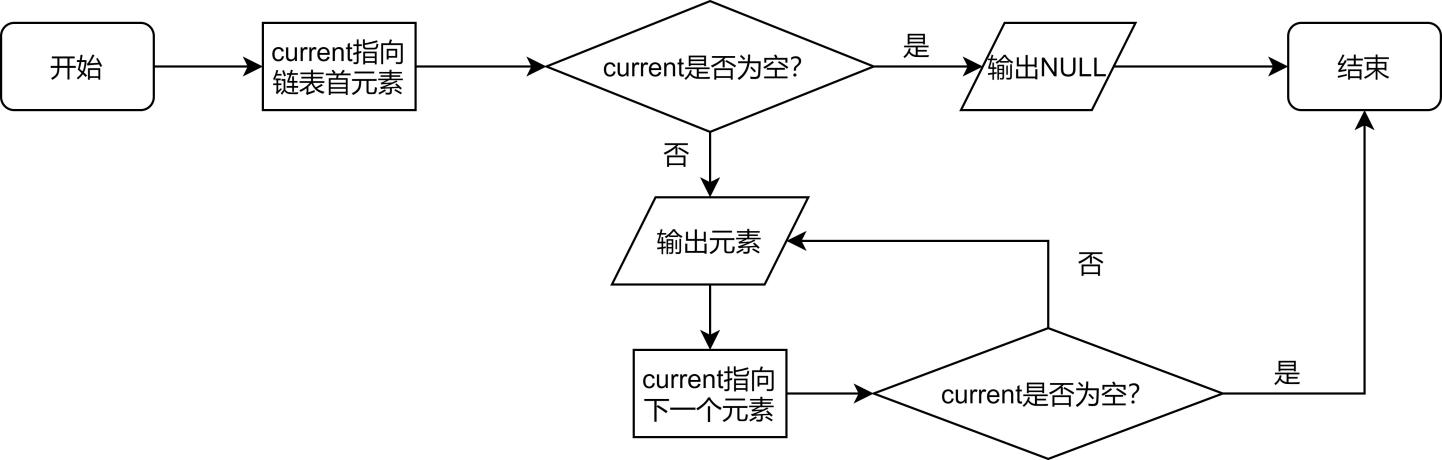
1. **template**<**class** Type>
2. **void** Linklist<Type>::Remove(Node<Type>\*Iterator)
3. {
4. **if** (Iterator->next == NULL)
5. {
6. Node<Type>\* temp = Iterator;
7. Iterator->last->next = NULL;
8. **delete** temp;
9. }
10. **else** **if** (Iterator->next != NULL)
11. {
12. Node<Type>\* temp = Iterator;
13. Iterator->last->next = Iterator->next;
14. Iterator->next->last = Iterator->last;
15. **delete** temp;
16. }
17. **this**->size--;
18. }

**3.2.3删除链表结点功能截屏示例**



**3.3 输出链表元素功能的实现**

**3.3.1输出链表元素功能流程图**

****

**3.3.2输出链表元素功能核心代码**

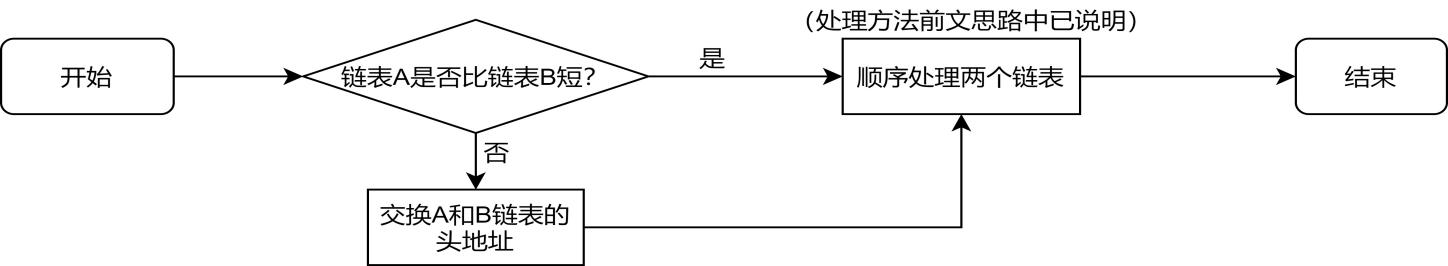
1. **template**<**class** Type>
2. **void** Linklist<Type>::Display()
3. {
4. cout << "相同的序列元素为:";
5. Node<Type>\* current = **this**->head->next;
6. **if** (current == NULL)
7. {
8. cout << "NULL" << endl;
9. }
10. **else** **if** (current != NULL)
11. {
12. **while** (current != NULL)
13. {
14. cout << current->num << " ";
15. current = current->next;
16. }
17. cout << endl;
18. }
19. }

**3.3.3输出链表元素功能截屏示例**



**3.4 求交集函数功能的实现**

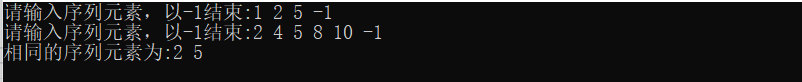
**3.4.1求交集函数功能流程图**

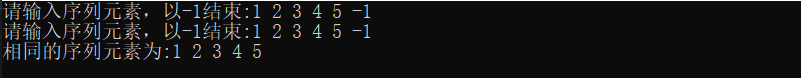
****

**3.4.2求交集函数功能核心代码**

1. **template**<**class** Type>
2. **void** Linklist<Type>::SeekIntersection(Linklist& S2)
3. {
4. Node<Type>\* Iterator1 = **this**->head;
5. Node<Type>\* Iterator2 = S2.head;
6. //以长度较短的链表作为S3的基链表
7. **if** (**this**->size > S2.size)
8. {
9. //交换头指针
10. **this**->head = Iterator2;
11. S2.head = Iterator1;
12. //交换链表长度
13. **int** tempsize = **this**->size;
14. **this**->size = S2.size;
15. S2.size = tempsize;
16. Node<Type>\* tempIterator = Iterator1->next;
17. Iterator1 = Iterator2->next;
18. Iterator2 = tempIterator;
19. }
20. **else** **if** (**this**->size <= S2.size)
21. {
22. Iterator1 = Iterator1->next;
23. Iterator2 = Iterator2->next;
24. }
25. **while** (Iterator1 != NULL && Iterator2 != NULL)
26. {
27. **if** (Iterator1->num < Iterator2->num)
28. {
29. Node<Type>\* temp = Iterator1;
30. Iterator1 = Iterator1->next;
31. **this**->Remove(temp);
32. }
33. **else** **if** (Iterator1->num > Iterator2->num)
34. {
35. Iterator2 = Iterator2->next;
36. }
37. **else** **if** (Iterator1->num == Iterator2->num && Iterator1->num != -1)
38. {
39. Iterator1 = Iterator1->next;
40. Iterator2 = Iterator2->next;
41. }
42. **else** **if** (Iterator1->num == Iterator2->num && Iterator1->num == -1)
43. {
44. **break**;
45. }
46. }
47. **while** (Iterator1 != NULL)
48. {
49. Node<Type>\* temp = Iterator1;
50. Iterator1 = Iterator1->next;
51. **this**->Remove(temp);
52. }
53. }

**3.4.3求交集函数功能截屏示例**





**4测试**

**4.1功能测试**

**4.1.1输入功能测试**

**测试用例：**1 2 3 4 5 6 7 -1

**预期结果：**1 2 3 4 5 6 7 -1

**实验结果：**

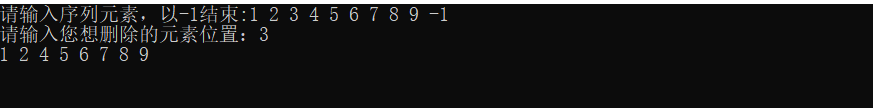


**4.1.2删除功能测试**

**测试用例：**1 2 3 4 5 6 7 8 9 -1（删除3）

**预期结果：**1 2 4 5 6 7 8 9

**实验结果：**



**4.1.3输出功能测试**

**测试用例：**1 2 3 4 5 6 7 8 99 -1

**预期结果：**1 2 3 4 5 6 7 8 99

**实验结果：**

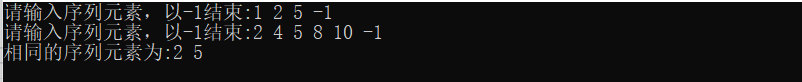


**4.1.4求交功能测试**

**测试用例：**1 2 5 -1 && 2 4 5 8 10 -1

**预期结果：**2 5

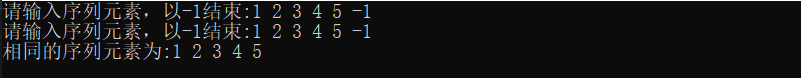
**实验结果：**



**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：**



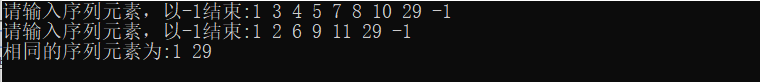
**4.2不同数据类型测试**

**4.2.1整型数据求交测试**

**测试用例：**1 3 4 5 7 8 10 29 -1 && 1 2 6 9 11 29 -1

**预期结果：**1 29

**实验结果：**

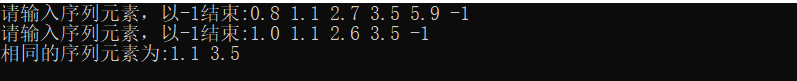


**4.2.2浮点型数据求交测试**

**测试用例：**0.8 1.1 2.7 3.5 5.9 -1 && 1.0 1.1 2.6 3.5 -1

**预期结果：**1.1 3.5

**实验结果：**

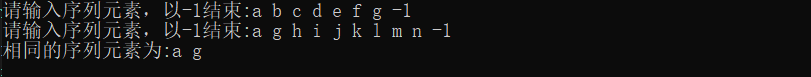


**4.2.3字符型数据求交测试**

**测试用例：**a b c d e f g -1 && a g h i j k l m n -1

**预期结果：**a g

**实验结果：**



**4.3边界测试**

**4.3.1两个链表都为空或其中一个为空**

**测试用例：**-1 && 1 2 3 4 -1

**预期结果：**NULL

**实验结果：**



**测试用例：**-1 && -1

**预期结果：**NULL

**实验结果：**

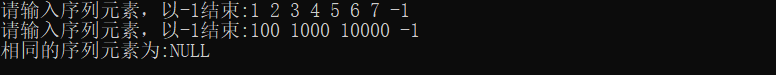


**4.3.2交集为空**

**测试用例：**1 2 3 4 5 6 7 -1 && 100 1000 10000 -1

**预期结果：**NULL

**实验结果：**



**4.3.3某个链表全是交集**

**测试用例：**1 2 3 4 -1 && 1 2 3 4 5 -1

**预期结果：**1 2 3 4

**实验结果：**

