项目说明文档

数据结构课程设计

——两个有序链表序列的交集

作 者 姓 名： 祝新元

学 号： 1751629

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 1](#_Toc531103296)

[1.1 背景分析 1](#_Toc531103297)

[1.2 功能分析 1](#_Toc531103298)

[2 设计 2](#_Toc531103299)

[2.1 数据结构设计 2](#_Toc531103300)

[2.2 类结构设计 2](#_Toc531103301)

[2.3 成员与操作设计 2](#_Toc531103302)

[2.4 系统设计 3](#_Toc531103303)

[3 实现 3](#_Toc531103304)

[3.1 创建并且读入链表S1和S2 3](#_Toc531103305)

[3.1.1 插入功能核心代码 3](#_Toc531103306)

[3.2 求两个集合（S1和S2）的交集S3 4](#_Toc531103307)

[3.2.1 求交集核心代码 4](#_Toc531103308)

[3.3 总体系统的实现 5](#_Toc531103309)

[3.3.1 总体系统核心代码 5](#_Toc531103310)

[4 测试 6](#_Toc531103311)

[4.1 案例测试 6](#_Toc531103312)

[4.1.1 一号测试 6](#_Toc531103313)

[4.1.2 二号测试 7](#_Toc531103314)

[4.1.3 三号测试 8](#_Toc531103315)

[4.1.4 四号测试 9](#_Toc531103316)

[4.1.5 五号测试 9](#_Toc531103317)

[4.2 自己测试 10](#_Toc531103318)

[4.2.1 两链表均为空 10](#_Toc531103319)

# 1 分析

## 1.1 背景分析

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。

## 1.2 功能分析

1. 输入说明：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。
2. 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。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求大量的增加、删除、修改操作，而链表进行增加、删除等操作十分简便，因此考虑使用链表数据结构。同时，为了实现简易，在第一个结点之前附加一个头结点，这样就使得增加或者删除头结点与处理其他结点方法相同，使得程序简洁。

## 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（LinkNode）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。为方便处理，本系统采用struct描述链表结点类（LinkNode），这样使得链表结点类（List）可以访问链表结点。

## 2.3 成员与操作设计

**链表结点类（**LinkNode**）**

T data;//节点值的值

LinkNode<T> \*link;//下一个节点的地址

**公有操作：**

LinkNode(LinkNode<T> \*next=NULL):link(next){}//构造函数

LinkNode(const T& d,LinkNode<T> \*next=NULL):data(d),link(next){}//构造函数

**链表类（**List**）**

**私有成员：**

LinkNode<T> \*first;//链表的头节点

**公有操作：**

List::List(){first=new LinkNode<T>;}

//构造函数，设置头节点

List<T>::List(List<T>& L)

//List的复制构造函数，通过另一个list构造

List::~List(){makeEmpty();}

//list的析构函数，通过调用makeEmpty()实现对内存的回收

int List<T>::Length()const

//得到List的长度

void List<T>::makeEmpty()

//收回链表所占的内存，防止内存泄漏

LinkNode<T> \*List<T>::Search(T x)

//寻找值为x的节点

bool List<T>::Insert(int i, T& x)

//在第i位插入值为x的节点

LinkNode<T>\* List<T>::Locate(int i)const

//寻找第i位节点

void List<T>::output()

//输出整个链表

## 2.4 系统设计

分别输入两个链表S1和S2，元素之间用空格作为间隔，以-1结尾。系统自动输出他们两的交集，新的链表S3.

# 3 实现

## 3.1 创建并且读入链表S1和S2

### 3.1.1 插入功能核心代码

template<class T>

bool List<T>::Insert(int i, T& x)

//前一个参数i是为辨别此节点是否第一个元素

{

LinkNode<T> \*current = Locate(i);//找到要插入的位置

LinkNode<T> \*newNode = new LinkNode<T>(x);

//创建节点失败

if (newNode == NULL) { cerr << "储存分配错误！！！" << endl; exit(1); }

//插入

if (i == 1)

{

first->link = newNode;

}

else

{

newNode->link=current->link;

current->link=newNode;

}

return true;

}

## 3.2 求两个集合（S1和S2）的交集S3

### 3.2.1 求交集核心代码

bool createNew(List<int>& s1,List<int>& s2,List<int> &s3)

{

//assert(s3.getHead() != NULL);

LinkNode<int>\* p = s1.getHead()->link;//得到s1的第一个元素

LinkNode<int>\* q = s2.getHead()->link;//得到s2的第一个元素

int count = 0; //记录s3的元素个数

const int x = -1;//结束标志

//结束

if (p->data == x || q->data == x)

{

return 0;

}

//求交集

while (p->link->data != x || q->link->data != x)

{

//如果两个元素相等，则把这个元素加入到s3里面

if (p->data == q->data)

{

count++;

s3.Insert(count, p->data);

if (p->link->data != x)

{

p = p->link;

}

if (q->link->data != x)

{

q = q->link;

}

}

//如果q大，且p没有到达最后，则p后移；如果p已经是最后一个元素了则q后移。p大，同理。

else if (p->data < q->data)

{

if (p->link->data != x)

{

p = p->link;

}

else if(q->link->data != x)

{

q = q->link;

}

}

else if (p->data > q->data)

{

if (q->link->data != x)

{

q = q->link;

}

else if(p->link->data != x)

{

p = p->link;

}

}

}

//比较最后一个元素

if (p->data == q->data)

{

count++;

s3.Insert(count, p->data);

}

return true;

}

## 3.3 总体系统的实现

### 3.3.1 总体系统核心代码

List<int> s1,s2,s3;

//init s1

bool flag1 = false;

int InputNum=0;

int count = 0;

while (InputNum>=0)

{

cin >> InputNum;

count++;

flag1=s1.Insert(count, InputNum);

}

s1.Insert(count, InputNum);

//init s2

InputNum = 0;

count = 0;

while (InputNum >= 0)

{

cin >> InputNum;

count++;

flag1=s2.Insert(count, InputNum);

}

s2.Insert(count, InputNum);

bool flag = createNew(s1, s2, s3);

s3.output();

# 4 测试

## 4.1 案例测试

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

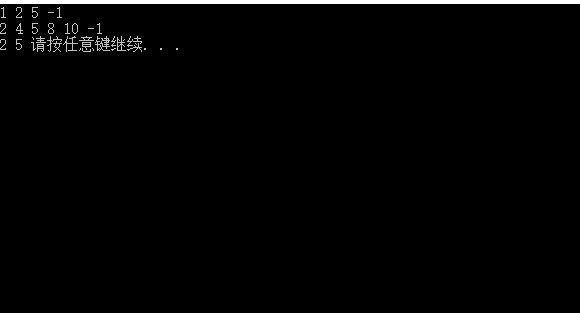
### 4.1.1 一号测试

**测试用例**：S1=1 2 5 -1

S2=2 4 5 8 10 -1

**预期结果**：2 5

**实验结果**



### 4.1.2 二号测试

**测试用例：**

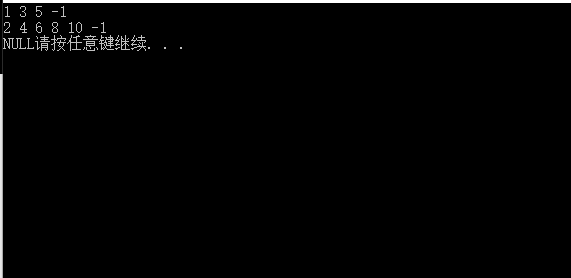
S1=1 3 5 -1

S2=2 4 6 8 10 -1

**预期结果：**

**NULL**

**实验结果：**



### 4.1.3 三号测试

**测试用例：**

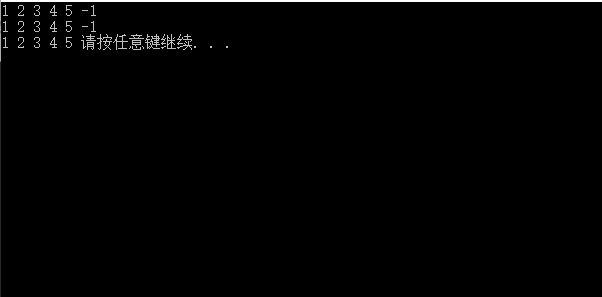
S1=1 2 3 4 5 -1

S2=1 2 3 4 5 -1

**预期结果：**

1 2 3 4 5

**实验结果：**



### 4.1.4 四号测试

**测试用例：**

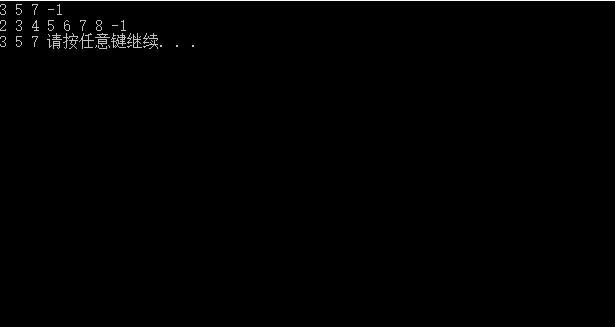
S1=3 5 7 -1

S2=2 3 4 5 6 7 8 -1

**预期结果：**

3 5 7

**实验结果：**



### 4.1.5 五号测试

**测试用例：**

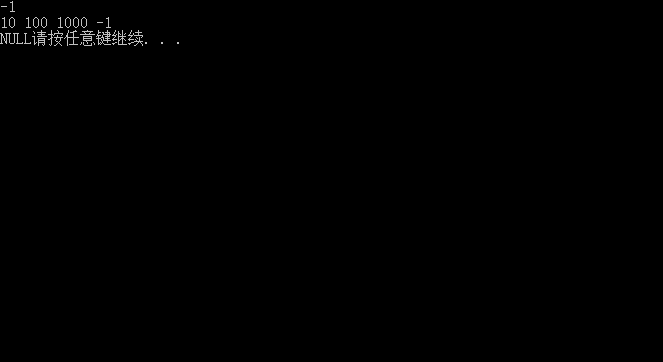
S1=-1

S2=10 100 1000 -1

**预期结果：**

NULL

**实验结果：**



## 4.2 自己测试

### 4.2.1 两链表均为空

**测试用例：**

**S1=-1**

**S2=-1**

**预期结果：**

**NULL**

**实验结果：**

