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： 李佳诺

学 号： 1751188

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 3

1.1 背景分析 3

空集 3

子集 3

交集 3

1.2 功能分析 4

2 设计 4

2.1 算法设计 4

2.2 数据结构设计 4

2.3 类结构设计 5

链表结点（linkedNode） 5

链表类（linkedList） 5

3 实现 7

3.1 构造函数linkedList() 7

构造函数流程图 7

3.2 插入函数insertData() 9

插入函数流程图 9

3.3 删除函数deleteData() 11

删除功能流程图 11

3.4 主函数 13

主函数流程图 13

4 测试 16

4.1功能性测试 16

4.1.1一般情况 16

4.1.2交集为空的情况 17

4.1.3完全相交的情况 17

4.1.4其中一个序列完全属于交集的情况 18

4.1.5其中一个序列为空的情况 18

# 1 分析

## 背景分析

集合是具有某种特定性质的具体的抽象的或抽象的对象汇总而成的集体。其中，构成集合的这些对象成为该集合的元素。给定一个集合一般来说，都有确定性、互异性、无序性。

### 空集

空集是一类特殊的集合。它不包含任何元素，记为**∅。**空集是个特殊的集合，它有两个特点：

1. 空集是任意一个非空集合的真子集。
2. 空集是任何一个集合的子集。

### 子集

设S，T是两个集合，如果S的所有元素都属于T ，即https://gss3.bdstatic.com/-Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D96/sign=a2e1f7afb33533faf1b69f28a9d31bfd/f703738da97739120822fbc0fa198618367ae20f.jpg，则称S是T的**子集**，记为https://gss3.bdstatic.com/-Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D39/sign=3e14b66bad51f3dec7b2bf6d95ee39c9/d01373f082025aaf070d006af9edab64034f1a57.jpg。显然，对任何集合S ，都有https://gss0.bdstatic.com/94o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D84/sign=dff755868882b90139adce37728da7cd/d439b6003af33a87f95b7ebbc45c10385343b558.jpg。其中，符号https://gss1.bdstatic.com/-vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D11/sign=04e9ba9646c2d562f608d4ece51124c5/7a899e510fb30f24ff76f8f5c195d143ac4b0353.jpg读作包含于，表示该符号左边的集合中的元素全部是该符号右边集合的元素。如果S是T的一个子集，即https://gss3.bdstatic.com/-Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D39/sign=3e14b66bad51f3dec7b2bf6d95ee39c9/d01373f082025aaf070d006af9edab64034f1a57.jpg，但在T中存在一个元素[x](https://baike.baidu.com/item/x/2659992)不属于S ，即https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D39/sign=a68b938faa014c081d3b2eac0b7b8cff/8c1001e93901213ff7a150fa57e736d12f2e9501.jpg，则称S是T的一个**真子集**。

### 交集

由属于A且属于B的相同元素组成的集合，记作A∩B（或B∩A），读作“A交B”（或“B交A”），即A∩B={x|x∈A,且x∈B}，注意交集越交越少。若A包含B，则A∩B=B，A∪B=A。

## 功能分析

本程序的基本需求是要求两个集合的交集。即对于用户输入的两个集合，要能正确的返回两个集合的交集。除此之外，该程序还要有一定的健壮性，减少因用户的操作失误而导致崩溃的可能性。

## 设计

## 算法设计

要求两条有序链表的交集，即求两条链表上的共有元素。为了节省空间，我们打算用其中一条链表作为结果链表。这里因为原本两条链表就是有序的，大大减少了工作量。我们就只要遍历A链（结果链），对于其中每一个元素，在B链中查找，如果查找到，就指向下一个结点，如果没找到，则在结果链中删除该结点，并指向下一个结点。最后将结果链作为结果输出给用户。

## 数据结构设计

实现集合的数据结构有很多，有数组、链表等。由于本题的题目明确要求要用链表实现，于是我们如题，打算用链表来实现。其优点显而易见，比如可以支持大量数据的增加、删除、修改。

## 类结构设计

#### 链表结点（linkedNode）

//链表结点

struct linkedNode

{

int value; //存储数据

linkedNode\* next; //指向下一结点

};

#### 链表类（linkedList）

//链表类

class linkedList

{

public:

linkedList(); //构造函数

linkedNode\* first; //指向首元结点

linkedNode\* last; //指向尾节点

int size; //链表大小

void print()const; //打印链表

void insertData(int pos,int &numToInsert); //插入函数：pos为插入位置，numToInsert为插入的数据

void deleteData(int &numToDelete); //删除函数：numToDelete为删除的数据

bool find(int num)const; //查找函数：所查找的数据为num

};

# 实现

## 构造函数linkedList()

#### 构造函数流程图

开 始

输入数据

输入是否合法？

录入数据至链表

结 束

否

是

创建一条空链

构造函数的主要功能是创建一个空链表，并通过与用户交互完善链表结构，具体代码如下：

linkedList::linkedList()

{

first = nullptr;

size=0;

cout << "请输入集合中的的元素（非降序的正整数），每个元素之间用空格隔开，以-1结尾： ";

first = new linkedNode;

cin >> first->value;

this->last = first;

if (first->value != -1)

{

size++;

int num = 0;

while (num >= 0)

{

cin >> num;

last->next = new linkedNode;

last = last->next;

last->value = num;

size++;

}

last->next = nullptr;

}

}

## 插入函数insertData()

#### 插入函数流程图

开 始

结 束

是否为插入位置

否

是

检查当前结点

插入

指向下一结点

插入函数其基本功能是在链表指定位置插入新的结点，以此来调整整个链表，具体代码如下：

//在链表pos位后插入numToInsert的数据

void linkedList::insertData(int pos, int &numToInsert)

{

auto newNode = new linkedNode;

newNode->value = numToInsert;

auto p = first;

for (int i = 0; i < pos - 1; i++)

{

//先找到所要插入的位置

if (p->next != nullptr)

{

p = p->next;

}

}

if (p->next == nullptr)

{

//已是链表尾部，则开辟新结点，并接上原链表

p->next = newNode;

this->last = newNode;

}

else

{

newNode->next = p->next;

p->next = newNode;

}

}

## 删除函数deleteData()

#### 删除功能流程图

开 始

查 找

是否找到

传入待删除的数据

结 束

输出报错

删 除

是

否

删除函数的基本功能是在链表中删除传入数据所在的结点。其实现的基本逻辑是在遍历链表的时候，检查当前结点的下一个，是否为所要删除的结点，如果是就将下一个结点从整个链表中摘除。当遍历完成仍未找到，则返回错误信息，具体代码如下：

//删除值为numToDelete的结点

void linkedList::deleteData(int &numToDelete)

{

//首元结点为所要删除的

if (this->first->value == numToDelete)

{

auto temp = first;

first = first->next;

delete temp;

this->size--;

return;

}

//非首元结点

auto p = first;

while (p->next != nullptr)

{

if (p->next->value == numToDelete)

{

auto q = p->next;

p->next = p->next->next;

delete q;

this->size--;

return;

}

p = p->next;

}

//遍历完没找到所要删除的结点

if (p->next == nullptr)

{

cout << "找不到元素：" << numToDelete << endl;

}

}

## 主函数

#### 主函数流程图

开 始

分别输入A,B链数据

指向A链头部

是否找到

在B链中搜索

A中指向下一个

添加到A链

A链是否完

结 束

是

是

否

否

删除该结点

主函数基本思路就是在算法设计中所讲，但加入了些人机交互的操作，使得界面友好并且不易崩溃，具体代码如下：

int main()

{

linkedList A;

linkedList B;

if (A.size == 0 || B.size == 0)

{//如果A,B链都为空

cout << "A和B的交集为： NULL";

}

else

{

auto p = A.first;

int numToSearch = 0;

while (p != nullptr)

{

auto temp = p->next;

numToSearch = p->value;

if (B.find(numToSearch) == false)

{//在B链中没招到该元素，则在A链中删除该元素所在结点

A.deleteData(numToSearch);

}

p = temp;

}

//输出结果

cout << "A和B的交集为：";

if (A.size > 1)

{

A.print();

}

else

{

cout << " NULL";

}

}

return 0;

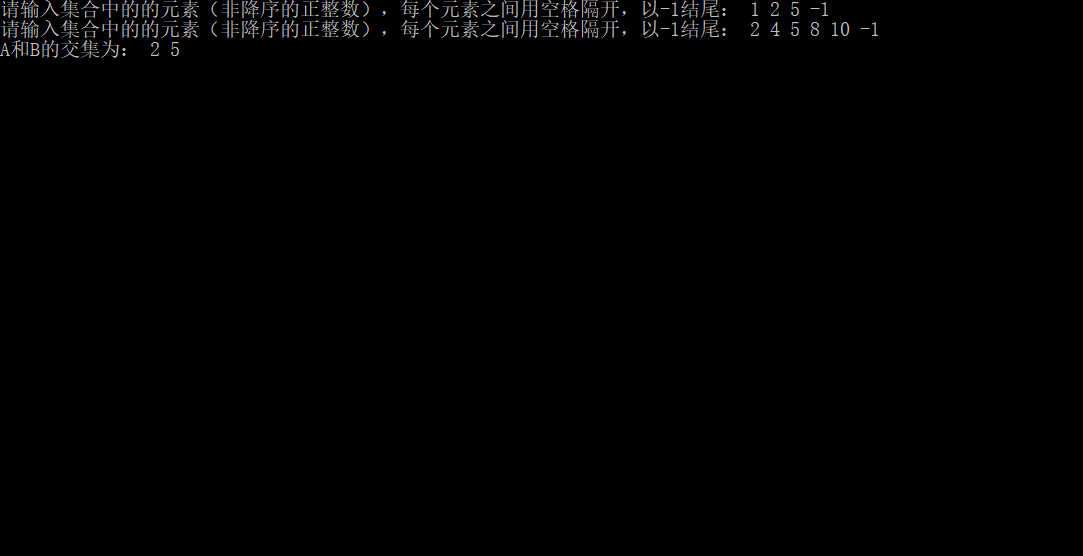
}

# 测试

## 4.1功能性测试

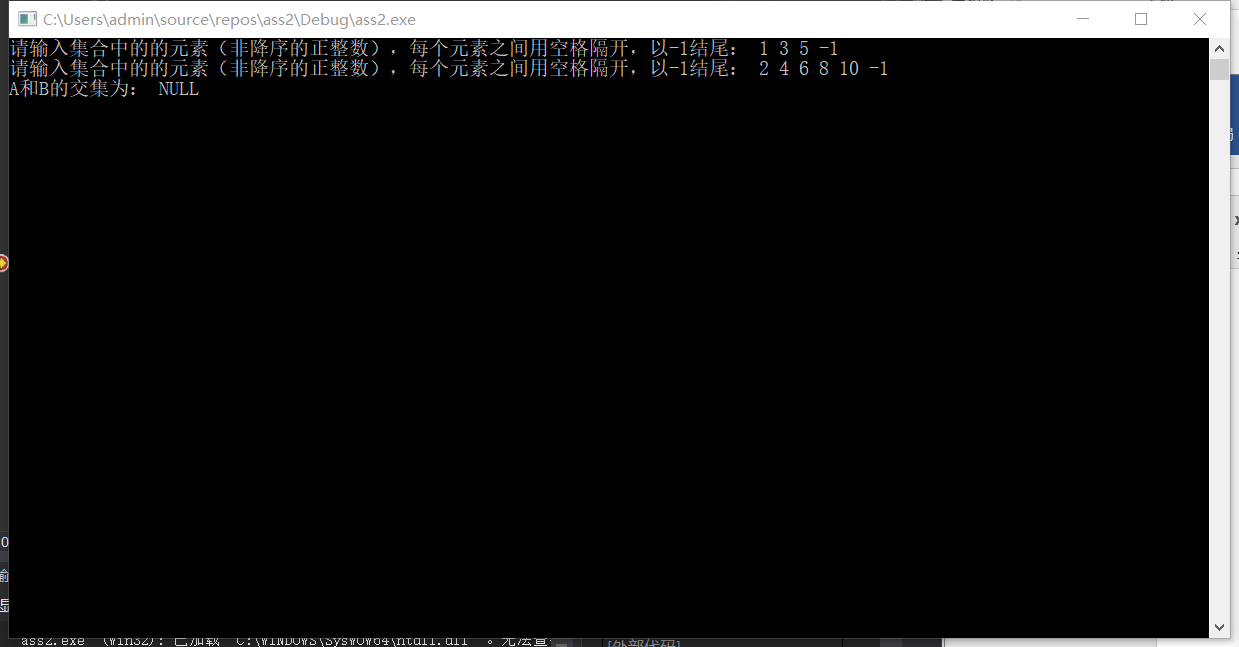
#### 4.1.1一般情况

实验结果如下：



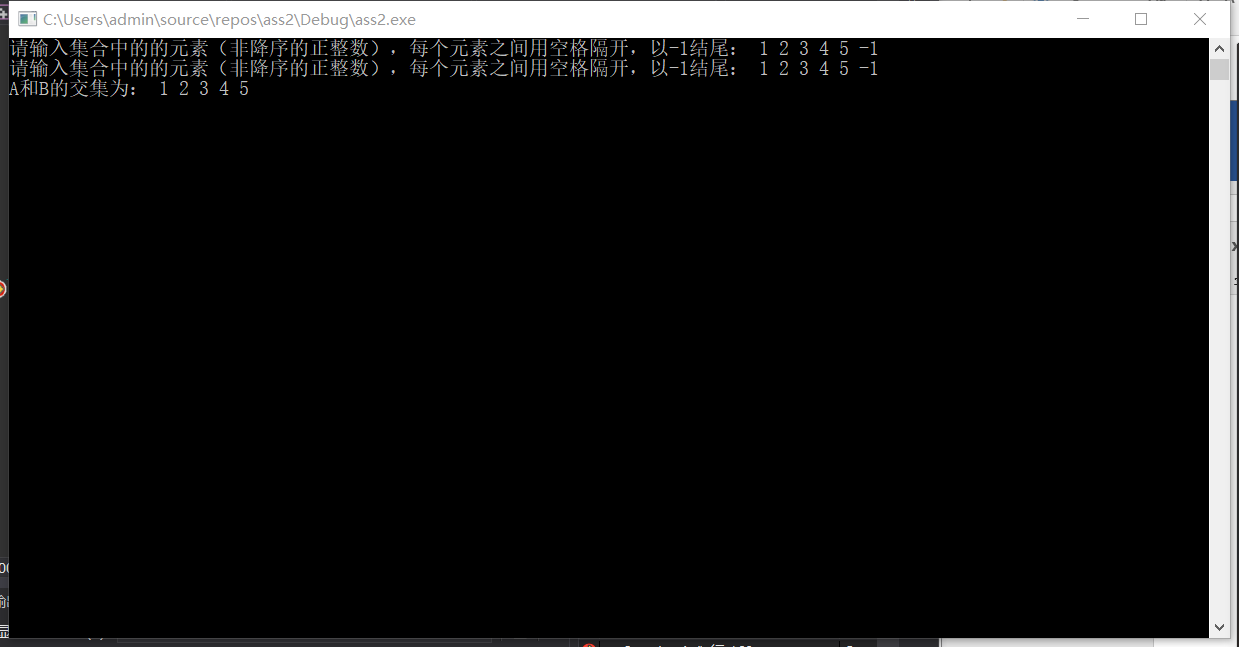
#### 4.1.2交集为空的情况

实验结果如下：



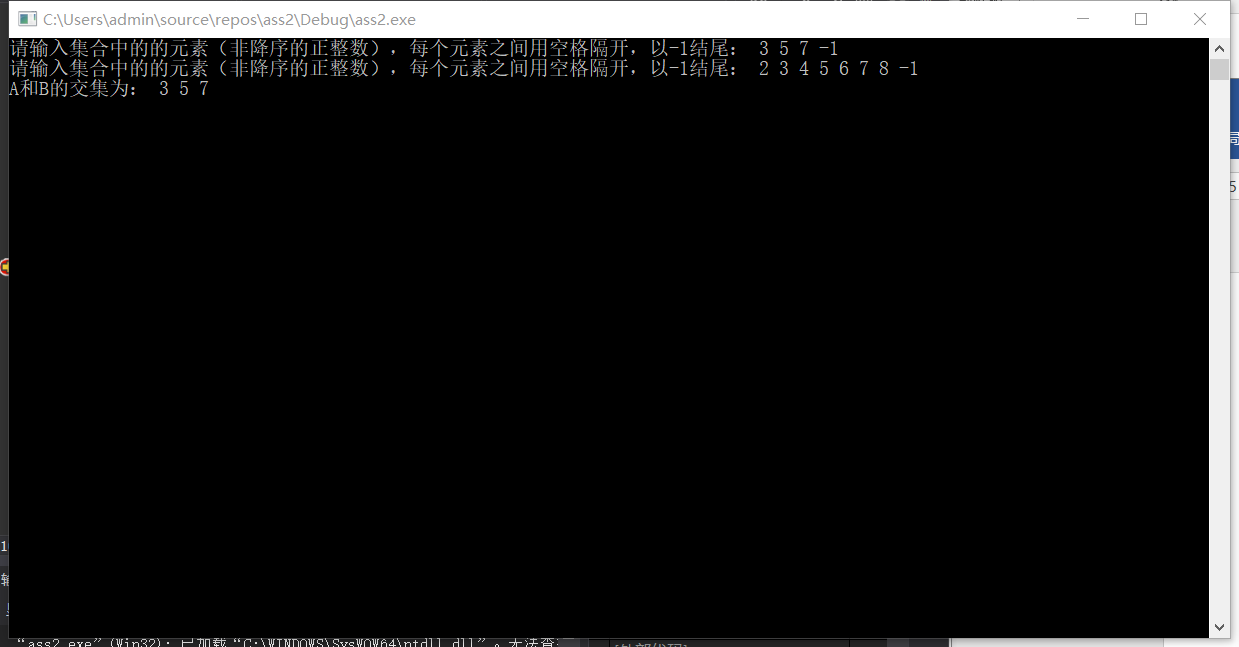
#### 4.1.3完全相交的情况

实验结果如下：



#### 4.1.4其中一个序列完全属于交集的情况

实验结果如下：



#### 4.1.5其中一个序列为空的情况

实验结果如下：

