项目说明文档

数据结构课程设计

——两个有序链表的交集

同济大学

Tongji University

目 录

[1 分析 1](#_Toc59940144)

[1.1 功能分析 1](#_Toc59940145)

[2 设计 1](#_Toc59940146)

[2.1 数据结构设计 1](#_Toc59940147)

[2.2 链表结构设计 1](#_Toc59940148)

[2.3 成员与操作设计 1](#_Toc59940149)

[2.4 系统设计 2](#_Toc59940150)

[3 实现 3](#_Toc59940151)

[3.1 插入功能的实现 3](#_Toc59940152)

[3.1.1 插入功能流程图 3](#_Toc59940153)

[3.1.2 插入功能核心代码 4](#_Toc59940154)

[3.2 合并功能的实现 5](#_Toc59940155)

[3.2.1 合并功能流程图 5](#_Toc59940156)

[3.2.2 合并部分核心代码 6](#_Toc59940157)

[3.3 展示功能的实现 7](#_Toc59940158)

[3.3.1 展示功能流程图 7](#_Toc59940159)

[3.3.2 展示功能核心代码 8](#_Toc59940160)

[3.3.3 展示功能截图示例 8](#_Toc59940161)

[4 测试 9](#_Toc59940162)

[4.1 基本测试 9](#_Toc59940163)

[4.1.1 一般情况 9](#_Toc59940164)

[4.1.2 交集为空的情况 9](#_Toc59940165)

[4.1.3 完全相交的情况 10](#_Toc59940166)

[4.1.4 其中一个序列完全属于交集的情况 10](#_Toc59940167)

[4.1.5 其中一个序列为空的情况 11](#_Toc59940168)

[4.2 出错测试 11](#_Toc59940169)

# 1 分析

## 1.1 功能分析

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。

根据要求，首先使用链表储存S1和S2（虽然对于解题，可以只记录S1，而后在读入S2的过程中删除S1中的元素），再用链表S0记录二者的交集。

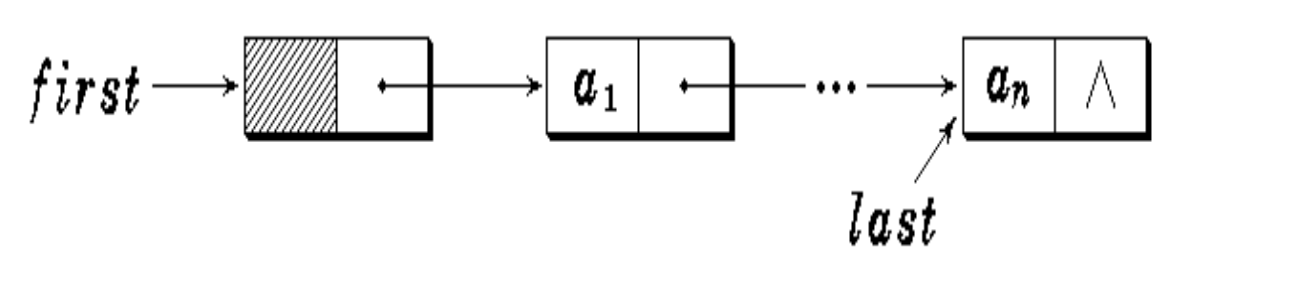
而具体看，输入分2行，分别在每行给出由若干个正整数构成的非降序序列，用-1表示序列的结尾（-1不属于这个序列），需要根据这个-1来判断输入结束；而输出需要保证在末尾不能有多余的空格。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述（以及题目明确要求），需要使用链表。本程序使用带表头的单链表。

## 2.2 链表结构设计

存储结构：  


## 2.3 成员与操作设计

**链表结点结构体（LNode）**

typedef struct LNode

{

int data;

LNode\* next;

}\*Linklist;

**链表类（LinkList）**

**私有成员：**

Linklist first[3];

//first[0]储存交集，first[1]储存第一个链表，first[2]储存第二个链表

**公有操作：**

List();

//构造函数，读入L1和L2

void insert(Linklist& tail,int x);

//在tail之后插入值为x的节点

void merge()

//关键操作，链表合并

void display()

//输出结果

## 2.4 系统设计

系统给出提示语句，由用户输入两个链表，之后输出结果。当用户的输入不是非降序列时，给出报错提示。

# 3 实现

## 3.1 插入功能的实现

### 3.1.1 插入功能流程图

开始

新建一个节点p，指向空指针，值为x

结束

tail指向p(使用指针引用)

### 3.1.2 插入功能核心代码

void insert(Linklist& tail,int x)

{

LNode\* p = new LNode{ x,NULL };

tail->next = p;

tail = p;

}

## 3.2 合并功能的实现

### 3.2.1 合并功能流程图

开始

p = first[1]->next

q = first[2]->next

first[0]=new LNode{ 0,NULL }

tail=first[0]

结束

p&&q?

比较p->data和q->data

p=p->next

q=q->next

insert(tail,q->data)

p=p->next

q=q->next

<

=

<

y

n

### 3.2.2 合并部分核心代码

void merge()

{

Linklist p = first[1]->next, q = first[2]->next;

Linklist tail = first[0] = new LNode{ 0,NULL };

while (p && q)

{

if (p->data < q->data)p = p->next;

else if (p->data == q->data) { insert(tail, q->data); p = p->next; q = q->next; }

else q = q->next;

}

}

## 3.3 展示功能的实现

### 3.3.1 展示功能流程图

开始

结束

是否是第一个?

打印空格+内容

n

y

打印内容

### 3.3.2 展示功能核心代码

void display()

{

if (!first[0]->next)cout << "NULL";

else

{

bool fir = 1;

for (Linklist p = first[0]->next; p; p = p->next)

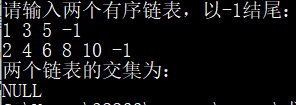
if (fir) { fir = 0; cout << p->data; }

else cout << ' ' << p->data;

}

}

### 3.3.3 展示功能截图示例



# 4 测试

## 4.1 基本测试

### 4.1.1 一般情况

**测试用例**：

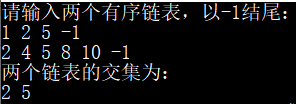
1 2 5 -1

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：**

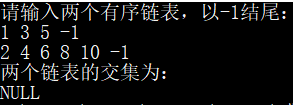
1 3 5 -1

2 4 6 8 10 -1

**预期结果：**

NULL

**实验结果：**



### 4.1.3 完全相交的情况

**测试用例：**

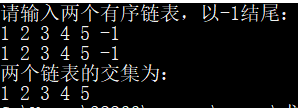
1 2 3 4 5 -1

1 2 3 4 5 -1

**预期结果：**

1 2 3 4 5

**实验结果：**



### 4.1.4 其中一个序列完全属于交集的情况

**测试用例：**

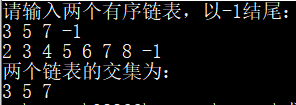
3 5 7 -1

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：**

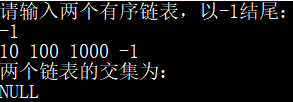
-1

10 100 1000 -1

**预期结果：**

NULL

**实验结果：**

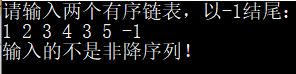


## 4.2 出错测试

**测试用例：**输入非降序链表

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

****