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： 陈翔飞

学 号： 1851756

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](#_Toc27344261)

[1.1 项目要求分析 3](#_Toc27344262)

[1.2 功能分析 3](#_Toc27344263)

[2 设计 3](#_Toc27344264)

[2.1 数据结构设计 3](#_Toc27344265)

[2.2 类结构设计 3](#_Toc27344266)

[2.3 成员与操作设计 3](#_Toc27344267)

[2.4 系统设计 6](#_Toc27344268)

[3 实现 7](#_Toc27344269)

[3.1 构造序列功能的实现 7](#_Toc27344270)

[3.1.1输入功能流程图 7](#_Toc27344271)

[3.1.2 构造序列功能核心代码 7](#_Toc27344272)

[3.1.3 构造序列功能截屏示例 9](#_Toc27344273)

[3.2 构造交集功能的实现 9](#_Toc27344274)

[3.2.1 构造交集功能核心代码 9](#_Toc27344275)

[3.2.3 构造交集功能截屏示例 10](#_Toc27344276)

[3.3 总体系统的实现 10](#_Toc27344277)

[3.3.1 总体系统核心代码 10](#_Toc27344278)

[3.6.2 总体系统截屏示例 11](#_Toc27344279)

[4 测试 12](#_Toc27344280)

[4.1 基本功能测试 12](#_Toc27344281)

[4.1.1 普通交集测试 12](#_Toc27344282)

[4.1.2 交集为空测试 12](#_Toc27344283)

[4.1.3 完全相交测试 13](#_Toc27344284)

[4.1.4 一个序列属于完全交集测试 13](#_Toc27344285)

[4.2 边界测试 14](#_Toc27344286)

[4.2.1 一个序列为空 14](#_Toc27344287)

[4.2.2 两个序列为空 15](#_Toc27344288)

[4.2.3 序列中数字极大 15](#_Toc27344289)

[4.3 出错测试 16](#_Toc27344290)

[4.3.1 询问是否求解时输入未知字符 16](#_Toc27344291)

[4.3.2 输入序列中含有非数字 16](#_Toc27344292)

[4.3.3 输入序列中含有非-1负数 17](#_Toc27344293)

[4.3.4 输入序列存在非降序部分 17](#_Toc27344294)

# 1 分析

## 项目要求分析

已知两个非降序序列S1和S2，设计构造函数构造出S1和S2的交集新链表。

## 1.2 功能分析

项目要求对链表进行操作，首先需要对链表进行建立，需要用到插入功能。构造出交集新链表之后要打印链表元素，需要用到输出功能。退出系统时要回收链表元素内存，需要用到清空功能。

综上所述，该考试管理系统需要有输入、输出、插入、清空的功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，项目需要使用链表的数据结构

## 2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将Node类和List类都设计为模板类，Node类储存实际的数据，List类储存头尾节点，并提供插入、查找、删除等接口。为了让模板类能实际应用于本项目，还设计了BigInteger类来适应超大的整数输入。

## 2.3 成员与操作设计

**结点类（Node）**

**类定义：**

1. **template** <**typename** ElementType> **class** Node
2. {
3. **public**:
4. Node<ElementType>();
5. Node<ElementType>(ElementType E);;
6. ~Node<ElementType>();
7. Node<ElementType>\* Insert(Node<ElementType>\* P, ElementType E);
8. Node<ElementType>\* Delete(Node<ElementType>\* P);
9. **void** DeleteList();
10. ElementType GetElement();
12. Node<ElementType>\* GetNext();
13. **private**:
14. ElementType Element;
15. Node<ElementType>\* Next = nullptr;
16. };

**私有成员：**

ElementType Element;//节点保存的元素

Node<ElementType>\* Next;//指向下一节点的指针

**公有操作：**

Node<ElementType>();//默认构造函数

Node<ElementType>(ElementType E);//参数为节点值的构造函数

~Node<ElementType>();//析构函数

Node<ElementType>\* Insert(Node<ElementType>\* P, ElementType E);

//在节点P后面插入一个值为E的新节点

Node<ElementType>\* Delete(Node<ElementType>\* P);

//删除节点P后面的一个节点

void DeleteList();

//清空当前节点和其之后的所有节点

ElementType GetElement();

//获取当前节点的值

**链表类（List）**

**类定义：**

1. **template** <**typename** ElementType> **class** List
2. {
3. **public**:
4. List<ElementType>();
5. ~List<ElementType>();
6. **void** Clear();
7. ElementType GetCurElement();
8. Node<ElementType>\* CurToNext();
9. Node<ElementType>\* ResetCur();
10. Node<ElementType>\* GetCur();
11. Node<ElementType>\* GetHead();
12. Node<ElementType>\* GetLast();
13. **int** GetLength();
14. Node<ElementType>\* PushBack(ElementType E);
15. Node<ElementType>\* PushFront(ElementType E);
16. **const** List< ElementType>& operator =(**const** List< ElementType>&L);
17. **void** Display(ostream& os);
18. **private**:
19. Node<ElementType>\* Head = nullptr, \*Last = nullptr, \*Cur = nullptr;
20. **int** Length = 0;
21. };

**私有成员：**

Node<ElementType>\* Head ;

//链表头节点，不储存实际数据，仅作为一个标志

Node<ElementType>\* Last ;

//链表尾节点，储存实际数据

Node<ElementType> \*Cur;

//当前正在访问的节点

int Length = 0;

//链表的长度

**公有操作：**

List<ElementType>();

//默认构造函数

~List<ElementType>();

//析构函数，调用Clear()来释放内存

void Clear();

//清空链表并释放内存

ElementType GetCurElement();

//返回当前访问节点的值

Node<ElementType>\* CurToNext();

//将当前访问节点向后移动一个

Node<ElementType>\* ResetCur();

//将当前访问节点重设为头节点后一个

Node<ElementType>\* GetCur();

//返回当前节点的指针

Node<ElementType>\* GetHead();

//返回头节点

Node<ElementType>\* GetLast();

//返回尾节点

int GetLength();

//返回链表长度

Node<ElementType>\* PushBack(ElementType E);

//向链表尾部插入元素

Node<ElementType>\* PushFront(ElementType E);

//向链表头部插入元素

const List< ElementType>& operator =(const List< ElementType>&L);

//重载=运算符，使得该类支持赋值运算

void Display(ostream& os);

//打印整个链表

**大整数类（BigInteger）**

**类定义：**

1. **class** BigInteger
2. {
3. **public**:
4. BigInteger(string Integer = {}) :Integer(Integer) {};
5. string Integer = {};
6. **bool** operator > (**const** BigInteger B) **const**;
7. **bool** operator < (**const** BigInteger B) **const**;
8. **bool** operator ==(**const** BigInteger B) **const**;
10. BigInteger& operator =(string s);
11. **friend** ostream &operator <<(ostream &os, **const** BigInteger &B);
12. **friend** istream& operator >>(istream& is, BigInteger &B);
13. };

**公有成员：**

string Integer ;//用字符串储存的整数

**公有操作：**

BigInteger(string Integer = {}) :Integer(Integer) {};

//带默认参数的构造函数

bool operator > (const BigInteger B) const;

//重载>运算符使该类可以进行>运算

bool operator < (const BigInteger B) const;

//重载<运算符使该类可以进行<运算

bool operator ==(const BigInteger B) const;

//重载==运算符使该类可以进行==运算

BigInteger& operator =(string s);

//重载=运算符使该类可以进行赋值运算

friend ostream &operator <<(ostream &os, const BigInteger &B);

//重载<<运算符使该类可以进行<<运算

friend istream& operator >>(istream& is, BigInteger &B);

//重载>>运算符使该类可以进行>>运算

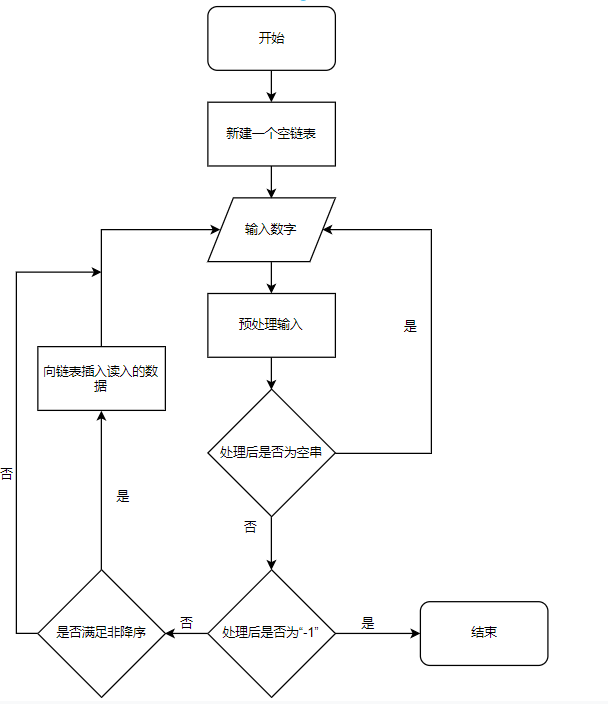
## 2.4 系统设计

系统首先会询问用户是否进行求解，若结果为是则进入求解模块，调用Read（）函数来读取数据，用户根据提示输入两个非降序序列，以 '-1'作为序列结束的标志，然后系统会构造出这两个序列的交集新链表，并且输出。然后继续询问是否要求解。

# 3 实现

## 3.1 构造序列功能的实现

### 3.1.1输入功能流程图



### 3.1.2 构造序列功能核心代码

**用户界面接口：**

1. List<BigInteger>\* Read()
2. {
3. Error1 = Error2 = **false**;
4. List<BigInteger>\* TempList = **new** List<BigInteger>();
5. string S;
6. BigInteger DisposedB, PreB;
7. **while** (cin >> S)
8. {
9. DisposedB = CheckString(S);
10. **if** (DisposedB == BigInteger("-1")) **return** TempList;
11. **if** (DisposedB.Integer.empty()) **continue**;
12. **if** (PreB > DisposedB)
13. {
14. **if** (!Error2)
15. {
16. cout << "存在数字小于上一个数，不满足非降序，已忽略不符合要求的整数" << endl;
17. }
18. Error2 = **true**;
19. **continue**;
20. }
21. TempList->PushBack(DisposedB);
22. PreB = DisposedB;
23. }
24. **return** TempList;
25. }

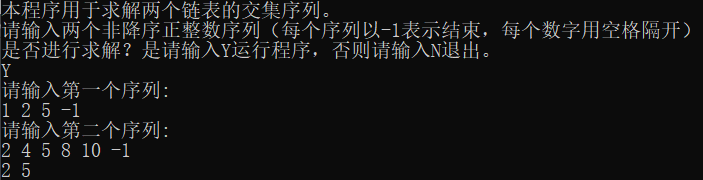
**链表接口：**

1. **template**<**typename** ElementType>
2. **inline**  Node<ElementType>\* List<ElementType>::PushFront(ElementType E)
3. {
4. Length++;
5. **return** Head->Insert(Head, E);
6. }

**链表插入内部实现：**

1. **template**<**typename** ElementType>
2. **inline**  Node<ElementType>\* Node<ElementType>::Insert(Node<ElementType>\* P, ElementType E)
3. {
4. **if** (P != nullptr)
5. {
6. **if** (P->Next == nullptr)
7. {
8. P->Next = **new** Node<ElementType>(E);
9. **return** P->Next;
10. }
11. **else**
12. {
13. Node<ElementType>\* TempCell\_1 = P->Next;
14. Node<ElementType>\* TempCell\_2 = **new** Node<ElementType>(E);
15. P->Next = TempCell\_2;
16. TempCell\_2->Next = TempCell\_1;
17. **return** TempCell\_1;
18. }
19. }
20. **else** **return** nullptr;
21. }

### 3.1.3 构造序列功能截屏示例

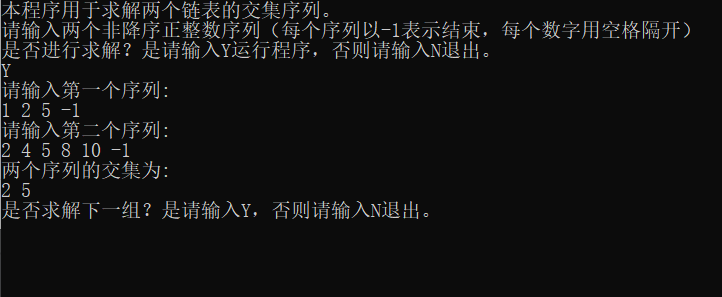


## 3.2 构造交集功能的实现

### 3.2.1 构造交集功能核心代码

1. List<BigInteger> \* GetIntersection(List<BigInteger> \*List\_1, List<BigInteger>\*List\_2)
2. {
3. List<BigInteger> \*Intersection = **new** List<BigInteger>();
4. List\_1->ResetCur(); List\_2->ResetCur();
5. List\_1->CurToNext(); List\_2->CurToNext();
6. **while** (List\_1->GetCur() != nullptr&&List\_2->GetCur()!= nullptr)
7. {
8. **if** (List\_1->GetCurElement() > List\_2->GetCurElement()) List\_2->CurToNext();
9. **else** **if**(List\_2->GetCurElement() > List\_1->GetCurElement()) List\_1->CurToNext();
10. **else**
11. {
12. Intersection->PushBack(BigInteger(List\_1->GetCurElement()));
13. List\_1->CurToNext(); List\_2->CurToNext();
14. }
15. }
16. **return** Intersection;
17. }

### 3.2.3 构造交集功能截屏示例

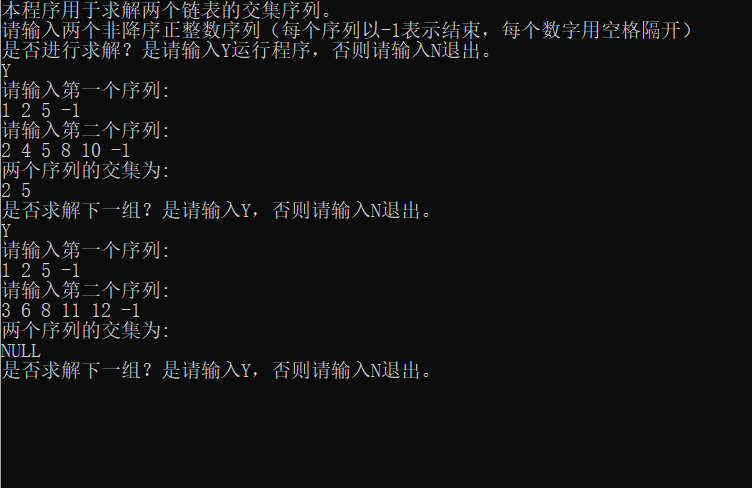


## 3.3 总体系统的实现

### 3.3.1 总体系统核心代码

1. **int** main(**void**)
2. {
3. cout << "本程序用于求解两个链表的交集序列。" << endl;
4. cout << "请输入两个非降序正整数序列（每个序列以-1表示结束，每个数字用空格隔开）" << endl;
5. cout << "是否进行求解？是请输入Y运行程序，否则请输入N退出。" << endl;
6. string s;
7. **while** (cin>>s)
8. {
9. **if** (s == "N"||s=="n") **break**;
10. **if** (s != "Y"&&s!="y")
11. {
12. cout << "请输入Y或者N，不要输入其他字符串。" << endl;
13. **continue**;
14. }
15. List<BigInteger> \*List\_1, \*List\_2;
16. cout << "请输入第一个序列:" << endl;
17. List\_1 = Read();
18. cout << "请输入第二个序列:" << endl;
19. List\_2 = Read();
20. List<BigInteger> \*Intersection = GetIntersection(List\_1, List\_2);
21. cout << "两个序列的交集为:" << endl;
22. Intersection->Display(cout);
23. cout << endl;
24. cout << "是否求解下一组？是请输入Y，否则请输入N退出。" << endl;
25. List\_1->Clear(); List\_2->Clear(); Intersection->Clear();
26. }
27. **return** 0;
28. }

### 3.6.2 总体系统截屏示例



# 4 测试

## 4.1 基本功能测试

### 4.1.1 普通交集测试

**测试用例**：

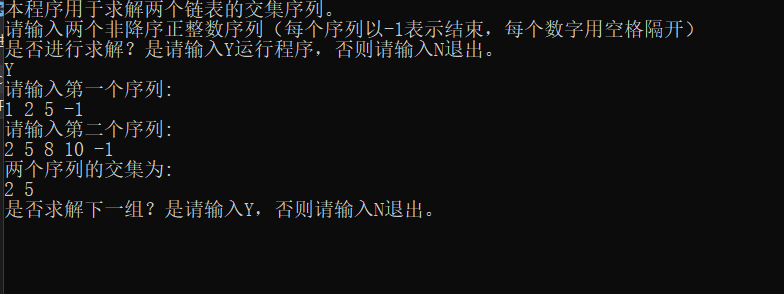
1 2 5 -1

2 5 8 10 -1

**预期结果**：

2 5

**实验结果：**



### 4.1.2 交集为空测试

**测试用例：**

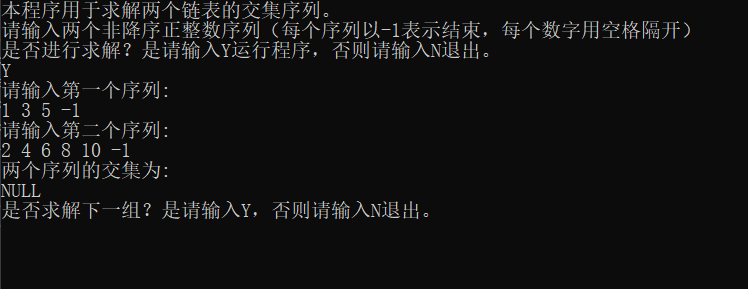
1 3 5 -1

2 4 6 8 10 -1

**预期结果：**

NULL

**实验结果：**



### 4.1.3 完全相交测试

**测试用例：**

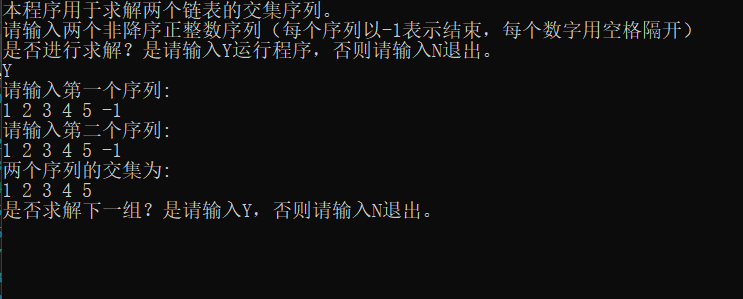
1 2 3 4 5 -1

1 2 3 4 5 -1

**预期结果：**

1 2 3 4 5

**实验结果：**



### 4.1.4 一个序列属于完全交集测试

**测试用例：**

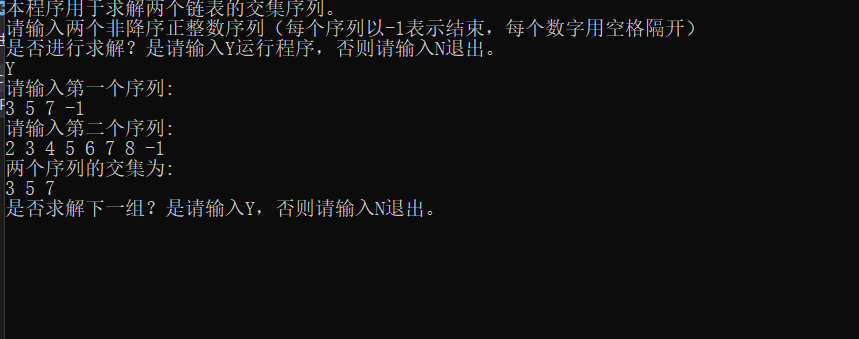
3 5 7 -1

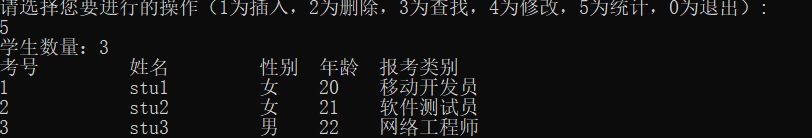
2 3 4 5 6 7 8 -1

**预期结果：**

3 5 7

**实验结果：**





## 4.2 边界测试

### 4.2.1 一个序列为空

**测试用例：**

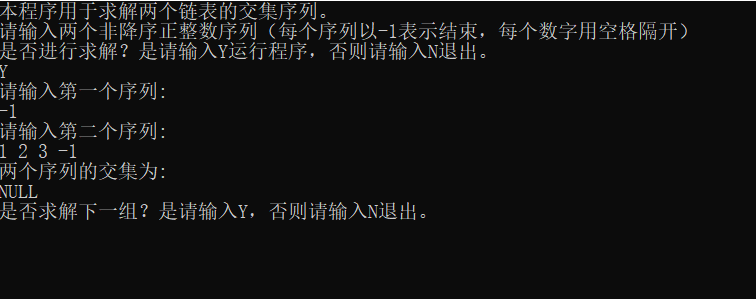
-1

1 2 3 -1

**预期结果：**

NULL

**实验结果：**



### 4.2.2 两个序列为空

**测试用例：**

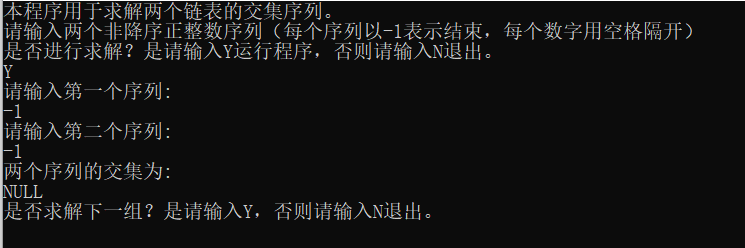
-1

-1

**预期结果：**

NULL

**实验结果：**



### 4.2.3 序列中数字极大

**测试用例：**

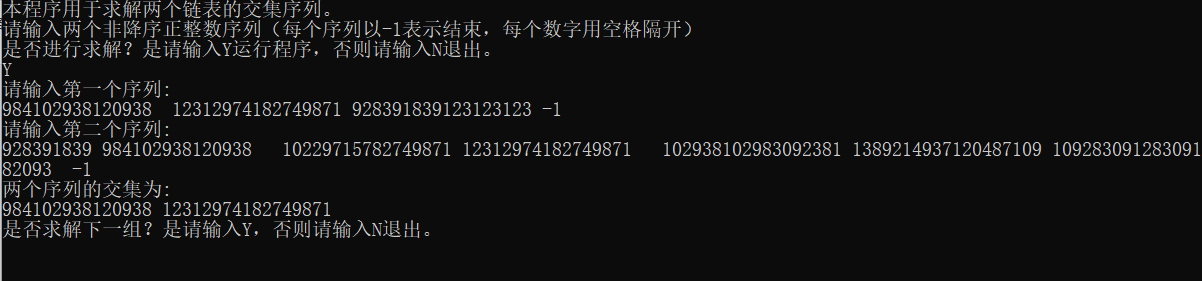
984102938120938 12312974182749871 928391839123123123 -1

928391839 984102938120938 10229715782749871 12312974182749871 102938102983092381 1389214937120487109 10928309128309182093 -1

**预期结果：**

984102938120938 12312974182749871

**实验结果：**



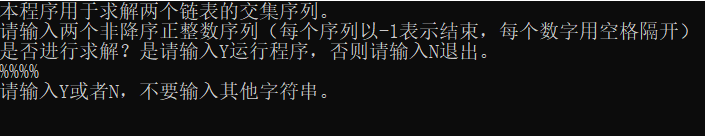
## 4.3 出错测试

### 4.3.1 询问是否求解时输入未知字符

**测试用例：** 询问是否求解时输入“%%%%%”

**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：**



### 4.3.2 输入序列中含有非数字

**测试用例：**

1 ASODJAWOIDJ111 2 3a 4 6 -1

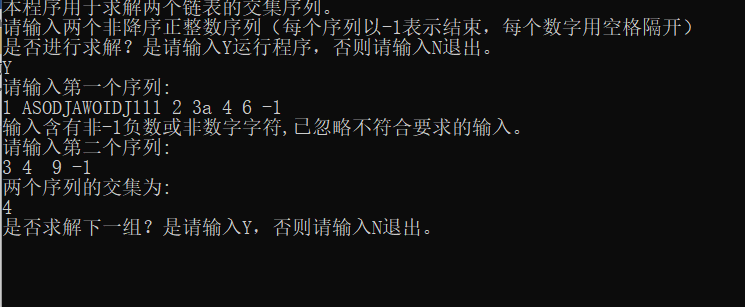
3 4 9 -1

**预期结果：**

程序不崩溃，忽略异常输入，输出提示，正常求解，输出：

4

**实验结果：**



### 4.3.3 输入序列中含有非-1负数

**测试用例：**

-22 -3 1 4 6 7 -1

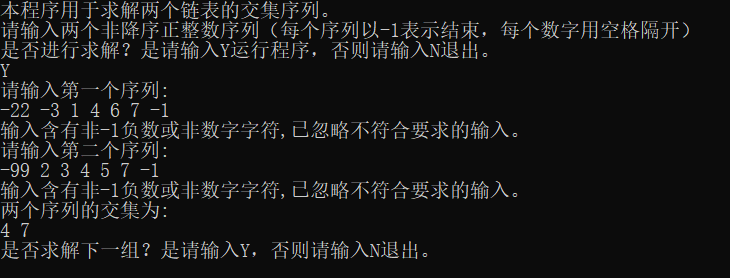
-99 2 3 4 5 7 -1

**预期结果：**

程序不崩溃，忽略异常输入，输出提示，正常求解，输出：

4 7

**实验结果：**



### 输入序列存在非降序部分

**测试用例：**

2 1 4 8 9 -1

4 1 9 8 10 -1

**预期结果：**

程序不崩溃，忽略非降序部分，输出提示，正常求解，输出：

4 9

**实验结果：**

