

数据结构课程设计

——两个有序链表序列的交集

项目说明文档

作 者 姓 名： 韦世贸

学 号： 2351131

指 导 老 师： 软件学院

专 业： 软件工程

同济大学

Tongji University

1 介绍

1.1 背景介绍

在求解实际问题中经常会遇见部分操作与求两个有序链表交集思维相似的情况，求解两个有序链表的交集这一题虽然不难但实际上是代表了一种思维方式，这种思维方式可以适用于更多的有序链表操作中。

1.2 功能介绍

作为一个有序链表求交集系统，首先应该有的功能就是可以建立链表的存储功能。其次，它还应该具有比较链表、进行链表交集求解以及正常输出功能。

2 设计

2.1 数据结构设计

该本项目的实质是两个有序链表序列的比较以及求交集运算。因为涉及到链表交集，于是理所应当考虑采用链表进行存储，建立有头结点的链表，方便进行操作。

2.2 类结构设计

在该项目中创建一个节点类和一个链表类，将链表类定义为节点类的友元类，从而建立起一个链表，因为该系统数据类型确定，无使用抽象数据类型的必要故直接定义了类型。

2.3 成员与操作设计

链表结点类和Mylist类

链表结点类

struct LinkNode //链表节点

{

int data; //链表元素

LinkNode\* link; //链表指针

LinkNode(LinkNode\* ptr = NULL) //输入信息

: data(0)

{

link = ptr;

}

};

Mylist类

私有数据成员

LinkNode\* first; //链表头指针

公有成员

MyList() //构造函数

~MyList() //析构函数

void CreateNewList() //建立新链表（后插法）

void output\_subset(int length) //输出子表

void Check\_repeat(MyList List1, MyList List2) //两表查重

3 实现

3.1 建立链表功能的实现

3.1.1 建立链表功能流程简介

建立链表功能即为最基础的创建链表，在创建新结点时进行健壮性的检测，逐个输入创建链表元素的值进行创建。

3.1.2 建立链表功能核心代码

LinkNode\* newNode, \* last;

last = first;

while(1) { //按顺序输入元素

newNode = new LinkNode();

cin >> newNode->data;

if (newNode == NULL) {

cout << "存储分配错误！" << endl;

return;

}

else if (newNode->data < 0) {

delete newNode;

return;

}

last->link = newNode; //尾指针指向新结点，连接起新结点

last = newNode; //更新尾指针的位置

}

last->link = NULL; //链表结尾

return;

3.2 链表求交集功能的实现

3.2.1 链表求交集功能流程简介

链表求交集功能是将s1链表中的数据与s2链表中的数据进行比较从而将交集输入到子表的过程，先对s1中的内容进行遍历，与s2中元素进行比对，若相等，则将该元素置入子表，子表长度加一。

3.2.2 链表求交集功能核心代码

if (List1.first == NULL || List2.first == NULL) //两表有一为空

cout << "NULL" << endl;

else{

LinkNode\* first\_current = List1.first->link, //从第一个有效结点开始遍历

\* subset\_current = first;

int subset\_length = 0; //子表的长度

//遍历两表

while (first\_current != NULL ) {

LinkNode\* second\_current = List2.first->link; //每次循环从第一个有效结点开始遍历

while(second\_current != NULL)

{

if (first\_current->data == second\_current->data) {

LinkNode\* newNode = new LinkNode();

newNode->data = first\_current->data;

subset\_current->link = newNode;

subset\_current = newNode; //保存结点位置，相当于last

subset\_length++; //更新子表的长度

}

second\_current = second\_current->link;

}

first\_current = first\_current->link;

}

if (subset\_length == 0) //两表没有交集

cout << "NULL";

else

output\_subset(subset\_length); //打印子表内容

cout << endl;

}

3.3 输出功能的实现

3.3.1 输出功能流程简介

输出功能即为逐个子表并且循环进行输出。

3.3.2 输出功能核心代码

LinkNode\* current = first->link;

while (length > 0) {

cout << current->data;

if (length != 1)

cout << " ";

current = current->link;

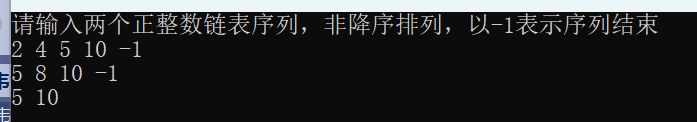
length--;

}

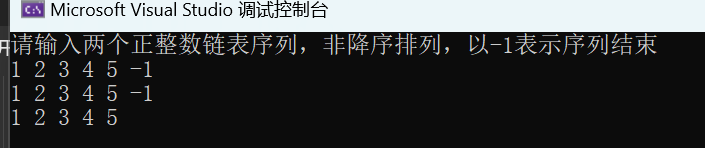
3.4 总体功能的演示

3.4.1 总体功能截屏示例

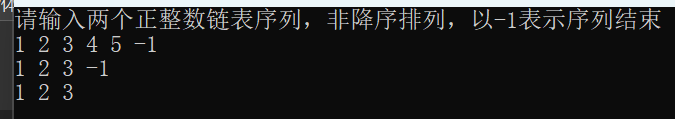
1.一般情况：



2.完全相交的情况

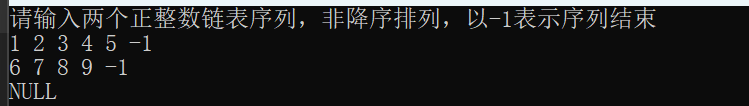


3.一个属于另一个的交集的情况

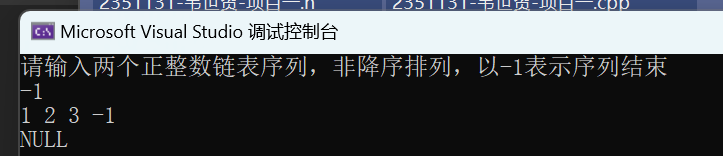


3.4.2 总体功能测试以及边界测试

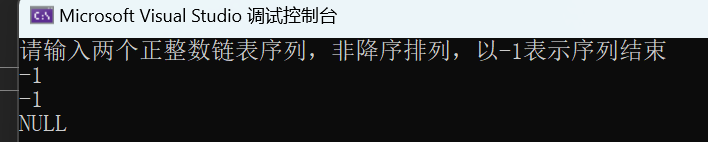
1.交集为空的情况



2.其中有一个为空集情况



3.两个都为空集的情况



4.感觉可以改进的地方

我觉得在遍历的算法时间复杂度上可以优化，现在的算法时间复杂度是o(n2)