

线性表实验报告

课程名称：数据结构与算法

学 院： 计算机与信息安全学院

专业班级：19003603

学 号：1900301517

姓 名：陆洪业

报告日期：2021年 01 月 10 日

目录

**[1. 求顺序线性表中连续子表（最少有一个元素）的最大和](#_Toc285254522_WPSOffice_Level1)** **[1](#_Toc285254522_WPSOffice_Level1)**

[1.1. 实验要求](#_Toc236156139_WPSOffice_Level2) [1](#_Toc236156139_WPSOffice_Level2)

[1.2. 相关代码](#_Toc1877410202_WPSOffice_Level2) [1](#_Toc1877410202_WPSOffice_Level2)

[1.3. 系统测试与结果](#_Toc1609705657_WPSOffice_Level2) [5](#_Toc1609705657_WPSOffice_Level2)

[1.4. 实验总结](#_Toc939456515_WPSOffice_Level2) [5](#_Toc939456515_WPSOffice_Level2)

**[2. 寻找缺失的第一个正数](#_Toc236156139_WPSOffice_Level1)** **[6](#_Toc236156139_WPSOffice_Level1)**

[2.1. 实验要求](#_Toc426281906_WPSOffice_Level2) [6](#_Toc426281906_WPSOffice_Level2)

[2.2. 实验代码](#_Toc2125679673_WPSOffice_Level2) [6](#_Toc2125679673_WPSOffice_Level2)

[2.3. 系统测试与结果](#_Toc158668215_WPSOffice_Level2) [7](#_Toc158668215_WPSOffice_Level2)

[2.4. 实验总结](#_Toc1025272159_WPSOffice_Level2) [8](#_Toc1025272159_WPSOffice_Level2)

**[3. 在有序的顺序线性表中找目标值出现的第一个和最后一个位置](#_Toc1877410202_WPSOffice_Level1)** **[8](#_Toc1877410202_WPSOffice_Level1)**

[3.1. 实验要求](#_Toc1997310754_WPSOffice_Level2) [8](#_Toc1997310754_WPSOffice_Level2)

[3.2. 实验代码](#_Toc348384930_WPSOffice_Level2) [8](#_Toc348384930_WPSOffice_Level2)

[3.3. 系统检测与结果](#_Toc521471381_WPSOffice_Level2) [11](#_Toc521471381_WPSOffice_Level2)

[3.4. 实验总结](#_Toc2037816043_WPSOffice_Level2) [12](#_Toc2037816043_WPSOffice_Level2)

**[4. 一元多项式相加](#_Toc1609705657_WPSOffice_Level1)** **[12](#_Toc1609705657_WPSOffice_Level1)**

[4.1. 实验目的](#_Toc1627006098_WPSOffice_Level2) [12](#_Toc1627006098_WPSOffice_Level2)

[4.2. 程序设计概要](#_Toc1573435854_WPSOffice_Level2) [12](#_Toc1573435854_WPSOffice_Level2)

[4.3. 系统测试与结果](#_Toc580643715_WPSOffice_Level2) [18](#_Toc580643715_WPSOffice_Level2)

[4.4. 实验小结](#_Toc585425990_WPSOffice_Level2) [18](#_Toc585425990_WPSOffice_Level2)

**[5. 利用单链表实现集合运算](#_Toc939456515_WPSOffice_Level1)** **[18](#_Toc939456515_WPSOffice_Level1)**

[5.1. 实验目的](#_Toc1746830539_WPSOffice_Level2) [18](#_Toc1746830539_WPSOffice_Level2)

[5.2. 程序设计概要](#_Toc1728458845_WPSOffice_Level2) [19](#_Toc1728458845_WPSOffice_Level2)

[5.3. 系统测试与结果](#_Toc1570800127_WPSOffice_Level2) [24](#_Toc1570800127_WPSOffice_Level2)

[5.4. 实验小结](#_Toc815694573_WPSOffice_Level2) [24](#_Toc815694573_WPSOffice_Level2)

**[6. 大数相加](#_Toc426281906_WPSOffice_Level1)** **[25](#_Toc426281906_WPSOffice_Level1)**

[6.1. 实验目的](#_Toc209735999_WPSOffice_Level2) [25](#_Toc209735999_WPSOffice_Level2)

[6.2. 程序设计概要](#_Toc192306863_WPSOffice_Level2) [25](#_Toc192306863_WPSOffice_Level2)

[6.3. 系统测试与结果](#_Toc2042455129_WPSOffice_Level2) [29](#_Toc2042455129_WPSOffice_Level2)

[6.4. 实验小结](#_Toc1688454942_WPSOffice_Level2) [29](#_Toc1688454942_WPSOffice_Level2)

# 求顺序线性表中连续子表（最少有一个元素）的最大和

## 实验要求

输入若干个整型数据（可能有正数和负数），存放在顺序表中，求顺序表中连续子表（最少有一个元素）的最大和。

提升要求：时间复杂度不超过O(n)

## 相关代码

#include <stdio.h>

#include <stdlib.h>

/\*此处是顺序线性表数据结构定义\*/

typedef int DataType;

struct seqList

{//有3个数据成员

int MAXNUM;//用于记录顺序线性表中能存放的最大元素个数的 整型 MAXNUM

int curNum;//用于存放顺序线性表中数据元素的个数 整型 curNum

DataType \*element;//用于存放顺序线性表数据元素的连续空间的起始地址

};

typedef struct seqList \*PseqList;

/\*创建空的顺序线性表，能存放的最大元素个数为 m\*/

PseqList createNullList\_seq(int m)

{ //若m=0，则返回NULL

PseqList plist = (struct seqList \*)malloc(sizeof(struct seqList));

if(plist == NULL) return NULL; //分配空间失败

plist->MAXNUM = m ;

plist->curNum = 0;

plist->element = (DataType \*)malloc(sizeof(DataType)\*m);

if(plist->element == NULL)

{

free(plist);

return NULL;

}

return plist;

}

/\*在线性表表尾插入数据元素,返回值0表示插入失败，返回值1表示在表尾插入成功\*/

int insertP\_tail(PseqList plist , int x)

{

if(plist->curNum == plist->MAXNUM) //若表满，则无法插入

{

printf("list if full !");

return 0;

}

plist->element[plist->curNum] = x ;

plist->curNum++;

return 1;

}

/\*回收线性表占用的空间\*/

int destroyList\_seq(PseqList plist)

{

//返回值为销毁的线性表中现有数据元素的个数，若待销毁的线性表不存在，则返回0

if(plist == NULL) return 0;

int m = plist->curNum;

free(plist->element);

free(plist);

return m;

}

void printList\_seq(PseqList plist)

{//逐个输出线性表的元素，相邻的两个数据元素之间以一个空格为分隔符隔开

for(int i=0;i<plist->curNum;i++)

printf("%d ",plist->element[i]);

}

//第一关:求顺序线性表中连续子表（最少有一个元素）的最大和并输出

int seqMaxSum(PseqList plist)

{

int max = 0, sum = 0;

int i;

for (i = 0; i < plist->curNum; ++i) {

if (sum < 0)

sum = 0;

sum += plist->element[i];

if (sum > max)

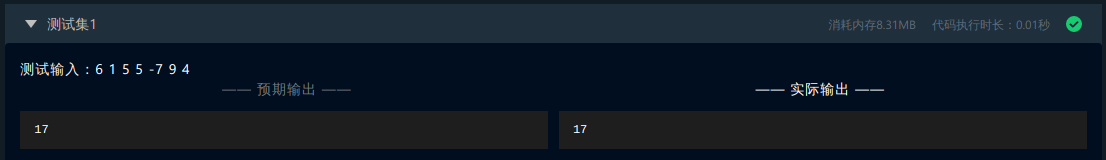
max = sum;

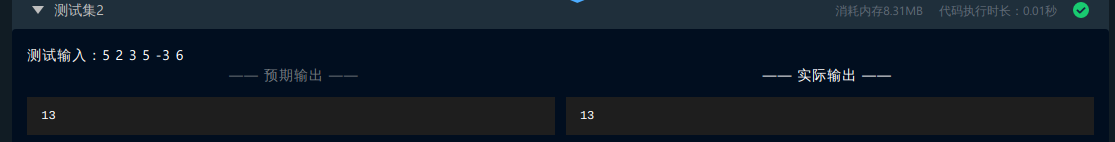
}

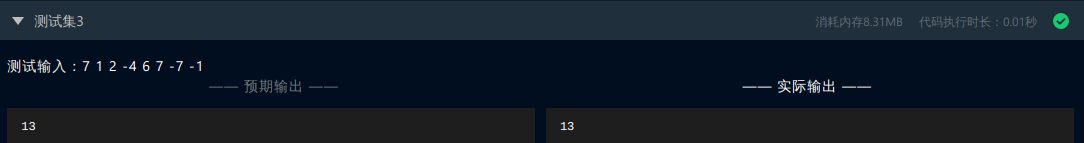
return max;

}

## 系统测试与结果







复杂度满足要求, 并且通过了系统测试, 所以实现了复杂度不超过O(n)顺序线性表中连续子表（最少有一个元素）的最大和.

## 实验总结

本次实验, 我完成了提升任务, 在完成过程中, 我学会了一些降低时间复杂度的算法.

# 寻找缺失的第一个正数

## 实验要求

给定含n(n>=1)个未排序的整数，放在顺序线性表中，请设计一个在时间上尽可能高效的算法(算法时间复杂度O(n) )，找出其中没有出现的最小的正整数。

例如，数组{-5，3，2，3}中未出现的最小正整数是1；数组{1，2，3}中未出现的最小正整数是4。

提升要求：时间复杂度不超过O（n)

## 实验代码

//第二关：寻找线性表中没有出现的最小的正整数

int findMinNumber(PseqList plist)

{

//若线性表为空，则返回0

int i = 0, tmp = 0;

while (i < plist->curNum) {

if (plist->element[i] <= 0 || plist->element[i] > plist->curNum) {

plist->element[i] = plist->element[--plist->curNum];

} else if (plist->element[i] != i+1 && plist->element[plist->element[i]-1] != plist->element[i]) {

tmp = plist->element[plist->element[i]-1];

plist->element[plist->element[i]-1] = plist->element[i];

plist->element[i] = tmp;

} else {

i++;

}

}

for (i = 0; i < plist->curNum; ++i){

if (plist->element[i] != i+1)

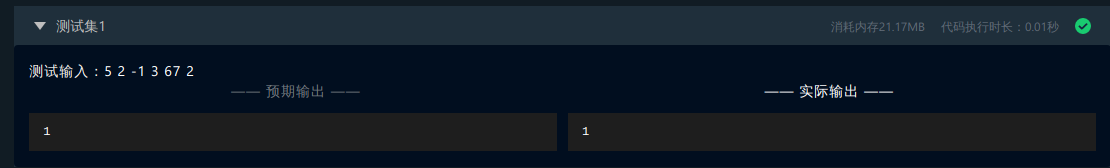
break;

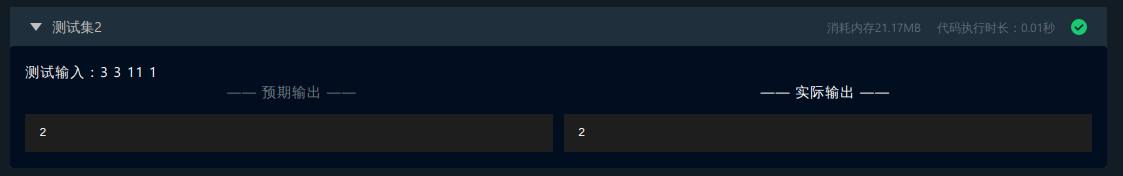
}

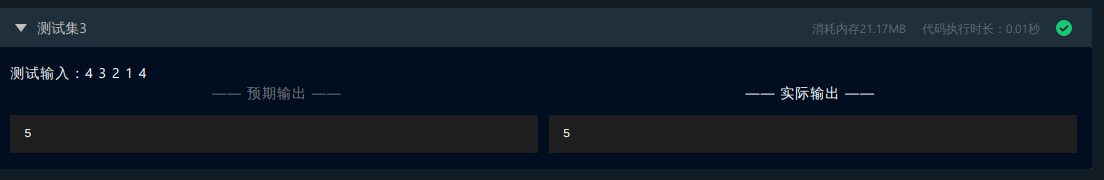
return i+1;

}

## 系统测试与结果







复杂度为O(n)且通过系统测试, 所以完成了在复杂度O(n)之内寻找缺失的第一个正数的任务.

## 实验总结

通过本次实验, 我更进一步了解了顺序线性表, 当严格按照顺序表的排列顺序的话, 删除或者插入一个元素及其麻烦, 但当不要求保持原顺序, 删除一个元素只需要将最后一个元素来代替要删除的元素即可.

# 在有序的顺序线性表中找目标值出现的第一个和最后一个位置

## 实验要求

给定一个按照升序排列的整数顺序线性表 ，和一个目标值 target。找出给定目标值在顺序表中的开始位置和结束位置。

提升要求：算法时间复杂度不超过O(log n)。

## 实验代码

//第三关：找出给定目标值target在有序线性表中出现的起始位置和结束位置

void findPos(PseqList plist,int target, int \*pos)

{

//起始位置放在pos[0], 结束位置放在pos[1]

// 先找到这个元素在哪个范围里面.

int mid, start, end;

int left = 0, right = plist->curNum - 1;

int start\_l, start\_r, end\_l, end\_r;

int flag = 0;

if (target < plist->element[0] || target > plist->element[plist->curNum-1]) {

pos[0] = -1;

pos[1] = -1;

return;

}

while (left <= right) {

mid = (left + right)/2;

if (target < plist->element[mid]) {

right = mid - 1;

} else if (plist->element[mid] < target) {

left = mid + 1;

} else {

flag = 1;

break;

}

}

start\_l = left;

start\_r = start = mid;

while (start\_l != start\_r) {

start = (start\_l + start\_r)/2;

if (target < plist->element[start]) {

start\_r = start - 1;

} else if (plist->element[start] < target) {

start\_l = start + 1;

} else {

start\_r = start;

}

}

end\_l = end = mid;

end\_r = right;

while (end\_l != end\_r) {

end = (end\_l + end\_r)/2;

if (target < plist->element[end]) {

end\_r = end - 1;

} else if (plist->element[end] < target) {

end\_l = end + 1;

} else {

end\_l = end;

if (end\_l + 1 == end\_r && plist->element[end\_r]==target) {

end = end\_r;

break;

}

}

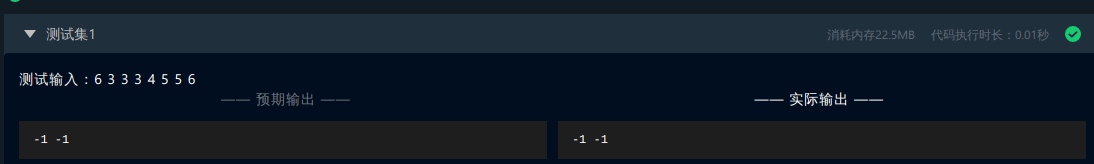
}

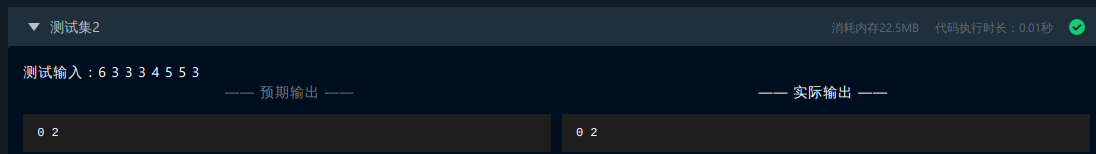
pos[0] = flag == 1?start\_l:-1;

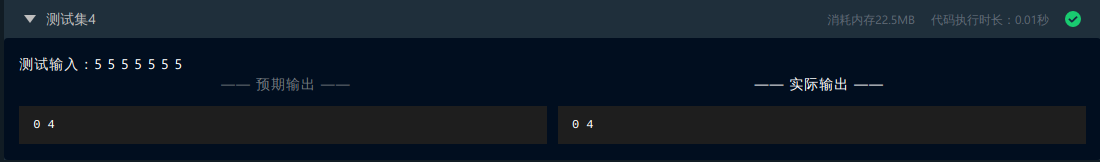
pos[1] = flag == 1?end\_r:-1;

}

## 系统检测与结果







复杂度为O(logn), 且通过系统测试, 所以完成了复杂度要求内在有序的顺序线性表中找目标值出现的第一个和最后一个位置的位置.

## 实验总结

通过这次实验, 我对顺序表和二分查找法有了进一步的了解, 可以灵活运用二分查找来满足不同的要求, 从而在较低的复杂度之内完成任务.

# 一元多项式相加

## 实验目的

设计一种单链表存储结构，每个结点存储一项的系数和指数，类型都是整型，编写完成产生多项式的函数、多项式相加及输出多项式的函数。

## 程序设计概要

### 设计目标

设计一种单链表存储结构，每个结点存储一项的系数和指数，类型都是整型，编写完成产生多项式的函数、多项式相加及输出多项式的函数。

### 选用储存结构

链式储存

### 有关类型和函数的定义

存放多项式某项的结点结构:

struct node

{

int exp ; // 表示指数

int coef ; //表示系数

struct node \*next; //指向下一个结点的指针

};

typedef struct node \* PNODE ;

函数的定义:

(1)PNODE createPoly(void)

函数功能：生成多项式

函数名：createPoly

函数参数：无

返回值：指向多项式的头指针

(2)PNODE addPoly(PNODE polyAddLeft , PNODE polyAddRight)

函数功能：进行多项式相加

函数名：addPoly

函数参数：polyAddLeft ：加法左边多项式头指针, polyAddRight：加法右边多项式头指针

返回值：指向结果多项式的头指针

(3) void printPoly(PNODE poly)

函数功能：输出多项式

函数名：printPoly

函数参数：待输出多项式的头指针poly

返回值：无

### 数据结构的设计以及函数的设计

一个链表就是一个多项式, 链表的节点就是多项式的项, 系数为零那么该项对应的节点就不存在.

1. PNODE createPoly(void)实现多项式的创建

原理:(1)给节点分配空间(2)输入系数和指数(3)判断系数是否为0, 为0就停止输入(4)将系数和指数保存到节点中,并把节点添加到链表上

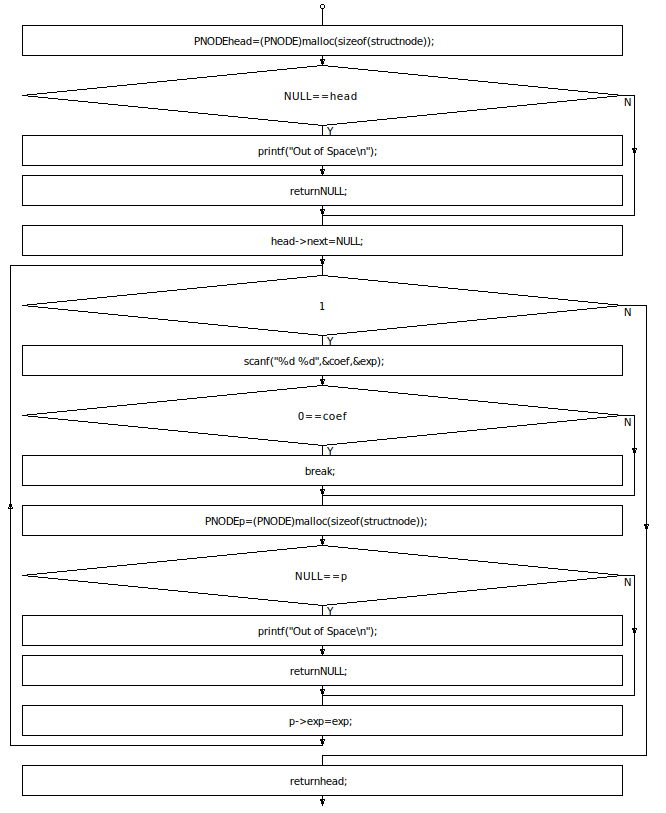


图 1creatPoly

1. PNODE addPoly(PNODE polyAddLeft , PNODE polyAddRight)实现多项式的相加

原理:(1)如果指数为i的项都不为0, 那么和为两相系数和(2)如果指数为i的项有其中一项为0, 那么和就为另一项系数

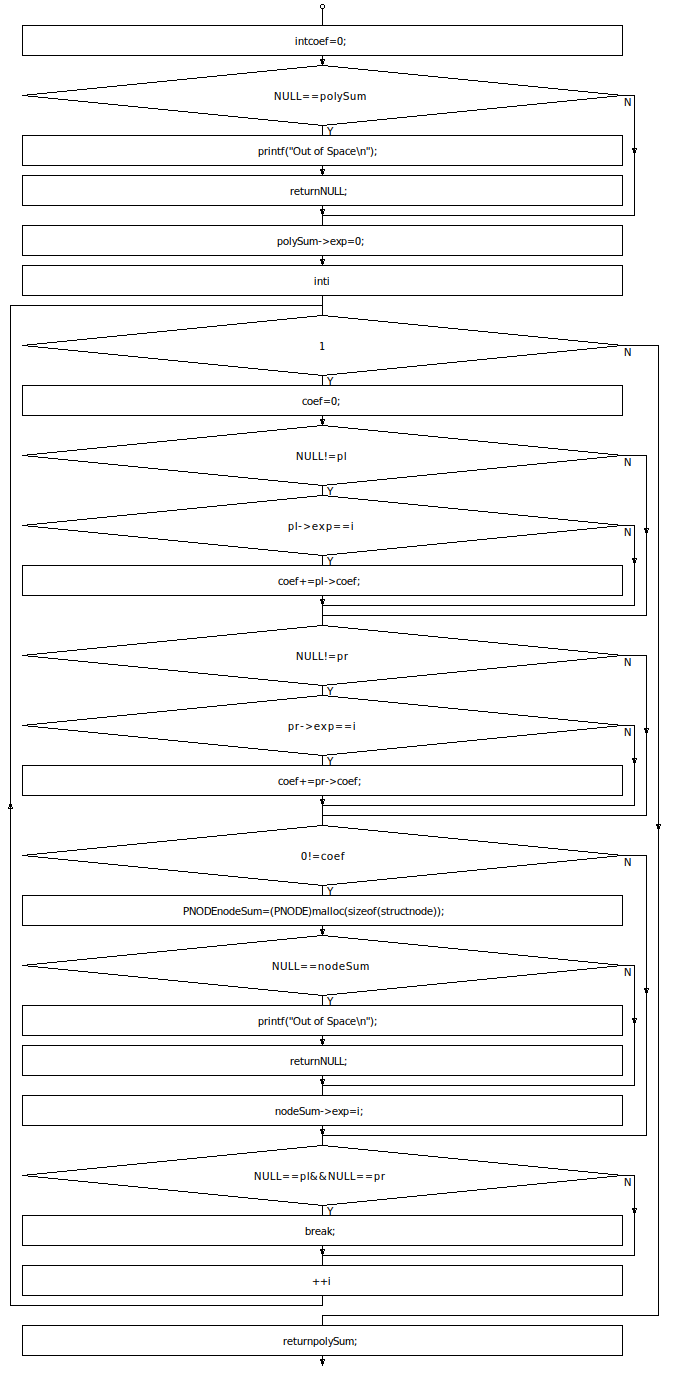


图 2addPoly

1. void printPoly(PNODE poly)实现多项式的输出

原理: 遍历输出

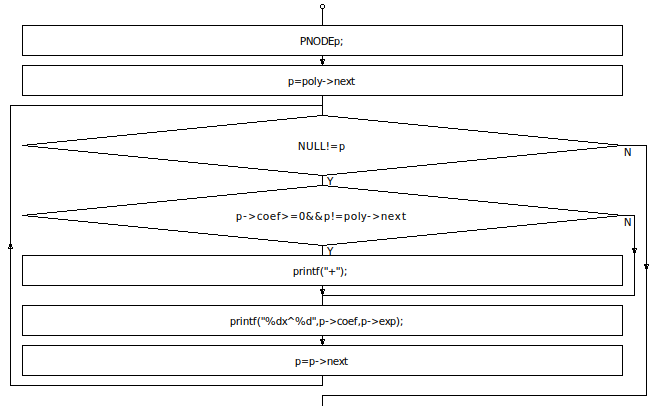


图 3printPoly

### 复杂度的分析

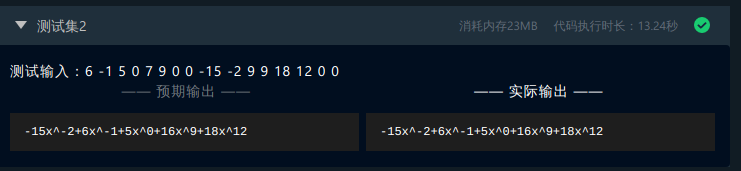
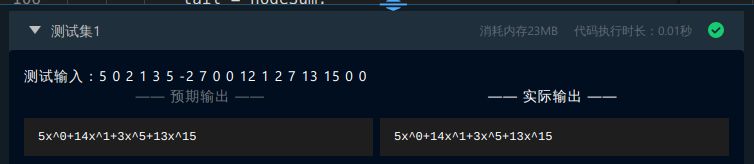
链表插入时间复杂度为O(1). 那么createPoly创建n个多项式, 循环了n次,则复杂度为O(n)

addPoly每次都会相加一个指数的项, 每相加一项复杂度为O(1), 那么相加n项复杂度就为O(n)

printPoly就是遍历链表, 复杂度为O(n)

|  |  |  |
| --- | --- | --- |
| createPoly | addPoly | printPoly |
| O(n) | O(n) | O(n) |

## 系统测试与结果



上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了多项的加,而且成功地输出多相式

## 实验小结

通过本次实验, 我对链表的操作更加熟悉, 了解了链表的优缺点, 能懂得什么时候选用链表.链式访问会比顺序访问慢, 但是链表插入会比顺序表方便, 顺序表插入需要将插入点的元素全都往后挪动, 链表就解决了这个问题.

# 利用单链表实现集合运算

## 实验目的

完成单链表基本操作，并使用基本操作实现集合的并、交和差功能。

## 程序设计概要

### 设计目标

设计一个单链表, 每个节点储存一个集合元素, 编写完成初始化集合和函数, 能复制集合的函数, 能求并集的函数, 能求差集的函数和输出集合的函数

### 选用储存结构

链式储存

### 有关类型和函数的定义

typedef int DataType;

struct node

{

DataType element;

struct node \*next;

};

typedef struct node \* SET;

void insert(DataType datax, SET set);

/\*

函数名： InitSet

函数功能：根据参数num，初始化集合

函数参数：集合元素的个数

返回值：集合头指针

\*/

SET InitSet(int num)

/\*

函数名： printSet

函数功能：输出集合的元素，以空格作为元素之间分界符

函数参数：set的头结点

返回值：无

\*/

void printSet(SET set)

/\*

函数名： setUnion

函数功能：求两个集合setA 和 setB的并集

函数参数：setA和setB的头结点

返回值：并集集合的头结点

\*/

SET setUnion(SET setA ,SET setB)

/\*

函数名： setIntersect

函数功能：求两个集合setA 和 setB的交集

函数参数：setA和setB的头结点

返回值：交集集合的头结点

\*/

SET setIntersect(SET setA ,SET setB)

/\*

函数名： setExcept

函数功能：求两个集合setA 和 setB的差

函数参数：setA和setB的头结点

返回值：结果集合的头结点

\*/

SET setExcept(SET setA ,SET setB)

### 数据结构的设计以及函数的设计

SET InitSet(int num)

原理: 循环输入集合元素, 并插入到链表中

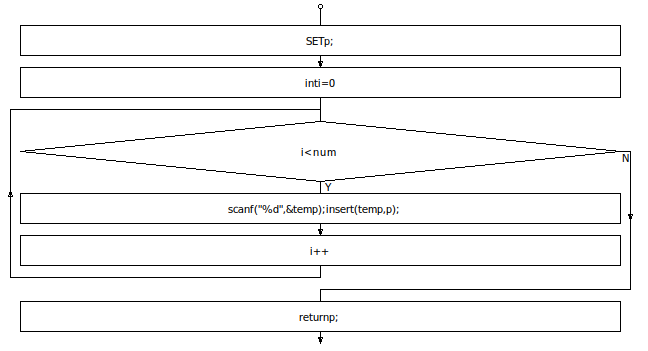


表 1initSet

void printSet(SET set)

原理: 遍历输出

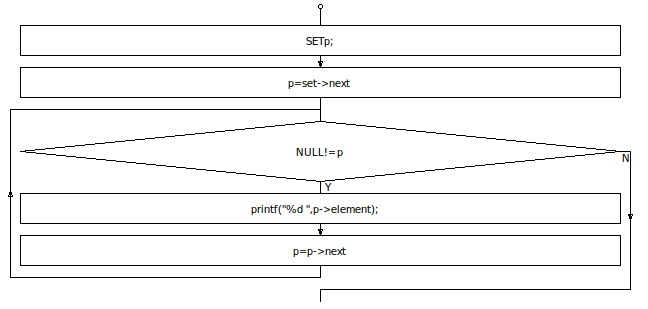


表 2printSet

SET setIntersect(SET setA ,SET setB)

原理: (1)将A复制到C中(2)将B复制到C中

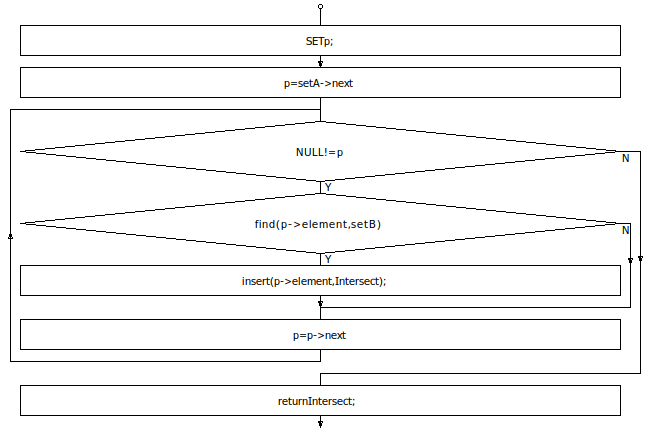


表 3setIntersect

SET setExcept(SET setA ,SET setB)

原理: (1)判断A中欲插入元素在B是否存在(2)不存在就将节点插入

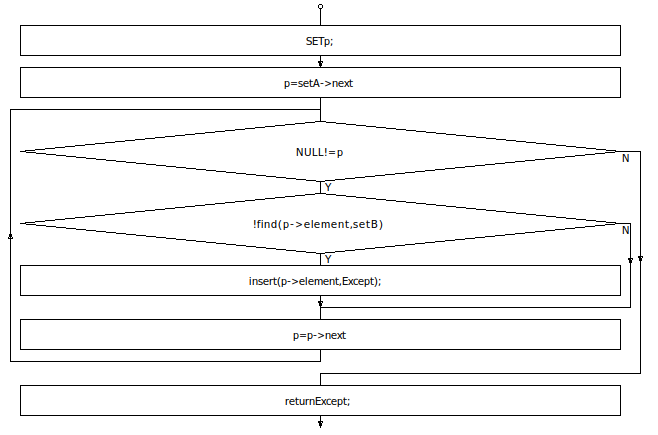


表 4setExcept

### 复杂度的分析

每插入一个元素复杂度为O(1),InitSet n次循环输入元素,并将元素插入到链表中, 复杂度为O(n)

printSet遍历链表, 复杂度为O(n)

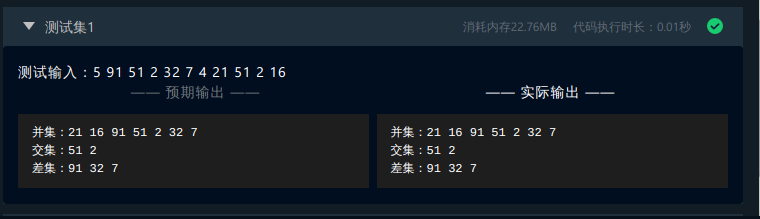
setUnion大循环复杂度O(n),每次循环都判断集合中是否存在同样的元素, 复杂度为O(n), 所以setUnion复杂度为O(n^2)

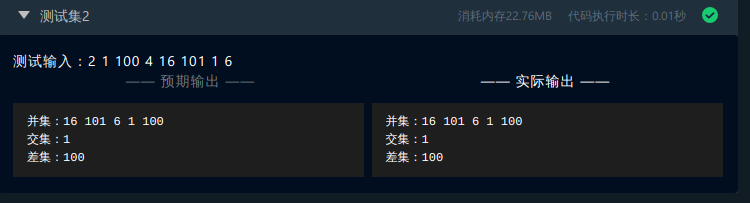
setInterect大循环复杂度O(n),每次循环都判断集合中是否存在同样的元素, 复杂度为O(n), 所以setInterect复杂度为O(n^2)

setExcept大循环复杂度O(n),每次循环都判断集合中是否存在同样的元素, 复杂度为O(n), 所以setExcept复杂度为O(n^2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| InitSet | printSet | setUnion | setInterect | setExcept |
| O(n) | O(n) | O(n^2) | O(n^2) | O(n^2) |

## 系统测试与结果





上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了集合的交并差,而且成功地输出集合

## 实验小结

通过这次实验, 我对链表又有了进一步的认识, 可以用链表建立一些数学模型.

# 大数相加

## 实验目的

在进行大数相加,大数就是超长位数（例如1000位的一个数字）

## 程序设计概要

### 设计目标

设计一个单链表用于储存大数, 一个链表就是一个数, 链表的每一个节点储存着每一位的数(int型), 设计一个初始化大数的函数, 一个输出大数的函数, 一个大数相加的函数.

### 选用储存结构

链式储存

### 有关类型和函数的定义

struct node

{//链表结点类型，包含一个存放整型数据的 data 成员，和一个指向下一个结点的next成员

int data ;

struct node \*next ;

};

void myprintList(struct node \*L)

/\*输出head为表头的链表中的数据元素，每输出一个数据空一格\*/

void genNumber( struct node \*A , int num)

//本函数用于接收输入的大数的各个位，返回大数链表表头，可使用上面已实现的链表插入函数

struct node \*addNumber(struct node \*A ,struct node \*B)

//此处实现函数求两数相加，并返回和值链表的表头；

### 数据结构的设计以及函数的设计

void myprintList(struct node \*L)

原理: 遍历输出链表

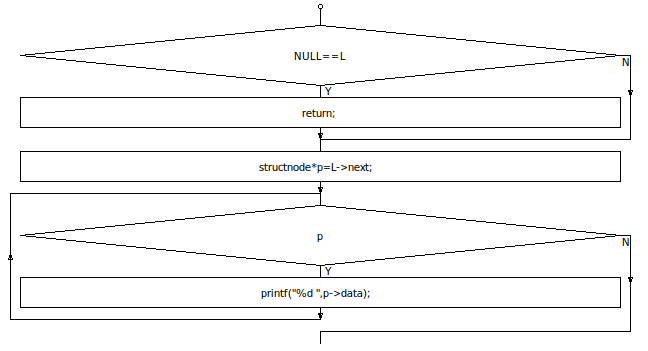


表 5myprintList

void getNumber( struct node \*A , int num)

原理: 循环读入数字, 并用头插法把数字插入到链表中

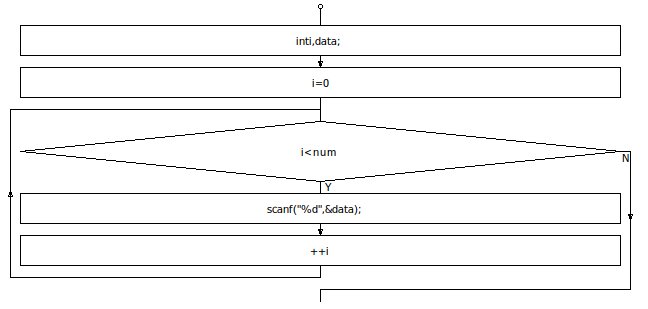


表 6getNumber

struct node \*addNumber(struct node \*A ,struct node \*B)

原理(1)每一项的加数有三个, AB两数和低位进位(2)从低位往高位加(3)AB的项都不为0, 那么和为两相系数和再加上低位的进位(4)AB有其中一项为0, 那么和就为另一项加上低位的进位(5)和超过10就将十位及以上的数进位为下一项的加数,这一项的和为余

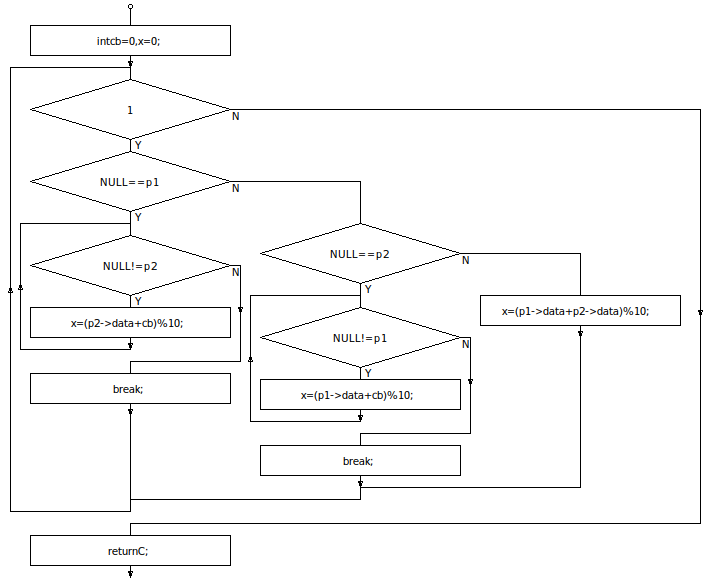


表 7addNumber

### 复杂度的分析

myprintList遍历链表, 复杂度为O(n)

getNumber循环头插法插入链表, 复杂度为O(n)

addNumber每次循环都会进行一位的相加, 每位的相加复杂度为O(1), 所以总复杂度为O(n)

|  |  |  |
| --- | --- | --- |
| myprintList | getNumber | addNumber |
| O(n) | O(n) | O(n) |

## 系统测试与结果

## 

上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了大数的相加,而且成功地输出大数

## 实验小结

通过本次实验, 我多懂了一种大数相加的方案——使用链表. 并且更深刻的认识到, 系统的数据类型是有位数上限的. 通过本次实验, 我懂得了系统数据类型不能满足需求时, 怎么使用数据结构设计一个自己的数据类型.