

链表实验报告

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. 一元多项式相加](#_Toc1514959104_WPSOffice_Level1)** **[1](#_Toc1514959104_WPSOffice_Level1)**

[1.1. 实验目的](#_Toc770049136_WPSOffice_Level2) [1](#_Toc770049136_WPSOffice_Level2)

[1.2. 程序设计概要](#_Toc144166176_WPSOffice_Level2) [1](#_Toc144166176_WPSOffice_Level2)

[1.3. 系统测试与结果](#_Toc1190084943_WPSOffice_Level2) [7](#_Toc1190084943_WPSOffice_Level2)

[1.4. 实验小结](#_Toc727815682_WPSOffice_Level2) [7](#_Toc727815682_WPSOffice_Level2)

**[2. 利用单链表实现集合运算](#_Toc770049136_WPSOffice_Level1)** **[7](#_Toc770049136_WPSOffice_Level1)**

[2.1. 实验目的](#_Toc842531121_WPSOffice_Level2) [7](#_Toc842531121_WPSOffice_Level2)

[2.2. 程序设计概要](#_Toc1190894746_WPSOffice_Level2) [8](#_Toc1190894746_WPSOffice_Level2)

[2.3. 系统测试与结果](#_Toc1993499711_WPSOffice_Level2) [13](#_Toc1993499711_WPSOffice_Level2)

[2.4. 实验小结](#_Toc1909512919_WPSOffice_Level2) [13](#_Toc1909512919_WPSOffice_Level2)

**[3. 大数相加](#_Toc144166176_WPSOffice_Level1)** **[14](#_Toc144166176_WPSOffice_Level1)**

[3.1. 实验目的](#_Toc1227799064_WPSOffice_Level2) [14](#_Toc1227799064_WPSOffice_Level2)

[3.2. 程序设计概要](#_Toc172513985_WPSOffice_Level2) [14](#_Toc172513985_WPSOffice_Level2)

[3.3. 系统测试与结果](#_Toc424129128_WPSOffice_Level2) [18](#_Toc424129128_WPSOffice_Level2)

[3.4. 实验小结](#_Toc48758845_WPSOffice_Level2) [18](#_Toc48758845_WPSOffice_Level2)

# 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

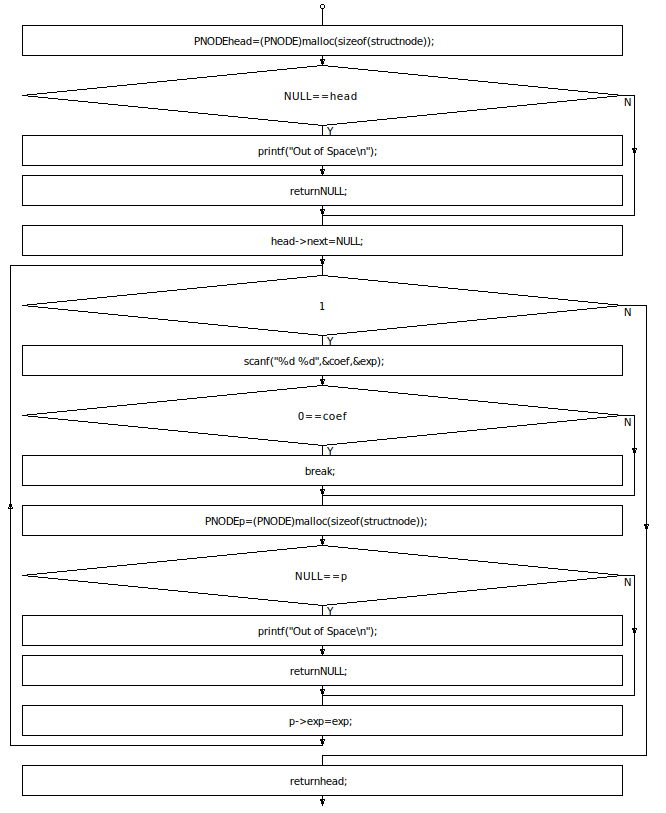


图 1creatPoly

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

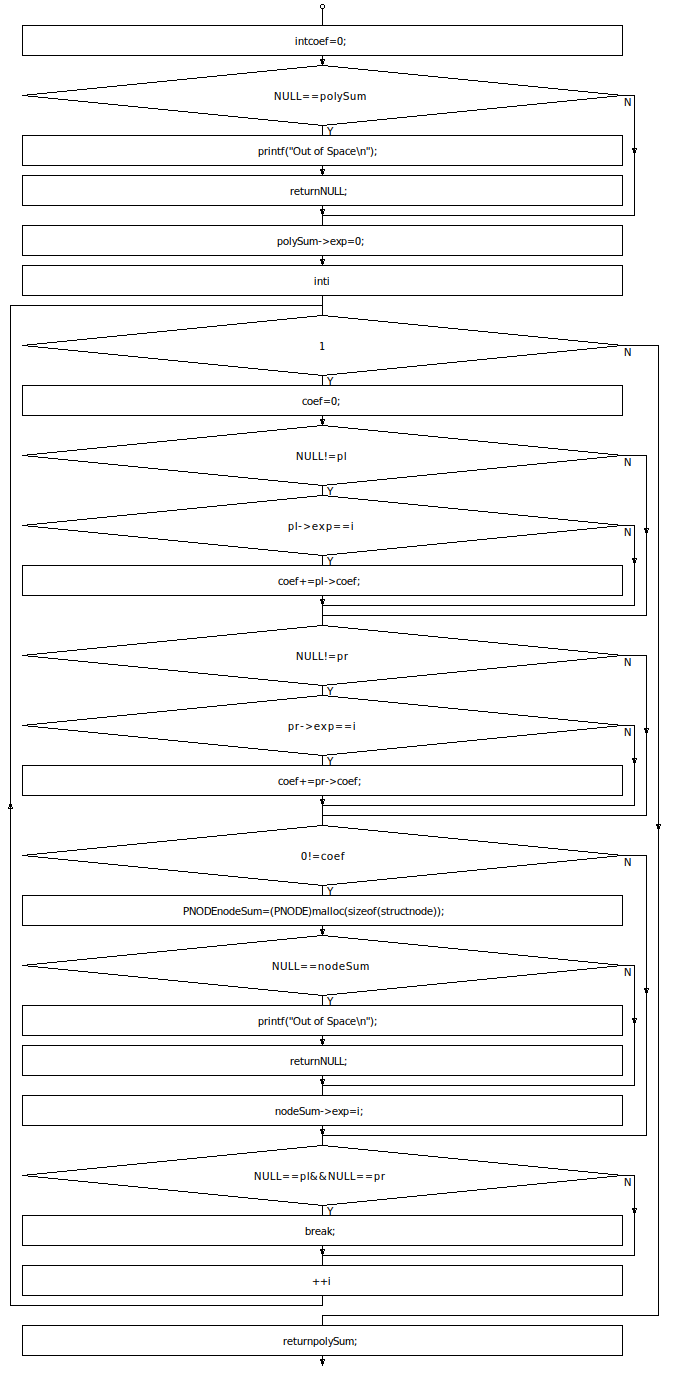


图 2addPoly

1. void printPoly(PNODE poly)实现多项式的输出

原理: 遍历输出

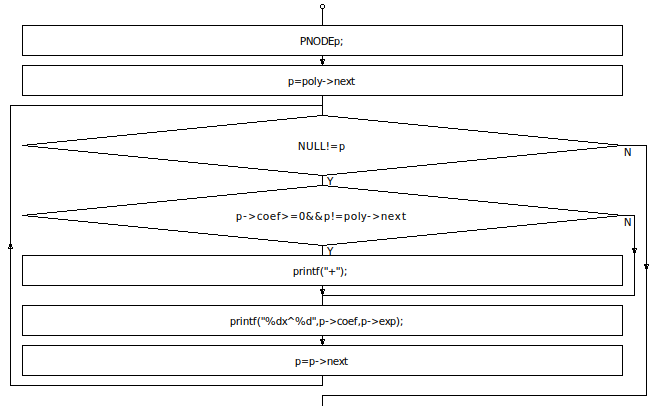


图 3printPoly

### 复杂度的分析

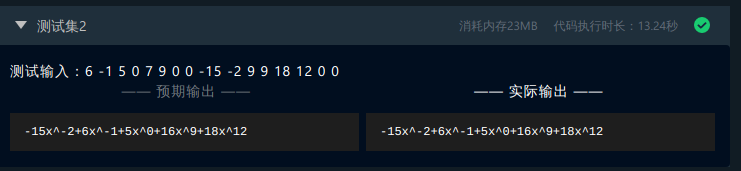
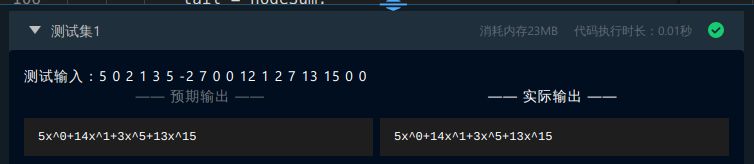
链表插入时间复杂度为O(1). 那么createPoly创建n个多项式, 循环了n次,则复杂度为O(n)

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

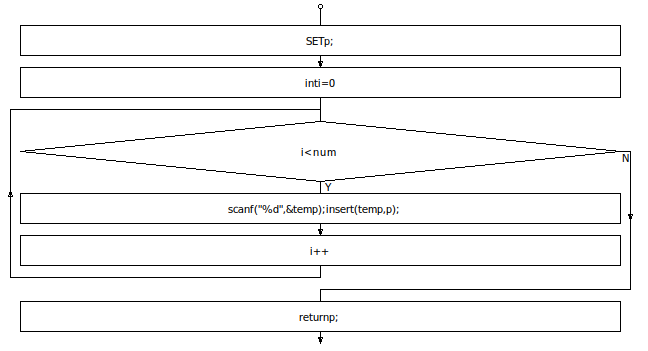


表 1initSet

void printSet(SET set)

原理: 遍历输出

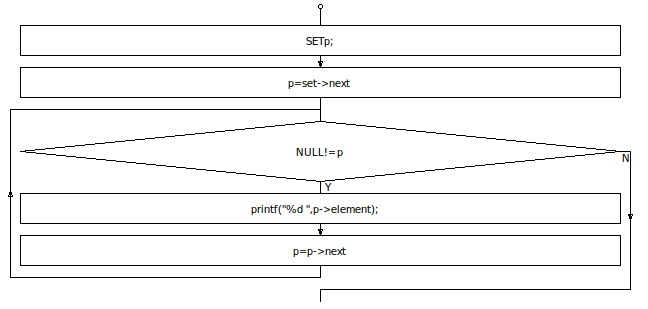


表 2printSet

SET setIntersect(SET setA ,SET setB)

原理: (1)将A复制到C中(2)将B复制到C中

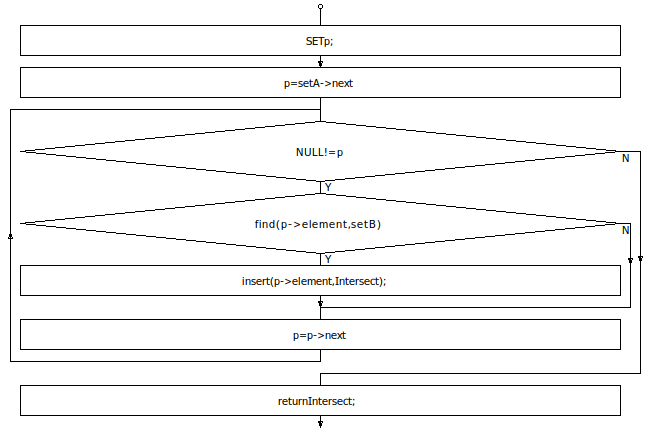


表 3setIntersect

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

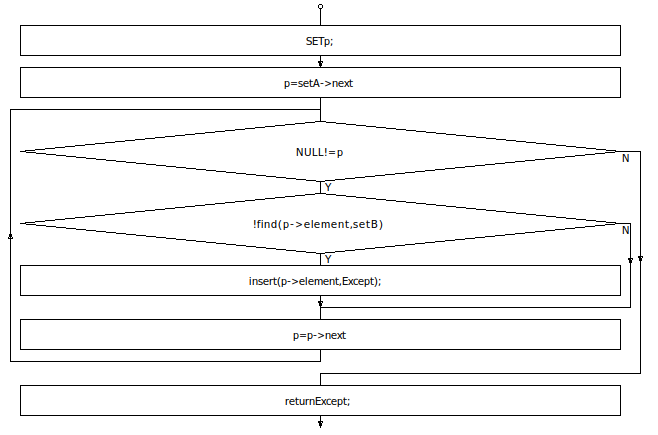


表 4setExcept

### 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)

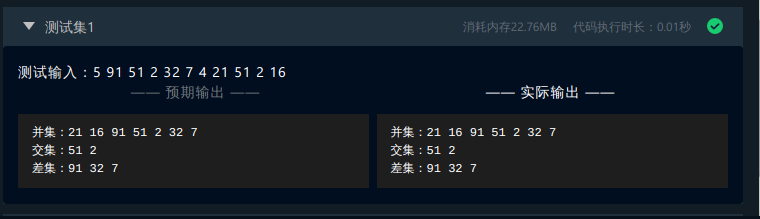
setUnion大循环复杂度O(n),每次循环都判断集合中是否存在同样的元素, 复杂度为O(n), 所以setUnion复杂度为O(n^2)

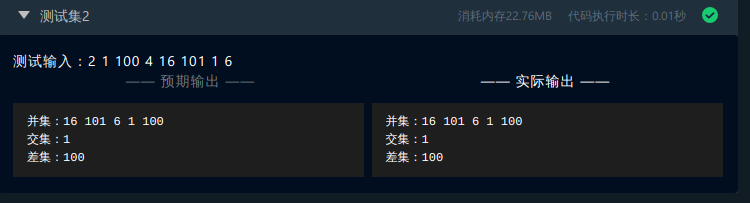
setInterect大循环复杂度O(n),每次循环都判断集合中是否存在同样的元素, 复杂度为O(n), 所以setInterect复杂度为O(n^2)

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

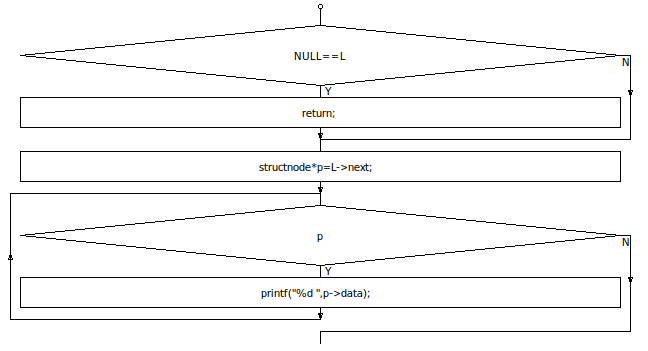


表 5myprintList

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

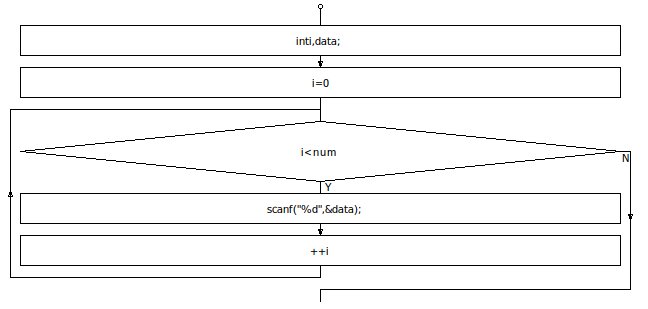


表 6getNumber

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

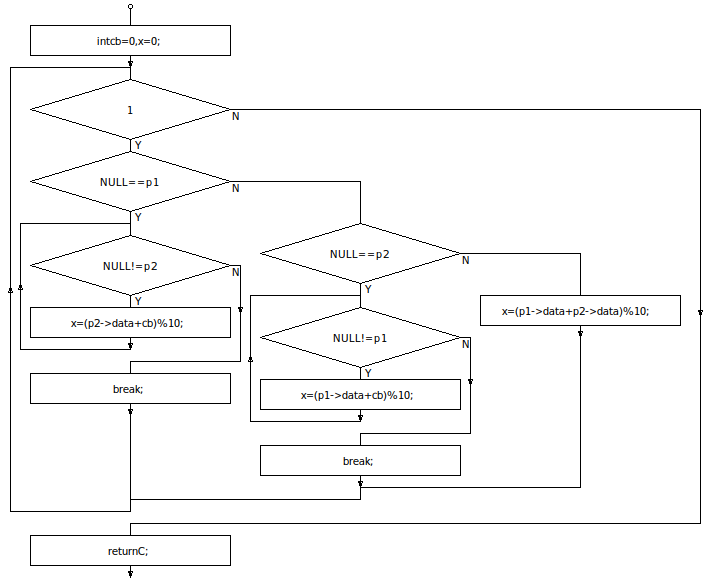


表 7addNumber

### 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.