**数据结构实验报告**

**实验题目：加里森任务**

|  |  |  |  |
| --- | --- | --- | --- |
| 班级：2020211305 | 姓名：缪奇志 | 学号：2020211345 | 分工：详细设计和调试分析报告 |
| 班级：2020211305 | 姓名：倪玮昊 | 学号：2020211346 | 分工：用户使用说明和测试结果 |
| 班级：2020211305 | 姓名：杨智杰 | 学号：2020211356 | 分工：需求分析和概要设计 |

1. **需求分析**

题目任务：

输入两个矩阵的三元表，判断两个矩阵是否能相加或者相乘，若能相加相乘则输出正确答案，若不能则输出不能相乘或者相加的原因。

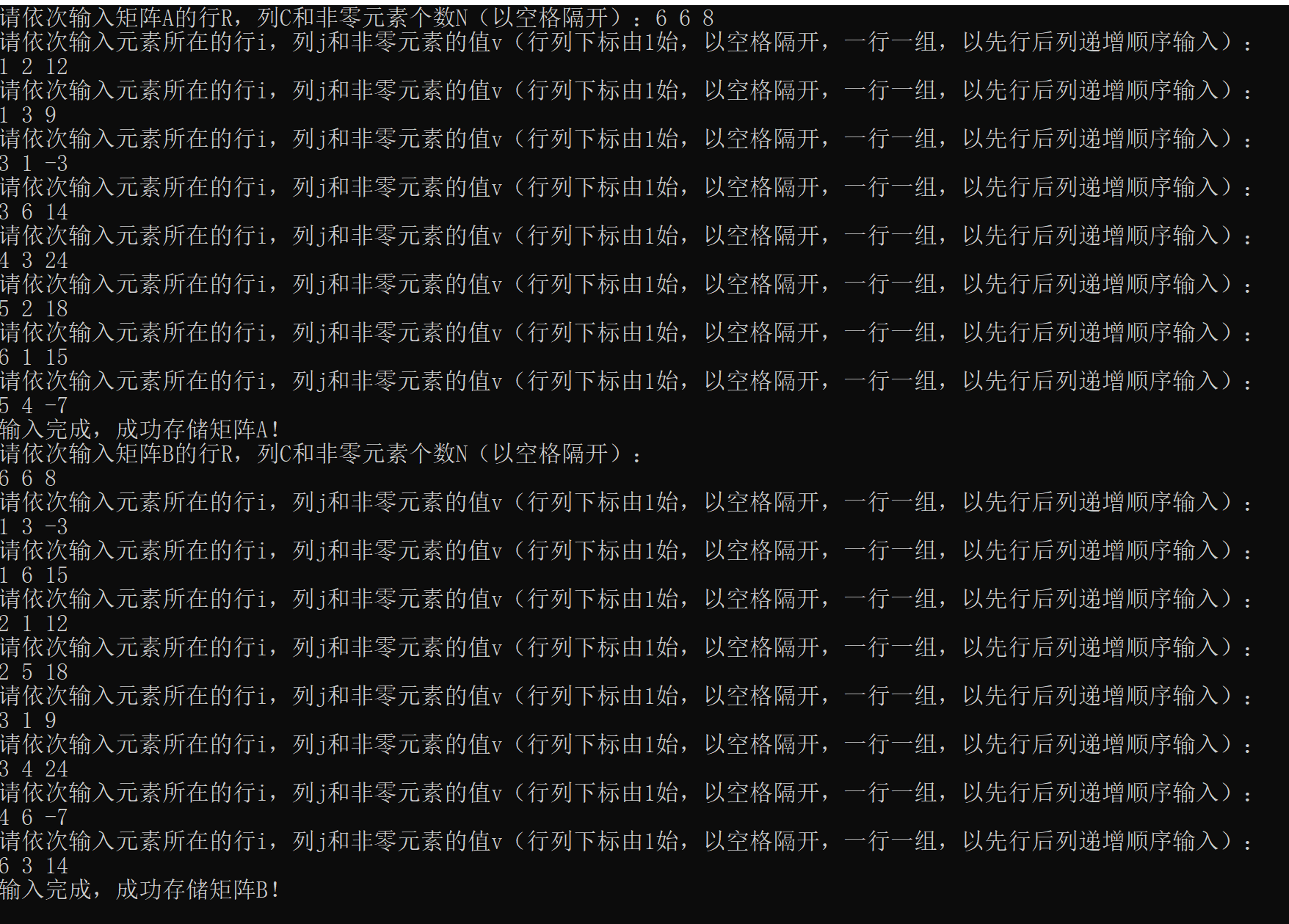
程序的输入和输出：

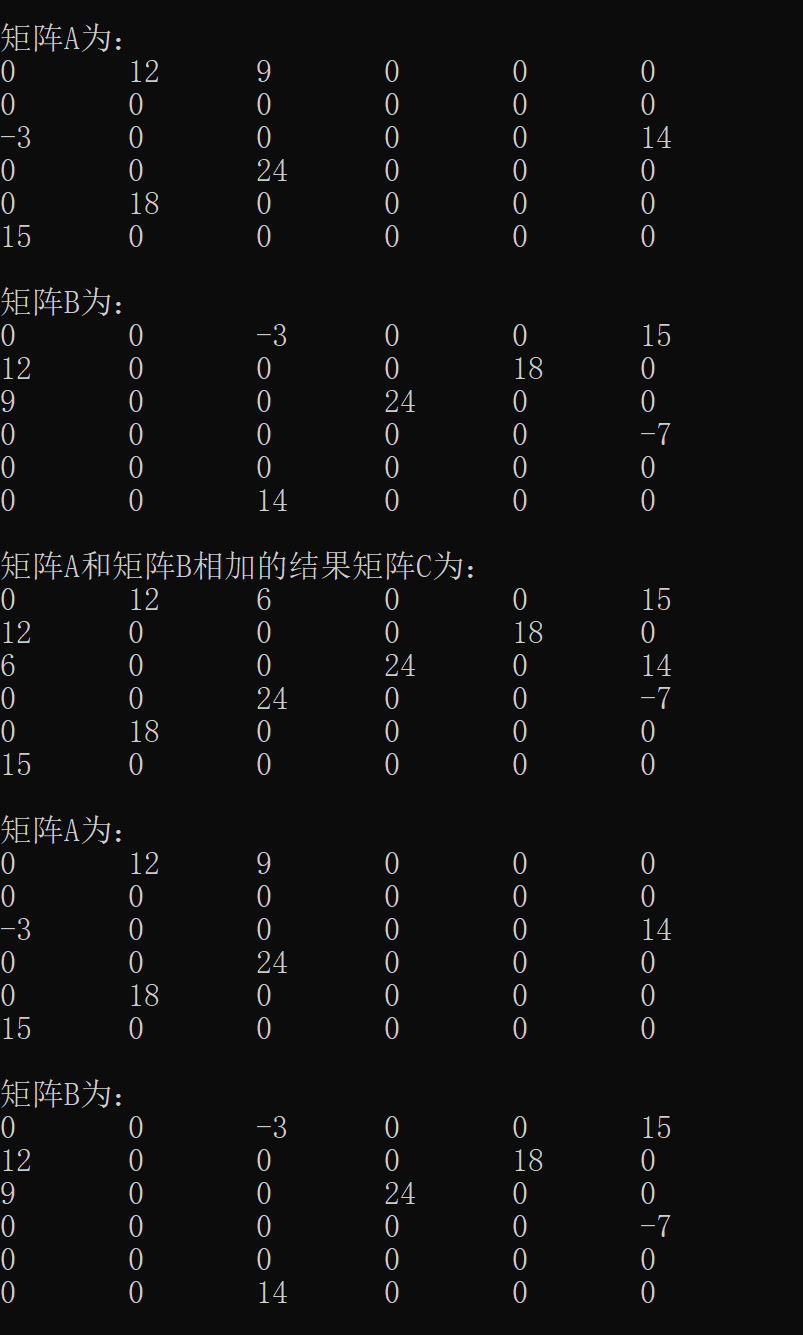
|  |  |  |  |
| --- | --- | --- | --- |
| 输入 | 数据类型 | 值的范围 | 代表意义 |
| M.row | 整型 | M.row>0 | 矩阵的行数 |
| M.column | 整型 | M.column>0 | 矩阵的列数 |
| M.num | 整型 | 整数 | 矩阵非0元素的个数 |
| ti | 整型 | ti>0 | 输入元素的行 |
| tj | 整型 | tj>0 | 输入元素的列 |
| tv | 整型 | 整数 | 输入元素的大小 |

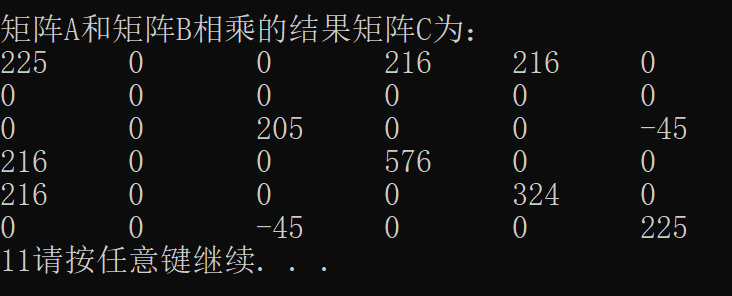
程序功能实现：

输入两个矩阵的三元表,若输入格式错误则退回重新输出，判断两个矩阵是否能相加或者相乘，若能相加相乘则输出经过相加或相乘之后的矩阵的三元表，若不能相加则输出不能相加的原因，如行列皆不相等，只有行不相等，只有列不相等.若不能相乘则停止。

正确输入：

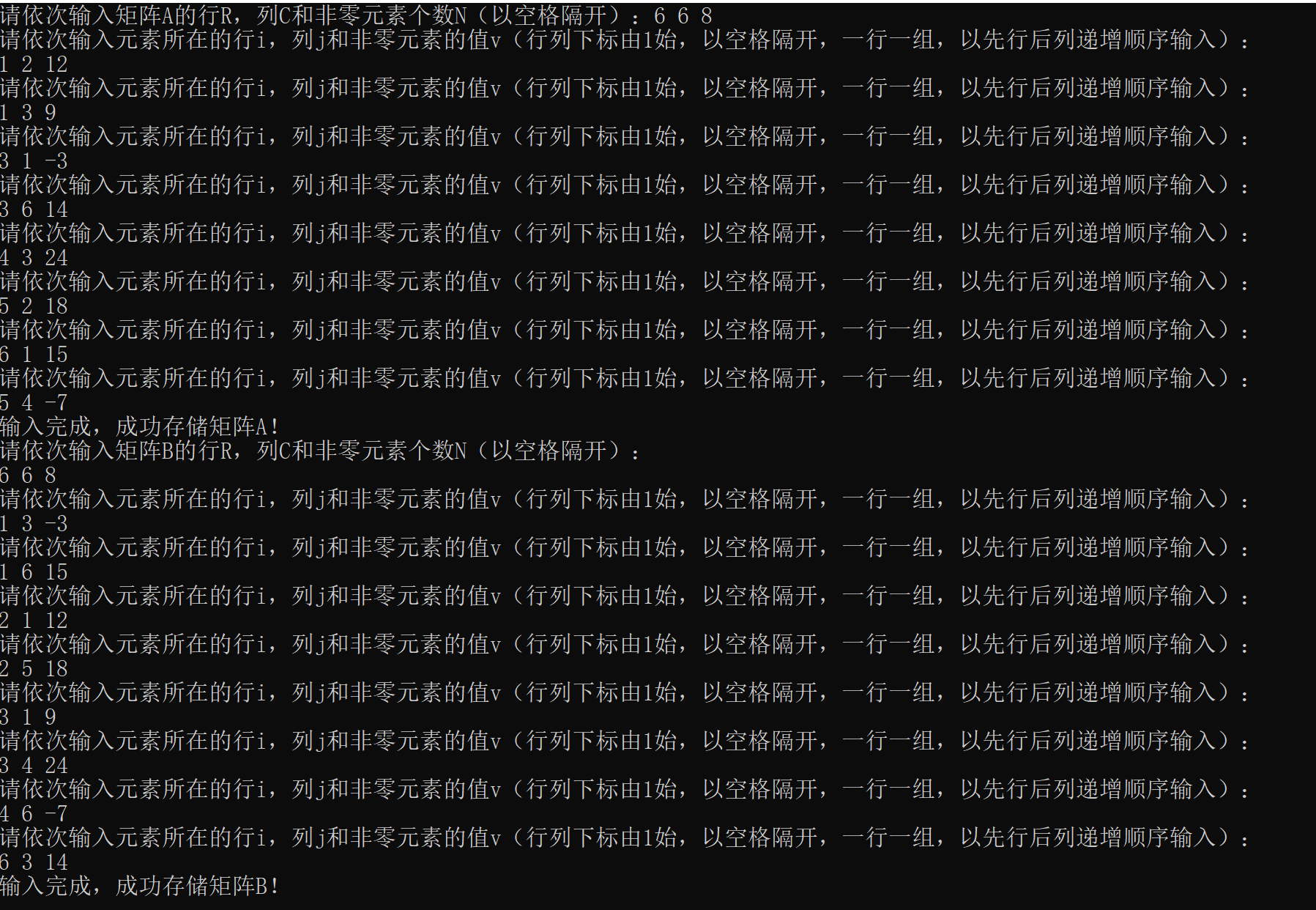






错误输入：





1. **概要设计**

**算法思路概述：**

本程序主要由四个主要的功能模块组成。

1. 模块一：完成稀疏矩阵三元组表的创建 思路：先创建好对应的数据结构，完成之后合理的利用该数据结构，依次输入需要录入的数据，即首先选择三元组表的行列以及非零数的个数，其次依次输入对应数据的值和对应在三元组表里的位置同时还完成对输入数据是否合法的判定。
2. 模块二：完成对两矩阵是否可以完成加法的判断，如果可以则把加法运算后的值存入到对应的结果矩阵。 思路：依次寻找元素，利用循环实现加法的运算同时判断操作是否合理，同时把结果存入加法运算结果矩阵。
3. 功能三：完成两矩阵是否可以进行乘法运算的判断，如果可以则把乘法运算后的值存入到对应的结果矩阵。思路：遍历寻找矩阵中的对应元素，同时设置一个新的数组用于储存乘法结果表里的对应每元素乘积和同时判断数据、操作是否合乎规定。
4. 功能四：完成对矩阵的打印。 思路：遍历数据结构中的每一个元素，实现对矩阵的打印。

**数据结构类型的定义：**

定义一个整型的变量elemtype用来表示三元组表中非零元素的个数。

typedef int elemtype;

定义结构体tuple用来表示一个三元组表。

struct tuple

{

int i; //总的行数

int j; //中的列数

elemtype v;

};

定义结构体sparsematrix用来作为三元组表中的元素

struct sparsematrix

{

int row; //元素对应的行

int column; //元素对应的列

int num; //元素对应的数据值

tuple data[MAXSIZE];

};

**主程序流程：**

InputMaxtrix(A, "A")

InputMaxtrix(B, "B")

if (MatrixAddition(A, B, C))

then printf("\n矩阵A为：\n")

PrintMatrix(A)

printf("\n矩阵B为：\n")

PrintMatrix(B)

printf("\n矩阵A和矩阵B相加的结果矩阵C为：\n")

PrintMatrix(C)

if (MatrixMultiplication(A, B, C))

then printf("\n矩阵A为：\n")

PrintMatrix(A)

printf("\n矩阵B为：\n")

PrintMatrix(B)

printf("\n矩阵A和矩阵B相乘的结果矩阵C为：\n")

PrintMatrix(C)

printf("%d", C.num)

system("pause")

return 0

**各层次模块层次调用关系：**

main先调用了InputMaxtrix构建了三个循环链表并对其进行输入，调用MatrixAddition函数完成矩阵的加法，调用MatrixMultiplication用于判断矩阵能否相乘并把运算结果储存到结果矩阵，MatrixMultiplication函数调用时可以调用SearchElement来进行寻找对应进行乘积求和运算，主程序中可调用PrintMatrix来实现需要矩阵的输出。

1. **详细设计**

**各种函数的设计：**

1.输入稀疏矩阵：

printf("请依次输入矩阵%s的行R，列C和非零元素个数N（以空格隔开）：", name)

while (scanf("%d%d%d", &(M.row), &(M.column), &(M.num)) != 3 || M.row <= 0 || M.column <= 0 || M.num < 0 || M.row \* M.column < M.num)

do

printf("输入错误，请重新输入：")

int pos  <- 0

int ti, tj

elemtype tv

while (pos != M.num)

do

printf("请依次输入元素所在的行i，列j和非零元素的值v（行列下标由1始，以空格隔开，一行一组以先行后列递增顺序输入）：\n")

flag <- scanf("%d%d%d", &ti, &tj, &tv)

while (getchar() != &apos;\n&apos;) // 清空缓冲区

do ontinue

while (flag != 3 || ti < 1 || ti > M.row || tj < 1 || tj > M.column || tv == 0)

do printf("输入错误，请从上一行的数据开始重新往下输入：\n");

flag = scanf("%d%d%d", &ti, &tj, &tv);

while (getchar() != &apos;\n&apos;) // 清空缓冲区

do continue

(M.data[pos]).i  <- ti;

(M.data[pos]).j <- tj;

(M.data[pos]).v  <- tv;

pos<-=pos+1

printf("输入完成，成功存储矩阵%s！\n", name)

2.判断矩阵是否能相加，如果能则将结果存储在sum中：

printf("\n")

if (M1.row != M2.row && M1.column != M2.column)

then printf("二者行列均不同，无法相加!\n")

return false

else if(M1.row != M2.row && M1.column == M2.column)

then printf("二者列相同但行不同，无法相加!\n")

return false

else if (M1.row == M2.row && M1.column != M2.column)

then printf("二者行相同但列不同，无法相加!\n")

return false

sum.row  <- M1.row

sum.column  <- M1.column

sum.num  <- 0

int M1\_pos  <-0, M2\_pos  <- 0

while (M1\_pos != M1.num && M2\_pos != M2.num)

do tuple T1 = M1.data[M1\_pos], T2 = M2.data[M2\_pos];

if ((T1.i < T2.i) || (T1.i == T2.i && T1.j < T2.j)) // M1的元素T1次序靠前

then sum.data[sum.num]  <- T1;

sum.num <-sum.num+1

M1\_pos <-M1\_pos

else if (T1.i == T2.i && T1.j == T2.j) // M1和M2在同一位置有数

then

sum.data[sum.num].i  <- T1.i

sum.data[sum.num].j  <-T1.j

sum.data[sum.num].v  <- T1.v + T2.v;

sum.num <-sum\_num+1

M1\_pos <-M1\_pos+1

M2\_pos <-M2\_pos+1

else // M2的元素T2次序靠前

then

sum.data[sum.num]  <- T2;

sum.num <-sum.num+1

M2\_pos <-M2\_pos+1

while(M1\_pos != M1.num) // M1还未遍历结束，则M1剩下元素存入sum

do sum.data[sum.num++]  <- M1.data[M1\_pos++]

while (M2\_pos != M2.num) // M2还未遍历结束，则M1剩下元素存入sum

do

sum.data[sum.num++]  <- M2.data[M2\_pos++]

return true

3. 判断矩阵是否能相乘，如果能则将结果存储在mul中：

if (M1.column != M2.row)

return false

mul.row  <- M1.row;

mul.column  <-M2.column;

mul.num  <- 0;

if (M1.num != 0 || M2.num != 0)

then for (int i = 1; i <= mul.row; i++)

do for (int j = 1; j <= mul.column; j++)

do int temp = 0; // 存储A[i][k] \* B[k][j]求和

for (int k = 1; k <= M1.column; k++)

do temp += SearchElement(M1, i, k) \* SearchElement(M2, k, j);

if (temp != 0) // 添加非零元素

then

mul.data[mul.num].i  <- i

mul.data[mul.num].j  <- j

mul.data[mul.num].v  <-temp

mul.num <-mu1.cum+1;

return true

4. 返回M[i][j]的值：

int pos  <-0;

while (pos < M.num && M.data[pos].i <= i)

do

if (M.data[pos].i == i && M.data[pos].j == j) //如果找到就返回非零元素值

then return M.data[pos].v

pos <-pos+1

return 0// 否则返回0

5. 输出稀疏矩阵：

int pos  <- 0;

for (int i = 1; i <= M.row; i++)

then for (int j = 1; j <= M.column; j++)

then

if (pos < M.num && i == M.data[pos].i && j == M.data[pos].j) //在data中，是非零元素

then printf("%d ", M.data[pos++].v)

else //不在data中，为0

then printf("%d ", 0)

printf("\n")

**函数调用关系图：**

图示

描述已自动生成

1. **调试分析报告**

**调试过程中遇到的问题：**

1. 编写代码时直接复制粘贴代码但没有修改导致程序出错
2. 没有考虑到数组会被多次调用，在之前的运算中储存的值会存入到下一次运算当中导致结果出错。

**算法时空分析：**

InputMaxtrix:o(n^3)

MatrixAddition:o(n)

MatrixMultiplication:o(n^3)

SearchElement:o(n)

PrintMatrix:o(n^2)

main:o(n^3)(功能2，3时）

**改进设想：**

本次实验设计的矩阵加法，矩阵乘法对应的只是整型的数据，没有考虑到浮点型，也没有考虑到复数的情况可以尝试完成所有数据的矩阵加乘法。

同时乘法部分时间复杂度过高，可以尝试换一种思路来减少算法的时间复杂度

**经验体会**：

三元组表的数据结构用来储存稀疏矩阵很大程度上减小了程序所占用的空间，在编写代码的时候，对矩阵的加法、乘法有了更加深刻的认识。对二元数组的储存也有新的看法。同时编写代码的时候需要耐心细致，避免出现一些小的错误。

1. **用户使用说明**

步骤1：

用户打开程序，输入需要建立代码的行数，列数，非零元素个数来建立一个三元数组，输入三个数据时需要空行。

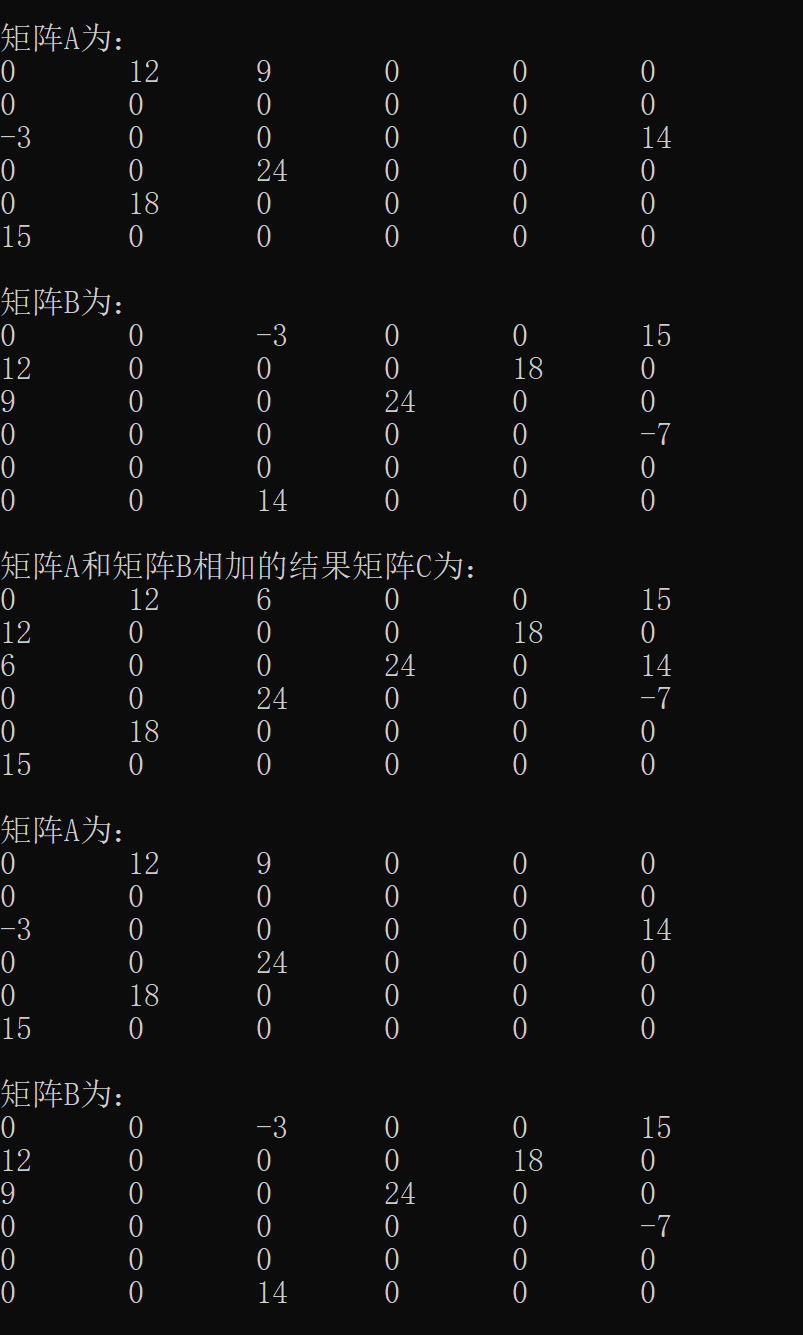
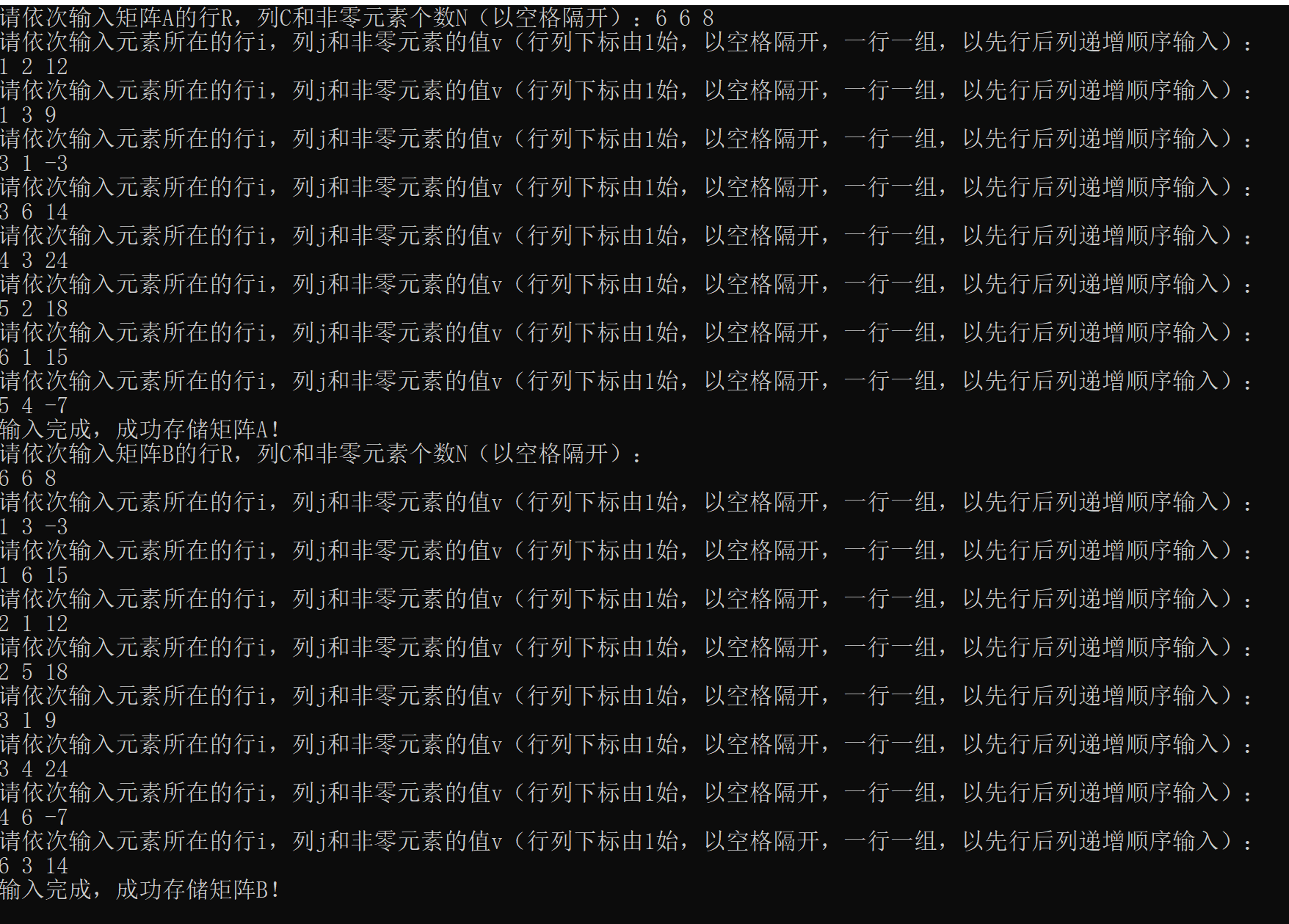
步骤2：

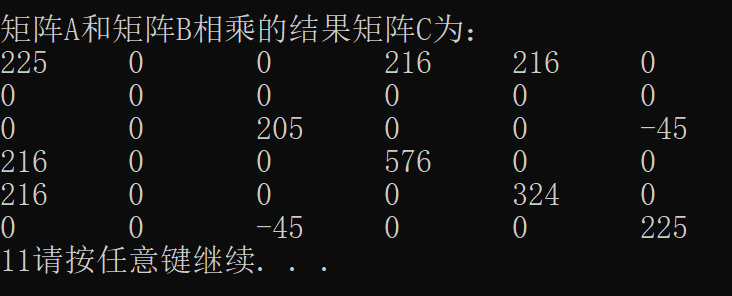
当建立好一个三元数组后，在输入每一个元素在表中的行、列以及数值，以这三个数据为一组，输入三个元素时空行，输入完一组后换行继续输入下一组数据直到输入完成所有数据。

步骤3：

重复步骤1、步骤2的操作，完成第二个三元组表的建立和数据输入。

1. **测试结果**

样例1：



样例二：

