# 四川师范大学 实验报告

学期： 2024 至 2025 第一学期 实验成绩：

课程名字：《程序设计基础——数据结构(C语言版)》 专业： 信息与计算科学

班级： 2023 级 9 班 实验编号： 05

实验项目： 实验五 指导老师： 冯山

姓名： 刘智恒 学号： 2023060522

**一、实验题目**

稀疏矩阵运算的三元组实现

**二、实验目的及要求**

掌握稀疏矩阵的三元组表示方法、算法设计与实现方法。

**三、实验内容：(类C算法的程序实现，任选其二)**

1.设计并实现基于三元组表示的稀疏矩阵表示与输入、输出算法。(必做)

2.设计并实现基于三元组表示的稀疏矩阵加法算法。

3.设计并实现基于三元组表示的稀疏矩阵转置算法。

4.设计并实现基于三元组表示的稀疏矩阵的乘法算法。

**四、实验准备**

1.计算机设备;

2.程序调试环境的准备，本实验采用**Microsoft Visual Studio**环境;

3.实验内容的算法分析与代码设计与分析准备；

4.实验源程序**Exp\_5**准备。

**注：本实验中将实验内容1至4进行了整合，故以下的实验测试用例用于4个问题的测试。**

5.实验测试用例：

**注：为了简化输入过程，用于相加与相乘运算均使用稀疏矩阵A和B，由此可推知，此时A与B均为方阵，并且行数(row）与列数(col)均对应相等。**

**（1）矩阵S的行数与列数：row=6, col=7;**

**矩阵S的非零元个数：n=8;**

**矩阵S的data信息如下：**

|  |  |  |
| --- | --- | --- |
| **i(元素行下标)** | **j(元素列下标)** | **v(元素值)** |
| **1** | **2** | **12** |
| **1** | **3** | **9** |
| **3** | **1** | **-3** |
| **3** | **6** | **14** |
| **4** | **3** | **24** |
| **5** | **2** | **18** |
| **6** | **1** | **15** |
| **6** | **4** | **-7** |

**表4.1稀疏矩阵S的三元组表信息**

**矩阵A&B的行数与列数：rowA & rowB=4, colA & colB=4;**

**矩阵A的非零元个数：nA=4;**

**矩阵B的非零元个数：nB=5;**

**矩阵A的data信息如下：**

|  |  |  |
| --- | --- | --- |
| **i(元素行下标)** | **j(元素列下标)** | **v(元素值)** |
| **1** | **1** | **3** |
| **1** | **4** | **5** |
| **2** | **2** | **-1** |
| **3** | **1** | **2** |

**表4.2稀疏矩阵A的三元组表信息**

**矩阵B的data信息如下：**

|  |  |  |
| --- | --- | --- |
| **i(元素行下标)** | **j(元素列下标)** | **v(元素值)** |
| **1** | **2** | **2** |
| **2** | **1** | **1** |
| **3** | **1** | **-2** |
| **3** | **2** | **4** |
| **4** | **3** | **7** |

**表4.3稀疏矩阵B的三元组表信息**

**五、实验内容**

**（一）问题分析**

（1）本实验中的所有内容均针对于**稀疏矩阵**的表达及其运算，并且基于**三元组表示**的数据结构。稀疏矩阵的衡量标准如下：

假设在**m\*n**矩阵中，有**t个元素不为零**，令：

称为矩阵的**稀疏因子**，通常认为：时称为稀疏矩阵。

由于稀疏矩阵本身的**非零元素较少，**故本实验选择**静态**的**数组**对稀疏矩阵的各项有效信息进行存储，并提前规定其存储大小。稀疏矩阵的**三元组顺序表**存储结构如下：

**#define MAXSIZE 1000**

**typedef struct { //当前非零元信息**

**int i, j; //非零元行下标和列下标**

**ElemType e; //非零元素值**

**}Triple;**

**typedef struct { //非零元三元组表**

**Triple data[MAXSIZE];**

**int row, col, num; //矩阵的行数、列数和非零元个数**

**}TSMatrix;**

（2）各项操作基本分析：

**[1]转置**：根据测试用例中的矩阵S的三元组表数据可知，若要实现稀疏矩阵的转置，只要做到：【1】**将矩阵的行列值互换**；【2】**将每个三元组中的i和j值相互调换**；【3】**重排三元组之间的次序**，这便可实现矩阵的转置。

而关键在于步骤【3】的实现：为了找到**S**中的每一列中所有的**非零元素**，则需要对其三元组表数据从第一行起整个**扫描一遍**。

可设定**Col（列）=1**，根据**Col<=S.col**，将**S.data[n].j（列数据）**依次**遍历匹配Col**的值，若匹配则把对应的**data**依次存储到转置矩阵**T**中。

**[2]乘法**：一般的矩阵相乘的公式为：

经典算法中，**M**矩阵的**列数(k)**必须等于**N**矩阵的**行数(k)**。不论**M(i,k)**和**N(k,j)**的值是否为0，都要进行一次乘法运算，而实际上，这**两者有一个值为0时，其乘积也为0**。因此，在进行稀疏矩阵运算时，应该免去这种无效操作。

换而言之，为求**Q**的值，只需要通过遍历**M**和**N**中各自**非零元的个数（假设M和N中非零元个数分别为m和n，当前遍历到的非零元数的值分别为m0<=m,n0<=n）**，找到当**M.data**中的**j**值和**N.data**中的**i**值**相等时的各对元素进行相乘**即可，即当前**M.data[m0].j == N.data[n0].i**，以此求得**M.data[m0].e \* N.data[n0].e**。最终将相乘结果存储到另一个矩阵中。

**（注：乘积矩阵Q中的每个元素的值是个累计和，该乘积只是Q(i,j)的一部分）**

**[3]加法**：一般的矩阵相加公式为：

其中，矩阵**M**和矩阵**N**的**行数与列数必须对应相同**，即对应位置上的**aij**进行相加。同样地，可以通过遍历**M**和**N**中各自**非零元的个数（假设M和N中非零元个数分别为m和n，当前遍历到的非零元数的值分别为m0<=m,n0<=n）**，然后根据当前非零元的**行列值相等**，即**M.data[m0].i == N.data[n0].i && M.data[m0].j == N.data[n0].j**；以及当前对应元素**相加后值不为零**，即**M.data[m0].e + N.data[n0].e != 0**以上条件，对当前位置**（i,j）**进行对应元素相加操作。最终将相加结果存储到另一个矩阵中。

**（二）算法描述**

（1）**Status InitMatrix(TSMatrix \*M, int row, int col)**;

//本算法用于初始化一个稀疏矩阵M：通过输入行数**（row）**，列数**（col）**，以此设置稀疏矩阵的行数、列数，并初始化M的非零元素的数量为0。

（2）**Status InputElem(TSMatrix \*M, int i, int j, ElemType e)**;

//本算法用于插入一个非零元素到稀疏矩阵M中：输入该非零元素的行索引**（i）**、列索引**（j）**以及元素值**（e）**，检查当前输入的下标**（i,j）**是否合法，以及非零元素的数量是否超过最大限制，如果没有则将该元素插入到M中，并增加非零元素的计数。

（3）**Status PrintSMatrix(TSMatrix M)**;

//本算法用于打印稀疏矩阵：遍历矩阵M的每一个位置，检查当前位置是否有非零元素，如果有则打印该元素，否则打印0。

（4）**Status TransposeSMatrix** **(TSMatrix M, TSMatrix \*T)**;

//本算法用于稀疏矩阵的转置：首先交换矩阵M和T的行和列，并根据原矩阵M中的非零元素创建转置矩阵T（具体见以上分析部分）。

（5）**Status AddSMatrix(TSMatrix \*C, TSMatrix A, TSMatrix B)**;

//本算法用于稀疏矩阵的相加操作：首先检查两个矩阵A和B的维度是否一致，如果一致则将每个位置上相同的非零元素进行相加，将结果保存到矩阵C中（具体见以上分析部分）。

（6）**Status MultiplySMatrix(TSMatrix \*S, TSMatrix A, TSMatrix B)**;

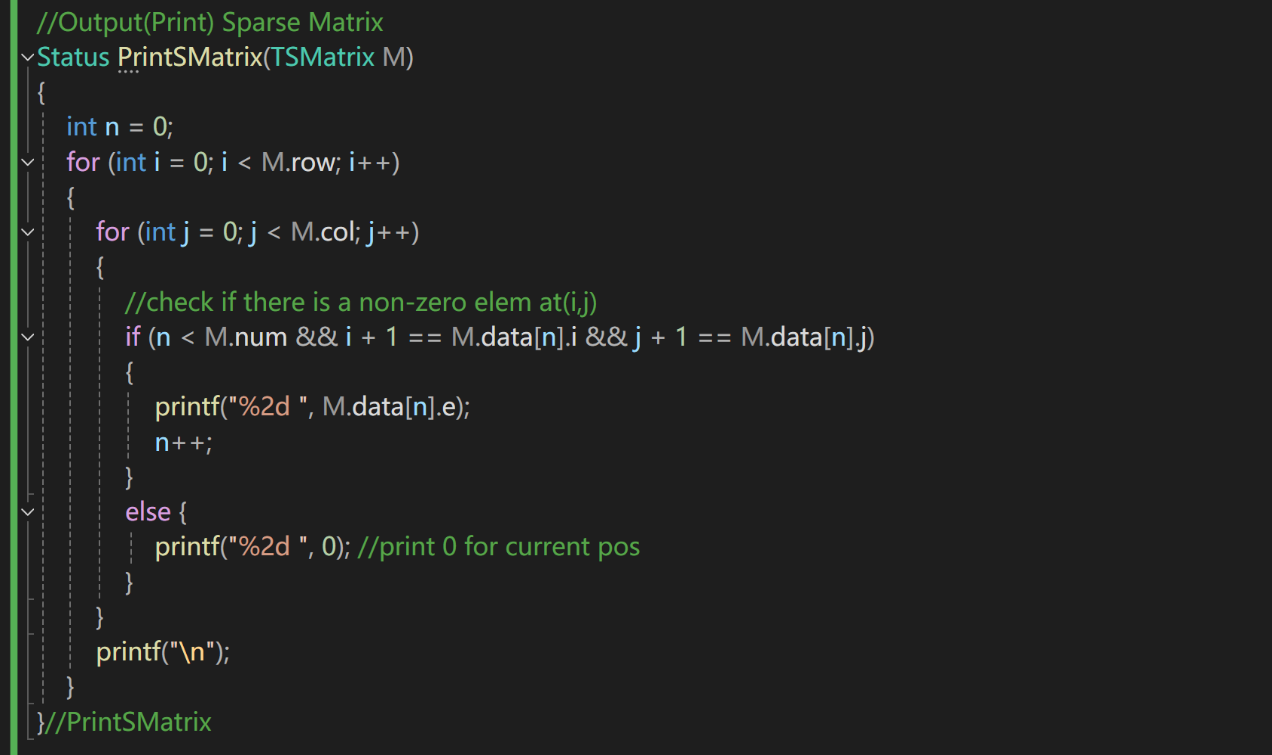
//本算法用于稀疏矩阵的相乘操作：首先检查矩阵A的列数是否等于矩阵B的行数，如果符合条件则计算每个位置的非零元素乘积，并将结果累加到矩阵S中。

**（三）程序代码**

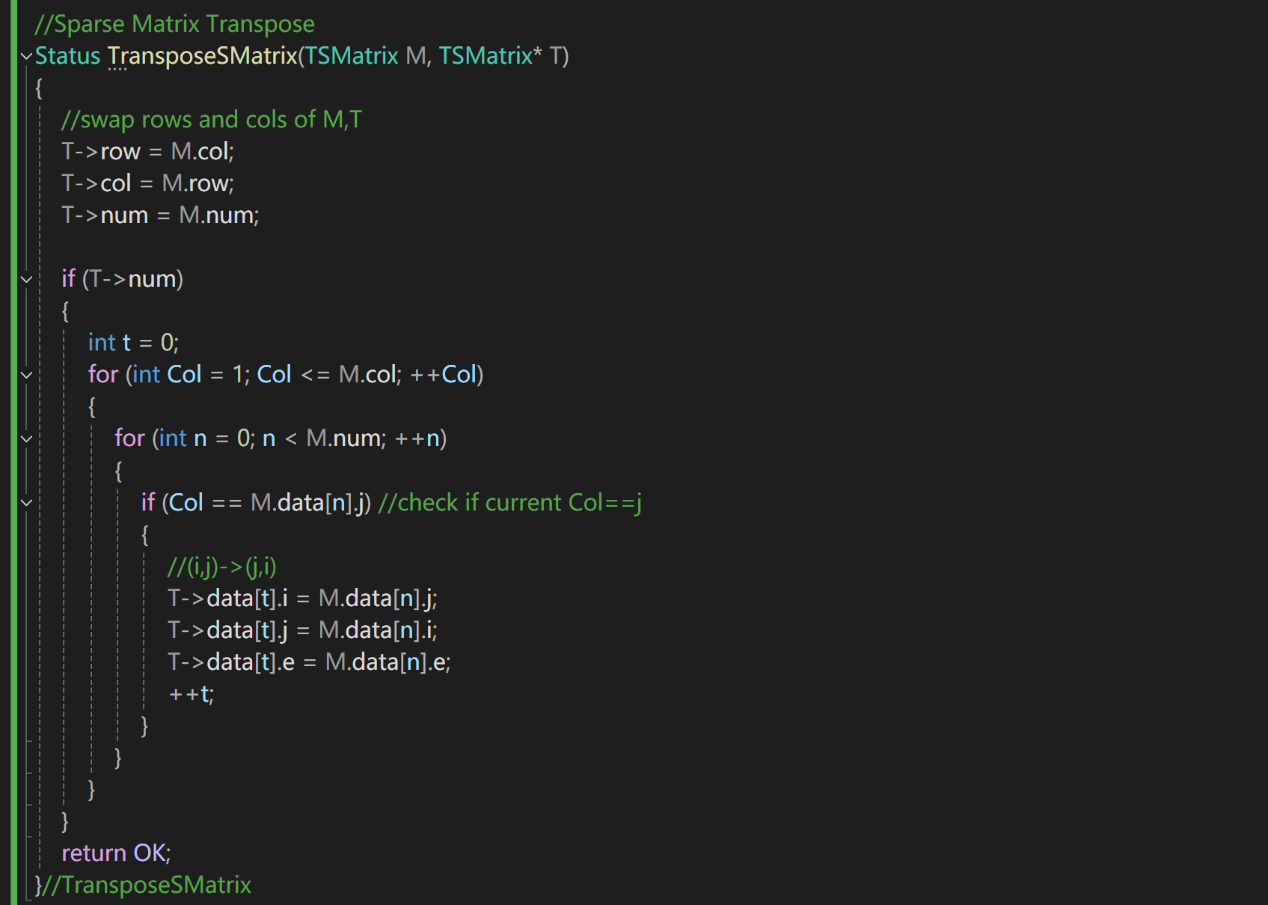
根据以上对**实验内容**的算法描述，求解的程序代码如下：



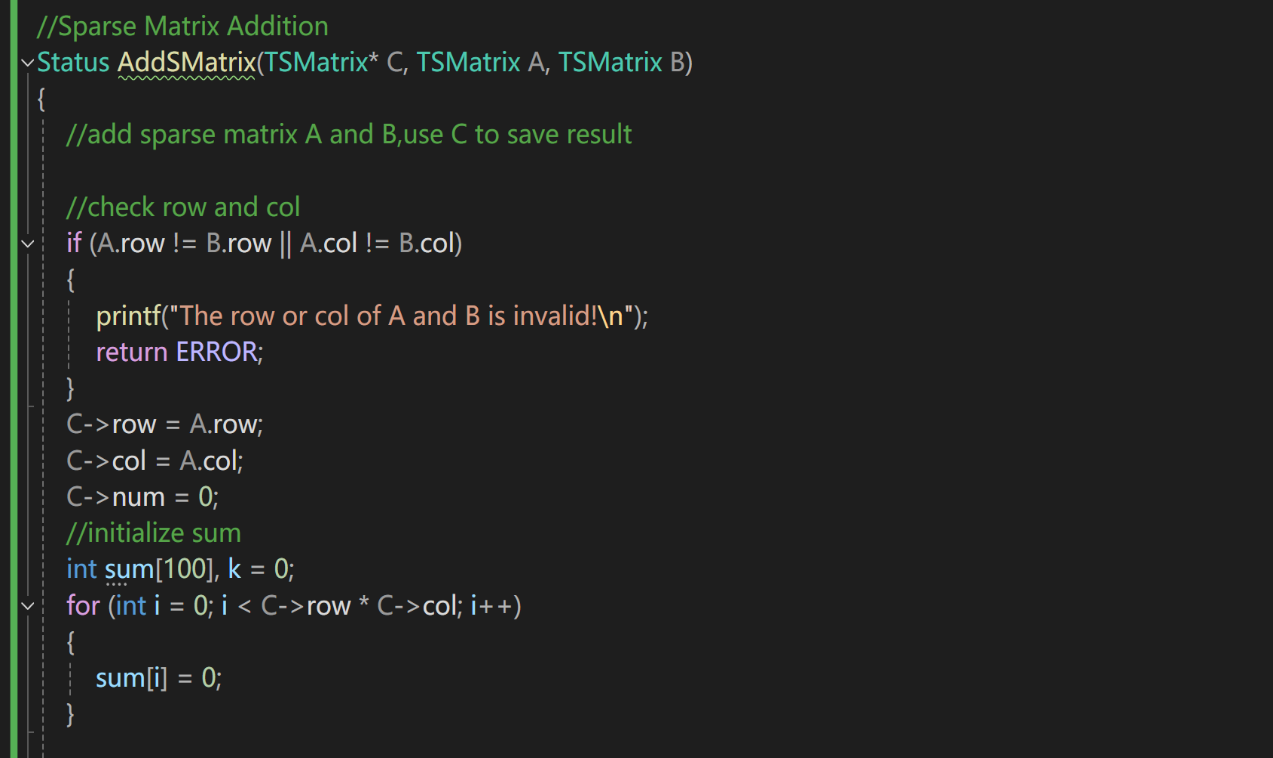
**注：此处只展示主要算法的程序代码图。**



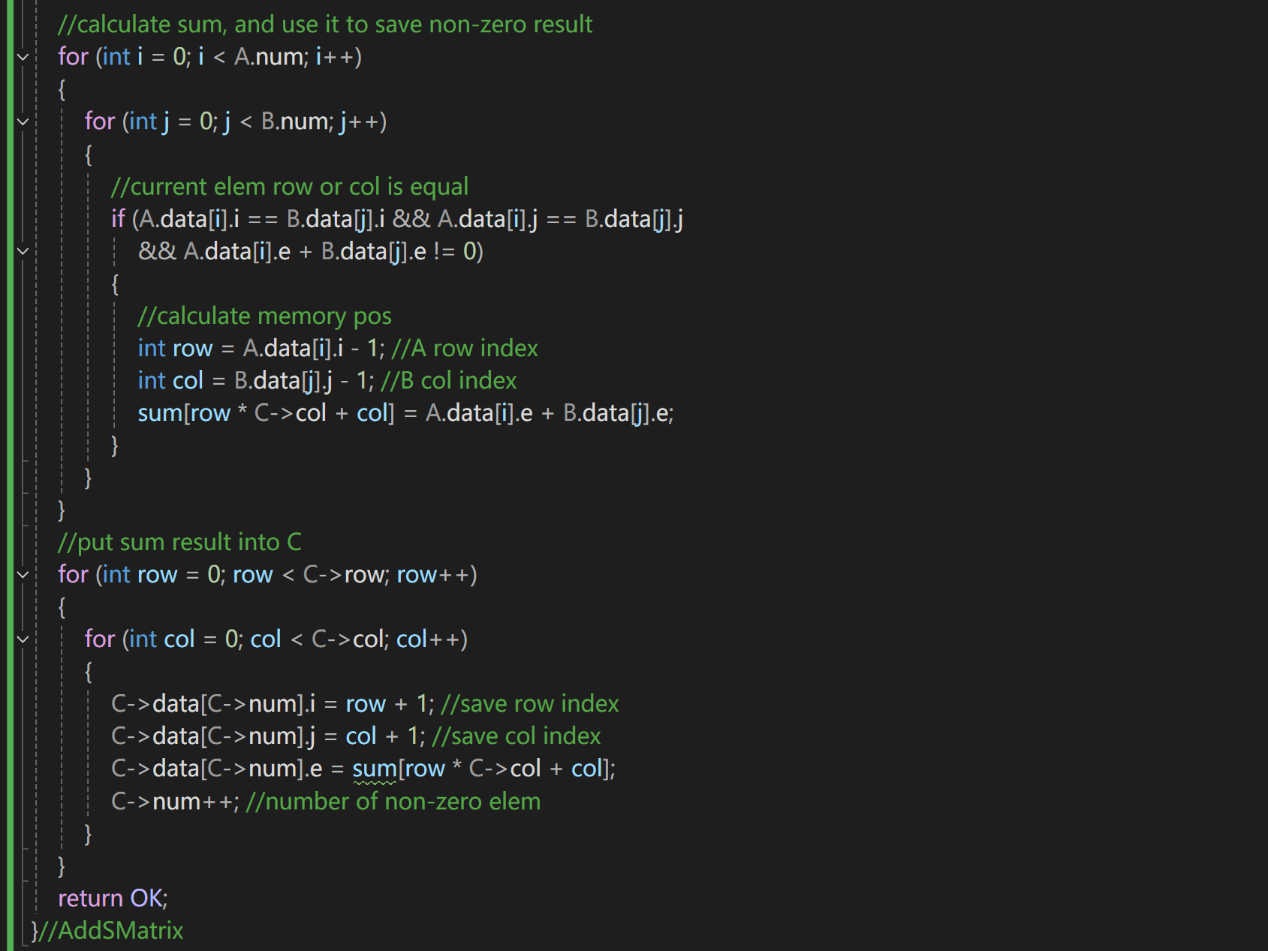
**图5.1稀疏矩阵打印**



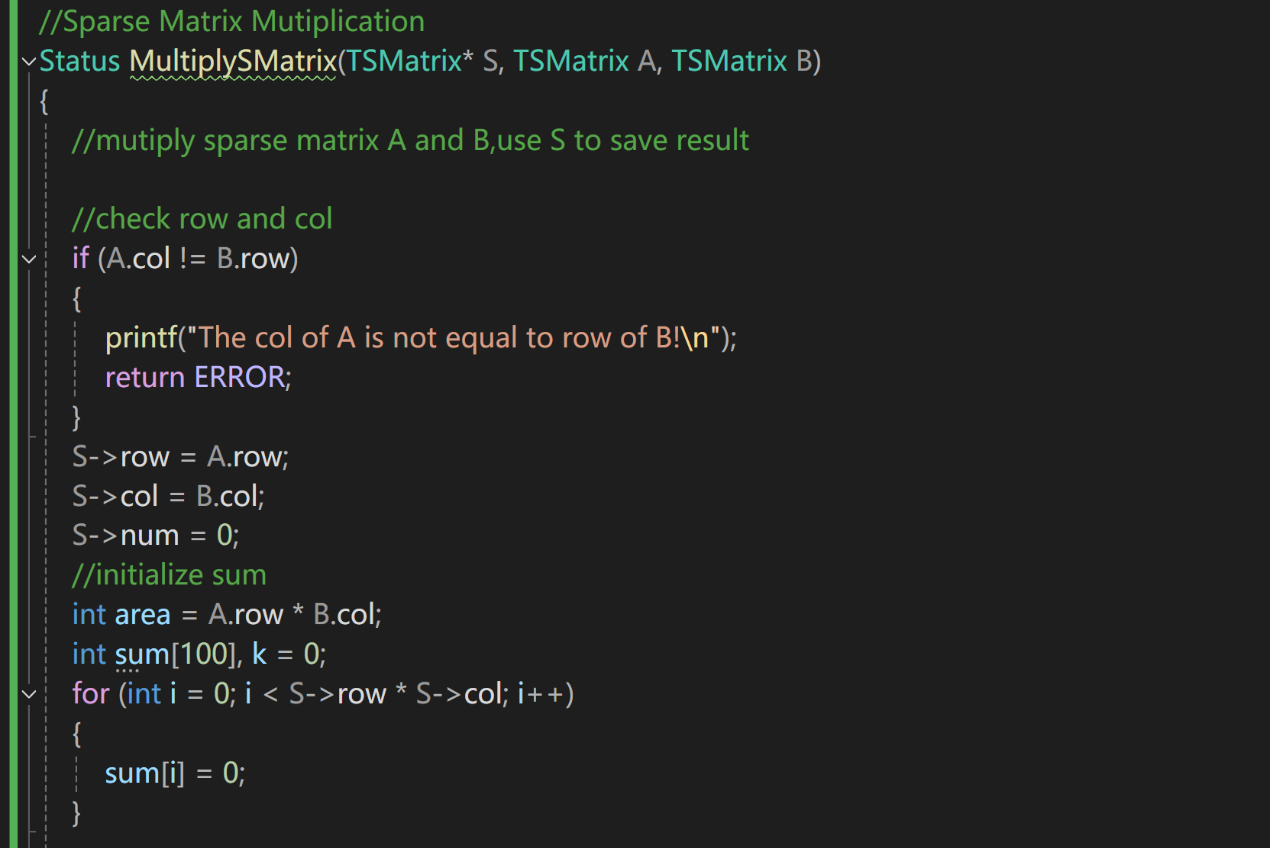
**图5.2稀疏矩阵转置**



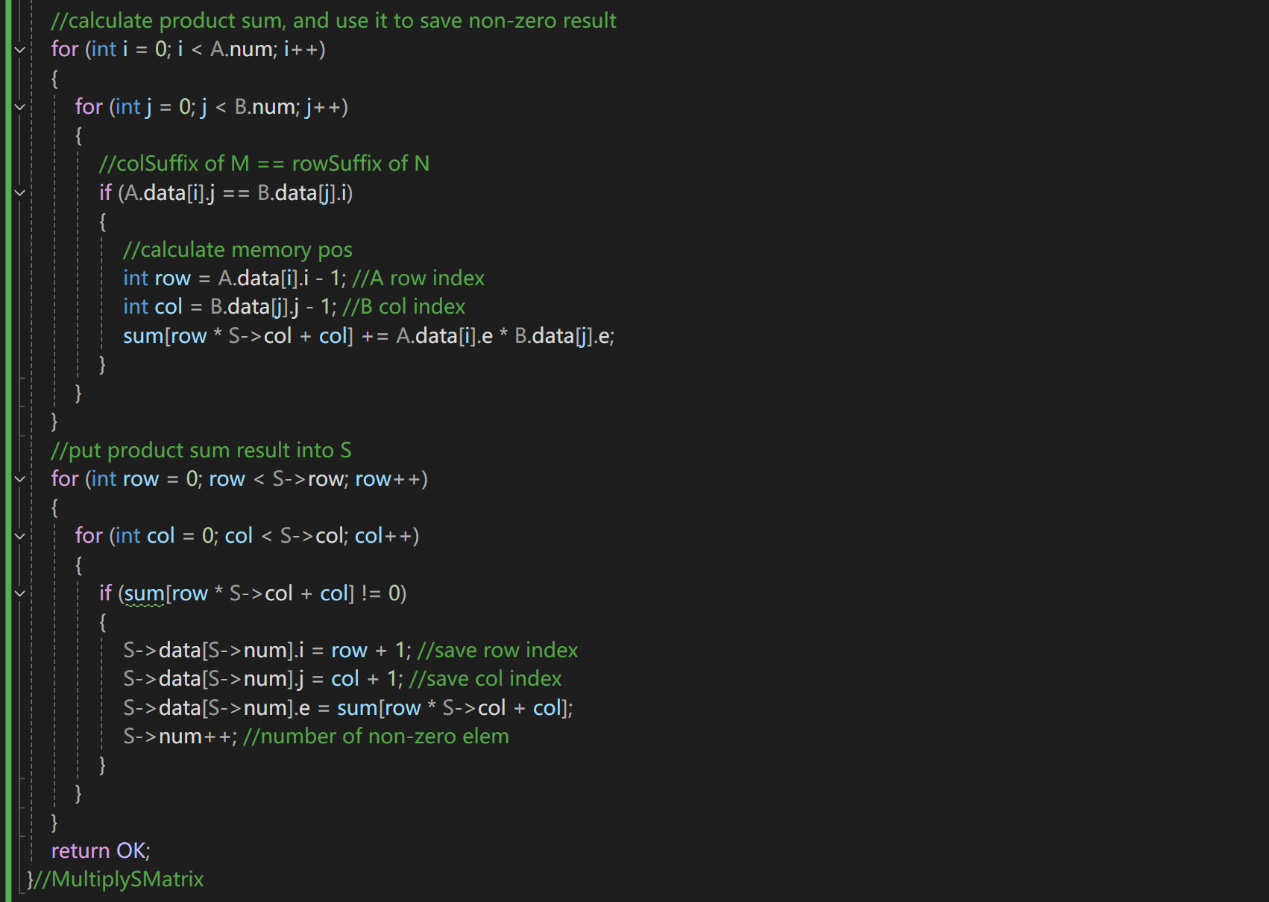
**图5.3稀疏矩阵相加（part\_1）**



**图5.4稀疏矩阵相加（part\_2）**



**图5.5稀疏矩阵相乘（part\_1）**

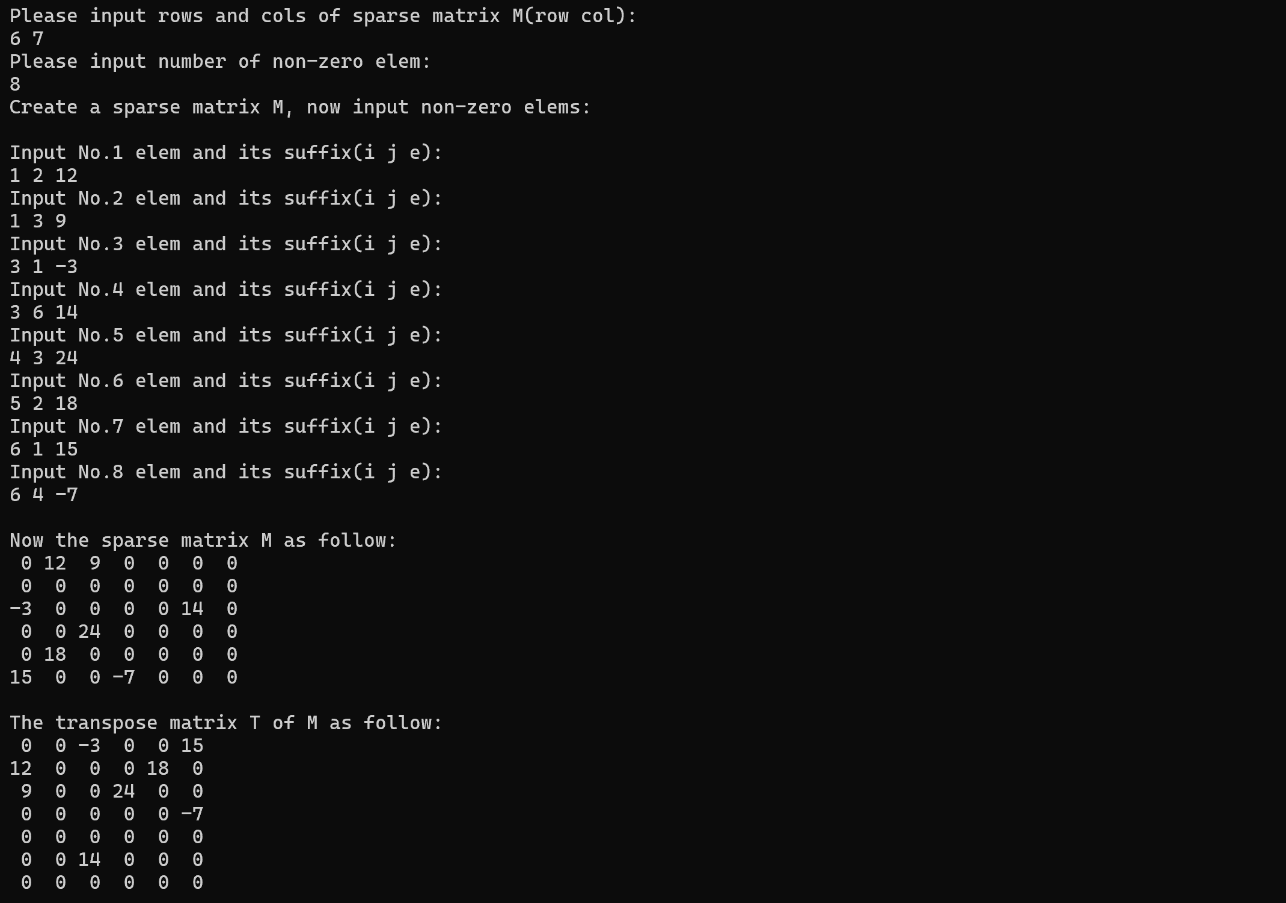


**图5.6稀疏矩阵相乘（part\_2）**

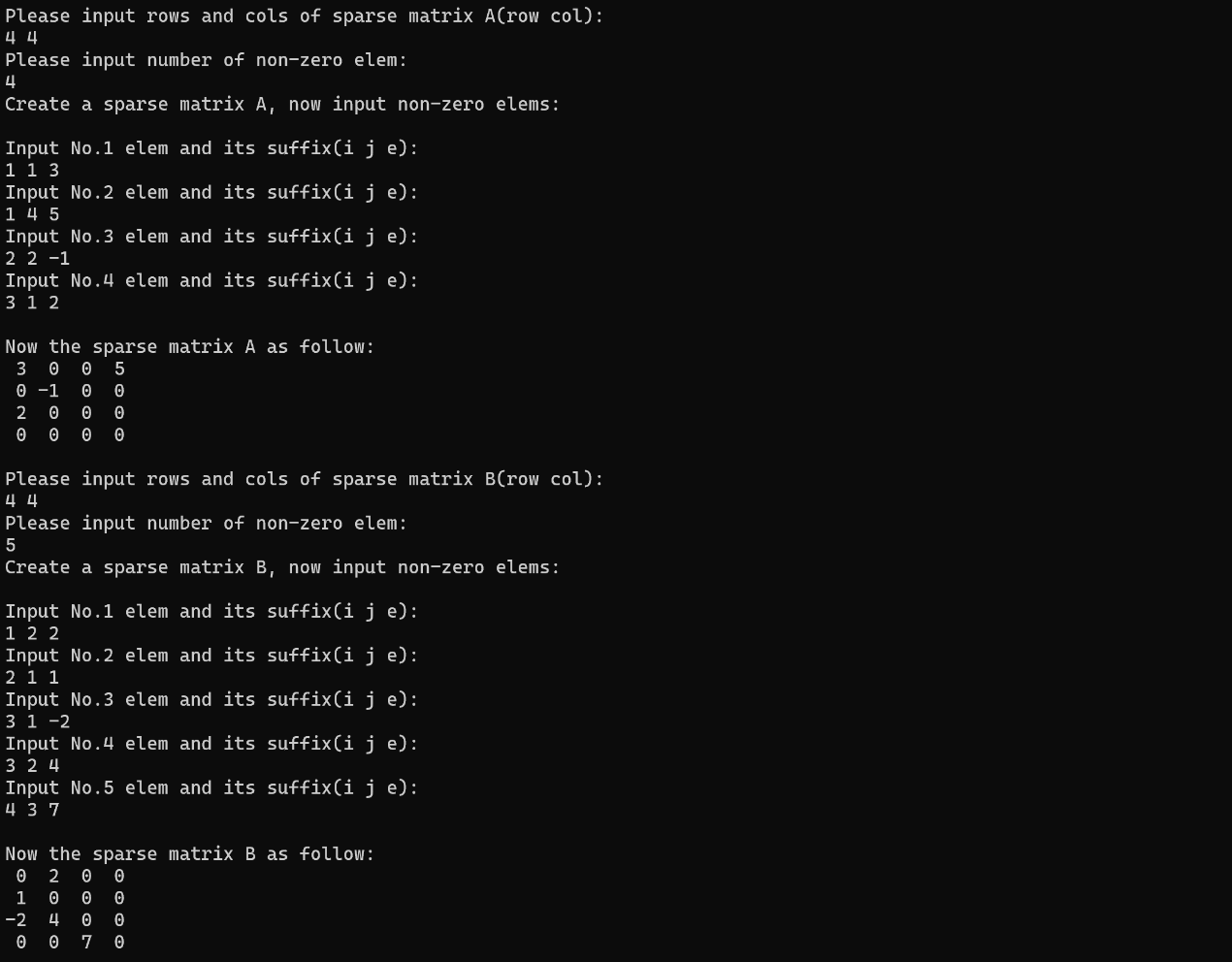
**六、实验结果**

**（一）结果呈现：**

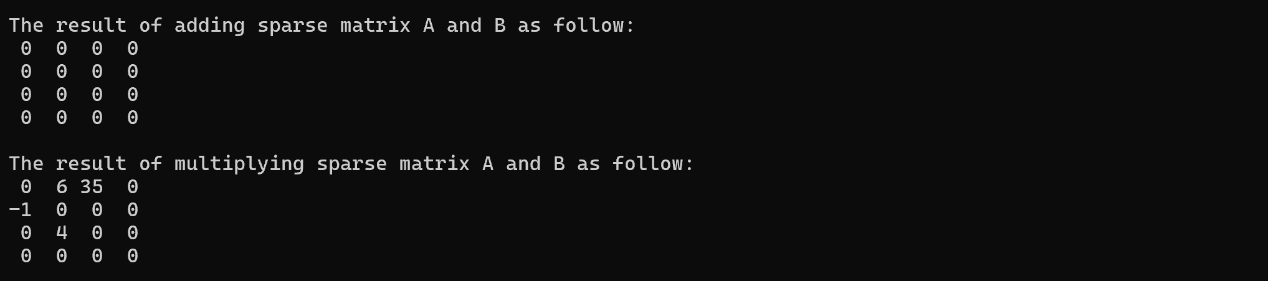
1.根据以上**实验内容**的程序代码运行后进行多次测试，以下是根据**实验内容测试用例**进行测试后的结果示意图：



**图6.1稀疏矩阵S及其转置矩阵T**



**图6.2稀疏矩阵A和B**



**图6.3稀疏矩阵A和B相加和相乘结果**

**（二）结果分析：**

**1.时间复杂度分析**

**【InitMatrix】—— 初始化矩阵**

只是在结构体中初始化一些参数，故时间复杂度为**O(1)**。

**【InputElem】—— 插入元素到矩阵**

直接将元素插入到数组中，并且只进行简单的赋值操作，故时间复杂度为**O(1)**。

【**PrintSMatrix**】**—— 打印矩阵元素**

函数需遍历整个矩阵并打印每个元素，故时间复杂度为**O(n)**，其中**n**为矩阵的行数乘以列数（即总元素数）。

【**TransposeSMatrix**】**—— 稀疏矩阵转置**

需要遍历原矩阵中的所有非零元素进行转置，故时间复杂度为**O(n)**,其中**n**为非零元素数量。

【**AddSMatrix**】**—— 稀疏矩阵相加**

需要比较**A**和**B**中所有的非零元素，故时间复杂度为**O(nA\*nB)**，其中**nA**和**nB**分别为矩阵**A**和**B**中非零元素的数量。

【**MultiplySMatrix**】**—— 稀疏矩阵相乘**

需要遍历**A**的每个非零元素，并对每个非零元素再遍历**B**进行乘法计算，故时间复杂度为**O(nA\*nB)**，其中**nA**和**nB**分别为矩阵**A**和**B**中非零元素的数量。

**综上所述：在最坏情况下，时间复杂度主要由AddSMatrix函数和MultiplySMatrix函数主导，故总的时间复杂度为：O(nA\*nB)，其中nA和nB分别为矩阵A和B中非零元素的数量。**

**2.空间复杂度分析**

**【稀疏矩阵装置】**

在最坏情况下需要存储所有非零元素的位置，故空间复杂度为**O(n)**。

**【稀疏矩阵相加】**

该函数中需要额外的数组**sum**来存储每个元素的和，故空间复杂度为**O(n)。**

**【稀疏矩阵相乘】**

最坏情况下是**nA\*nB**个非零元素都存在，故空间复杂度为**O(m)**，其中**m**为结果矩阵非零元素的数量。

**综上所述：空间复杂度主要由存储矩阵非零元素的数组和计算和或乘积的数组决定，故整体的空间复杂度为O(n)。**

**七、实验总结**

1.本实验中使用了固定大小的数组**（MAXSIZE）**，这可能会导致**内存浪费或溢出**。在创建稀疏矩阵时，可以使用**动态数组**（例如**malloc**）来**根据实际需要分配内存**。

2.在**AddSMatrix**和**MultiplySMatrix**函数中，当前的实现需要**两层嵌套循环**来查找和计算非零元素。在这些过程中可以考虑使用**哈希表**或其他数据结构来存储行或列的非零元素，以**提高查找效率**。