《数据结构》实验报告

班　级:U10P53013.04　　姓　名:梁桐　　学　号:2023370018
E-mail:3211583077@qq.com　　日　期: 2024.4.14

◎实验题目:

实验 2.1：稀疏矩阵转置

◎实验内容:

第一行输入两个正整数 n 和 m,分别表示矩阵的行数和列数, 然后输入矩阵三元组， 最后输入（0 0 0）表示结束输入。输出稀疏矩阵的转置矩阵。（行列均不大于 20）

一、需求分析

1. 输入的形式和输入值的范围：

输入包括两个内容，第一行输入两个正整数 m 和 n,分别表示矩阵的行数和列数，用空格隔开，后续输入矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

2. 输出的形式：

输出为转置后的稀疏矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

3. 程序所能达到的功能：

将稀疏矩阵转置。

测试数据：

输入：

4 4
1 1 1
2 1 2
3 2 3
0 0 0

输出：

1 1 1
1 2 2
2 3 3

二 概要设计

1. Tran 函数:

名称: Tran

功能: 这是一个转置函数，用于将输入的稀疏矩阵 A 转置为矩阵 B。

参数:

TSMatrix A: 输入的稀疏矩阵 A。

TSMatrix *B: 指向输出转置后的矩阵 B 的指针。

实现:

函数首先初始化了转置后矩阵 B 的属性，包括行数、列数和非零元素个数。

它统计了矩阵 A 中每列非零元素的个数，并计算了每列非零元素在矩阵 B 中的起始位置。

它按列遍历矩阵 A，将非零元素转置到矩阵 B 中相应位置。

2. main 函数：

名称：main

功能：这是程序的主函数，用于执行程序的主要逻辑。

实现：

在函数内部，首先定义了两个稀疏矩阵 A 和 B。

然后，通过标准输入读入矩阵 A 的行数和列数。

接着，循环读入非零元素的行号、列号和值，直到遇到输入的行号和列号均为零的情况，表示输入结束。

调用 Tran 函数将矩阵 A 转置为矩阵 B。

最后，输出矩阵 B 的内容。

3. 其他函数和类型定义：

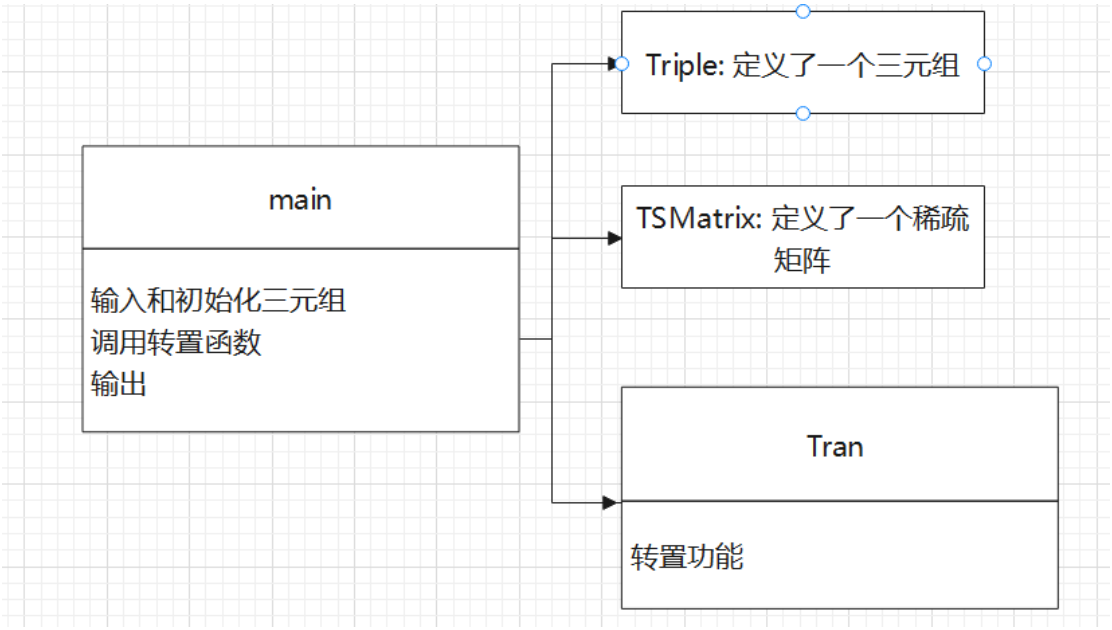
Triple：定义了一个三元组，包含了一个稀疏矩阵中的一个非零元素的行号、列号和值。

TSMatrix：定义了一个稀疏矩阵，包含了一个三元组表存储非零元素、矩阵的行数、列数以及非零元素个数。

MAXSIZE：定义了三元组表的最大容量。

ElementType：定义了矩阵中非零元素的数据类型，这里为整数类型。

除此之外，还包含了标准库头文件 <stdio.h> 和 <stdlib.h>，以及一些基本的 C 语言语法和控制流程语句。



三 详细设计

伪代码如下

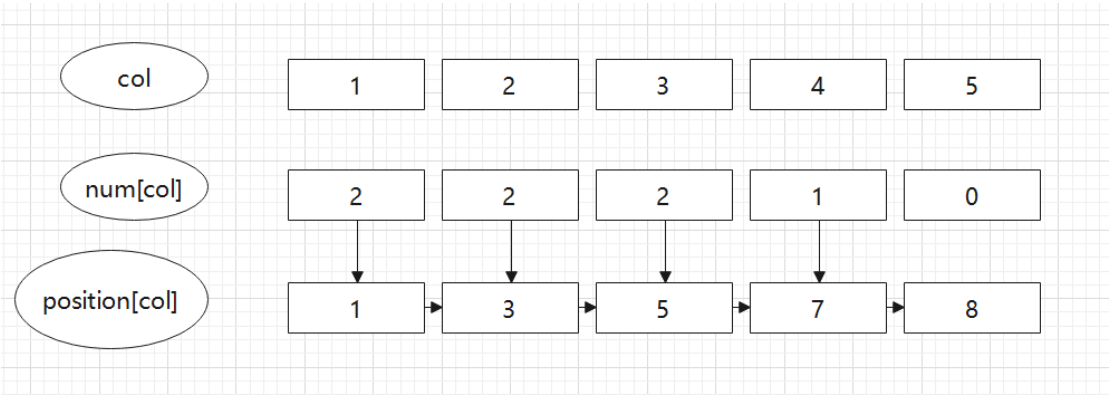
定义结构体 Triple:

属性：行号(row)，列号(col)，元素值(e)

定义结构体 TSMatrix:

属性：三元组表 data[MAXSIZE + 1]，行数 m，列数 n，非零元素个数 len

定义函数 Tran(A, B):
    参数: 稀疏矩阵 A, 输出矩阵 B
    初始化 B 的属性: len = A.len, n = A.m, m = A.n
    若 B 的非零元素个数不为零:
        初始化 num 数组，记录每列非零元素个数为 0
        统计每列非零元素个数，存储在 num 数组中
        计算每列非零元素在 B 中的起始位置，存储在 position 数组中
        开始转置:
            遍历 A 中的每个非零元素:
                获取当前元素的列号 col
                获取当前列的起始位置 q
                将当前元素转置到 B 中对应的位置:
                    行列互换
                    更新元素值
                    更新当前列的起始位置
    返回
算法核心思路如下图：



定义函数 main():
    声明变量: i, 稀疏矩阵 A, B
    读入矩阵 A 的行数和列数
    初始化 i 为 0，A 的非零元素个数 len 为 0
    读入非零元素，存入矩阵 A 中，直到遇到行号、列号和值均为零的情况，表示输入结束
    调用 Tran 函数将矩阵 A 转置为矩阵 B
    输出矩阵 B 的内容
    返回

四 使用说明、测试分析及结果

1.说明如何使用你编写的程序；

1、 输入输出均使用标准的 scanf()和 printf()函数。
2、 使用链表的思路，程序基于 C89 标准编写
3、 运行环境为 Codeblocks

2.测试结果与分析；

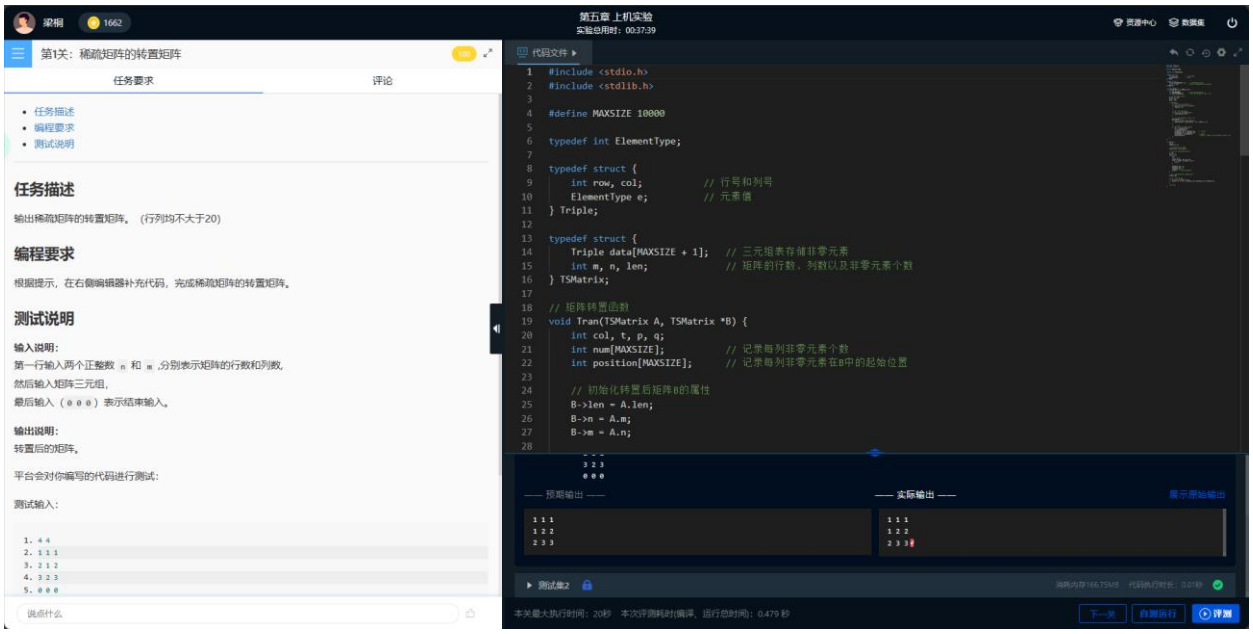
输入：
4 4
1 1 1
2 1 2
3 2 3
0 0 0
输出：
1 1 1
1 2 2
2 3 3
输出正常

4、 调试过程中遇到的问题是如何解决提以及对设计与实现的回顾讨论和分析

问题：未考虑行与列均以 0 开始，导致测试集 2 不通过

解决：更改边界值判断条件

5、 运行界面



五、实验总结

在进行这个稀疏矩阵转置的实验过程中，我学到了许多关于稀疏矩阵表示和转置算法的知识。以下是我在实验中的一些总结和心得：

1. 稀疏矩阵的表示：

稀疏矩阵是一种大部分元素为零的矩阵，通常采用三元组表示法来节省存储空间。

三元组表示法使用一个数组存储非零元素的行号、列号和值，以及矩阵的行数、列数和非零元素个数。

2. 转置算法：

矩阵转置是指将矩阵的行和列互换的操作，对于稀疏矩阵而言，转置需要考虑如何有效地处理非零元素。

在实现稀疏矩阵转置算法时，可以利用统计每列非零元素个数和计算每列非零元素在转

置后矩阵中的起始位置的方法，从而提高效率。

3. 编程技巧：

在编写转置算法的过程中，需要考虑如何合理地利用数据结构和算法，以及如何进行边界条件的处理。

使用指针作为函数参数可以有效地传递和修改数据，提高程序的灵活性和效率。

4. 调试和测试：

在实验过程中，我重点关注了转置算法的正确性和性能。

通过编写测试用例和进行调试，我确保了程序能够正确地处理各种情况，并且在处理大规模稀疏矩阵时也能够保持较好的性能。

通过这次实验，我不仅加深了对稀疏矩阵表示和转置算法的理解，还提升了编程能力和问题解决能力。我将继续努力学习，深入探索更多与数据结构和算法相关的知识。

◎实验题目：

2.2：稀疏矩阵加法,实现 C=A+B

◎实验内容：

输入两个稀疏矩阵，输出它们相加的结果。

input 第一行输入四个正整数，分别是两个矩阵的行 m、列 n、第一个矩阵的非零元素的个数 t1 和第二个矩阵的非零元素的个数 t2。 接下来的 t1+t2 行是三元组，分别是第一个矩阵的数据和第二个矩阵的数据。三元组 的第一个元素表示行号，第二个元素表示列号，第三个元素是该项的值。

输出相加后的矩阵三元组。

一、需求分析

1. 输入的形式和输入值的范围：

输入包括两个内容，第一行输入四个正整数 m 和 n,分别表示两个矩阵的行数和列数，用空格隔开，后续输入矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

2. 输出的形式：

输出为相加后的稀疏矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

3. 程序所能达到的功能：

将稀疏矩阵相加。

测试数据：

输入：

3 4 3 2
1 1 1
1 3 1
2 2 2
1 2 1
2 2 3
输出：
1 1 1
1 2 1
1 3 1
2 2 5

二 概要设计

函数模块设计与功能：

input_matrix 函数:

功能: 该函数用于从标准输入读取稀疏矩阵的数据，包括行数、列数以及非零元素的位置和值，并将数据存储在对应的矩阵结构体中。

add_matrices 函数:

功能: 这个函数实现了两个稀疏矩阵的加法运算。它遍历了第二个矩阵的非零元素，并在第一个矩阵中找到相同位置的非零元素进行相加。如果第一个矩阵中的元素值为零，则标记为无效元素。最后，将第二个矩阵中未被标记的非零元素添加到第一个矩阵中。

sort_matrix 函数:

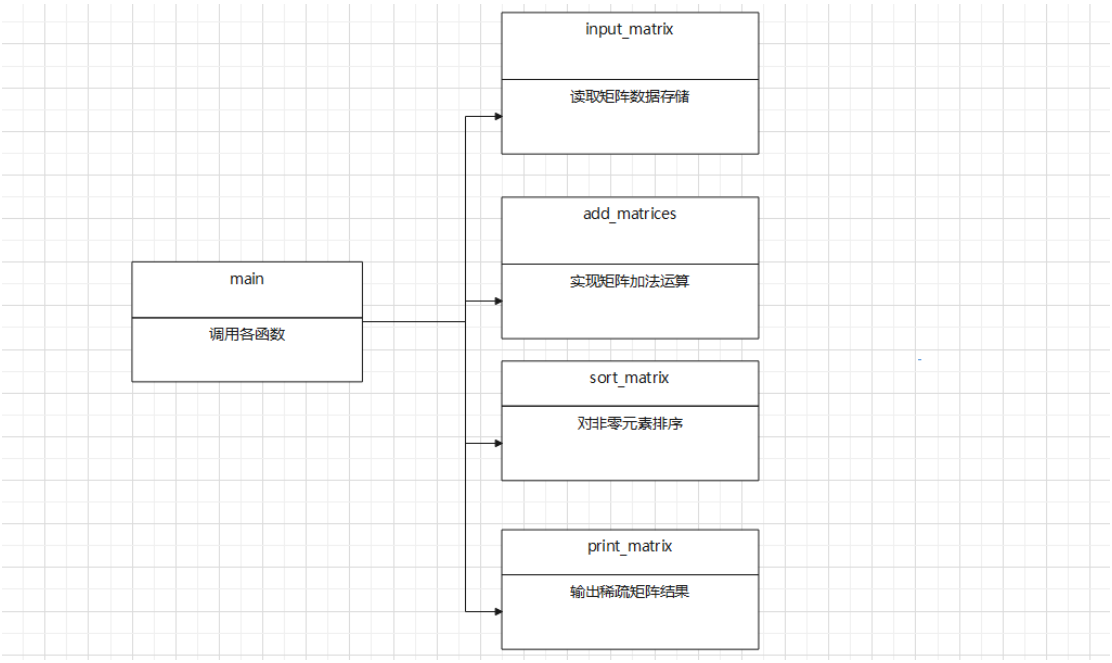
功能: 这个函数用于对矩阵中的非零元素按照行列坐标进行排序，以便输出结果时按照从左上到右下的顺序输出。

print_matrix 函数:

功能: 这个函数用于将稀疏矩阵的结果输出到标准输出。它遍历矩阵中的非零元素，并将其打印出来。

主函数 main:

功能: 主函数负责程序的整体逻辑。它首先读取两个矩阵的行数、列数以及非零元素的个数，然后调用 input_matrix 函数读取矩阵数据，接着调用 add_matrices 函数进行矩阵加法运算，再调用 sort_matrix 函数对结果进行排序，最后调用 print_matrix 函数输出结果。这些函数共同实现了稀疏矩阵的加法运算，并按照要求输出结果。



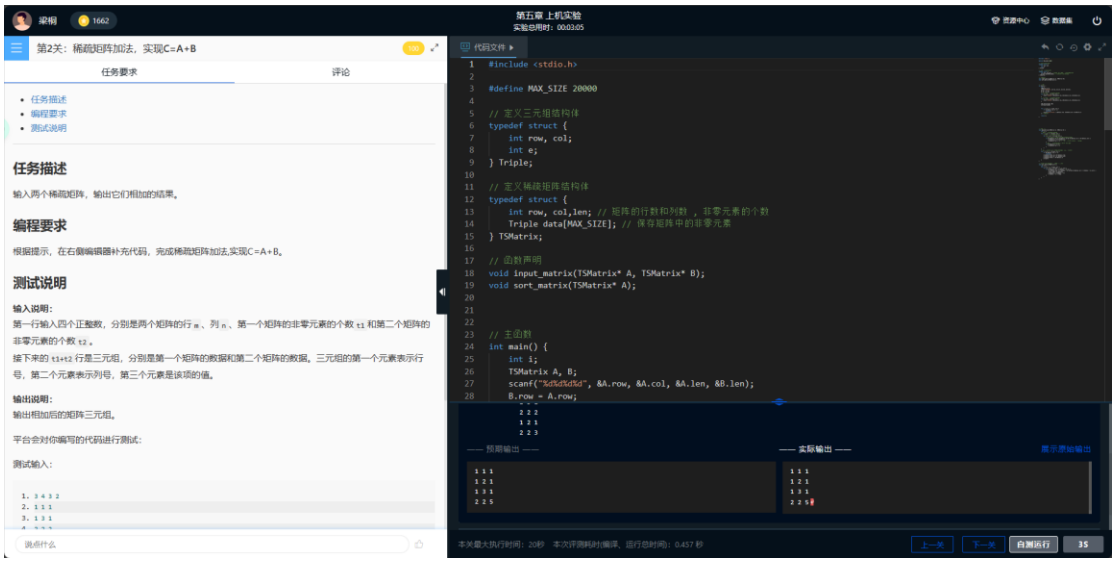
三 详细设计

伪代码如下

```
定义 MAX_SIZE 为 20000

定义结构体 triple 包含
    整数 x, y
    整数 value
结束结构体

定义结构体 matrix 包含
    整数 row, col
    整数 count
    三元组 data[MAX_SIZE]
结束结构体

函数 输入矩阵数据(matrix1, matrix2)
    对于 i 从 0 到 matrix1.count - 1 循环执行
        从输入中读取 matrix1.data[i].x, matrix1.data[i].y, matrix1.data[i].value
    结束循环
    对于 i 从 0 到 matrix2.count - 1 循环执行
        从输入中读取 matrix2.data[i].x, matrix2.data[i].y, matrix2.data[i].value
    结束循环
结束函数

函数 矩阵相加(matrix1, matrix2)
    对于 i 从 0 到 matrix2.count - 1 循环执行
        对于 j 从 0 到 matrix1.count - 1 循环执行
            如 果 matrix1.data[j].x 等 于 matrix2.data[i].x 并 且
```

```
matrix1.data[j].y 等于 matrix2.data[i].y 则
                matrix1.data[j].value += matrix2.data[i].value
                matrix2.data[i].x = -1
            结束如果
            如果 matrix1.data[j].value 等于 0 则
                matrix1.data[j].x = -1
            结束如果
        结束循环
    结束循环
    对于 i 从 0 到 matrix2.count - 1 循环执行
        如果 matrix2.data[i].x 等于 -1 则
            继续
        结束如果
        matrix1.data[matrix1.count].x = matrix2.data[i].x
        matrix1.data[matrix1.count].y = matrix2.data[i].y
        matrix1.data[matrix1.count].value = matrix2.data[i].value
        matrix1.count++
    结束循环
结束函数

函数 排序矩阵(matrix1)
    对于 i 从 0 到 matrix1.count - 1 循环执行
        对于 j 从 0 到 matrix1.count - i - 2 循环执行
            如果 matrix1.data[j].x 大于 matrix1.data[j + 1].x 或者
               (matrix1.data[j].x 等 于 matrix1.data[j + 1].x 并 且
matrix1.data[j].y 大于 matrix1.data[j + 1].y) 则
                交换 matrix1.data[j] 与 matrix1.data[j + 1]
            结束如果
        结束循环
    结束循环
结束函数

函数 打印矩阵(matrix1)
    对于 i 从 0 到 matrix1.count - 1 循环执行
        如果 matrix1.data[i].x 等于 -1 则
            继续
        结束如果
        输出 matrix1.data[i].x, matrix1.data[i].y, matrix1.data[i].value 到 输出
流
    结束循环
结束函数

函数 主函数
    声明 matrix1, matrix2 为 matrix 类型
```

从输入流中读取 matrix1.row, matrix1.col, matrix1.count, matrix2.count
matrix2.row 赋值为 matrix1.row
matrix2.col 赋值为 matrix1.col
输入矩阵数据(matrix1, matrix2)
矩阵相加(matrix1, matrix2)
排序矩阵(matrix1)
打印矩阵(matrix1)
结束函数
```

**四 使用说明、测试分析及结果**

**1.说明如何使用你编写的程序；**

1.输入输出均使用标准的 scanf()和 printf()函数。
2.运行环境为 Codeblocks

**2.测试结果与分析；**

输入：
3 4 3 2
1 1 1
1 3 1
2 2 2
1 2 1
2 2 3
输出：
1 1 1
1 2 1
1 3 1
2 2 5

**3.调试过程中遇到的问题是如何解决提以及对设计与实现的回顾讨论和分析**

逻辑错误:在矩阵相加函数 `add_matrices` 中存在逻辑错误,导致矩阵相加的结果不正确。

解决方法：
逐步调试：通过添加输出语句或者使用调试工具，逐步跟踪程序的执行过程，观察变量的取值和程序的流程，从而找出问题所在。
代码审查：仔细审查代码，特别是涉及数组索引和循环边界的地方，确保没有越界访问的问题。

**4.运行界面**
```



五、实验总结

在完成稀疏矩阵相加与排序的实验过程中，我收获了许多宝贵的经验和体会，以下是我的心得与感悟：

1. 系统学习理论知识： 在实验开始之前，我系统地学习了稀疏矩阵的相关理论知识，包括稀疏矩阵的定义、表示方法以及相加与排序的算法原理。这为我后续的实验操作提供了坚实的理论基础。

2. 加强编程能力： 实验过程中，我不仅需要运用所学的数据结构和算法知识，还需要灵活运用编程语言来实现稀疏矩阵的相加与排序功能。通过编写代码、调试程序，我进一步提高了自己的编程能力和实践能力。

3. 探索解决问题的方法： 在实验过程中，我遇到了各种各样的问题，包括编译错误、逻辑错误以及边界情况处理等。针对这些问题，我学会了通过查阅资料、调试程序、与同学讨论等方式来解决问题，不断积累了解决问题的经验。

4. 不断总结与反思： 在实验完成之后，我及时总结了实验过程中遇到的问题、解决方法以及心得体会，不断反思自己的不足之处，并且记录下来，以便今后查阅和借鉴，这有助于我在以后的学习和工作中更好地提高自己。

◎实验题目：

实验 2.3：稀疏矩阵加法，用十字链表实现 C=A+B

◎实验内容：

第一行输入四个正整数，分别是两个矩阵的行 m、列 n、第一个矩阵的非零元素的个 数 t1 和第二个矩阵的非零元素的个数 t2。 接下来的 t1+t2 行是三元组，分别是第一个矩阵的数据和第二个矩阵的数据。三元组 的第一个元素表示行号，第二个元素表示列号，第三个元素是该项的值。

输出相加后的矩阵三元组。

一、需求分析

1. 输入的形式和输入值的范围：

输入包括两个内容，第一行输入四个正整数 m 和 n,分别表示两个矩阵的行数和列数，用空格隔开，后续输入矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

2. 输出的形式：

输出为相加后的稀疏矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

3. 程序所能达到的功能：

将稀疏矩阵相加。

测试数据：

输入：

3 4 3 2
1 1 1
1 3 1
2 2 2
1 2 1
2 2 3

输出：

1 1 1
1 2 1
1 3 1
2 2 5

二 概要设计

概要设计思路：

1. 选择数据结构：

选择一种适合表示稀疏矩阵的数据结构。在这个代码中，选择了十字链表来表示稀疏矩阵。这种数据结构既能高效地存储非零元素，又能方便地进行行和列的操作。

2. 设计数据结构：

设计稀疏矩阵的数据结构，包括矩阵节点的结构和存储矩阵的结构。在这个代码中，使用了`OLNode`结构表示矩阵节点，包含行列坐标和元素值等信息；使用`CrossList`结构表示稀疏矩阵，包含行列指针数组以及矩阵的行数、列数和非零元素个数等信息。

3. 设计基本操作：

设计操作稀疏矩阵的基本函数，包括初始化矩阵、插入节点、创建矩阵、矩阵相加和打印矩阵等。这些函数实现了对稀疏矩阵的初始化、插入节点、创建、相加和打印等操作。

4. 设计主程序流程：

设计主函数，组织调用上述函数完成整个程序的功能。主函数主要是调用初始化矩阵、创建矩阵、相加矩阵和打印矩阵等步骤，完成稀疏矩阵的相加功能。

5. 测试与调试：

设计测试用例，对程序进行测试，确保程序在各种情况下都能正确运行。通过调试解决程序中的问题，确保程序的正确性和稳定性。

概要设计思路主要是围绕稀疏矩阵的数据表示和操作展开,通过选择合适的数据结构和设计合理的函数，实现稀疏矩阵的创建、相加和打印等功能。

各函数模块功能：

1. `initMatrix`: 初始化矩阵 A 和矩阵 B，从标准输入中读取矩阵的行数、列数和非零元素个数。

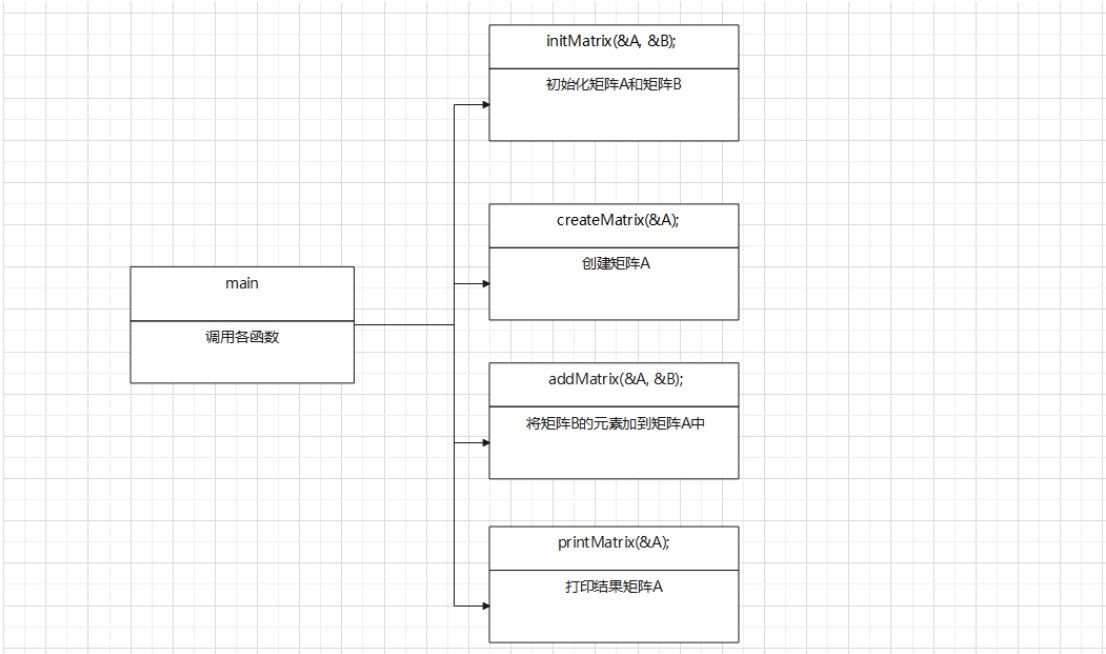
2. `insertNode`: 向矩阵中插入节点。根据节点的行列坐标以及元素值，在十字链表中找到合适的位置插入节点。

3. `createMatrix`: 创建矩阵。根据输入的非零元素信息，创建稀疏矩阵，并利用`insertNode`函数将节点插入到十字链表中。

4. `addMatrix`: 将矩阵 B 的元素加到矩阵 A 中。遍历矩阵 B 的每一行，将每个节点的元素加到矩阵 A 对应位置的元素上。

5. `printMatrix`: 打印矩阵。遍历十字链表，输出每个非零元素的行列坐标和元素值。

6. `main`: 主函数。初始化矩阵 A 和矩阵 B，创建它们，将矩阵 B 的元素加到矩阵 A 中，然后打印结果矩阵 A。



这些函数一起完成了稀疏矩阵的创建、相加以及结果的打印功能。

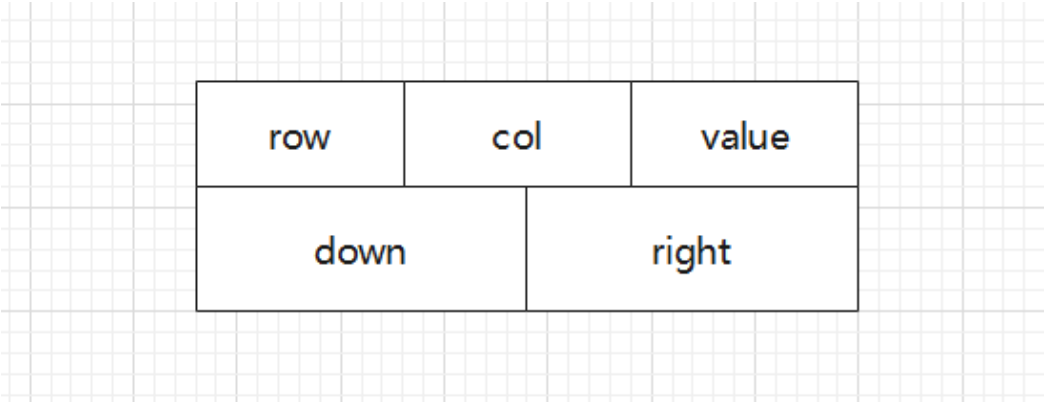
三 详细设计

伪代码：

plaintext

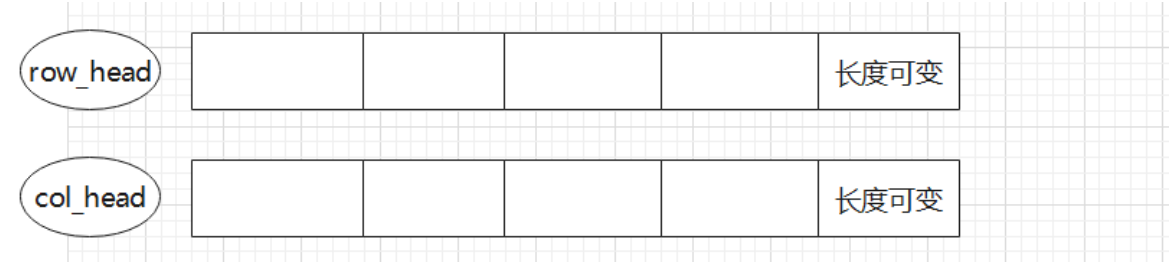
定义结构体 OLNode:

成员变量：row, col, e, right, down



定义结构体 CrossList:

成员变量：row_head, col_head, m, n, len



函数 initMatrix(A: CrossList, B: CrossList):

```
    读取 A 的行数、列数、非零元素个数
    读取 B 的非零元素个数
    设置 B 的行数为 A 的行数
    设置 B 的列数为 A 的列数

函数 insertNode(L: CrossList, P: OLNode):
    创建新节点 N
    设置 N 的行、列、元素值为 P 的行、列、元素值
    如果 L 的行指针数组中当前行为空或新节点的列坐标小于当前行的第一个节点的列坐标:
        将新节点的右指针指向当前行的第一个节点
        更新当前行的第一个节点为新节点
    否则:
        遍历当前行找到新节点应该插入的位置:
            如果当前节点的右节点为空或新节点的列坐标小于当前节点的右节点的列坐标:
                将新节点的右指针指向当前节点的右节点
                将当前节点的右指针指向新节点
    如果 L 的列指针数组中当前列为空或新节点的行坐标小于当前列的第一个节点的行坐标:
        将新节点的下指针指向当前列的第一个节点
        更新当前列的第一个节点为新节点
    否则:
        遍历当前列找到新节点应该插入的位置:
            如果当前节点的下节点为空或新节点的行坐标小于当前节点的下节点的行坐标:
                将新节点的下指针指向当前节点的下节点
                将当前节点的下指针指向新节点

函数 createMatrix(M: CrossList):
    创建 M 的行指针数组和列指针数组
    初始化行指针数组和列指针数组为 NULL
    循环读取非零元素的信息并插入矩阵:
        创建新节点 p
        读取非零元素的行、列、值
        将新节点插入矩阵

函数 addMatrix(A: CrossList, B: CrossList):
    循环遍历矩阵 B 的每一行:
        如果当前行为空:
            继续下一次循环
        否则:
            如果矩阵 A 的当前行为空:
                将矩阵 B 的当前行节点复制到矩阵 A 中
```

否则:
遍历矩阵 B 的当前行的每个节点:
遍历矩阵 A 的当前行的每个节点:
如果当前节点的列坐标相同:
将对应位置的元素相加
跳出内层循环
如果当前节点的列坐标大于当前矩阵 A 的节点的列坐标，并且小于下一个节点的列坐标:
将当前节点插入矩阵 A 中
跳出内层循环
如果已经遍历到矩阵 B 的当前行的最后一个节点:
跳出外层循环

函数 printMatrix(L: CrossList):
循环遍历矩阵的每一行:
遍历当前行的每个节点:
如果当前节点的元素值不为 0:
打印当前节点的行、列、元素值

主函数:
创建 CrossList 结构体 A, B
初始化矩阵 A 和矩阵 B
创建矩阵 A
创建矩阵 B
将矩阵 B 的元素加到矩阵 A 中
打印结果矩阵 A

这些伪代码描述了程序的基本逻辑和各个函数的功能。

四 使用说明、测试分析及结果

1.说明如何使用你编写的程序；

1.输入输出均使用标准的 scanf()和 printf()函数。
2.使用十字链表链表的思路，程序基于 C99 标准编写
3.运行环境为 VS

2.测试结果与分析；

输入：
3 4 3 2
1 1 1
1 3 1
2 2 2
1 2 1
2 2 3

输出：
1 1 1
1 2 1
1 3 1
2 2 5
结果正确

3. 调试过程中遇到的问题是如何解决提以及对设计与实现的回顾讨论和分析

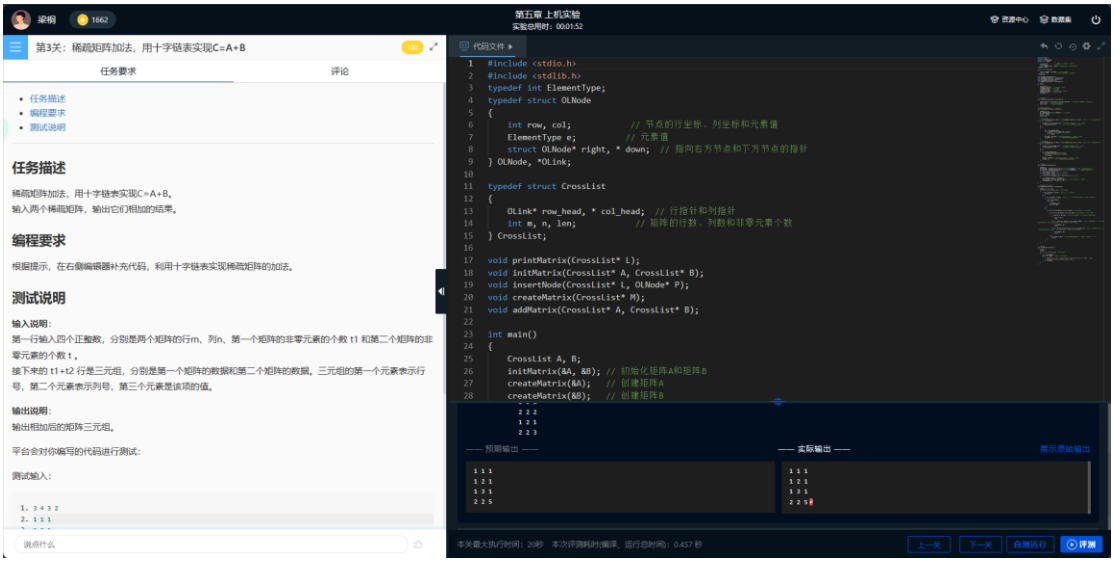
控制流不完整：在 `addMatrix()` 函数中，循环控制结构不够优化，导致意外行为。

解决：确保循环条件和控制流结构良好，并覆盖所有可能的情况。

未定义行为：如果未正确初始化输入矩阵或提供了意外输入，则可能出现未定义行为。

解决：添加错误检查机制，确保输入矩阵已正确初始化，并处理意外输入。

4 运行界面



五、实验总结

在完成这次关于稀疏矩阵加法的实验过程中，我学到了许多关于数据结构和算法的知识，并且提升了我的编程技能。以下是我在实验中的总结和心得体会：

1. 理解稀疏矩阵表示方法：稀疏矩阵与常规矩阵不同，它们主要由大量的零元素组成，因此采用特定的表示方法可以节省存储空间和提高运算效率。在本次实验中，我了解了使用十字链表表示稀疏矩阵的方法，并学会了如何操作这种数据结构。

2. 熟悉链表操作：实验中涉及了链表的插入、遍历等操作。通过编写代码实现这些操作，我加深了对链表数据结构的理解，并学会了如何在程序中灵活运用链表。

3. 细心和耐心是关键：在处理指针操作和链表插入时，细心和耐心是非常重要的。一些小错误可能导致程序出现难以察觉的 bug，因此我学会了在编程过程中注重细节，并且耐心地调试和修改代码。

4. 对边界条件的处理：稀疏矩阵加法涉及到处理不同的边界情况，比如矩阵为空、矩阵行

列数不匹配等。在编程过程中，我学会了如何预先考虑这些边界情况，并编写代码来处理它们，以确保程序的健壮性和稳定性。

◎实验题目：

实验 2.4：稀疏矩阵的乘法

◎实验内容：

计算两个稀疏矩阵的乘法

输入第一个矩阵的行数和列数，再输入该矩阵的三元组形式，以 0 0 0 结束。 然后输入第二个矩阵的行数和列数，再输入该矩阵的三元组形式，以 0 0 0 结束。

输出相加后的矩阵三元组。

一、需求分析

1. 输入的形式和输入值的范围：

输入包括两个内容，第一行输入 2 个正整数 m 和 n,分别表示矩阵的行数和列数，用空格隔开，后续输入矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

2. 输出的形式：

输出为相加后的稀疏矩阵三元组，数字以空格隔开，一个元素的三个相关参数占一行。

3. 程序所能达到的功能：

将稀疏矩阵相乘。

测试数据：

输入：

3 3
1 1 1
2 2 2
2 3 4
3 1 -4
0 0 0
3 3
1 3 -2
2 3 -5
3 1 8
3 2 -6
0 0 0

输出：

1 3 -2
2 1 32
2 2 -24
2 3 -10
3 3 8

二 概要设计

这段代码解决了稀疏矩阵的乘法问题，即给定两个稀疏矩阵 A 和 B，求它们的乘积矩阵 C。下面是解决问题的思路概要：

1.稀疏矩阵的表示：

使用三元组结构体来表示稀疏矩阵，其中每个非零元素由行号、列号和元素值组成。

通过结构体数组来存储稀疏矩阵的非零元素，使得每个元素都可以用一个结构体表示。

2. 矩阵乘法的实现：

遍历矩阵 A 的每一行，以及矩阵 B 的每一列。

在遍历过程中，查找可以相乘的元素对，即 A 矩阵的当前行和 B 矩阵的当前列以及它们对应的转置位置是否存在非零元素。

如果存在相乘的元素对，则将其乘积累加到结果矩阵 C 的对应位置。

注意，为了减少时间复杂度，可以使用稀疏矩阵的特性，只查找与当前元素位置相关的非零元素，而不是遍历整个矩阵。

3. 输出结果：

将得到的乘积矩阵 C 的每个非零元素输出到标准输出，按照行号、列号和元素值的格式输出。

函数功能模块：

1. `init()` 函数：

功能：用于初始化稀疏矩阵。

输入：无。

输出：返回一个初始化后的稀疏矩阵。

实现：该函数首先创建一个空的稀疏矩阵 `A`，然后通过用户输入获取矩阵的行数、列数以及非零元素的信息，构建稀疏矩阵的三元组表示。

2. `mult()` 函数：

功能：实现稀疏矩阵的乘法运算。

输入：两个稀疏矩阵 `A` 和 `B`。

输出：返回两个稀疏矩阵的乘积矩阵 `C`。

实现：该函数通过遍历 `A` 的行和 `B` 的列，找到对应位置的非零元素，然后进行相乘累加操作,得到乘积矩阵的每个元素。如果乘积结果不为零,则将结果添加到结果矩阵 `C` 中。

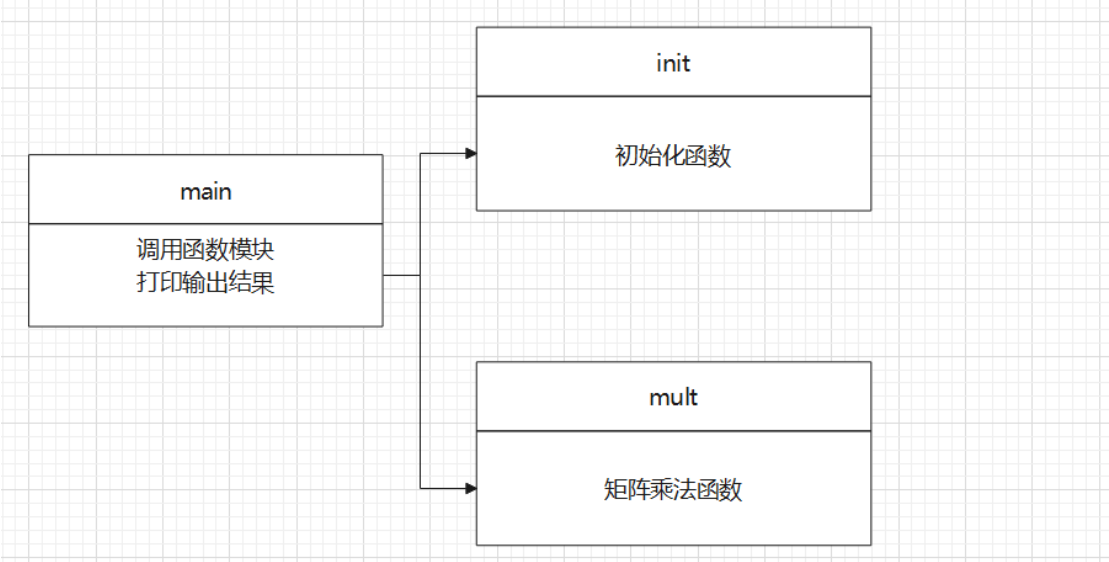
3. `main()` 函数：

功能：程序的入口，用于调用其他函数完成稀疏矩阵乘法并输出结果。

输入：用户从标准输入中输入稀疏矩阵的信息。

输出：输出稀疏矩阵乘法的结果。

实现：该函数首先调用 `init()` 函数初始化两个输入的稀疏矩阵 `A` 和 `B`，然后调用 `mult()` 函数计算它们的乘积矩阵 `C`，最后输出乘积矩阵的每个元素。



三 详细设计

定义三元组结构体 Triple:

属性：行号 row、列号 col、元素值 e

定义稀疏矩阵结构体 TSMatrix:

属性：数据 data、行数 m、列数 n、非零元素个数 len

初始化函数 init(P:指向 TSMatrix 的指针):

```
创建稀疏矩阵 A
初始化 A.m 和 A.n
初始化 A.len 为 0
读取非零元素信息，直到输入 0 0 0 为止：
    读取行号 x、列号 y、元素值 z
    将(x, y, z) 添加到 A.data 中
    A.len 自增
将 A 赋值给 P 指向的变量
返回 A
```

矩阵乘法函数 mult(A: TSMatrix, B: TSMatrix) -> C: TSMatrix:

```
创建稀疏矩阵 C
初始化 C.m 为 A.m，C.n 为 B.n，C.len 为 0
遍历 A 的每一行 i：
    遍历 B 的每一列 j：
        将 temp 初始化为 0
        在 A.data 中查找与当前行 i 和列 j 相乘的元素：
            若存在元素 (i, p, x) 且 B 中存在元素 (p, j, y)：
                将 temp 加上 x * y
        若 temp 不为 0：
            将 (i, j, temp) 添加到 C.data 中
```

C.len 自增

返回 C

主函数 main:

定义整数变量 i

定义稀疏矩阵变量 A、B、C

初始化矩阵 A 和 B，分别赋值给 A 和 B

计算矩阵 A 和 B 的乘积，结果赋值给 C

输出结果矩阵 C 的非零元素信息

四 使用说明、测试分析及结果

1.说明如何使用你编写的程序；

1.输入输出均使用标准的 scanf()和 printf()函数。

2.使用三元组的思路，程序基于 C99 标准编写

3.运行环境为 VS

2 测试结果与分析；

输入：

3 3

1 1 1

2 2 2

2 3 4

3 1 -4

0 0 0

3 3

1 3 -2

2 3 -5

3 1 8

3 2 -6

0 0 0

输出：

1 3 -2

2 1 32

2 2 -24

2 3 -10

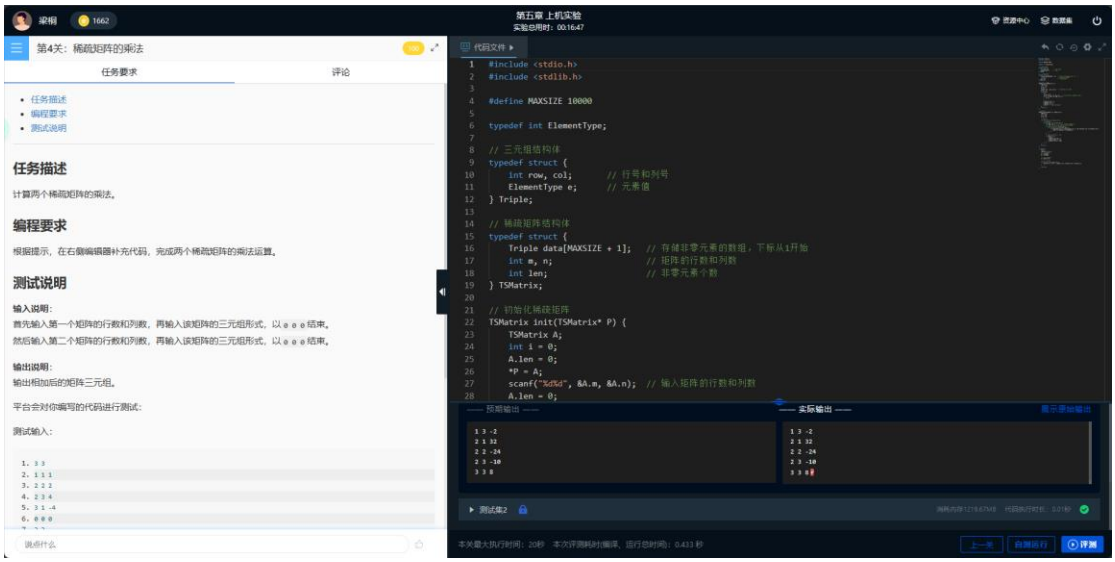
3 3 8

结果正确

3.调试过程中遇到的问题是如何解决提以及对设计与实现的回顾讨论和分析

内存越界或溢出问题：由于稀疏矩阵的数据结构使用了数组，可能会导致数组越界或内存溢出的问题。解决方法是确保数组的索引值在合法范围内，并且适当地控制数据结构的大小，以避免内存溢出。

4.运行界面



五、实验总结

实验心得总结反思：

在完成稀疏矩阵乘法的代码实现过程中，我遇到了一些挑战和问题，但通过不断地思考和尝试，最终成功地完成了任务。以下是我的实验心得总结和反思：

1. 深入理解算法原理：实现稀疏矩阵乘法需要理解矩阵乘法的基本原理，并且针对稀疏矩阵的特点进行算法设计。通过深入理解算法原理，我能够更好地理解代码逻辑，准确地实现算法。

2. 调试和测试：在编写代码的过程中，我遇到了一些语法错误和逻辑错误，通过调试工具逐行调试代码，查找并解决了这些问题。同时，我编写了一些测试用例，验证代码的正确性和稳定性。

通过这次实验，我不仅掌握了稀疏矩阵乘法算法的实现方法，还提高了自己的编程能力和问题解决能力。我相信这些经验和收获将对我的编程之路产生积极的影响。

教师评语：

实验成绩：

指导教师签名：

批阅日期：