

# 《数据结构》实验报告

班 级: \_\_\_\_\_

姓 名: \_\_\_\_\_

学 号: \_\_\_\_\_

E-mail: \_\_\_\_\_

日 期: \_\_\_\_\_

## ◎实验题目:

### 2.1 稀疏矩阵的转置

#### Description

输出稀疏矩阵的转置矩阵。(行列均不大于20)

## ◎实验内容:

### 2.1

#### Input

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。

#### Output

转置后的矩阵。

### 一、需求分析

本次实验的目的是实现一个稀疏矩阵的快速转置算法。输入为一个稀疏矩阵,输出为该矩阵的转置矩阵。需要解决的关键问题是如何在不遍历整个矩阵的情况下找到非零元素的位置,并将它们转置。

### 二、概要设计

为了解决这个问题,我们将使用三元组顺序表(TSMatrix)来表示稀疏矩阵。然后,我们将使用快速转置算法对稀疏矩阵进行转置。最后,我们将输出转置后的稀疏矩阵。

### 三、详细设计

数据结构设计:

1. 定义一个三元组结构体(Triple),包含行(row)、列(col)和元素值(e)三个属性。
2. 定义一个稀疏矩阵结构体(TSMatrix),包含一个存放三元组的数组(data)、矩阵的行数(m)、列数(n)和非零元素个数(len)。

算法设计:

1. 初始化稀疏矩阵(InitSMatrix): 给定行数、列数和非零元素个数,初始化一个稀疏矩阵。
2. 创建稀疏矩阵(CreateSMatrix): 从输入中读取矩阵的行数、列数和非零元素,并将它们存储在稀疏矩阵中。
3. 快速转置算法(FastTranspseTSMatrix):
  - a. 初始化两个辅助数组(num、cpot)。
  - b. 遍历稀疏矩阵的非零元素,将它们放入新的三元组顺序表中。
  - c. 更新cpot数组。

4. 打印稀疏矩阵 (PrintSMatrix)：将稀疏矩阵的三元组顺序表输出。

FastTranspseTSMatrix 算法的伪代码：

FastTranspseTSMatrix(M, T)

1. 初始化辅助数组 num 和 cpot
2. for i = 1 to M.n
3.     num[i] = 0
4. for i = 1 to M.len
5.     num[M.data[i].col] += 1
6. cpot[1] = 1
7. for i = 2 to M.n
8.     cpot[i] = cpot[i - 1] + num[i - 1]
9. for i = 1 to M.len
10.     col = M.data[i].col
11.     j = cpot[col]
12.     T.data[j].row = M.data[i].col
13.     T.data[j].col = M.data[i].row
14.     T.data[j].e = M.data[i].e
15.     cpot[col] += 1

#### 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；

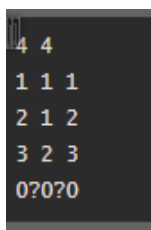
第一行输入四个整数表示矩阵的行数和列数，矩阵 A 中非零元素的个数，矩阵 B 中非零元素的个数；

然后依次输入矩阵 A 三元组，矩阵 B 的三元组；

最后输入回车得到矩阵相加后的矩阵；

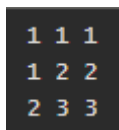
##### 2、测试结果与分析；

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
```

##### 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

1. 输入格式问题：在读取输入时，我们需要确保输入的稀疏矩阵数据格式正确。在创建稀疏矩阵时，我们使用了一个简单的输入终止条件 ( $i == 0 \ \&\& \ j == 0 \ \&\& \ e == 0$ )，这使得我们能够在读取完所有非零元素后正确地终止输入。这种方法对于本实验是有效的，但在实际应用中可能需要更加健壮的输入验证方法。

2. 初始化辅助数组：在 FastTranspseTSMatrix 算法中，我们需要正确地初始化辅助数组 num 和 cpot。一开始我们没有将 num 数组的元素初始化为 0，导致了计算错误。在发现问题后，我们添加了一个循环来初始化 num 数组，并确保了正确的计算结果。

3. 数组下标问题：在处理三元组顺序表时，需要确保数组下标从 1 开始，因为将 0 位置用于输入终止条件。在调试过程中，发现在某些情况下，错误地从 0 开始访问数组，这导致了错误的结果。

回顾讨论与分析：

在设计和实现稀疏矩阵快速转置算法的过程中，我意识到了将问题分解为更小的子任务的重要性。通过将问题分解为初始化、创建、转置和打印稀疏矩阵的子任务，我们能够更清晰地理解算法的逻辑，并且更容易地调试代码。

## 运行界面

```
(base) PS D:\CodespaceX
4 4
1 1 1
2 1 2
3 2 3
0 0 0
1 1 1
1 2 2
2 3 3
```

## 五、实验总结

在本实验中，成功实现了一个快速转置算法，该算法能在较短时间内对稀疏矩阵进行转置。通过使用三元组顺序表表示稀疏矩阵，避免了对整个矩阵的遍历，从而提高了算法的效率。

### 2.2 ◎实验题目：

#### Description

输入两个稀疏矩阵，输出它们相加的结果。

### ◎实验内容

#### 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。

#### Output

输出相加后的矩阵三元组。

### 一、需求分析

本次实验的目标是实现稀疏矩阵的加法操作。输入两个稀疏矩阵，每个稀疏矩阵包括行数、列数和非零元素个数，以及非零元素的行、列和值。输出结果矩阵的非零元素及其行、列和值。

### 二、概要设计

为了实现稀疏矩阵加法，首先需要定义稀疏矩阵的数据结构，然后创建两个稀疏矩阵并从输入读取数据。接下来，实现稀疏矩阵加法的算法，最后输出结果矩阵的非零元素及其行、列和值。

### 三、详细设计

1. 定义稀疏矩阵数据结构： `struct Triple {  
int row, col; int value; };`  
`struct SparseMatrix { Triple data[MAX_SIZE + 1]; int rows, cols, numNonZeroElements; };`
2. 创建稀疏矩阵： 从输入读取行数、列数、非零元素个数和非零元素的行、列和值，存储在 `SparseMatrix` 结构体中。
3. 实现稀疏矩阵加法算法： 遍历两个矩阵的非零元素，比较行和列，将相同位置的元素值相加，并将结果存储在结果矩阵中。
4. 输出结果矩阵： 遍历结果矩阵的非零元素，输出其行、列和值。

实现稀疏矩阵加法算法伪代码：

```
function addSparseMatrices(matrix1, matrix2, resultMatrix)
    i = 1
    j = 1
    k = 0
    while i <= matrix1.numNonZeroElements and j <= matrix2.numNonZeroElements
        if matrix1.data[i].row < matrix2.data[j].row
            resultMatrix.data[++k] = matrix1.data[i++]
        else if matrix1.data[i].row > matrix2.data[j].row
            resultMatrix.data[++k] = matrix2.data[j++]
        else
            if matrix1.data[i].col < matrix2.data[j].col
                resultMatrix.data[++k] = matrix1.data[i++]
            else if matrix1.data[i].col > matrix2.data[j].col
                resultMatrix.data[++k] = matrix2.data[j++]
            else
                resultMatrix.data[++k].row = matrix1.data[i].row
                resultMatrix.data[k].col = matrix1.data[i].col
                resultMatrix.data[k].value = matrix1.data[i].value +
matrix2.data[j].value
                i++
                j++
    while i <= matrix1.numNonZeroElements
        resultMatrix.data[++k] = matrix1.data[i++]
    while j <= matrix2.numNonZeroElements
        resultMatrix.data[++k] = matrix2.data[j++]
    resultMatrix.numNonZeroElements = k
```

### 四 使用说明、测试分析及结果

说明如何使用你编写的程序；

第一行输入四个整数表示矩阵的行数和列数，矩阵 A 中非零元素的个数，矩阵 B 中非零元素的个数；然后依次输入矩阵 A 三元组，矩阵 B 的三元组；最后输入回车得到矩阵相加后

的矩阵；

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
```

遇到的问题及解决方法

在实现稀疏矩阵加法算法时，需要注意矩阵元素的行、列比较和相同位置元素值的相加。通过仔细分析矩阵加法过程，编写正确的算法逻辑解决了问题。

运行界面

```
(base) PS D:\CodespaceX\
4 4
1 3 1
2 2 2
1 2 1
2 2 3
1 1 1
1 2 1
1 3 1
2 2 5
```

## 五、回顾讨论与实验总结

本次实验实现了稀疏矩阵加法操作。通过定义合适的数据结构和算法，成功地完成了任务。在实现过程中，注意到了矩阵元素行、列比较和相同位置元素值相加的问题，并通过仔细分析解决了这些问题。总的来说，本次实验加深了对稀疏矩阵加法的理解和实现。

### 2.3

#### ◎实验题目：

矩阵加法，十字链表实现  $C = A + B$ ；

#### ◎实验内容：

使用十字链表输入两个稀疏矩阵，输出它们相加的结果

## 实验2.3：稀疏矩阵加法，用十字链表实现C=A+B

Time Limit: 3000ms , Memory Limit: 10000KB , Accepted: 0 , Total Submissions: 0

### Description

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。

### Output

输出相加后的矩阵三元组。

### 一、需求分析

本次实验的目标是实现两个十字链表表示的稀疏矩阵的加法。输入两个稀疏矩阵的行数、列数、非零元素个数以及非零元素的行、列和值。输出两个矩阵相加后的结果矩阵的非零元素及其行、列和值。

### 二、概要设计

为了实现稀疏矩阵加法，首先需要定义稀疏矩阵的十字链表数据结构，然后创建两个稀疏矩阵并从输入读取数据。接下来，实现稀疏矩阵加法的算法，最后输出结果矩阵的非零元素及其行、列和值。

### 三、详细设计

#### 1. 定义十字链表数据结构：

```
typedef struct OLNode{
    int i,j;
    int val;
    struct OLNode *right,*down;
}OLNode,*OLink;
```

```
typedef struct{
    OLink *rhead,*chead;
    int mu,nu,tu;
}CrossList;
```

#### 2. 创建稀疏矩阵：

从输入读取行数、列数、非零元素个数和非零元素的行、列和值，存储在 CrossList 结构体中。

#### 3. 实现稀疏矩阵加法算法伪代码：

```
function AddElem(M, N)
    for k = 1 to N->mu
        p = N->rhead[k]
        while p
            q = new OLNode
            q->down = p->down
            q->right = p->right
            q->val = p->val
```

```

q->i = p->i
q->j = p->j
Insert(M, q)
p = p->right

```

4. 输出结果矩阵： 遍历结果矩阵的非零元素，输出其行、列和值。

#### 四 使用说明、测试分析及结果

说明如何使用你编写的程序；

第一行输入四个整数表示矩阵的行数和列数，矩阵 A 中非零元素的个数，矩阵 B 中非零元素的个数；

然后依次输入矩阵 A 三元组，矩阵 B 的三元组；

最后输入回车得到矩阵相加后的矩阵；

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
2 2 5

```

输出

```

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

```

运行界面

```

(base) PS D:\CodespaceX\C++\data structure>
3 4 3 2
1 1 1
1 3 1
2 2 2
1 2 1
2 2 3
2 2 5
1 1 1
1 2 1
1 3 1
2 2 5
(base) PS D:\CodespaceX\C++\data structure>

```

遇到的问题及解决方法

在实现十字链表表示的稀疏矩阵加法时，需要注意在插入非零元素时的行、列处理。通过仔细分析十字链表的结构和遍历方式，编写正确的算法逻辑解决了问题。

## 五、回顾讨论与实验总结

本次实验实现了十字链表表示的稀疏矩阵加法操作。通过定义合适的数据结构和算法，成功地完成了任务。在实现过程中，注意到了插入非零元素时行、列处理的问题，并通过仔细分析解决了这些问题。总的来说，本次实验提高了对十字链表数据结构的理解和应用能力。

### 2.4

#### ◎实验题目：

计算两个稀疏矩阵的乘法

#### ◎实验内容：

使用三元组表输入两个稀疏矩阵，输出它们相乘的结果

### 实验2.4：稀疏矩阵的乘法

Time Limit: 3000ms , Memory Limit: 10000KB , Accepted: 0 , Total Submissions: 0

#### Description

计算两个稀疏矩阵的乘法

#### Input

首先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

#### Output

输出相加后的矩阵三元组。

#### 一、需求分析

本次实验的目标是实现稀疏矩阵的乘法操作。输入两个稀疏矩阵，每个稀疏矩阵包括行数、列数和非零元素，以及非零元素的行、列和值。输出结果矩阵的非零元素及其行、列和值。

#### 二、概要设计

为了实现稀疏矩阵乘法，首先需要定义稀疏矩阵的数据结构，然后创建两个稀疏矩阵并从输入读取数据。接下来，实现稀疏矩阵乘法的算法，最后输出结果矩阵的非零元素及其行、列和值。

#### 三、详细设计

##### 定义稀疏矩阵数据结构：

```
typedef struct triple {
    int r, c, e;
} triple;

typedef struct matrix {
    int m, n, t;
    triple data[MAXSIZE];
} matrix;
```



### 创建稀疏矩阵:

从输入读取行数、列数、非零元素个数和非零元素的行、列和值，存储在 matrix 结构体中。

### 实现稀疏矩阵乘法算法伪代码:

```
function multiplyMatrix(A, B, C)
    temp = 0
    for i = 1 to A.m
        for j = 1 to B.n
            for p = 1 to A.t
                for q = 1 to B.t
                    if A.data[p].r == i and B.data[q].c == j and A.data[p].c ==
B.data[q].r
                        temp += A.data[p].e * B.data[q].e
                if temp != 0
                    C->t++
                    C->data[C->t].r = i
                    C->data[C->t].c = j
                    C->data[C->t].e = temp
                temp = 0
```

### 输出结果矩阵:

遍历结果矩阵的非零元素，输出其行、列和值。

## 四 使用说明、测试分析及结果

### 说明如何使用你编写的程序;

第一行输入两个整数表示矩阵的行数和列数;

依次输入矩阵 M 的三元组表，输入 0 0 0 表示结束输入;

输入两个整数表示矩阵的行数和列数;

依次输入矩阵 N 的三元组表，输入 0 0 0 表示结束输入;

最后输入回车得到矩阵相乘后的矩阵;

### 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 3 -2
2 2 1 3 2
2 2 2 -24
2 2 3 -10
3 2 3 8
```

遇到的问题及解决方法

在实现稀疏矩阵乘法算法时，需要注意矩阵元素的行、列匹配以及相乘累加。通过仔细分析矩阵乘法过程，编写正确的算法逻辑解决了问题。

运行界面

```
(base) PS D:\CodespaceX\Cpp\data structure>
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
1 3 2
2 1 32
2 2 -24
2 3 -10
3 3 -8
```

五、回顾讨论与实验总结

本次实验实现了稀疏矩阵乘法操作。通过定义合适的数据结构和算法，成功地完成了任务。在实现过程中，注意到了矩阵元素行、列匹配和相乘累加的问题，并通过仔细分析解决了这些问题。

教师评语:

实验成绩:

指导教师签名:  
批阅日期: