

《数据结构》实验报告2

班级：10012006

姓名：夏卓

学号：2020303245

E-mail: 2769223717@qq.com

日期：2022.4.9

2.2 稀疏矩阵转置

实验内容：

实验2.1：稀疏矩阵转置

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

VIDEO1

Description

输出稀疏矩阵的转置矩阵。（行列均不大于20）

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。

Sample Input

```
4 4
1 1 1
2 1 2
3 2 3
0?0?0
```

Sample Output

```
1 1 1
1 2 2
2 3 3
```

© 2002–2012 JDBSoft.

一、需求分析：

1. 输入：

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。

2. 输出：

输出转置后的矩阵

3. 程序所能达到的功能：

能够将输入的稀疏矩阵转置，能够检测(0?0?0)来结束输入

二、概要设计：

核心思想：

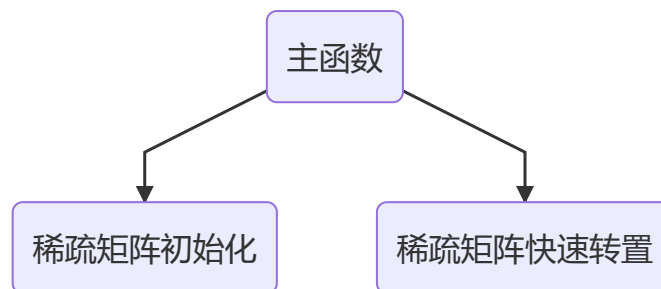
- 利用矩阵三元组表示法存储稀疏矩阵，检测(0?0?0)结束输入
- 利用快速转置法将稀疏矩阵转置，即先通过扫描A的每一列元素个数来预处理出B每一行开始的下标，以免重新对B的三元组进行排序

程序框架：

本程序包含三个模块：

1. 主程序模块；
2. 初始化三元组矩阵模块；
3. 支持稀疏矩阵快速转置的函数模块；


模块调用图：



三、详细设计：

算法实现图：

快速转置法的实现过程

矩阵 A	row	col	e		矩阵 B	row	col	e
1	1	2	12		1	3	-3	
2	1	3	9		2	1	6	15
3	3	1	-3		3	2	1	12
4	3	6	14		4	2	5	18
5	4	3	24		5	3	1	9
6	5	2	18		6	3	4	24
7	6	1	15		7	4	6	-7
8	6	4	-7		8	6	3	14

预处理B每行起始下标

col (三元 组表A)	1	2	3	4	5	6	7
num[col]	2	2	2	1	0	1	0
position[col] (三元组表B)	1	3	5	7	8	8	9

核心算法的伪代码框架：

```

1 //数据结构(矩阵三元组表示的稀疏矩阵)
2 struct Triple
3 {
4     int row, col, x;
5 };
6
7 struct TSMatrix
8 {
9     Triple data[MAXSIZE];
10    int m, n, len;
11 };
12
13 //核心算法
14 void FastTransposeTSMatrix(TSMatrix* A, TSMatrix* B)
15 {
16     B->len = A->len;
17     B->m = A->n;
18     B->n = A->m; //初始化B矩阵行列信息
19     int pos[MAXSIZE], num[MAXSIZE];
20     memset(num, 0, sizeof(num));
21     //计算A的每一列中元素个数, num[i]表示第i列的元素个数
22     for (int i = 0; i < (*A).len; i++)
23         num[(*A).data[i].col]++;

```

```

24 //根据num[i]计算B的每一行元素起始位置
25 pos[1] = 0;
26 for (int i = 2; i <= (*A).n; i++)
27     pos[i] = pos[i - 1] + num[i - 1];
28 //转置
29 for (int i = 0; i < (*A).len; i++)
30 {
31     int col = (*A).data[i].col; //通过A的列col确定B的行pos[col]
32     B->data[pos[col]].col = (*A).data[i].row;
33     B->data[pos[col]].row = (*A).data[i].col;
34     B->data[pos[col]].x = (*A).data[i].x;
35     pos[col]++;
36 }
37 }

```

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

本程序的运行环境为visual studio 2019。

第一行输入矩阵的行数和列数，然后输入矩阵三元组 (*row, col, value*)，输入 (0?0?0) 结束输入

程序会输出按行序为主序排列的转置后的矩阵三元组

2. 测试结果与分析

本程序较好的实现了实验需求，经分析，利用快速转置法可使时间复杂度降为 $O(n + len)$ ，其中 n 为待转置矩阵列数， len 为待转置矩阵元素个数，但其存储空间较列序递增法增加了 $O(n)$ 。

3. 调试过程中遇到的问题及解决方法

- 预处理A的每一列的第一个元素在B中的下标有点繁琐，在最后转置时较容易出错，需要小心仔细

4. 运行界面

输入1:

```
4 4
1 1 1
2 1 2
3 2 3
0?0?0
1 1 1
1 2 2
2 3 3
```

D:\Code\VS code\homework\稀疏矩阵快速转置
按任意键关闭此窗口. . .

输入2:

```
6 6
1 2 12
1 3 9
3 1 -3
3 6 14
4 3 24
5 2 18
6 1 15
6 4 -7
0?0?0
1 3 -3
1 6 15
2 1 12
2 5 18
3 1 9
3 4 24
4 6 -7
6 3 14
```

D:\Code\VS code\homework\稀疏矩阵快速转置\Debug\稀疏矩阵快速转置.

五、实验总结

- 本实验我在编程中用时15分钟
- 在调试中用时3分钟
- 因为样例较为简单，我又重新举了个例子，画出了草稿图分析，再写代码，本题的关键在于预处理出B的每一行首元素下标，然后对其进行转置

2.2 稀疏矩阵加法

实验2.2：稀疏矩阵加法,实现 $C=A+B$

Time Limit: 3000ms , Memory Limit: 1000KB , Accepted: 0 , Total Submissions: 0

VIDEO1

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 t_1 和第二个矩阵的非零元素的个数 t_2 。接下来的 $t_1 + t_2$ 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。

Sample Input

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

Sample Output

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

一、需求分析：

1. 输入：

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 t_1 和第二个矩阵的非零元素的个数 t_2 。接下来的 $t_1 + t_2$ 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。

2. 输出：

输出相加后的矩阵三元组。

3. 程序所能达到的功能：

输入两个稀疏矩阵，输出它们相加的结果。

二、概要设计：

核心思想：

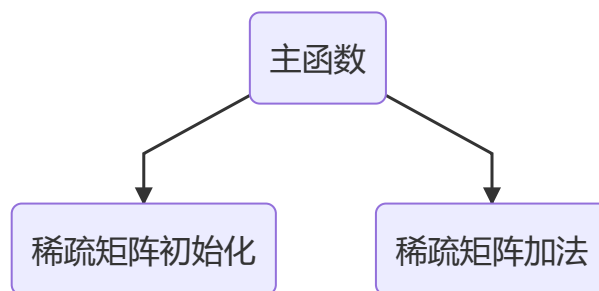
- 利用矩阵三元组表示法存储稀疏矩阵
- 扫描矩阵 A, B ，比较元素的位置下标，分成以下三大类情况：
 1. A 中元素的行小于 B 中元素的行
则直接将 A 中元素拷贝到 C 中对应位置
 2. A 中元素的行大于 B 中元素的行
则直接将 B 中元素拷贝到 C 中对应位置
 3. A 中元素的行等于 B 中元素的行
则又分成三类情况
 - 1) A 中元素的列小于 B 中元素的列
则直接将 A 中元素拷贝到 C 中对应位置
 - 2) A 中元素的列大于 B 中元素的列
则直接将 B 中元素拷贝到 C 中对应位置
 - 3) A 中元素的列等于 B 中元素的列
将二者元素相加，若不为零则将结果存入 C 中对应位置
否则，什么也不做
- 最后将 A 和 B 中剩下的元素存入 C 中对应位置

程序框架：

本程序包含三个模块：

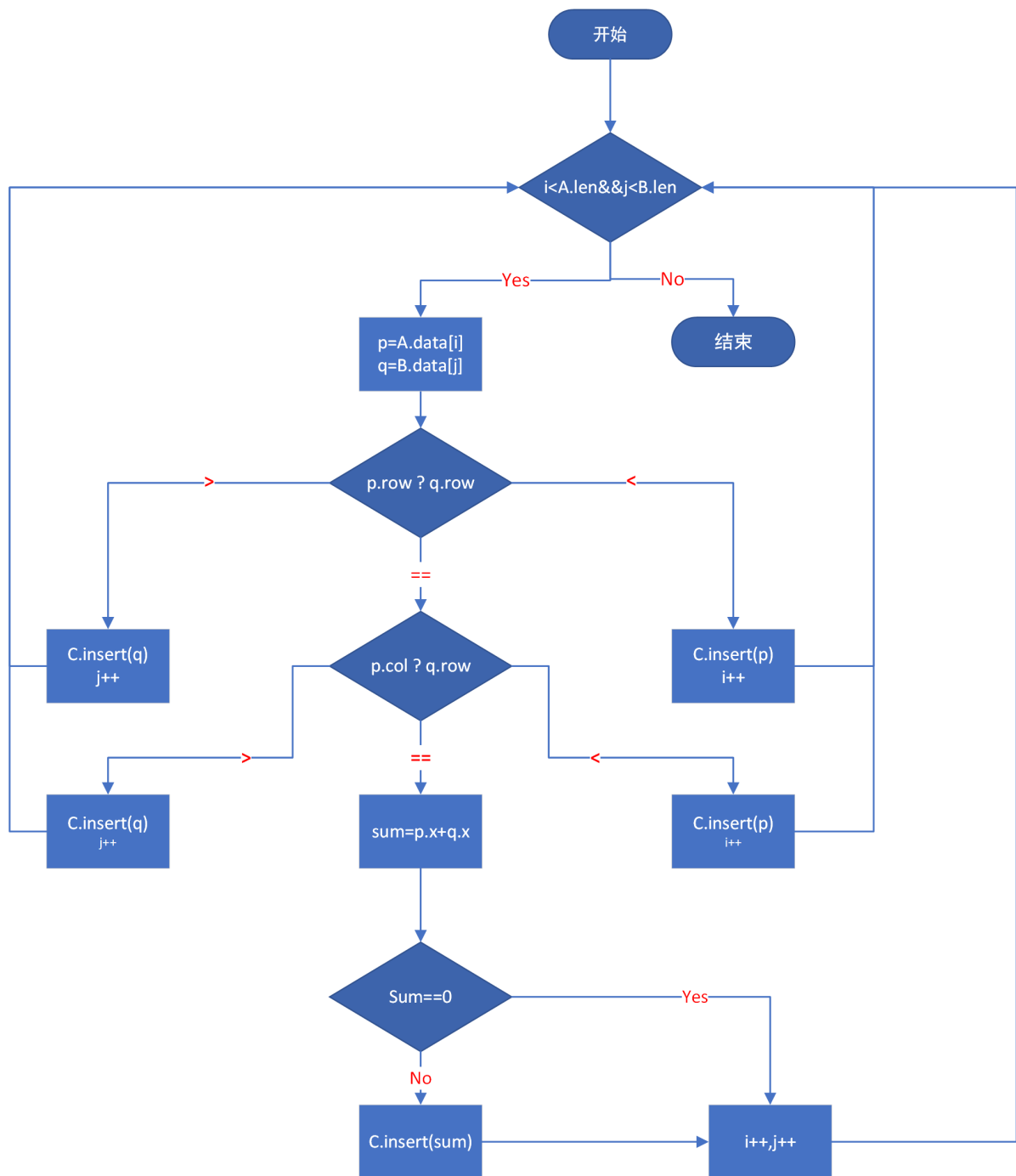
1. 主程序模块；
2. 初始化三元组矩阵模块；
3. 支持稀疏矩阵加法的函数模块；

模块调用图：



三、详细设计：

算法实现图：



核心算法的伪代码框架：

```

1  //数据结构(矩阵三元组表示的稀疏矩阵)
2  struct Triple
3  {
4      int row, col, x;
5  };
6
7  struct TSMatrix
8  {
9      Triple data[MAXSIZE];
10     int m, n, len;
11 };
12
13 //核心算法
14 void MatrixAdd(TSMatrix *A, TSMaTriX *B)

```



```

15 {
16     int i = 0, j = 0, k = 0, sum;
17     while (i < A.len && j < B.len) //扫描A,B中元素
18     {
19         if (A.data[i].row == B.data[j].row)
20         {
21             if (A.data[i].col == B.data[j].col) //若行列相等
22             {
23                 sum = A.data[i].x + B.data[j].x;
24                 //若和不为零则拷贝到C中对应位置
25                 if (sum != 0){
26                     C.data[k].insert(sum);
27                     k++;
28                 }
29                 i++;j++;
30             }
31             else if (A.data[i].col > B.data[j].col) {
32                 C.data[k].insert(B.data[j]); //若A中元素列大,则插入B.data[j]
33                 j++;k++;
34             }
35             else{
36                 C.data[k].insert(A.data[i]); //若B中元素列大,则插入A.data[i]
37                 i++;k++;
38             }
39         }
40         else if (A.data[i].row < B.data[j].row){
41             C.data[k].insert(A.data[i]); //若B中元素行大, ,则插入A.data[i]
42             i++;k++;
43         }
44         else{
45             C.data[k].insert(B.data[j]); //若A中元素行大,则插入B.data[j]
46             j++;k++;
47         }
48     }
49     while (i < A.len){
50         C.data[k].insert(A.data[i]); //将A中剩余元素拷贝到C中
51         i++;k++;
52     }
53     while (j < B.len){
54         C.data[k].insert(B.data[j]); //将B中剩余元素拷贝到C中
55         j++;k++;
56     }
57 }

```

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

本程序的运行环境为visual studio 2019。

第一行分别输入两个矩阵的行数、列数以及矩阵中非零元素个数。接下来分别输入第一个矩阵和第二个矩阵的三元组表示的数据，其中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。

程序会输出按行序为主序排列的两矩阵相加后的矩阵三元组

2. 测试结果与分析

本程序较好的实现了实验需求，分情况讨论即可得出结果

3. 调试过程中遇到的问题及解决方法

无，注意若两元素行列相等时需要判断相加结果是否为0即可

4. 运行界面



```
c:\n Microsoft Visual Studio 调试控制台
3 4 3 2
1 1 1
1 3 1
2 2 2
1 2 1
2 2 3
1 1 1
1 2 1
1 3 1
2 2 5
D:\Code\VS_code\homework\稀疏矩阵加法
```

五、实验总结

- 本实验我在编程中用时12分钟
- 在调试中用时2分钟
- 本题较为简单，需要先分好不同情况，分别处理即可

2.3 稀疏矩阵加法（十字链表法）

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

VIDEO1

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 t_1 和第二个矩阵的非零元素的个数 t_2 。接下来的 t_1+t_2 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。

Sample Input

```
3?4?3?2
1?1?1
1?3?1
2?2?2
1?2?1
2?2?3
```

Sample Output

```
1?1?1
1?2?1
1?3?1
2?2?5
```

一、需求分析：

1. 输入：

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 t_1 和第二个矩阵的非零元素的个数 t_2 。接下来的 t_1+t_2 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。

2. 输出：

输出相加后的矩阵三元组。

3. 程序所能达到的功能：

输入两个稀疏矩阵，输出它们相加的结果，此题需要用十字链表法存储稀疏矩阵

二、概要设计：

核心思想：

- 利用十字链表法存储稀疏矩阵
- 同时遍历A,B的每一行，同样分类讨论即可。因为这里遍历时可以保证A、B中元素的行相等。故仅需讨论列即可：
 - 1) A中元素的列小于B中元素的列
则直接将A中元素拷贝到C中对应位置

2) A中元素的列大于B中元素的列

则直接将B中元素拷贝到C中对应位置

3) A中元素的列等于B中元素的列

将二者元素相加，若不为零则将结果存入C中对应位置

否则，什么也不做

- 将元素存入C中时也要用十字链表法插入，具体流程为：

若待插入元素的行在C中没有元素，则直接插入即可

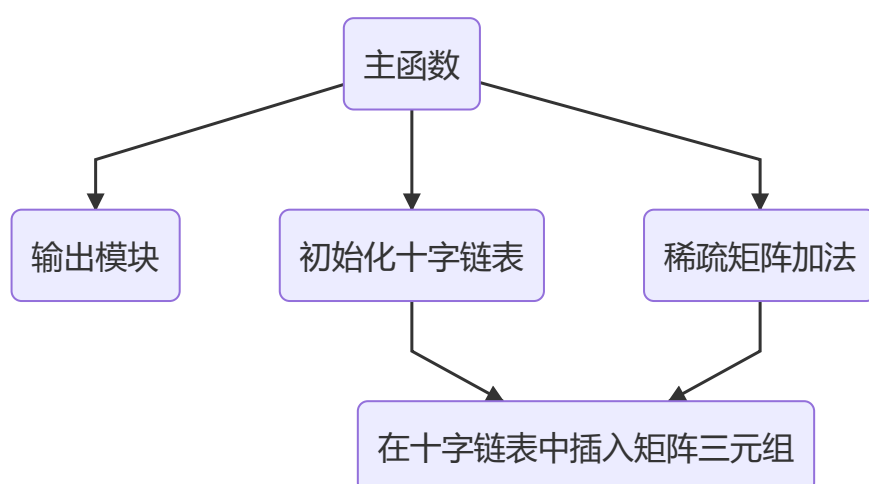
否则需要先根据元素的列在C中找到第一个列大于该列的元素位置，然后将待插入元素插入到此元素之前，若都比待插入元素的列小，则将其插入最后一列

程序框架：

本程序包含五个模块：

1. 主程序模块；
2. 初始化矩阵十字链表模块；
3. 支持稀疏矩阵加法的函数模块；
4. 在十字链表中插入三元组的函数模块；
5. 输出模块

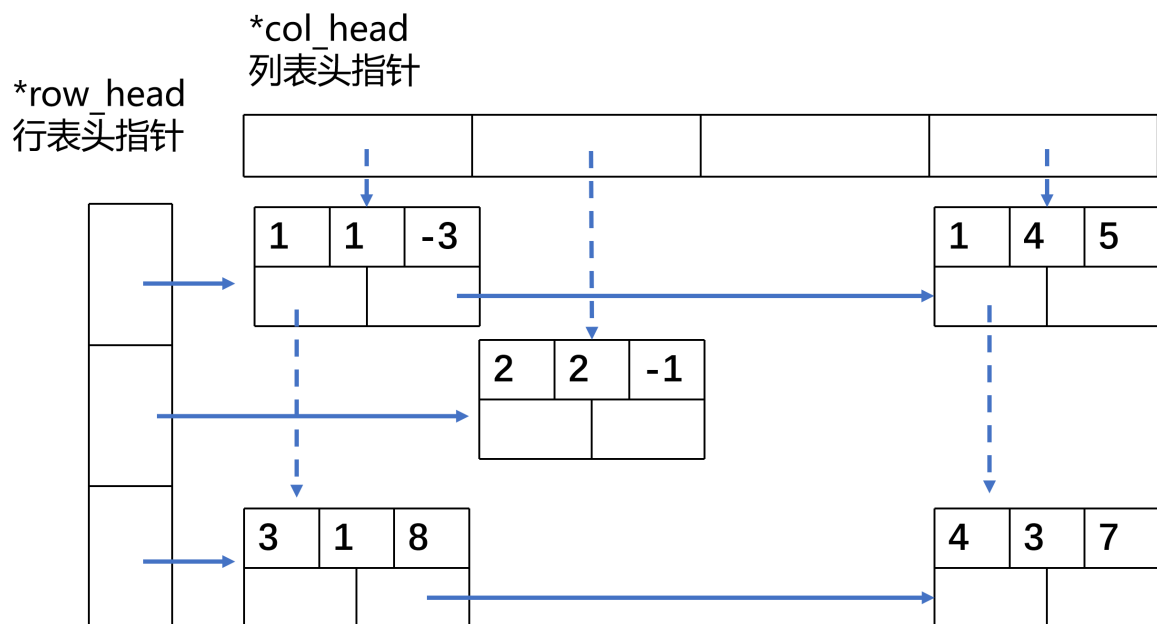
模块调用图：



三、详细设计：

算法实现图：

十字链表法的存储结构



核心算法的伪代码框架

```

1  //数据结构(十字链表存储的稀疏矩阵)
2  typedef struct OLNode
3  {
4      int row, col, x;
5      OLNode* right, * down;
6  }OLNode, * OLink;
7
8  struct CrossList
9  {
10     OLink* row_head, * col_head;
11     int m, n, len;
12 };
13
14 //核心算法
15 //在十字链表中插入三元组的函数模块:
16 void Insert(CrossList* M, OLink p) //在矩阵M中插入三元组p
17 {
18     OLink q;
19     //若p所在行对应的M行没有元素则直接插入
20     if (M->row_head[p->row] == NULL) M->row_head[p->row] = p;
21     //否则找到合适位置进行插入
22     else{
23         q = M->row_head[p->row];
24         while (q->right != NULL && q->right->col < p->col)
25             q = q->right;
26         p->right = q->right; q->right = p;
27     }
28     //同理对列进行处理, 若p所在列对应的M列没有元素则直接插入
29     if (M->col_head[p->col] == NULL) M->col_head[p->col] = p;
30     //否则找到合适位置进行插入
31     else{
32         q = M->col_head[p->col];
33         while (q->down != NULL && q->down->row < p->row)
34             q = q->down;

```

```

35     p->down = q->down; q->down = p;
36 }
37 }
38
39 //初始化矩阵十字链表模块:
40 void CreatCrossList(CrossList* M)
41 {
42     //初始化行列头指针节点
43     M->row_head = (OLink*)malloc((M->m + 1) * sizeof(OLink));
44     M->col_head = (OLink*)malloc((M->n + 1) * sizeof(OLink));
45     for (int i = 1; i <= M->m; i++)
46         M->row_head[i] = NULL;
47     for (int i = 1; i <= M->n; i++)
48         M->col_head[i] = NULL;
49     //处理输入
50     for (int i = 0; i < M->len; i++){
51         OLink p = (OLink)malloc(sizeof(OLNode));
52         cin >> p->row >> p->col >> p->x;
53         Insert(M, p);
54     }
55 }
56
57 //支持稀疏矩阵加法的函数模块:
58 void MatrixAdd(CrossList* A, CrossList* B, CrossList* C)
59 {
60     OLink p, q;
61     for (int i = 1; i <= A->m; i++)
62     {
63         p = A->row_head[i], q = B->row_head[i];
64         while (p != NULL && q != NULL)
65         {
66             //若p的列与q的列相等
67             if (p->col == q->col){
68                 int sum = p->x + q->x;
69                 //若和不为0则插入
70                 if (sum != 0){
71                     OLNNode* m = (OLNNode*)malloc(sizeof(OLNode));
72                     m->x = sum; m->row = i; m->col = p->col;
73                     Insert(C, m);
74                     C->len++;
75                 }
76                 p = p->right; q = q->right;
77             }
78             //若p的列小于q的列, C插入p
79             else if (p->col < q->col){
80                 OLNNode* m = (OLNNode*)malloc(sizeof(OLNode));
81                 m->x = p->x; m->row = i; m->col = p->col;
82                 Insert(C, m);
83                 C->len++;
84                 p = p->right;
85             }
86             //若p的列大于q的列, C插入q
87             else{
88                 OLNNode* m = (OLNNode*)malloc(sizeof(OLNode));
89                 m->x = q->x; m->row = i; m->col = q->col;
90                 Insert(C, m);
91                 C->len++;
92                 q = q->right;

```

```

93         }
94     }
95     //将A中剩余元素插入
96     while (p != NULL)
97     {
98         OLNNode* m = (OLNNode*)malloc(sizeof(OLNNode));
99         m->x = p->x; m->row = i; m->col = p->col;
100        Insert(C, m);
101        C->len++;
102        p = p->right;
103    }
104    //将B中剩余元素插入
105    while (q != NULL)
106    {
107        OLNNode* m = (OLNNode*)malloc(sizeof(OLNNode));
108        m->x = q->x; m->row = i; m->col = q->col;
109        Insert(C, m);
110        C->len++;
111        q = q->right;
112    }
113 }
114 }

```

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

本程序的运行环境为visual studio 2019。

第一行分别输入两个矩阵的行数、列数以及矩阵中非零元素个数。接下来分别输入第一个矩阵和第二个矩阵的三元组表示的数据，其中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。

程序会输出按行序为主序排列的两矩阵相加后的矩阵三元组

2. 测试结果与分析

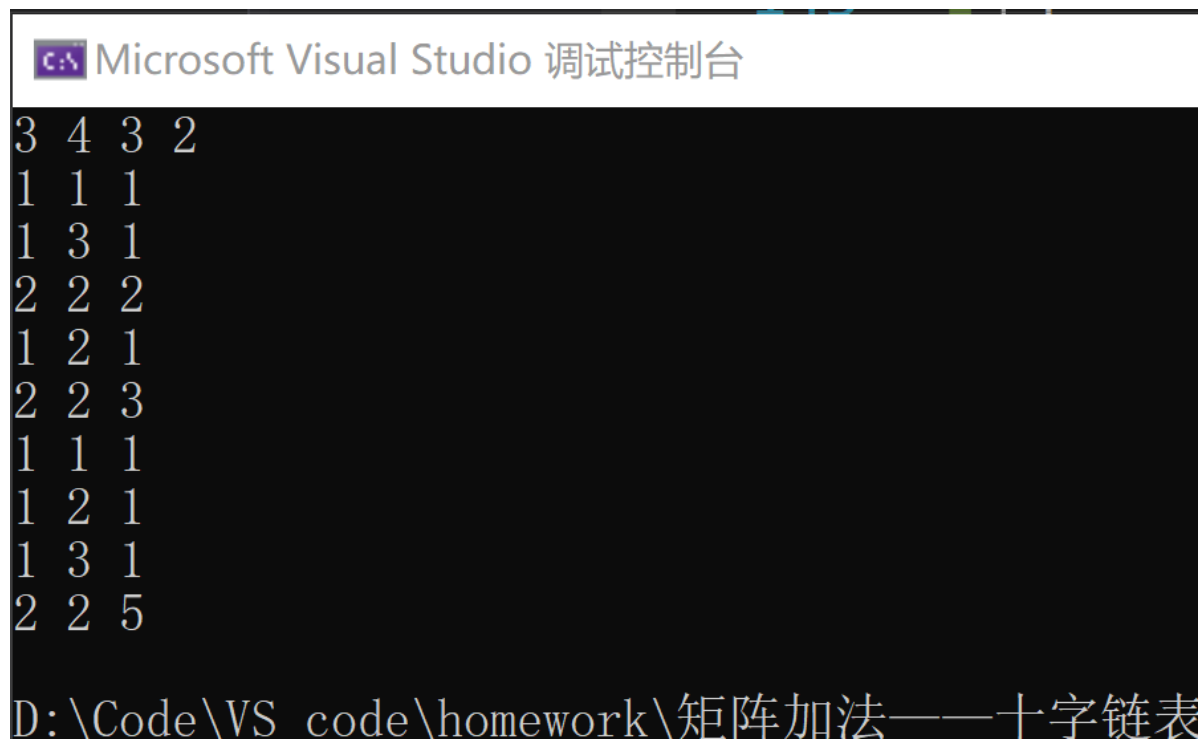
本程序较好的完成了实验需求，经分析，十字链表创建的时间复杂度为 $O(t \times s)$ ，矩阵加法的时间复杂度为 $O(t)$ ，其中 t 为稀疏矩阵中非零元素个数， $s = \max(m, n)$

3. 调试过程中遇到的问题及解决方法

在处理两个矩阵相加元素的插入时遇到些麻烦，于是我将插入函数单独提取了出来，这也既可以在初始化十字链表时使用也可以在矩阵的相加操作中使用，虽然会略微提高一些时间复杂度，但避免了重复劳动降低了出错概率

另外十字链表的存储结构较为复杂，初始化时很容易出错，一开始我没有理顺这一点，花费了较多时间在这上面调试，后来对照PPT和图解进行修改才得以解决

4.运行界面



```
Microsoft Visual Studio 调试控制台

3 4 3 2
1 1 1
1 3 1
2 2 2
1 2 1
2 2 3
1 1 1
1 2 1
1 3 1
2 2 5

D:\Code\VS code\homework\矩阵加法——十字链表
```

五、实验总结

- 本实验我在编程中花费26分钟
- 在调试中花费13分钟，主要在十字链表的初始化操作方面耽搁太久
- 十字链表的初始化需要先建好行列元素头指针数组，并先赋初值`NULL`
- 插入操作时需要分类讨论，如果该行/列没有元素直接插入即可，否则需要先找到合适位置再进行插入
- 需要时刻维护三元组中新加的`right`和`down`指针的关系

2.4 稀疏矩阵乘法

实验2.4：稀疏矩阵的乘法

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

VIDEO1

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。

Sample Input

```
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
```

Sample Output

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

© 2002-2012 IDRSoft

一、需求分析：

1. 输入：

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，同样以(0 0 0)结束

2. 输出：

输出相乘后的矩阵三元组。

3. 程序所能达到的功能：

输入两个稀疏矩阵，输出它们相乘的结果，并能检测出(0 0 0)以结束输入

二、概要设计：

核心思想：

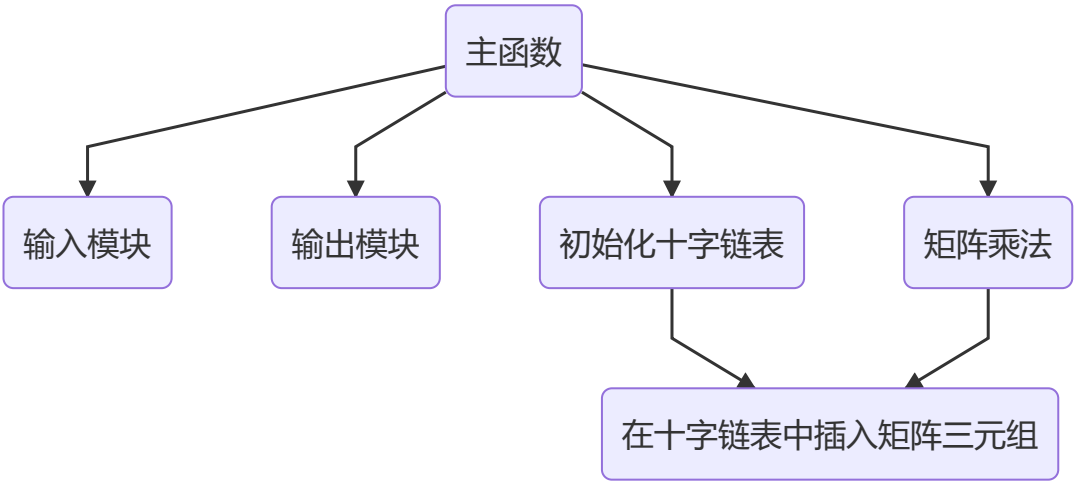
- 利用十字链表法存储稀疏矩阵
- 本题大致解题思路与上一题类似，对于A的每一行元素p，遍历B的每一列元素q
若p的列与q的行相等，则二者相乘加到sum中，待这一行或这一列遍历完毕，判断sum的值是否为0，若不为0，则插入到C中对应位置，否则什么也不做
- 同样，将元素存入C中时也要用十字链表法插入，具体流程为：
若待插入元素的行在C中没有元素，则直接插入即可
否则需要先根据元素的列在C中找到第一个列大于该列的元素位置，然后将待插入元素插入到此元素之前，若都比待插入元素的列小，则将其插入最后一列
- 由于本题需要根据输入(0 0 0)判断何时结束输入，因此需要将十字链表的初始化模块与输入模块分开编写，否则难以处理C的初始化

程序框架：

本程序包含六个模块：

1. 主程序模块；
2. 初始化矩阵十字链表模块；
3. 输入模块
4. 支持稀疏矩阵乘法的函数模块；
5. 在十字链表中插入三元组的函数模块；
6. 输出模块

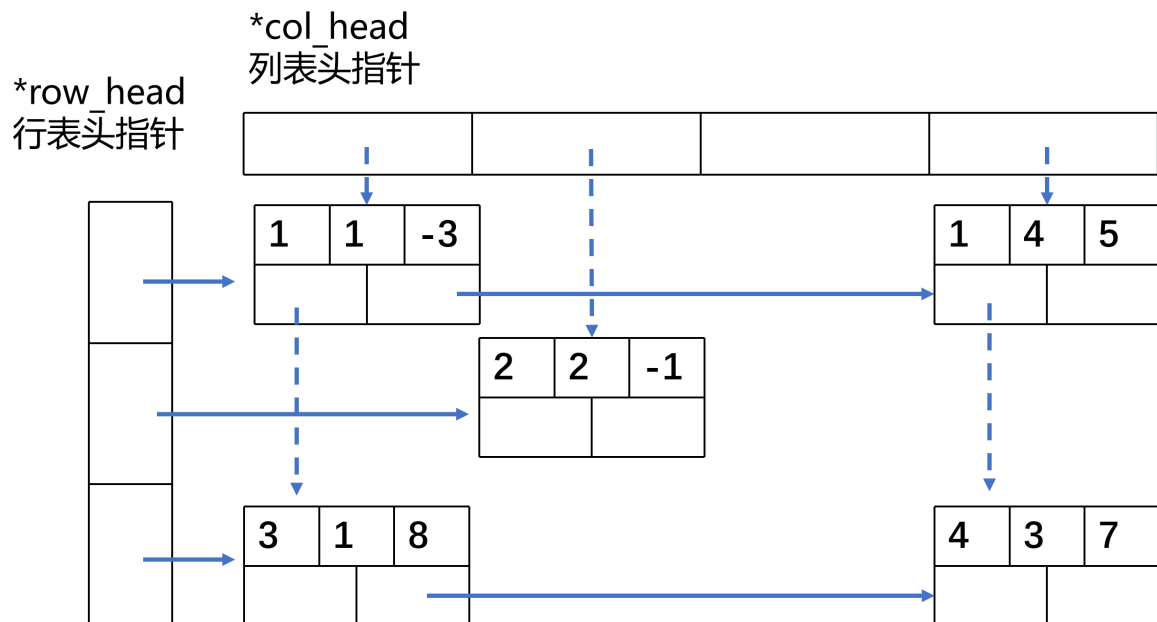
模块调用图：



三、详细设计：

算法实现图：

十字链表法的存储结构（同上一题）



核心算法的伪代码框架:

```

1  //数据结构(十字链表存储的稀疏矩阵)
2  typedef struct OLNode
3  {
4      int row, col, x;
5      OLNode* right, * down;
6  }OLNode, * OLink;
7
8  struct CrossList
9  {
10     OLink* row_head, * col_head;
11     int m, n, len;
12 };
13
14 //核心算法
15 //在十字链表中插入三元组的函数模块:
16 void Insert(CrossList* M, OLink p) //在矩阵M中插入三元组p
17 {
18     OLink q;
19     //若p所在行对应的M行没有元素则直接插入
20     if (M->row_head[p->row] == NULL) M->row_head[p->row] = p;
21     //否则找到合适位置进行插入
22     else{
23         q = M->row_head[p->row];
24         while (q->right != NULL && q->right->col < p->col)
25             q = q->right;
26         p->right = q->right; q->right = p;
27     }
28     //同理对列进行处理, 若p所在列对应的M列没有元素则直接插入
29     if (M->col_head[p->col] == NULL) M->col_head[p->col] = p;
30     //否则找到合适位置进行插入
31     else{
32         q = M->col_head[p->col];
33         while (q->down != NULL && q->down->row < p->row)
34             q = q->down;

```

```

35     p->down = q->down;q->down = p;
36 }
37 }
38
39 //初始化矩阵十字链表模块:
40 void CreatCrossList(CrossList* M)
41 {
42     //初始化行列头指针节点
43     M->row_head = (OLink*)malloc((M->m + 1) * sizeof(OLink));
44     M->col_head = (OLink*)malloc((M->n + 1) * sizeof(OLink));
45     for (int i = 1; i <= M->m; i++)
46         M->row_head[i] = NULL;
47     for (int i = 1; i <= M->n; i++)
48         M->col_head[i] = NULL;
49 }
50
51 //稀疏矩阵乘法:
52 void MatrixMul(CrossList* A, CrossList* B, CrossList* C)
53 {
54     OLink p, q;
55     for (int i = 1; i <= A->m; i++){
56         for (int j = 1; j <= B->n; j++){
57             //遍历A的第i行, B的第j列
58             p = A->row_head[i];q = B->col_head[j];
59             int sum = 0;
60             while (p != NULL && q != NULL){
61                 //若p的列等于q的行, 则将二者相乘加到sum中
62                 if (p->col == q->row){
63                     sum += p->x * q->x;
64                     p = p->right;q = q->down;
65                 }
66                 else if (p->col < q->row)
67                     p = p->right;
68                 else q = q->down;
69             }
70             //若sum不为0, 插入到C中对应位置
71             if (sum != 0){
72                 OLink r = (OLink)malloc(sizeof(OLNode));
73                 r->row = i;r->col = j;r->x = sum;
74                 Insert(C, r);
75                 C->len++;
76             }
77         }
78     }
79 }

```

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

本程序的运行环境为visual studio 2019。

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, 同样以(0 0 0)结束

程序会输出按行序为主序排列的两矩阵相乘后的矩阵三元组

2. 测试结果与分析

本程序较好的完成了实验需求，经分析，十字链表的创建和矩阵乘法的时间复杂度均为 $O(t \times s)$ ，其中 t 为稀疏矩阵中非零元素个数， $s = \max(m, n)$

3. 调试过程中遇到的问题及解决方法

本题的存储结构与上一题一样，再写完上一题后，我对矩阵的十字链表法掌握更加熟练，因此没有遇到什么大问题，只是在遍历时一开始将A中元素p放在了第一层循环里，导致运行时出错，后来发现问题后将他放在了第二层循环，这是因为在遍历B的列时，A的这一行虽然不变，但需要重头开始。

4. 运行界面



```
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
```

```
D:\Code\VS_code\homework\矩阵乘法\
以任意键退出此程序
```

五、实验总结

- 本实验我在编程中花费17分钟
- 在调试中花费5分钟，主要在矩阵乘法的元素遍历上出了点毛病
- 十字链表的初始化需要先建好行列元素头指针数组，并先赋初值`NULL`
- 插入操作时需要分类讨论，如果该行/列没有元素直接插入即可，否则需要先找到合适位置再进行插入
- 需要时刻维护三元组中新加的`right`和`down`指针的关系
- 需要注意矩阵乘法中元素的遍历的问题

