5.2 稀疏矩阵求和

稀疏矩阵因非零元多，一般采用只存储非零元的压缩存储方法。但不同的问题需采用不同的存储方法。例如在稀疏矩阵转置中稀疏矩阵采用三元组存储方法，就不适合用于稀疏矩阵求和中。本节通过疏矩阵的求和运算展示存储方法对算法的影响。学习中注意比对非稀疏矩阵求和方法与本节所采用的方法上的不同。

一、程序设计简介

本验证程序只有一个源程序文件 AddMatrix.cpp，实现两个稀疏矩阵的和（MC=MA+MB）。稀疏矩阵采用了带行指针向量的存储方式，矩阵元素类型为整型。稀疏矩阵的非零元存储在一个二维数组中，例如：

da[5][3]={{0,1,3},{1,1,2},{1,3,5},{3,0,9},{3,5,1}};

db[4][3]={{0,2,7},{1,1,6},{1,3,-5},{2,1,4}};

由函数 LMcreate()将上述存储的非零元生成带行逻辑向量存储的稀疏矩阵。为了增加直观性，矩阵显示给出了包括零元的完整矩阵显示，由函数 LMdisp()实现。

求 MC=MA+MB 时，和矩阵的元素来自 MA 或 MB，对于相同位置上的非零元，求和后为和矩阵中的元素。非相同位置上非零元，同行按列号升序形成和矩阵中的元素。实现中用函数 NodeCopy()实现结点拷贝。新结点链在同行非零元结点的表尾，用函数 AddNode()实现。函数的调用关系如图 1.5.3 示。

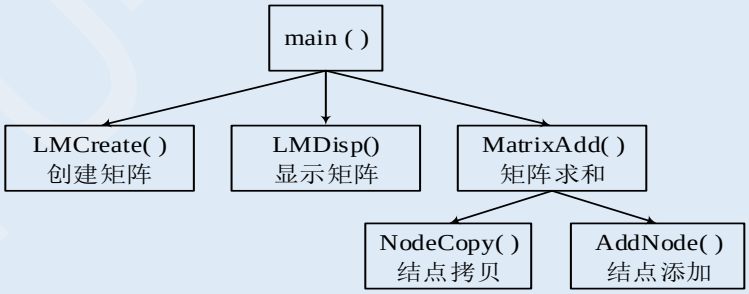

图 1.5.3 稀疏矩阵求和函数调用关系

二、运行说明

编译与链接成功后，屏幕显示运行结果如图 1.5.4 示。用户通过修改主函数中二维数组的值，可以求不同矩阵的和。

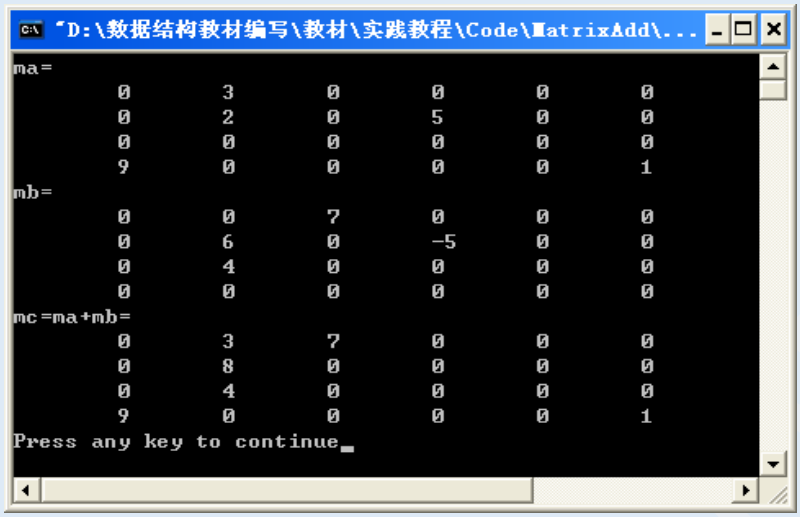

图 1.5.4　稀疏矩阵求和运行截图

三、思考题

1. 研读源程序，回答下列问题：

（1）和矩阵的行向量生成第一个结点和非第一个结点操作上有何区别？

（2）矩阵完整显示中，零元素的显示考虑了哪几种情况？

（3）如何修改程序，实现稀疏矩阵差运算？

（4）如果求稀疏矩阵的积，哪种存储结构比较合适？

2. 运行程序，回答下列问题

针对下列矩阵情况，求他们的和，并分析结果的正确性

（5）给出另外非零两个矩阵；

（6）一个零矩阵，一个非零矩阵；

（7）和为零的两个矩阵。