**实验四。2016.11.6**

**题目：稀疏的矩阵相加和相乘**

**班级：****2015211307 姓名：赵如茵 学号：2015211334** **分工： 下次实验报告**

**班级：2015211307 姓名：尹子长 学号：2015211921 分工： 此次实验代码**

**班级：2015211307 姓名：裴子祥 学号：2015211921 分工： 此次实验报告**

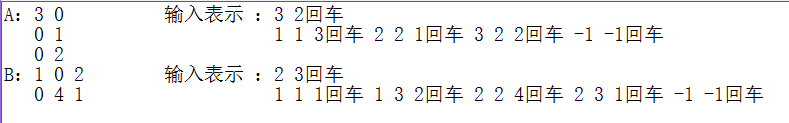
1. **需求分析**

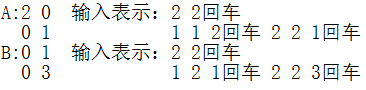
**判别回文字符串：**稀疏矩阵A和B均以三元组表作为存储结构，试写出矩阵相加和相乘的算法，另设三元组表C存放结果矩阵。

**输入：**矩阵A的三元组表，矩阵B的三元组表

**输出：**阵两矩阵的和或两矩阵的乘积

**测试数据：**





1. **概要设计**
2. 创建矩阵

Matrix Input(Matrix A)//输入矩阵，返回构造的矩阵

1. 矩阵相加函数

Matrix Add(Matrix A,Matrix B)//返回相加后的所得矩阵

矩阵相乘函数

Matrix Multiply(Matrix A,Matrix B)// 返回相乘后的所得矩阵

1. 输出矩阵

void Output(Matrix A)//输出矩阵

1. 模块框架：
2. input（）；依次输入两矩阵
3. add（）或Multiply（）
4. output（）//输出结果
5. 数据类型

typedef struct t //define the tri-tuple

{

int i = 0,j = 0;

double x = 0;

}tuple;

typedef struct m //define the matrix

{

int m = 0,n = 0,t = 0;

tuple data[99];

}Matrix;

1. **详细设计**
2. Matrix Add(Matrix A,Matrix B)

{

if(A.m == B.m && A.n == B.n) //judge if it can be added

{

Matrix C;

C.m = A.m;

C.n = B.n;

double x[99][99];

k = 1;//计数

for(int i = 1; i <= A.t ;i++) // add matrix A to a new array

{

x[A.data[i].i][A.data[i].j] = A.data[i].x;

}

for(int i = 1; i <= B.t ;i++) // add matrix A to the array

{

x[B.data[i].i][B.data[i].j] = x[B.data[i].i][B.data[i].j] + B.data[i].x;

}

for(int i = 1; i <= A.m ; i++) //transform matrix C to the sparse matrix

{

for(int j = 1;j <= B.n ; j++)

{

if(x[i][j])

{

C.data[k].i = i;

C.data[k].j = j;

C.data[k].x = x[i][j];

k++;

}

}

}

C.t = k;

return C;

}

else

{

cout << "ERROR" << endl;

Matrix C;

return C;

}

}

1. Matrix Multiply(Matrix A,Matrix B)

{

if(A.n == B.m)

{

Matrix C;

C.m = A.m;

C.n = B.n;

double x[99][99]; //new a array

int cnum[99] ; //记录每行非零个数

int cpos[99] ; //记录每行第一个非零元素位置。

cpos[1] = 1;

for(int i = 1; i <= B.t ; i++)

{

cnum[B.data[i].i]++; // compute the cnum array

}

int i;

for(i = 2; i <= B.m ; i++)

{

cpos[i] = cpos[i-1] + cnum[i-1]; //compute the cpos array

}

cpos[B.m+1] = B.t + 1;

for(int i = 1; i <= A.t ; i++) // for the i element in A

{

int j = cpos[A.data[i].j];

while(cpos[A.data[i].j+1] > j) // make the compution

{

x[A.data[i].i][B.data[j].j] = x[A.data[i].i][B.data[j].j]+ A.data[i].x \* B.data[j].x;

j++;

}

}

int k = 1;

for(int i = 1; i <= A.m ; i++)// transform the matrix C into sparse matrix

{

for(int j = 1;j <= B.n ; j++)

{

if(x[i][j])

{

C.data[k].i = i;

C.data[k].j = j;

C.data[k].x = x[i][j];

k++;

}

}

}

C.t = k;

return C;

}

else

{

Matrix C;

cout << "Can't multiplication for the column of A dosen't equal to the row of B." << endl;

cout << endl;

return C;

}

}

1. 函数关系调用图

main（）（一个while循环）

Input（） add（） mulyiply() Output()

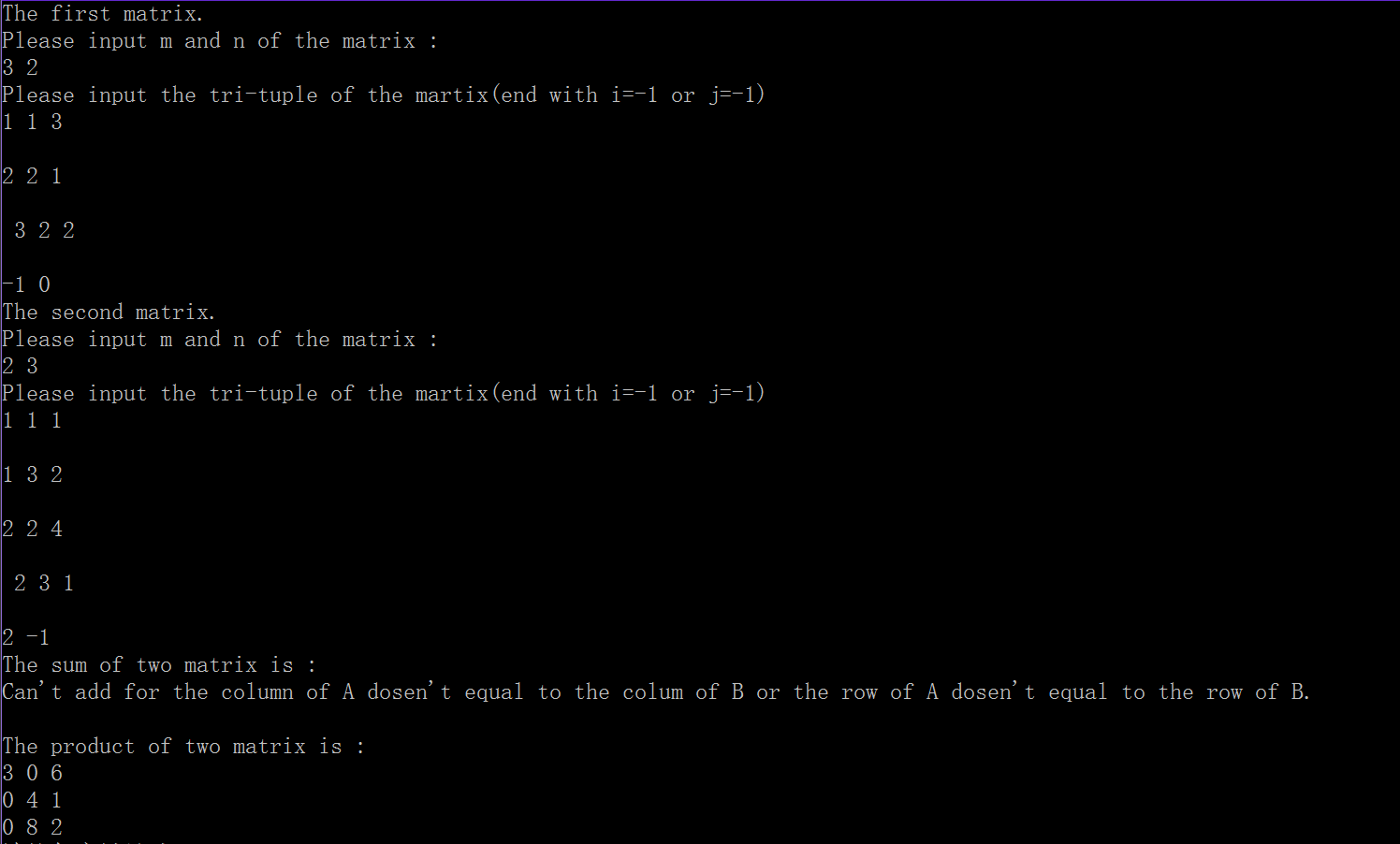
1. **调试分析报告**
2. 算法分析，

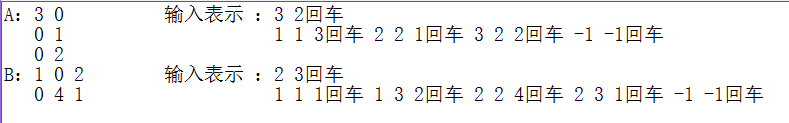
add()中将A中非零元素加入对应行列的二维数组中，再将B中非零元素加入对应行列的二维数组中，再将二维数组x[][]赋值给矩阵C，从而获得矩阵C算法复杂度O（m\*n）

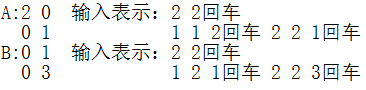
multiply()中。int cnum[99] ;//记录每行非零个数 int cpos[99] ;//记录每行第一个非零元素位置。O(m\*n).

1. 熟悉矩阵结构，使用三元组表储表示矩阵。熟悉用矩阵相乘的更优算法
2. 遇到经常数据类型不匹配的问题，注意数据类型定义
3. 体会：慢慢编总会对的
4. 输入操作选项时要用fflush(0)处理缓存区中enter字符
5. **用户使用说明**
6. 输入第一个矩阵A的行数与列数
7. 用三元组表表示矩阵（依次元素行数i，元素列数j，输入元素值x）直到（i=-1||j=-1时输入结束，其它不合法输入为错误，提示重新输入）
8. 同1,2步输入矩阵B
9. 先做加法，若矩阵形状不一样，则不能进行加法运算。再做乘法，若A的列数不等于B的行数，则不能进行乘法运算。
10. 若成功运算输出矩阵C
11. **测试结果**

**矩阵相乘：**







**矩阵相加与相乘：**

