**数据结构实验报告**

**实验六 稀疏矩阵的加法和乘法问题**

学 生 姓 名 ：杨仕振

学 号 ：2015301500349

二○一六年十一月

**一、简介**

【基本要求】

两个稀疏矩阵的三元组存储结构生成；

实现稀疏矩阵的转置；

实现两个稀疏矩阵的加运算；

实现两个稀疏矩阵的乘法运算。

**二、算法说明**

使用的数据结构：

Tupnode：记录矩阵中非零值的坐标和数值；

TSMatrix：矩阵的三元组表示

使用的算法：

CreatMat：根据稀疏矩阵的值进行判断，非零则生成对应的三元组项（课本P139）

TranTat：对换三元组的横纵坐标。并进行重新排序。（课本P140，添加了重新排序的修正）

sort：三元组转置算法中根据行序进行重新排序的函数

MatAdd：矩阵三元组相加算法（课本P140例6.3，有修正）

MatMul：矩阵三元组相乘算法

DispMat：输出矩阵三元组（课本P140）

具体的分析及解释见源代码注释

**三、测试结果**

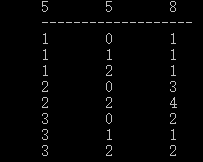
为了简化问题，这里用了一个特例进行检验，

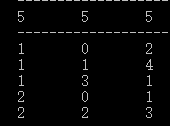
A={ {},{1,1,1,0,0},{3,0,4,0,0},{2,1,2,0,0},{} }

B={ {},{2,4,0,1,0},{1,0,3,0,0},{},{} }

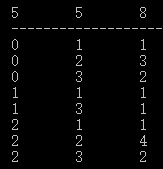
进行如下操作：

对A、B进行输出

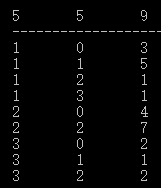




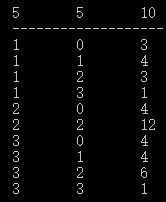
对A进行转置



将AB相加



将AB相乘



测试均通过。

**四、分析与探讨**

在设计的时候为了简化计算，突出重点，这儿只用了5\*5的矩阵作为例子，但是边长是用宏定义规定的，所以可以修改。而对于此程序，进一步的改善是可以添加手动输入或者文本导入的矩阵内容。

空间复杂度：所有变量的数目固定。空间复杂度为O(1);

时间复杂度：各个函数中矩阵乘法函数的时间复杂度最大，为O(n3)

**附录：源代码**

#include <stdio.h>

#include <stdlib.h>

#define M 5

#define N 5

#define Maxsize 25

typedef struct

{

int r;

int c;

int d;

}TupNode;//节点

typedef struct

{

int rows;

int cols;

int nums;

TupNode data[Maxsize];

}TSMatrix;//矩阵的三元组

void CreatMat(TSMatrix &t, int A[M][N])//创建二维稀疏矩阵的三元组表示

{

int i, j;

t.rows = M; t.cols = N; t.nums = 0;//初始化三元组

for (i = 0; i<M; i++)

{

for (j = 0; j<N; j++)

{

if (A[i][j] != 0)

{

t.data[t.nums].r = i;

t.data[t.nums].c = j;

t.data[t.nums].d = A[i][j]; t.nums++;//根据节点值进行赋值

}

}

}

}

void sort(TSMatrix &tb)

{

int tempr;

int tempc;

int tempd;

int i = 0, j = 0;

for (; i<tb.nums; i++)//以行序作为主排序依据进行排序

{

for (j = i + 1; j < tb.nums; j++)//扫描未排序的部分

{

if (tb.data[i].r > tb.data[j].r || tb.data[i].r == tb.data[j].r&&tb.data[i].c > tb.data[j].c)//若扫描的后者行序比前者大，或行序相同但列序比前者大，则交换两个节点

{

tempr = tb.data[i].r;

tempc = tb.data[i].c;

tempd = tb.data[i].d;

tb.data[i].r = tb.data[j].r;

tb.data[i].c = tb.data[j].c;

tb.data[i].d = tb.data[j].d;

tb.data[j].r = tempr;

tb.data[j].c = tempc;

tb.data[j].d = tempd;

}

}

}

}

void TranTat(TSMatrix t, TSMatrix &tb)//矩阵转置

{

int q, v = 0;

tb.rows = t.cols; tb.cols = t.rows; tb.nums = t.nums;//初始化三元组

if (t.nums != 0)

{

for (q = 0; q < t.cols; q++)//依列序转置

for (v = 0; v < t.nums; v++)

if (t.data[v].c == q)

{

tb.data[v].r = t.data[v].c;

tb.data[v].c = t.data[v].r;

tb.data[v].d = t.data[v].d;

}

}

sort(tb);//对转置完成的三元组进行重新排序

}

bool MatAdd(TSMatrix a, TSMatrix b, TSMatrix &c)//矩阵相加

{

int i = 0, j = 0, k = 0;

int v;

if (a.rows != b.rows || a.cols != b.cols)

return false;

c.rows = a.rows; c.cols = a.cols;

while (i<a.nums&&j<b.nums)//若其中一三元组扫描完成，则退出循环

{

if (a.data[i].r == b.data[j].r)//若两三元组节点行序相同

{

if (a.data[i].c<b.data[j].c)//若A三元组节点的列序小于B三元组的列序

{

c.data[k].r = a.data[i].r;

c.data[k].c = a.data[i].c;

c.data[k].d = a.data[i].d;

k++; i++;

}

else if (a.data[i].c>b.data[j].c)//若A三元组节点的列序大于B三元组的列序

{

c.data[k].r = b.data[i].r;

c.data[k].c = b.data[i].c;

c.data[k].d = b.data[i].d;

k++; j++;

}

else//若A三元组节点的列序等于B三元组的列序

{

v = a.data[i].d + b.data[j].d;

if (v != 0)

{

c.data[k].r = a.data[i].r;

c.data[k].c = a.data[i].c;

c.data[k].d = v;

k++;

}

i++; j++;

}

}

else if (a.data[i].r<b.data[j].r)//若A三元组节点的行序小于B三元组的行序

{

c.data[k].r = a.data[i].r;

c.data[k].c = a.data[i].c;

c.data[k].d = a.data[i].d;

k++; i++;

}

else//若A三元组节点的行序大于B三元组的行序

{

c.data[k].r = b.data[j].r;

c.data[k].c = b.data[j].c;

c.data[k].d = b.data[j].d;

k++; j++;

}

c.nums = k;//更新c三元组的节点个数

}

for (; i<a.nums; i++)//若A三元组尚未扫描完，则继续扫描

{

c.data[k].r = a.data[i].r;

c.data[k].c = a.data[i].c;

c.data[k].d = a.data[i].d;

k++;

c.nums = k;

}

for (; j<b.nums; j++)//若B三元组尚未扫描完，则继续扫描

{

c.data[k].r = b.data[j].r;

c.data[k].c = b.data[j].c;

c.data[k].d = b.data[j].d;

k++;

c.nums = k;

}

return true;

}

bool MatMul(TSMatrix a, TSMatrix b, TSMatrix &c)//矩阵相乘

{

int v = 0;

int m = 0, n = 0;

int l = 0;

int record = 0;

TSMatrix \*f;

f = (TSMatrix \*)malloc(sizeof(TSMatrix));

if (a.cols != b.rows)//如果前者列序不等于后者行序，则退出

return false;

c.rows = a.rows; c.cols = b.cols;

c.nums = 0;//三元组初始化

TranTat(b, \*f);//为了方便运算，此处将B矩阵进行了转置

for (int i = 0; i<c.rows; i++)

{

record = m;//记录A矩阵当前行的首个元素在三元组中的序号

for (int j = 0; j<c.cols; j++)//遍历结果矩阵的每一个元素

{

for (int k = 0; k<a.cols; k++)//求矩阵中此对应元素的值

{

l = k;

if (k == a.data[m].c&&l == f->data[n].c&&i == a.data[m].r&&j == f->data[n].r)//只有矩阵A，B的两个节点值都不为零的情况才可以加和

{

v = v + a.data[m].d\*f->data[n].d;

m++; n++;

}

if ((i == a.data[m].r) && (k == a.data[m].c))//若其中一个为零，则跳过该节点

{

m++;

}

if ((l == f->data[n].c) && (j == f->data[n].r))

{

n++;

}

}

if (v != 0)//若值不为零，创建对应三元组中的项

{

c.data[c.nums].d = v;

c.data[c.nums].r = i;

c.data[c.nums].c = j;

c.nums++;

}

v = 0;

if(j!=c.cols-1)//判断是否要将正在扫描的A矩阵对应的三元组序号复位为当前行的首个非零元素

m = record;

}

n = 0;//从头重新扫描B矩阵

}

return true;

}

void DispMat(TSMatrix t)//输出三元组

{

int i;

if (t.nums <= 0)

return;

printf("\t%d\t%d\t%d\n", t.rows, t.cols, t.nums);

printf("\t-------------------\n");

for (i = 0; i<t.nums; i++)//打印所有节点的行序，列序和值

printf("\t%d\t%d\t%d\n", t.data[i].r, t.data[i].c, t.data[i].d);

printf("\t-------------------\n");

}

int main()

{

TSMatrix a,b;

TSMatrix \*c,\*d,\*e;

c = (TSMatrix \*)malloc(sizeof(TSMatrix));

d = (TSMatrix \*)malloc(sizeof(TSMatrix));

e = (TSMatrix \*)malloc(sizeof(TSMatrix));

int A[M][N] = { {},{1,1,1,0,0},{3,0,4,0,0},{2,1,2,0,0},{} };

int B[M][N] = { {},{2,4,0,1,0},{1,0,3,0,0},{},{} };

CreatMat(a,A);

CreatMat(b,B);

DispMat(a);

DispMat(b);

TranTat(a,\*c);

DispMat(\*c);

MatAdd(a,b,\*d);

DispMat(\*d);

MatMul(a,b,\*e);

DispMat(\*e);

return 0;

}