《数据结构》实验报告T003

班 级: DL062138 姓 名: 张苏宇 学 号: 2021302853

E-mail:905159071@qq.com 日 期: 2022/4/21

◎实验题目: T003稀疏矩阵转置

◎实验目的：实现三元组表下的稀疏矩阵转置

◎实验内容：设计出能实现三元组表下的稀疏矩阵转置的算法

一 需求分析

输入: 第一行输入两个正整数n，m，分别表示矩阵的行数和列数。然后输入矩阵三元组，最后通过 0 0 0 表示输入结束。

输出: 按三元组的形式输出转置后的矩阵。

样例输入: 4 4

1 1 1

2 1 2

3 2 3

0 0 0

样例输出:1 1 1

1 2 2

2 3 3

关键问题分析：

本题问题的关键在于要确保转置后，行列的优先，也就是行小的在前，对于行相同的，列小的在前，这样的顺序。此外，我们也需要保证程序的时间复杂度较优。

二 概要设计

使用结构体来存储矩阵数据，包括行数、列数、非0元素个数以及元素内容。

由于三元组表是从左往右、从上到下的顺序存储矩阵元素，因此实现转置时行变成列，列变成行，只要将矩阵三元组表的数据中行数和列数转换并按照原来的顺序排序就可以实现转置的功能。

所以在主函数体内输入元素的数据时，记录元素个数，最后通过先对列数排序，筛选同列数中的矩阵元素，只需要按照原来的顺序输入就可以了。

三 详细设计

#include <stdio.h>

#define MAX 1000

//定义三元组结构体

typedef struct {

int r, c;

//所在行数和列数

int d;

//所表示的值

} Triple;

//存储矩阵的三元组顺序表

typedef struct {

Triple data[MAX];

int m, n, t;

//存储这个矩阵当中信息，m行，n列，t个非0元素

} TsMatrix;

//稀疏矩阵的转置函数

void Transpose(TsMatrix A, TsMatrix \*B, int i) {

int p, q, r, x = i;

//将矩阵 a 存储的行数、列数赋值给转置矩阵 b

B->m = A.n;

B->n = A.m;

//从第一行开始依次遍历

if (i) {

q = 0;

for (r = 0; r < A.n; r++)

for (p = 0; p < x; p++) {

if (A.data[p].c == r) {

//交换a,b的行数和列数，并把a的值赋给b

B->data[q].r = A.data[p].c;

B->data[q].c = A.data[p].r;

B->data[q].d = A.data[p].d;

q++;

}

}

}

}

int main() {

int m, n, i = 0, j = 0;

scanf("%d %d", &m, &n);

TsMatrix A, B;

A.m = m;

A.n = n;

//获取初始矩阵a

while (1) {

int r, c, d;

scanf("%d %d %d", &r, &c, &d);

if (r == 0 && c == 0 && d == 0)

break;

A.data[i].r = r, A.data[i].c = c, A.data[i].d = d;

i++;

}

Transpose(A, &B, i);

//逐个输出转置后的矩阵元素

for (j = 0; j < i; j++) {

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

}

return 0;

}

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；

输入时 第一行输入两个正整数n，m，分别表示矩阵的行数和列数。然后输入矩阵三元组，最后通过 0 0 0 表示输入结束。

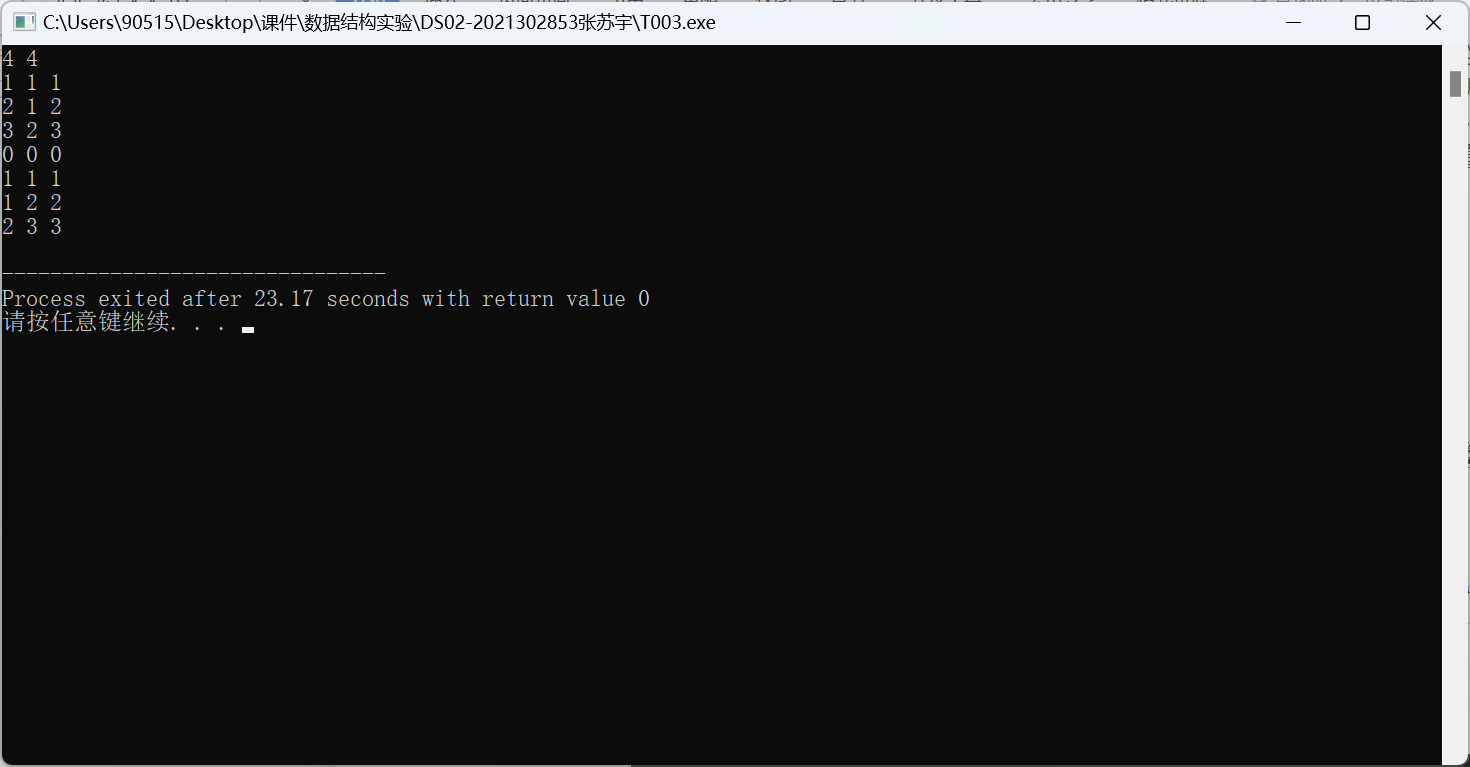
1. 测试结果与分析；

分析可以发现，时间复杂度为 O(n+T), n为矩阵列数，t为非零元素个数。

1. 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

没遇到什么问题。

1. 运行界面



五、实验总结

本题算是矩阵的一个初步算法转置操作，难度不大，算法上时间复杂度为O（n）；

并且我在本次实验中学会了优化转置算法的方法。

《数据结构》实验报告T004

班 级: DL062138 姓 名: 张苏宇 学 号: 2021302853

E-mail:905159071@qq.com 日 期: 2022/4/21

◎实验题目: T004稀疏矩阵加法.,实现C=A+B

◎实验目的：实现三元组表下的稀疏矩阵加法

◎实验内容：设计出能实现三元组表下的稀疏矩阵相加的算法

一 需求分析

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。

输出: 按三元组的形式输出相加后的矩阵。

样例输入: 3 4 3 2

1 1 1

1 3 1

2 2 2

1 2 1

2 2 3

样例输出:1 1 1

1 2 1

1 3 1

2 2 5

关键问题分析：

本题问题的关键在于要确保转置后，行列的优先，也就是行小的在前，对于行相同的，列小的在前，这样的顺序。而且也需要保证复杂度比较优秀。

二 概要设计

选用三元组表来存储矩阵信息，用一个结构体来表示矩阵。因为是矩阵相加，所以将两个矩阵相加后的数据存储进第三个矩阵里面并输出。使用两个指针分别指向A,B两个矩阵的三元组，当两者相加的时候，判断他们的行列的关系，分顺序将他们加到C矩阵中，不断遍历至将所有元素输入到C中然后输出就可以了。

三 详细设计

#include <stdio.h>

#define MAX 1000

//定义三元组结构体

typedef struct {

int r, c;

//所在行数和列数

int d;

//所表示的值

} Triple;

//存储矩阵的三元组顺序表

typedef struct {

Triple data[MAX];

int m, n, t;

//存储这个矩阵当中信息，m行，n列，t个非0元素

} TsMatrix;

//稀疏矩阵的加法，返回值为新矩阵

TsMatrix PlusMatrix(TsMatrix a, TsMatrix b) {

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

TsMatrix c;

c.t = 0;

c.m = a.m;

c.n = a.n;

//初始化返回的新矩阵

while (i <= a.t && j <= b.t && (!(i == a.t && j == b.t))) {

//判断条件

//按照从上到下从左到右的顺序依次遍历，先比较同行，再比较同列

if (a.data[i].r < b.data[j].r || (a.data[i].r == b.data[j].r && a.data[i].c < b.data[j].c)) {

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

c.t++;

} else if (a.data[i].r > b.data[j].r || (a.data[i].r == b.data[j].r && a.data[i].c > b.data[j].c)) {

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

c.t++;

}

//考虑相加和为0的情况，如果不为零则保留

else {

if (a.data[i].d + b.data[j].d != 0) {

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

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

c.t++;

}

}

}

return c;

}

int main() {

int m, n, t1, t2;

scanf("%d%d%d%d", &m, &n, &t1, &t2);

TsMatrix a, b, c;

a.m = b.m = m;

a.n = b.n = n;

a.t = t1, b.t = t2;

for (int i = 0; i < t1; i++) {

scanf("%d%d%d", &a.data[i].r, &a.data[i].c, &a.data[i].d);

}

for (int i = 0; i < t2; i++) {

scanf("%d%d%d", &b.data[i].r, &b.data[i].c, &b.data[i].d);

}

//获取要相加的两个矩阵a,b

c = PlusMatrix(a, b);

//逐个输出相加后的矩阵元素

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

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

}

return 0;

}

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；

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。

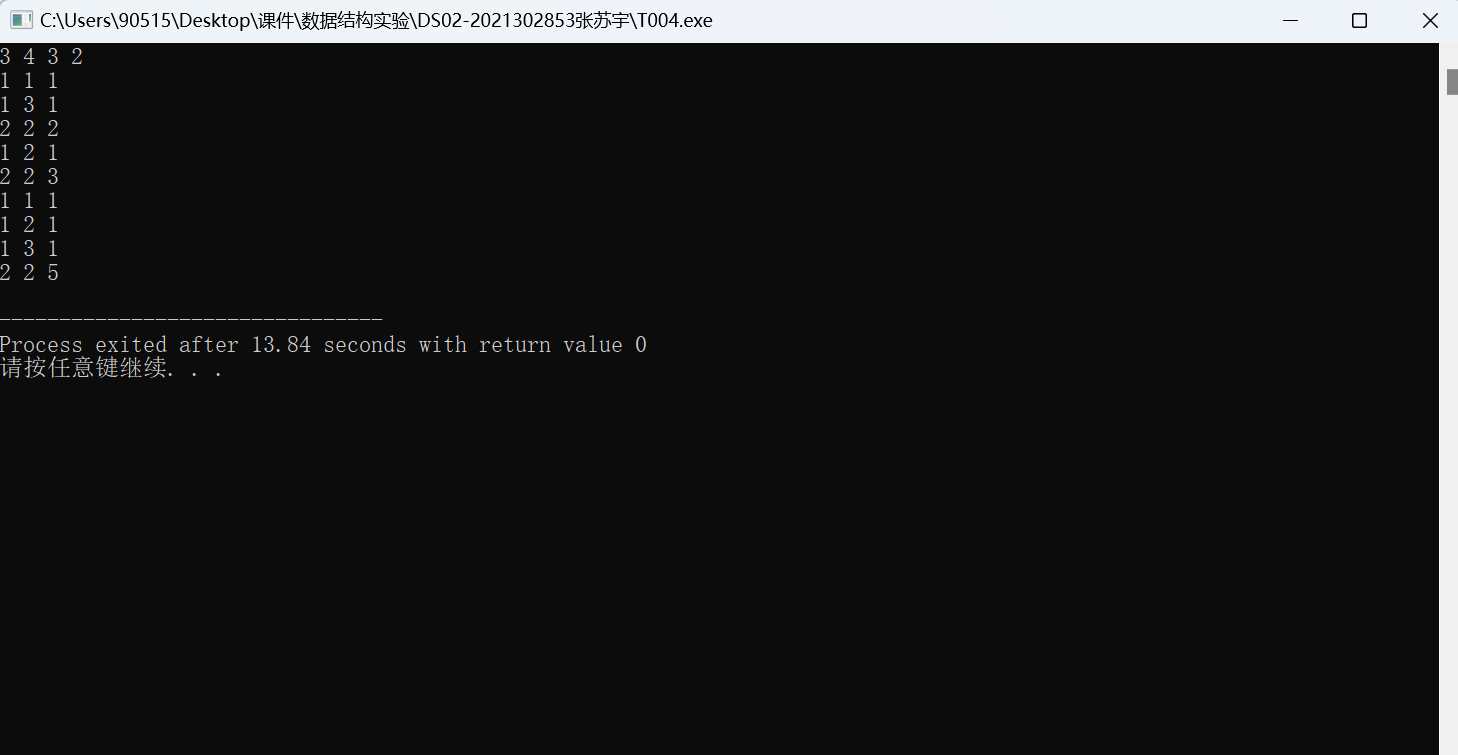
1. 测试结果与分析；

分析可以发现，时间复杂度为 O(t1+t2), t1为矩阵A非零元个数，t2为矩阵B非零元素个数。

1. 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

调试过程中遇到的问题就是一直PE,分析后发现是没有考虑当行列相等时他们的值加和为0不需要加入到矩阵C中。改正之后AC。

1. 运行界面



五、实验总结

通过本次实验，我学会了三元组表示矩阵的方法，积累了三元组表示矩阵的经验，并学会一些多线程遍历的算法。除此之外，在本次实验中，无数次的debug调试程序，其中收获的emmmm也是重大一环。

总而言之，我觉得应该还是要去多借鉴别人的算法或者是源码，自己想出的算法不一定有效而且很耗时，先去多参考类似代码，再去实际操作会更高效。

《数据结构》实验报告T005

班 级: DL062138 姓 名: 张苏宇 学 号: 2021302853

E-mail:905159071@qq.com 日 期: 2022/4/21

◎实验题目: T005稀疏矩阵加法，用十字链表实现C=A+B

◎实验目的：实现十字链表下的稀疏矩阵加法实现C=A+B

◎实验内容：设计出能实现十字链表下的稀疏矩阵加法的算法

1. 需求分析

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。

输出: 按三元组的形式输出相加后的矩阵。

样例输入: 3 4 3 2

1 1 1

1 3 1

2 2 2

1 2 1

2 2 3

样例输出:1 1 1

1 2 1

1 3 1

2 2 5

关键问题分析：

本题问题的关键在于如何使用十字链表存储稀疏矩阵以及如何用让两个十字链表表示的矩阵相加后依然保持能够有序输出。

二 概要设计

 建立两个十字链表存储矩阵信息，并通过加法函数将第二个矩阵和第一个矩阵相加的结果放入第一个矩阵当中。在加法函数当中，主要是不断析取第二个矩阵的节点，和创建链表一样，不断判断条件，添加进入第一个矩阵，若是行数列数相等则相加，若为0则删去此节点。

三 详细设计

#include <stdio.h>

//定义三元组结构体

typedef struct node {

int r, c, d;

//分别指向该结构体down和right的指针

struct node \*right;

struct node \*down;

} Node, \*NodePtr;

//定义十字链表

typedef struct {

int m, n, t;

//分别指向下和右的指针

NodePtr \*rp;

NodePtr \*cp;

} Matrix;

void CreateMatrix(Matrix \*p);//创造十字链表

void InitMatrix(Matrix \*p, int m, int n); //初始化稀疏矩阵

void OutputMatrix(Matrix p);//输出稀疏矩阵

void PlusMatrix(Matrix \*a, Matrix \*b); //两个十字链表相加

int rm, rn;//定义全局变量，函数中需使用

//通过主函数调用初始化函数，创建函数，相加函数和输出函数。

int main() {

Matrix a, b;

scanf("%d %d %d %d", &rm, &rn, &a.t, &b.t);

InitMatrix(&a, rm, rn);

InitMatrix(&b, rm, rn);

CreateMatrix(&a);

CreateMatrix(&b);

PlusMatrix(&a, &b);

OutputMatrix(a);

return 0;

}

//初始化稀疏矩阵

//增加指针数组，每一个元素都是指针，rp cp则是二级指针

void InitMatrix(Matrix \*p, int m, int n) {

p->m = m, p->n = n;

//申请内存

p->rp = (NodePtr \*)malloc((m + 1) \* sizeof(NodePtr));

p->cp = (NodePtr \*)malloc((n + 1) \* sizeof(NodePtr));

//初始化空链表

for (int i = 1; i < m + 1; i++)

p->rp[i] = NULL;

for (int i = 1; i < n + 1; i++)

p->cp[i] = NULL;

}

//创造十字链表

//通过判断节点行数列数和当前矩阵的状态之间的关系条件不断添加进入矩阵对应节点，分别从行和列两个顺序遍历插入。

void CreateMatrix(Matrix \*p) {

NodePtr k;

//逐一处理各三元组

for (int i = 0; i < p->t; i++) {

NodePtr q = (NodePtr)malloc(sizeof(Node));

scanf("%d%d%d", &q->r, &q->c, &q->d);

q->right = NULL, q->down = NULL;

// 行方向建立链表

if (p->rp[q->r] == NULL || p->rp[q->r]->c > q->c) {

q->right = p->rp[q->r];

p->rp[q->r] = q;

} else {

for (k = p->rp[q->r]; k->right != NULL && k->right->c < q->c; k = k->right) ;

q->right = k->right;

k->right = q;

}

// 列方向建立链表

if (p->cp[q->c] == NULL || p->cp[q->c]->r > q->r) {

q->down = p->cp[q->c];

p->cp[q->c] = q;

} else {

for (k = p->cp[q->c]; k->down != NULL && k->down->r < q->r; k = k->down) ;

q->down = k->down;

k->down = q;

}

}

}

//输出稀疏矩阵

void OutputMatrix(Matrix p) {

NodePtr k;

//逐一输出

for (int i = 1; i <= p.m; i++) {

k = p.rp[i];

while (k != NULL) {

printf("%d %d %d\n", k->r, k->c, k->d);

k = k->right;

}

}

}

//两个十字链表相加

void PlusMatrix(Matrix \*a, Matrix \*b) {

NodePtr k, j;

//按照从上到下从左到右的顺序依次相加

//不断析取第二个矩阵的节点，和创建链表一样，不断判断条件，添加进入第一个矩阵

for (int i = 1; i <= rm; i++) {

k = b->rp[i];

while (k != NULL) {

j = (NodePtr)malloc(sizeof(Node));

j->c = k->c;

j->r = k->r;

j->d = k->d;

j->down = NULL;

j->right = NULL;

//如果right为空

if (a->rp[j->r] == NULL) {

a->rp[j->r] = j;

a->t++;

} else {

NodePtr h = a->rp[j->r];

NodePtr pre = h;

while (h != NULL && h->c < j->c) {

pre = h;

h = h->right;

}

//若是行数列数相等则相加，若为0则删去此节点。

if (h != NULL && h->c == j->c) {

h->d = h->d + j->d;

if (h->d == 0) {

if (pre == h) {

a->rp[j->r] = h->right;

} else {

pre->right = h->right;

}

a->t--;

}

}

//将剩余的直接保存

else {

if (pre == h) {

j->right = a->rp[j->r];

a->rp[j->r] = j;

} else {

j->right = h;

pre->right = j;

}

a->t++;

}

}

k = k->right;

}

}

}

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；

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。

1. 测试结果与分析；

测试时输入数据为 3 4 1 1

1 2 1

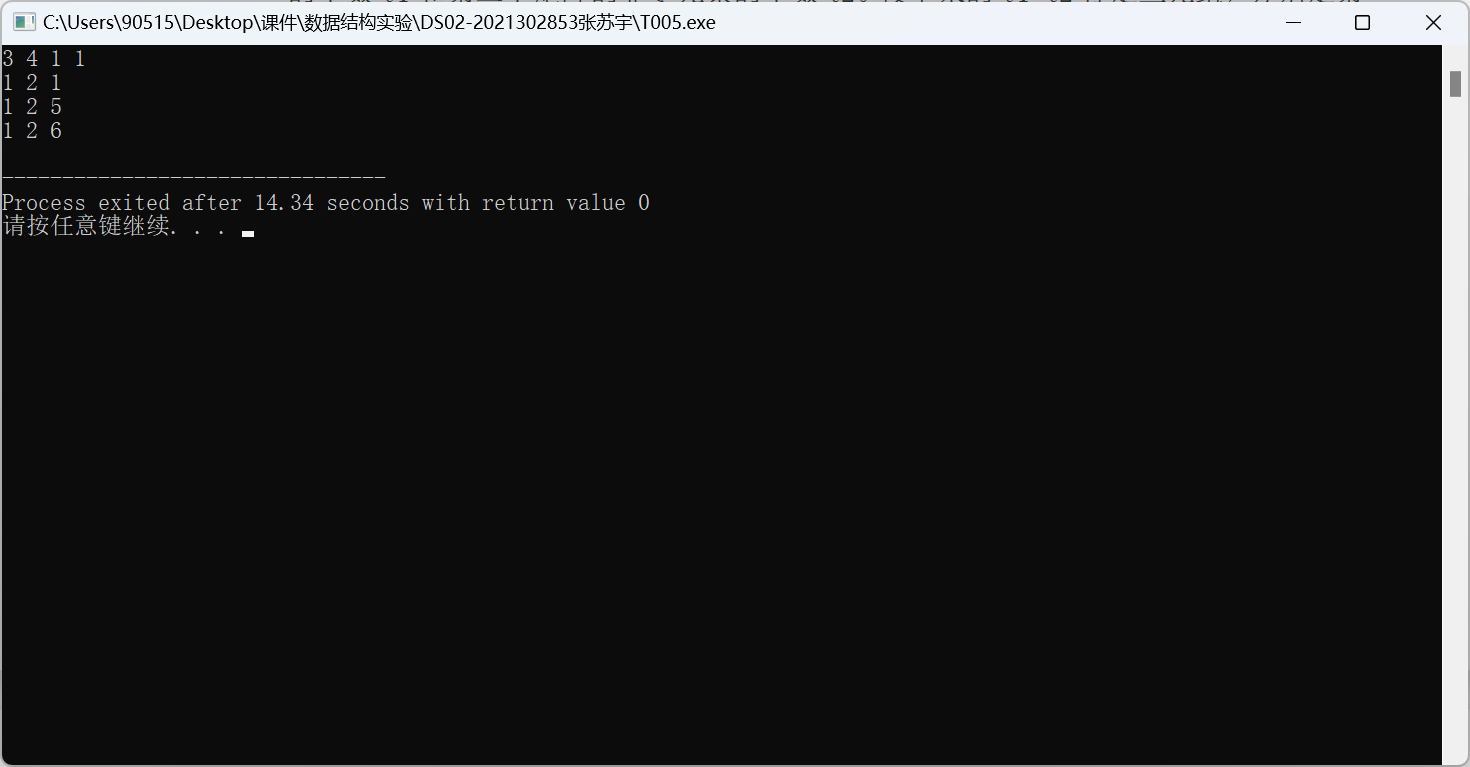
1 2 5

应输出 1 2 6 ，结果输出 1 2 6 ，结果正确。

1. 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

在debug过程中,删除节点一直有问题，就是当数据中存在和为0时，他会产生错误的输出（大概也许可能是乱码）。在无限次debug后，我发现是因为混淆了指针的内容中存的地址和指针本身地址这个问题。如果去改变它自身，那么它内容中的存的地址将无法访问，导致输出错误的结果。

1. 运行界面



五、实验总结

我能更加熟悉运用十字链表去操作矩阵，以及更好地debug程序。

《数据结构》实验报告T006

班 级: DL062138 姓 名: 张苏宇 学 号: 2021302853

E-mail:905159071@qq.com 日 期: 2022/4/21

◎实验题目: T006稀疏矩阵的乘法

◎实验目的：实现三元组表下的稀疏矩阵的乘法

◎实验内容：设计出能实现三元组表下的稀疏矩阵的乘法

1. 需求分析

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，以0 0 0结束。

输出: 输出相乘后的矩阵三元组。

样例输入: 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

样例输出:1 3 -2

2 1 32

2 2 -24

2 3 -10

3 3 8

关键问题分析：

本题问题的关于是用什么来存储稀疏矩阵，并且要确保乘完之后矩阵的行列序优先，以及如果优化他的时间复杂度。

二 概要设计

  通过十字链表存储矩阵信息，设计出三个功能函数：初始化矩阵函数、输出矩阵函数、矩阵相乘函数。在主函数中调用他们，在数据结构设计中加入一个rpos的前缀和数组来存储信息，即每行第一个非0元素在三元组表中的位置。然后在乘法算法中，将对应行和对应列的元素相乘，其结果放在t[]矩阵当中，最后遍历t矩阵，将相乘的结果放在对应的c矩阵当中然后输出。

三 详细设计

#include <stdio.h>

#define MAX 1000

//定义三元组结构体

typedef struct {

int r, c;

//所在行数和列数

int d;

//所表示的值

} Triple;

//一个矩阵的数据类型

typedef struct {

Triple list[MAX];

//1000个三元表

int m, n, t;

//存储这个矩阵当中信息，m行，n列，t个非0元素

int rpos[MAX] = {0};

} Matrix;

void InitMatrix(Matrix \*a);//初始化矩阵

void OutputMatrix(Matrix a);//输出函数，将三元组表内矩阵信息以三元组形式输出

void MutiplyMatrix(Matrix a, Matrix b, Matrix \*c); //矩阵相乘

int main() {

Matrix a, b, c;

InitMatrix(&a);

InitMatrix(&b);

MutiplyMatrix(a, b, &c);

OutputMatrix(c);

return 0;

}

//初始化矩阵

void InitMatrix(Matrix \*A) {

int i, j, k;

scanf("%d %d", &A->m, &A->n);

A->t = 0;

while (1) {

scanf("%d %d %d", &i, &j, &k);

if (i == 0 && j == 0 && k == 0)

break;

//当输入0 0 0时结束

A->t++;

//从1开始

A->list[A->t].r = i;

A->list[A->t].c = j;

A->list[A->t].d = k;

A->rpos[i]++;

//输入时设计rpos前缀和数组，记录第i行中的非0元素个数

}

//通过两个for循环，算出第i行中第一个元素在其三元组表中的位置，并将其存储在对应的rpos[i]数组当中

for (int i = 1; i <= A->m; i++)

A->rpos[i] += A->rpos[i - 1];

for (int i = A->m; i >= 1; i--)

A->rpos[i] = A->rpos[i - 1] + 1;

}

//矩阵相乘

void MutiplyMatrix(Matrix a, Matrix b, Matrix \*c) {

//对c矩阵的基本信息初始化赋值

c->m = a.m;

c->n = b.n;

c->t = 0;

int t[1000] = {0};

int x, y, z, s;

//判断条件是否不等于0

if (a.t \* b.t != 0) {

for (int i = 1; i <= a.m; i++) {

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

t[j] = 0;

c->rpos[i] = c->t + 1;

if (i < a.m)

x = a.rpos[i + 1];

else

x = a.t + 1;

for (int p = a.rpos[i]; p < x; p++) {

int c = a.list[p].c;

if (c < b.m)

y = b.rpos[c + 1];

else

y = b.t + 1;

for (int q = b.rpos[c]; q < y; q++) {

z = b.list[q].c;

t[z] += (a.list[p].d \* b.list[q].d);

}

}

//求a.list[p].d 和 b.list[q].d的乘积

for (int k = 1; k <= c->n; k++) {

if (t[k]) {

c->t++;

c->list[c->t].r = i;

c->list[c->t].c = k;

c->list[c->t].d = t[k];

}

}

//乘积矩阵Q中每个元素的值是满足条件乘积的累计和，因此我们选择一个累计和数组t来存储这些c[i][j] 的这个部分，初值为0，然后扫描每一个放进去，求得相应元素的乘积并累加到变量上，然后循环一遍将t数组里面的对应元素放到c中。

}

}

}

//输出函数，将三元组表内矩阵信息以三元组形式输出

void OutputMatrix(Matrix A) {

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

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

}

}

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；

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，以0 0 0结束。

1. 测试结果与分析；

测试输入: 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

测试输出:1 3 -2

2 1 32

2 2 -24

2 3 -10

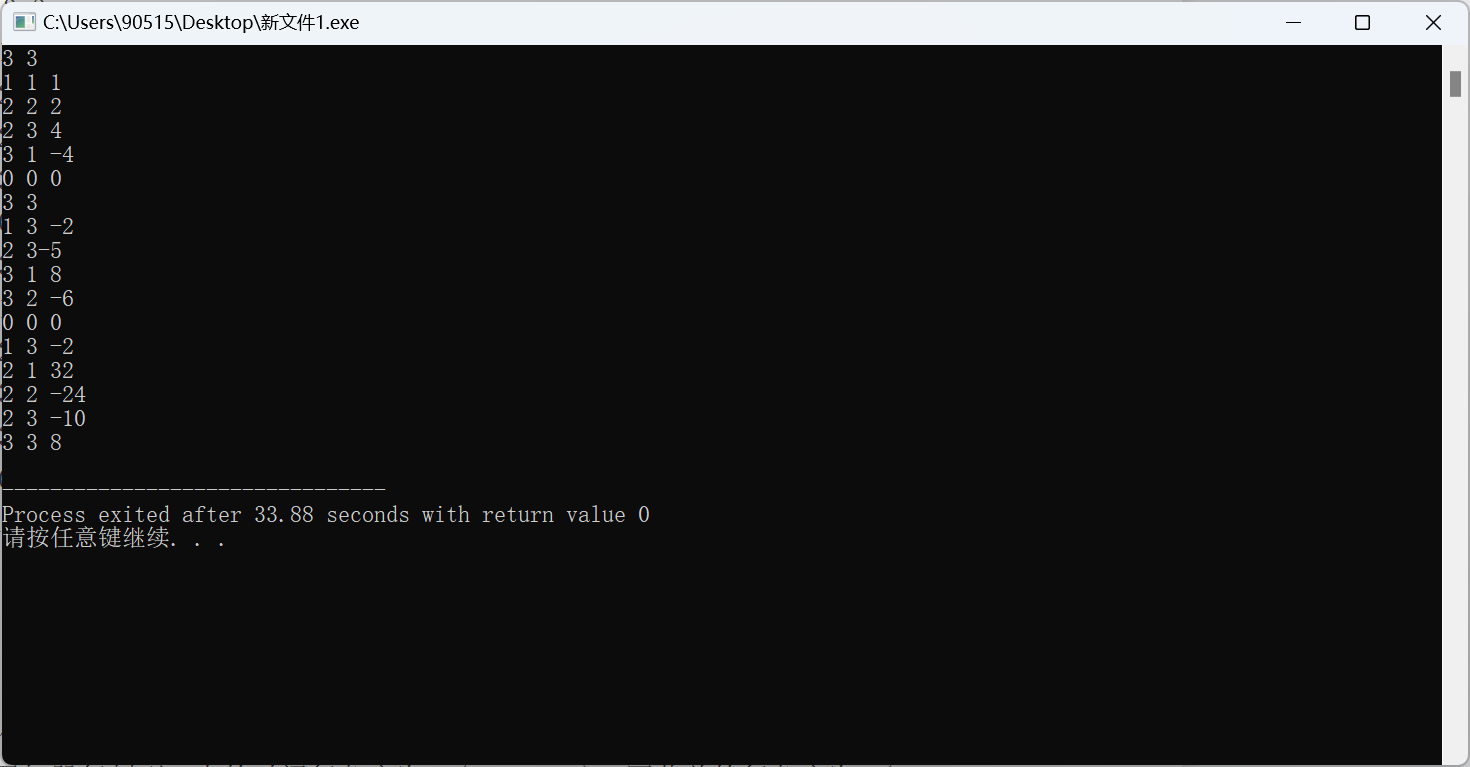
3 3 8

分析累加器初始化的时间复杂度为 O(M.m \* N.n) ，求Q所有的非零元的时间复杂度为O(M.t \* N.t/N.m) ，将累加器复制到Q中的时间复杂度为 O(M.m \* N.n) ，因此总的复杂度为O(M.m \* N.n + M.t \* N.t / N.m) 。

1. 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

过程中算乘积元素的时候对行列的分析，不够透彻。例如行列循环的问题，还有指针指向哪里，理解后得以解决。  
优化算法，将算出的结果存储在t数组中再遍历输入c数组中会节省不少时间。

1. 运行界面



五、实验总结

本实验最难的部分就是设计出相乘的算法，还要考虑到相乘相加后是否为0的问题，以及如何优化算法节省时间的问题。在本次实验中，需要不停的debug观察数组具体数值，判断是否有越界访问的问题。

最主要的还是乘法函数中相乘为0实现删除的具体操作，这是本实验的一大难点。