**武汉大 学计算机学院**

**本科生实验报告**

**数据结构实验报告**

**实验六：稀疏矩阵的加法和乘法问题**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：数据结构

指 导 教 师 ：安 扬

学 生 学 号 ：2017301500061

学 生 姓 名 ：彭 思 翔

学 生 班 级 ：计科二班

上 机 环 境 ：Visual Studio Code

二○一八 年 12 月

**一、实验题目**

实验六：稀疏矩阵的加法和乘法问题

【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、实验项目的**

深入掌握递稀疏矩阵的写法和应用。

1. **实验项目程序结构**

initMatrix

dispMatrix

transpositoni

main

findStart

multiply

addition

**四、实验项目中各文件函数功能描述**

void dispMatrix(Matrix); //输出稀疏矩阵

void initMatrix(Matrix &); //读入稀疏矩阵

Matrix tansposition(Matrix); //反转稀疏矩阵

void findStart(int\*, Matrix); //找到该矩阵每一行的开头是第几个元素

Matrix addition(Matrix, Matrix); //稀疏矩阵相加

Matrix multiply(Matrix, Matrix); //稀疏矩阵相乘

**五、算法描述**

【数据结构】

稀疏矩阵：Matrix用来存矩阵信息，TupNode用来存稀疏矩阵中结点的信息，我重定义了几个表示方便代码书写。

#define r(a, i) a.data[i].r

#define c(a, i) a.data[i].c

#define d(a, i) a.data[i].d

#define copy\_int(a, i, row, col, value) a.data[i].r = row, a.data[i].c = col, a.data[i].d = value

#define copy\_tup(a, i, b, j) a.data[i].r = b.data[j].r, a.data[i].c = b.data[j].c, a.data[i].d = b.data[j].d

typedef struct {

int r;

int c;

int d;

} TupNode;

typedef struct {

int rows;

int cols;

int nums;

TupNode data[MaxN];

} Matrix;

【设计思路】

矩阵转置：朴素的想法是枚举列号，对于每一列枚举元素逐行反转每一个元素的行列加入到新的矩阵中。这样的做法是O（Cols\*N）的，十分暴力。我们试着维护这样一个数组cstart[i]，表示第i列数未访问的开头元素在新的稀疏矩阵元素列表中的位置，这样一来我们只需遍历旧的稀疏矩阵，反转并按照cstart的位置加入到新的稀疏矩阵中即可。而维护cstart只需在开头的时候遍历旧稀疏矩阵记录每一列的数目，cstart就可以计算出来，从而每次访问后cstart++就可以保持了。

时间复杂度O(N)

Matrix tansposition(Matrix s) {

int j = 0;

Matrix c;

c.nums = s.nums;

c.rows = s.rows;

c.cols = s.cols;

int num[c.cols] = {0}, cstart[c.cols] = {0};

for (int i = 0; i < s.nums; i++) num[c(s, i)]++; //遍历旧稀疏矩阵统计每列数目

cstart[0] = 0;

for (int col = 1; col < c.cols; col++) //计算cstart即每一列开头在新矩阵中的位置

cstart[col] = cstart[col - 1] + num[col - 1];

for (int i = 0; i < s.nums; i++) {

j = cstart[c(s, i)];

copy\_int(c, j, c(s, i), r(s, i), d(s, i));

cstart[c(s, i)]++; //该列下一个访问的应该在当前的后一位

}

return c;

}

矩阵相加：类似于归并排序，两个指针从前往后分别遍历矩阵，注意归并大小比较变为了以行为第一关键字、列为第二关键字的双关键字比较。

时间复杂度O(N)

Matrix addition(Matrix a, Matrix b) {

Matrix c;

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

c.cols = a.cols;

c.rows = a.rows;

while (i < a.nums && j < b.nums) {

if (r(a, i) > r(b, j)) copy\_tup(c, k, b, j), j++, k++;

else if (r(a, i) < r(b, j)) copy\_tup(c, k, a, i), i++, k++;

else if (c(a, i) < c(b, j)) copy\_tup(c, k, a, i), i++, k++;

else if (c(a, i) > c(b, j)) copy\_tup(c, k, b, j), j++, k++;

else {

int dd = d(a, i) + d(b, j);

if (dd != 0) copy\_int(c, k, r(a, i), c(a, i), dd), k++;

i++, j++;

}

}

while (i < a.nums) copy\_tup(c, k, a, i), i++, k++;

while (j < b.nums) copy\_tup(c, k, b, j), j++, k++;

c.nums = k;

return c;

}

矩阵相乘：传统朴素的稀疏矩阵相乘就是枚举行列，再暴力找出第一个矩阵的行的每个元素和其对应的第二个矩阵的列的元素相乘的和。这样思路简单，但是时间复杂度为O(Rows\*Cols\*N\*N)。我们用转置矩阵类似的思路，找到A和B（假设矩阵A和B相乘）每一行开头元素在矩阵列表中的位置Astart和Bstart。枚举A的行Ri，通过Astart找到并遍历A中该行的所有元素j，对于每个元素j，找到j的列Cj，通过Bstart找到并遍历B中Cj行的所有元素k，将j×k的值存入temp[Ck]表示Ri行Ck列的值，这样一次性求出了Ri行的所有结果。看似有三层循环，但是由于A中每个元素只访问一次，对于每个A的元素只访问了对应行的B。设B平均每行有k个（k为比较小的常数），则时间复杂度为O(kN)

时间复杂度O(kN)

Matrix multiply(Matrix a, Matrix b) {

Matrix c;

int ta, tb;

int Arstart[a.rows] = {0}, Brstart[a.rows] = {0}, Crstart[a.rows] = {0};

c.rows = a.rows; c.cols = b.cols; c.nums = 0;

findStart(Arstart, a);

findStart(Brstart, b);

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

int t, temp[b.cols] = {0};

if (i < a.rows - 1) ta = Arstart[i + 1];

else ta = a.nums;

for (int j = Arstart[i]; j < ta; j++) {

if (c(a, j) < b.rows - 1) tb = Brstart[c(a, j) + 1];

else tb = b.nums;

for (int k = Brstart[c(a, j)]; k < tb; k++)

temp[c(b, k)] += d(a, j) \* d(b, k);

}

for (int col = 0; col < b.cols; col++)

if (temp[col])

copy\_int(c, c.nums, i, col, temp[col]), c.nums++;

}

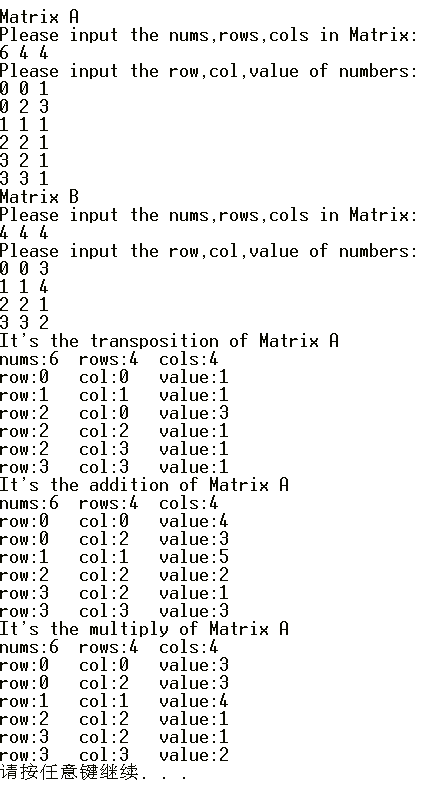
return c;

}

其中findStart的代码与矩阵转置时的类似，把列统计改为行统计即可，不再赘述。

**六、实验数据和实验结果分析**

运行结果良好。



**七、实验体会**

本次实验原本都是按照书上用暴力破解的，但是时间复杂度实在是高到不可攀登，所以搜索学习了优化算法，没想到记录了每一行/列开始位置以后时间复杂度大大下降了，非常开心。