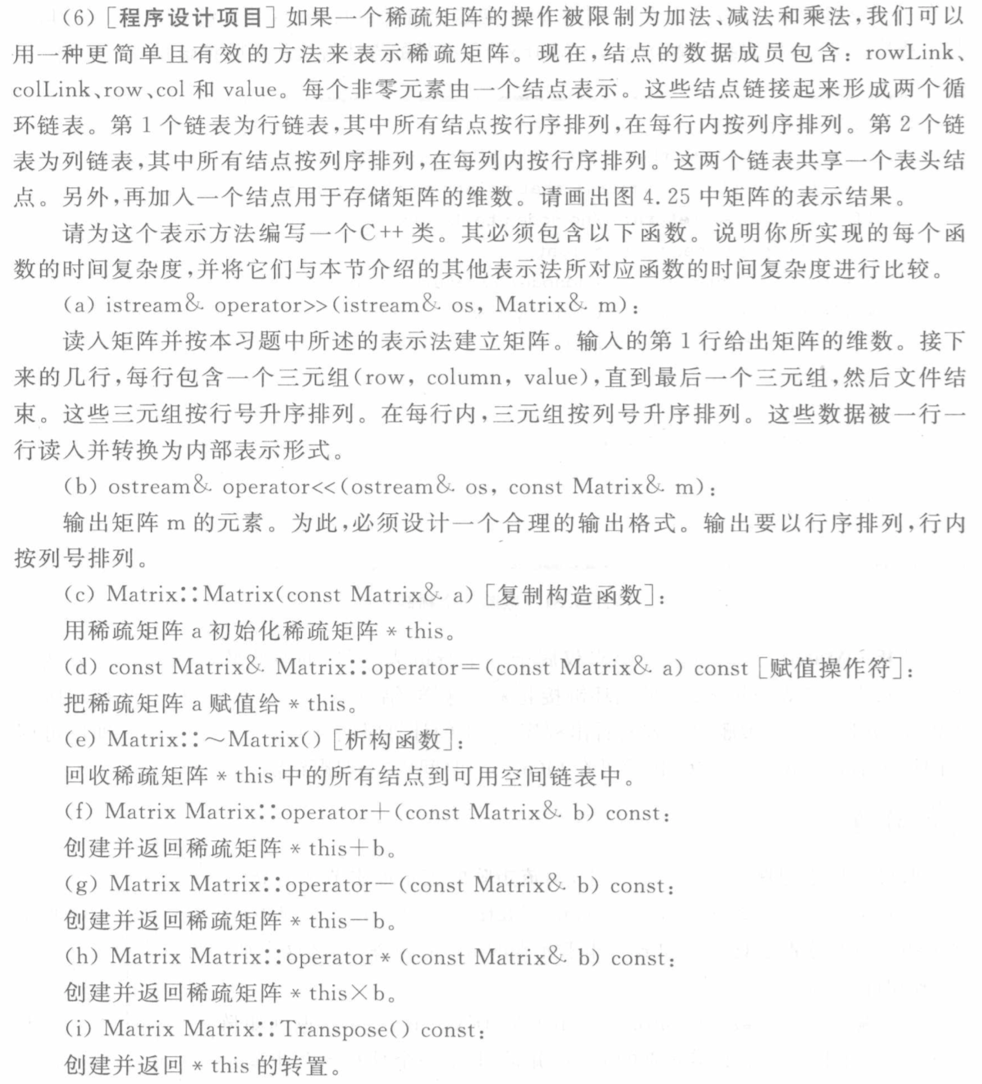
**《数据结构》实验报告（第四章）**

**学号：\_\_\_\_\_\_58122231\_\_\_\_\_\_\_\_ 姓名 ：\_\_\_\_\_\_\_陆文韬\_\_\_**

**实验题号：\_P223 6 实验日期：\_2023.12.05\_\_\_\_**

**实验一**

**1．问题描述：**描述实验内容和要求以及需要解决的问题。

****

**2．算法思想：**

首先定义一个Node类，包含行号，列号，数值，行指针，列指针这些数据乘员，再定义一个稀疏矩阵类，类内包含一个共用的head节点，他的行指针和列指针分别连接链表的行表头和列表头，再根据这个结构去定义具体的功能函数。

**3．功能函数：**

findNode：根据行号和列号，返回链表中的对应指针

setValue：根据行号列号和数值，如果已有节点则修改节点的数值，如果没有节点则创建节点

operator+：将两个矩阵相加

operator\*：将两个矩阵相乘

getValue：通过调用findNode函数获取节点的数值

printMatrix：打印出矩阵

**4．测试数据：**

SparseMatrix matrix1(3, 3);

matrix1.setValue(0, 1, 2);

matrix1.setValue(1, 1, 4);

matrix1.setValue(2, 0, 5);

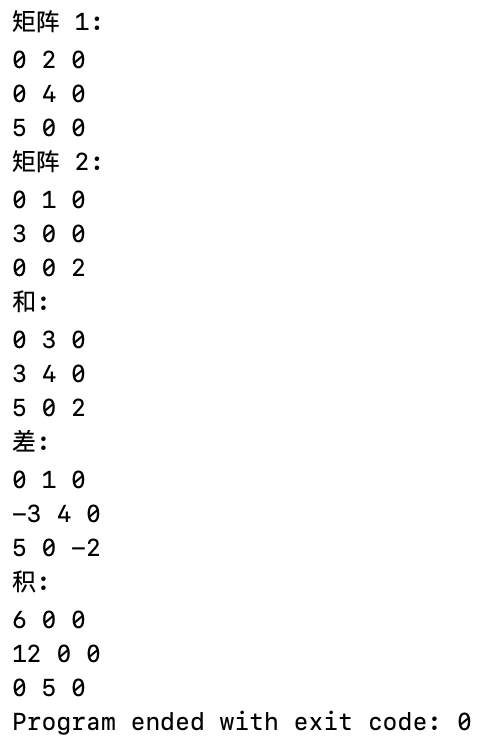
SparseMatrix matrix2(3, 3);

matrix2.setValue(0, 1, 1);

matrix2.setValue(1, 0, 3);

matrix2.setValue(2, 2, 2);

**5．测试情况：**

****

**6．实验总结：**

在节点类和稀疏矩阵类的定义上面，要注意自己设计的数据结构是否合理，跟题目的要求是否一致，好的设计能让程序的编写事半功倍，在功能函数的编写上，要注意指针的使用，要充分考虑到各种特殊情况，否则在程序调试的时候极易出现内存泄漏、边界溢出等问题。

**7. 源代码：**

实验源程序清单

#include <iostream>

**using** **namespace** std;

**class** SparseMatrix {

**private**:

**class** Node {

**public**:

**int** row, col, value;

Node\* rowLink;

Node\* colLink;

Node(**int** r, **int** c, **int** v) : row(r), col(c), value(v), rowLink(**nullptr**),colLink(**nullptr**) {}

};

Node\* head;//头结点

**int** numRows;//行数

**int** numCols;//列数

**public**:

SparseMatrix(**int** rows, **int** cols) : numRows(rows), numCols(cols) {

head = **new** Node(-1, -1, 0); // 头结点

head->rowLink = head; // 初始化行链表

head->colLink = head;

}

// 找到对应位置节点

Node\* findNode(**int** row, **int** col) **const** {

Node\* current = head->rowLink;

**while** (current->rowLink != head->rowLink) {

**if** (current->row == row && current->col == col) {

**return** current;

}

current = current->rowLink;

}

**return** **nullptr**;

}

**void** setValue(**int** row, **int** col, **int** value) {

Node\* node = findNode(row, col);

**if** (node) {

// 如果节点存在，只需要修改值

node->value = value;

**return**;

}

// 新建节点

Node\* newNode = **new** Node(row, col, value);

// 在行链表中插入节点

**if** (head->rowLink == head || head->rowLink->row > row) {

newNode->rowLink = head->rowLink;

head->rowLink = newNode;

}

**else** {

Node\* rowPrev = head;

**while** (rowPrev->rowLink != head && rowPrev->rowLink->row < row && (rowPrev->rowLink->row == row && rowPrev->rowLink->col<col)) {

rowPrev = rowPrev->rowLink;

}

newNode->rowLink = rowPrev->rowLink;

rowPrev->rowLink = newNode;

}

// 在列链表中插入节点

**if** (head->colLink == head || head->colLink->col > col) {

newNode->colLink = head->colLink;

head->colLink = newNode;

}

**else** {

Node\* colPrev = head;

**while** (colPrev->colLink != head && colPrev->colLink->col < col && (colPrev->colLink->col == col && colPrev->colLink->row < row)) {

colPrev = colPrev->colLink;

}

newNode->colLink = colPrev->colLink;

colPrev->colLink = newNode;

}

}

SparseMatrix **operator**+(**const** SparseMatrix& other) **const** {

**if** (numRows != other.numRows || numCols != other.numCols) {

**throw** runtime\_error("矩阵维度必须一致.");

}

SparseMatrix result(numRows, numCols);

**for** (**int** i = 0; i < numRows; i++) {

**for** (**int** j = 0; j < numCols; j++) {

**int** value = getValue(i, j) + other.getValue(i, j);

result.setValue(i, j, value);

}

}

**return** result;

}

SparseMatrix **operator**-(**const** SparseMatrix& other) **const** {

**if** (numRows != other.numRows || numCols != other.numCols) {

**throw** runtime\_error("矩阵维度必须一致");

}

SparseMatrix result(numRows, numCols);

**for** (**int** i = 0; i < numRows; i++) {

**for** (**int** j = 0; j < numCols; j++) {

**int** value = getValue(i, j) - other.getValue(i, j);

result.setValue(i, j, value);

}

}

**return** result;

}

SparseMatrix **operator**\*(**const** SparseMatrix& other) **const** {

**if** (numCols != other.numRows) {

**throw** runtime\_error("第一个矩阵的列数必须等于第二个矩阵的行数");

}

SparseMatrix result(numRows, other.numCols);

**for** (**int** i = 0; i < numRows; i++) {

**for** (**int** j = 0; j < other.numCols; j++) {

**int** value = 0;

**for** (**int** k = 0; k < numCols; k++) {

value += getValue(i, k) \* other.getValue(k, j);

}

result.setValue(i, j, value);

}

}

**return** result;

}

// 返回节点值

**int** getValue(**int** row, **int** col) **const** {

Node\* node = findNode(row, col);

**return** (node ? node->value : 0);

}

// 打印矩阵

**void** printMatrix() **const** {

**for** (**int** i = 0; i < numRows; i++) {

**for** (**int** j = 0; j < numCols; j++) {

cout << getValue(i, j) << " ";

}

cout << endl;

}

}

};

**int** main() {

SparseMatrix matrix1(3, 3);

matrix1.setValue(0, 1, 2);

matrix1.setValue(1, 1, 4);

matrix1.setValue(2, 0, 5);

SparseMatrix matrix2(3, 3);

matrix2.setValue(0, 1, 1);

matrix2.setValue(1, 0, 3);

matrix2.setValue(2, 2, 2);

SparseMatrix sum = matrix1 + matrix2;

SparseMatrix difference = matrix1 - matrix2;

SparseMatrix product = matrix1 \* matrix2;

cout << "矩阵 1:" << endl;

matrix1.printMatrix();

cout << "矩阵 2:" << endl;

matrix2.printMatrix();

cout << "和:" << endl;

sum.printMatrix();

cout << "差:"<< endl;

difference.printMatrix();

cout << "积:" << endl;

product.printMatrix();

**return** 0;

}