#include <iostream>

#include <vector>

using namespace std;

// 矩阵加法

vector<vector<int>> matrixAddition(const vector<vector<int>>& A, const vector<vector<int>>& B) {

int rows = A.size(), cols = A[0].size();

vector<vector<int>> result(rows, vector<int>(cols, 0));

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

for (int j = 0; j < cols; j++) {

result[i][j] = A[i][j] + B[i][j];

}

}

return result;

}

// 矩阵乘法

vector<vector<int>> matrixMultiplication(const vector<vector<int>>& A, const vector<vector<int>>& B) {

int rowsA = A.size(), colsA = A[0].size();

int rowsB = B.size(), colsB = B[0].size();

if (colsA != rowsB) {

throw invalid\_argument("无法进行矩阵乘法，A的列数必须等于B的行数");

}

vector<vector<int>> result(rowsA, vector<int>(colsB, 0));

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

for (int j = 0; j < colsB; j++) {

for (int k = 0; k < colsA; k++) {

result[i][j] += A[i][k] \* B[k][j];

}

}

}

return result;

}

// 打印矩阵

void printMatrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int val : row) {

cout << val << " ";

}

cout << endl;

}

}

int main() {

vector<vector<int>> A = {

{1, 2, 3},

{4, 5, 6}

};

vector<vector<int>> B = {

{7, 8, 9},

{10, 11, 12}

};

cout << "矩阵 A:" << endl;

printMatrix(A);

cout << "矩阵 B:" << endl;

printMatrix(B);

// 矩阵加法

if (A.size() == B.size() && A[0].size() == B[0].size()) {

vector<vector<int>> sum = matrixAddition(A, B);

cout << "A + B 矩阵结果:" << endl;

printMatrix(sum);

} else {

cout << "无法进行矩阵加法，矩阵维度不匹配！" << endl;

}

// 矩阵乘法

vector<vector<int>> C = {

{1, 2},

{3, 4},

{5, 6}

};

vector<vector<int>> D = {

{7, 8, 9},

{10, 11, 12}

};

cout << "矩阵 C:" << endl;

printMatrix(C);

cout << "矩阵 D:" << endl;

printMatrix(D);

try {

vector<vector<int>> product = matrixMultiplication(C, D);

cout << "C × D 矩阵结果:" << endl;

printMatrix(product);

} catch (const exception& e) {

cout << "错误: " << e.what() << endl;

}

return 0;

}

修改指定位置rc的元素值为v  
template <class ElemType>

Status TriSparseMatrix<ElemType>::SetElem(int r, int c, const ElemType &v) {

// 检查索引是否越界

if (r < 0 || r >= rows || c < 0 || c >= cols) {

return ERROR; // 越界错误

}

// 查找是否已有该位置的元素

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

if (triElems[i].row == r && triElems[i].col == c) {

if (v == 0) {

// 若新值为0，则删除该元素（后续元素前移）

for (int j = i; j < num - 1; j++) {

triElems[j] = triElems[j + 1];

}

num--; // 更新非零元素计数

} else {

triElems[i].value = v; // 直接修改已有元素的值

}

return OK;

}

}

// 若该位置不存在元素且值不为0，则新增元素

if (v != 0) {

if (num >= maxSize) return ERROR; // 超出最大存储限制

triElems[num++] = {r, c, v}; // 添加新的非零元素

}

return OK;

}

取指定位置的元素值

template <class ElemType>

Status TriSparseMatrix<ElemType>::GetElem(int r, int c, ElemType &v) const {

// 检查索引是否越界

if (r < 0 || r >= rows || c < 0 || c >= cols) {

return ERROR; // 越界错误

}

// 遍历查找元素

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

if (triElems[i].row == r && triElems[i].col == c) {

v = triElems[i].value; // 找到对应值

return OK;

}

}

v = 0; // 若未找到则返回0（默认值）

return OK;

}

复制构造函数

template <class ElemType>

TriSparseMatrix<ElemType>::TriSparseMatrix(const TriSparseMatrix<ElemType> &copy) {

// 复制基本信息

rows = copy.rows;

cols = copy.cols;

num = copy.num;

maxSize = copy.maxSize;

// 分配新内存存储三元组表

triElems = new Triple<ElemType>[maxSize];

// 复制三元组数据

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

triElems[i] = copy.triElems[i];

}

}

赋值运算符重载

template <class ElemType>

TriSparseMatrix<ElemType>&TriSparseMatrix<ElemType>::operator=(const TriSparseMatrix<ElemType> &copy) {

if (this != &copy) { // 避免自赋值

delete[] triElems; // 释放原有内存

// 复制基本信息

rows = copy.rows;

cols = copy.cols;

num = copy.num;

maxSize = copy.maxSize;

// 分配新内存并复制数据

triElems = new Triple<ElemType>[maxSize];

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

triElems[i] = copy.triElems[i];

}

}

return \*this;

}