**稀疏矩阵的逆置**

* 1. **稀疏**矩阵的逆置
     1. 我们知道稀疏矩阵是按照三元组来进行存储的
     2. 所以我们比如（1，3）存储的是3
     3. 那我们进行逆置一下是不是应该变成（3，1） 3
     4. 这样的形式了，那我们怎么对三元组进行逆置和保存下标是按照第一列的顺序进行排序的，因为我们知道，如果我们仅仅时互换i、j时达不到此目的的
     5. 代码

1. 头文件

#pragma once

#include <vector>

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class TSMatrix {

struct Triple {

//两个坐标

int i, j;

T e;

};

int x = 0;

int y = 0;

int t = 0; //非零个数

public:

vector<Triple> data;

//构造函数

TSMatrix() = default;

TSMatrix(int x1, int y1, int t1 = 0):x(x1),y(y1) {

if (t1 > 0) {

t = t1;

data.resize(t1); // 看data应该变成多少的

}

}

//下标取值函数,返回的是三元组的类型的,也就是说不会改变原来的值

Triple& operator[](int i)const {

return data[i];

}

Triple& operator[](int i) {

return data[i];

}

int size() {

return data.size();

}

//待会在进行实现

template<typename T>

friend void invTSMatrix(const TSMatrix<T> &m, TSMatrix<T> &T);

};

template<typename T>

void invTSMatrix(const TSMatrix<T> &M, TSMatrix<T> &T) {

//进来我们先初始化一下T

T.x = M.x;

T.y = M.y;

T.t = M.t;

T.data.resize(M.t);

if (T.t) {

int q = 0;

for (int col = 1; col <= M.x; col++) {

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

//我们要从i从小到大进行排序，那么我们应该对齐j进行排序，每次外层循环对j进行判断

if (M.data[i].j == col) {

T.data[q].i = M.data[i].j;

T.data[q].j = M.data[i].i;

T.data[q].e = M.data[i].e;

q++;

}

}

}

}

}

1. 主函数

// 三元组.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <iostream>

#include "TSMatrix.h"

int main()

{

//实例化三行四列，4个非零数组，其实我么也应该知道，我们非零的个数应该与我们的列数是相等的呀

// 既然是三元组，所以我们知道肯定是3列

TSMatrix<int> m(4, 3, 4);

//我们自己输入一些值来进行测试一下

m[0].i = 1; m[0].j = 2; m[0].e = 4;

m[1].i = 1; m[1].j = 3; m[1].e = 5;

m[2].i = 3; m[2].j = 1; m[2].e = 6;

m[3].i = 3; m[3].j = 4; m[3].e = 7;

std::cout << "x" << " " << "y" << " " << "e" << std::endl;

//现在我们进行打印输出一下

for (int i = 0; i < 4; i++) {

std::cout << m[i].i << " " << m[i].j << " " << m[i].e << std::endl;

}

std::cout << std::endl;

//我们来实验一下逆置

TSMatrix<int> T;

invTSMatrix(m, T);

std::cout << "x" << " " << "y" << " " << "e" << std::endl;

for (int i = 0; i < 4; i++) {

std::cout << T[i].i << " " << T[i].j << " " << T[i].e << std::endl;

}

}