**离散数学课程设计**

**项目说明文档**

**求关系的自反、对称和传递闭包**

作 者 姓 名： 苏家铭

学 号： 2151299

指 导 教 师： 唐剑锋

学院、 专业： 软件学院 软件工程

同济大学

**Tongji University**



**目录**

[1 项目分析 1](#_Toc6531)

[1.1 项目背景 1](#_Toc19350)

[1.2项目要求 1](#_Toc14156)

[1.2.1 功能要求 1](#_Toc19211)

[1.2.2 输入要求 1](#_Toc13594)

[1.2.3 输出要求 1](#_Toc18747)

[1.2.4 项目实例 1](#_Toc3931)

[2 项目设计及实现 3](#_Toc2344)

[2.3 项目算法 3](#_Toc31787)

[2.3.1 实现思路 3](#_Toc29458)

[2.3.2 代码实现 4](#_Toc2812)

[3 项目测试 7](#_Toc18448)

[4 算法性能分析 9](#_Toc21065)

[4.1 正确性 9](#_Toc16236)

[4.2 可使用性 9](#_Toc9230)

[4.3 可读性 9](#_Toc17341)

[4.4 效率 9](#_Toc18140)

[4.5 健壮性 10](#_Toc30246)

[5 实验感想 10](#_Toc24776)

# 1 项目分析

## 项目背景

本项目我们只研究关系的自反、对称、传递的闭包。设R是费控集合A上的关系，有时候人们希望R具有一些有用的性质，例如，自反性（对称性或传递性）。为此，需要在R中添加一些有序对而构成新的关系R’，使得R’具有所需要的性质，但又不希望R’变得“太大”。换句话说，希望添加的有序对尽可能少。满足这些要求的R’就是R的自反（对称或传递）闭包。

## 1.2项目要求

### 1.2.1 功能要求

对以矩阵表示的关系，用矩阵的方式表示出矩阵的自反、对称和传递闭包。

### 1.2.2 输入要求

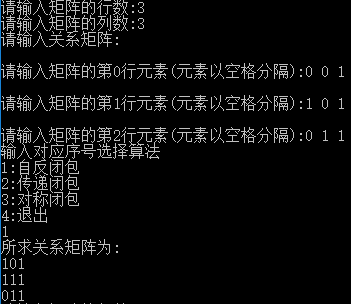
输入矩阵的行数和列数，以及初始未实现闭包的矩阵。

### 1.2.3 **输出要求**

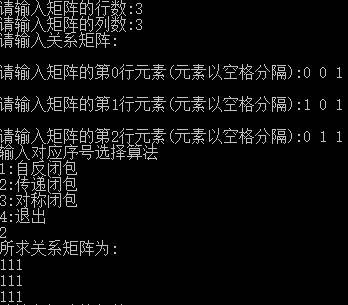
自反/传递/闭包关系的矩阵表示。

### 1.2.4 项目实例

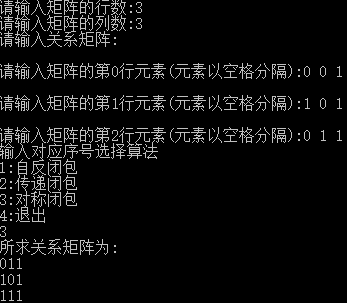
1.求关系的自反闭包：



2. 求关系的传递闭包：



3. 求关系的对称闭包：



# 2 **项目设计及实现**

## 2.3 **项目算法**

R的关系矩阵为M，相应的自反、对称、传递闭包的矩阵为Mr、Ms、Ml，则有

* Mr=M+E
* Ms=M+M’
* Ml=M+M2+M3+...

其中E表示同阶的单位矩阵（主对角线为1，其他元素为0），M’表示M的转置，而+均表示矩阵中对应元素的逻辑加。

### 2.3.1 实现思路

1. 自反：

对于单位矩阵，只需要利用两个循环，将数组同行同列的元素置为1即可。

1. 对称：

原矩阵逻辑加转置矩阵，实际上只需要在原矩阵一遍遍历的过程中，遇到arr[i][j]为1,那么将对称位置arr[j][i]也置为1即可。

1. 传递：

本质上需要Ml=M+M2+M3+...,那么可以利用矩阵的相乘，将M的各次方计算出来（一直到N次方，N为行/列数，即变量个数），因为在程序中没法做到每个次方的矩阵都算好后再相加，那么就需要每计算一次矩阵，就在结果矩阵中加上这次计算的矩阵（总共加N-1次）。

### 2.3.2 代**码实**现

#include<iostream>

using namespace std;

void duichen();

void chuandi();

void zifan();

void exit();

void show();

void init();

void select();

void delete\_shuzu(int\*\* arrx, int n);

int \*\*arr,n,choice;

int main() {

init();//初始化矩阵

show();//展示初始化的矩阵

select();//选择算法,choice

switch (choice) {

case 1:

zifan();

show();

break;

case 2:

chuandi();

show();

break;

case 3:

duichen();

show();

break;

case 4:

exit();

break;

}

delete\_shuzu(arr, n);

return 0;

}

void delete\_shuzu(int\*\* arrx, int n) {//清理数组的空间的函数，以免损害内存

for (int i = 0; i < n; ++i)

delete []arrx[i];

}

void exit() {//退出的函数

cout << "按回车退出......." << endl;

cin.get();

}

void init() {//初始化矩阵

cout << "请输入矩阵行(列)数:";

while (1) {//健壮性，以防非法输入

cin >> n;

if (cin.fail() || n < 0) {

cout << "输入非法，请再次输入:" << endl;

cin.clear();

cin.ignore(65535, '\n');

}

else

break;

}

cout << "创建初始矩阵:" << endl;

arr = new int\* [n];

for (int i = 0; i < n; ++i)

arr[i] = new int[n];

for (int i = 0; i < n; i++) {//用两个 循环输入矩阵

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

cin >> arr[i][j];

}

}

}

void select() {

cout << "=======================" << endl;

cout << "=======1:自反闭包======" << endl;

cout << "=======2:传递闭包======" << endl;

cout << "=======3:对称闭包======" << endl;

cout << "=======4:退出==========" << endl;

cout << "=======================" << endl;

cout << "请输入相应序号选择对应算法:";

while (1) {//健壮性

cin >> choice;

if (cin.fail()|| choice < 0 || choice>4) {

cout << "输入错误，请再次输入:";

cin.clear();

cin.ignore(65535, '\n');

}

else

break;

}

}

void zifan() {//自反,主对角线置1

for (int i = 0; i < n; i++)

arr[i][i] = 1;

}

void duichen() {//遍历，如果ij是1，那么ji也要是1

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

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

if (arr[i][j] == 1)

arr[j][i] = 1;

}

}

}

void chuandi() {//传递，时间复杂度O(n4)，未用沃舍尔算法

int \*\*next,\*\*ans,\*\*M,temp=0;

next = new int\* [n];

for (int i = 0; i < n; ++i)

next[i] = new int[n];

ans = new int\* [n];

for (int i = 0; i < n; ++i)

ans[i] = new int[n];

M = new int\* [n];

for (int i = 0; i < n; ++i)

M[i] = new int[n];

//memset();

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

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

next[i][j] = 0;

ans[i][j] = arr[i][j];

M[i][j]= arr[i][j];

}

}

for (int m = 0; m < n; m++) {//n个矩阵相加

for (int i = 0; i < n; i++) {//第i行每个元素\*第j列每个元素

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

temp = 0;

for (int d=0; d < n; d++){

temp+=(M[i][d] \* arr[d][j]);

}

if (temp >= 1)

next[i][j] = 1;

}

}

for (int i = 0; i < n; i++) {//把arr和M矩阵加起来

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

M[i][j] = next[i][j];

ans[i][j] = ans[i][j] + M[i][j];

next[i][j] = 0;

if (ans[i][j] > 1)//bool

ans[i][j] = 1;

}

}

}

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

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

arr[i][j] =ans[i][j];

}

}

delete\_shuzu(next, n);//清理空间

delete\_shuzu(ans, n);

delete\_shuzu(M, n);

}

void show() {//展示矩阵

cout << "矩阵：" << endl;

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

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

cout << arr[i][j] << " ";

}

cout << endl;

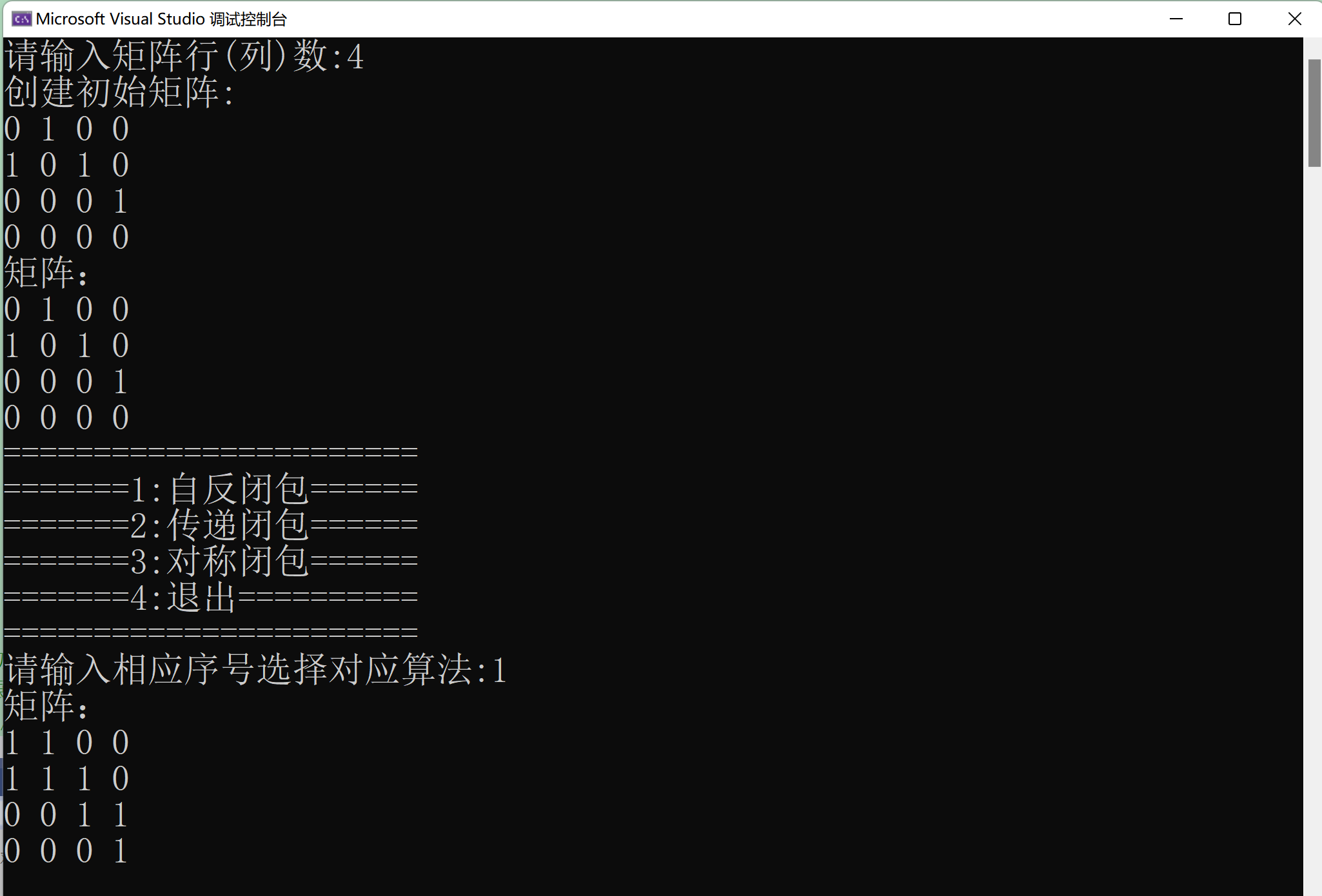
}

}

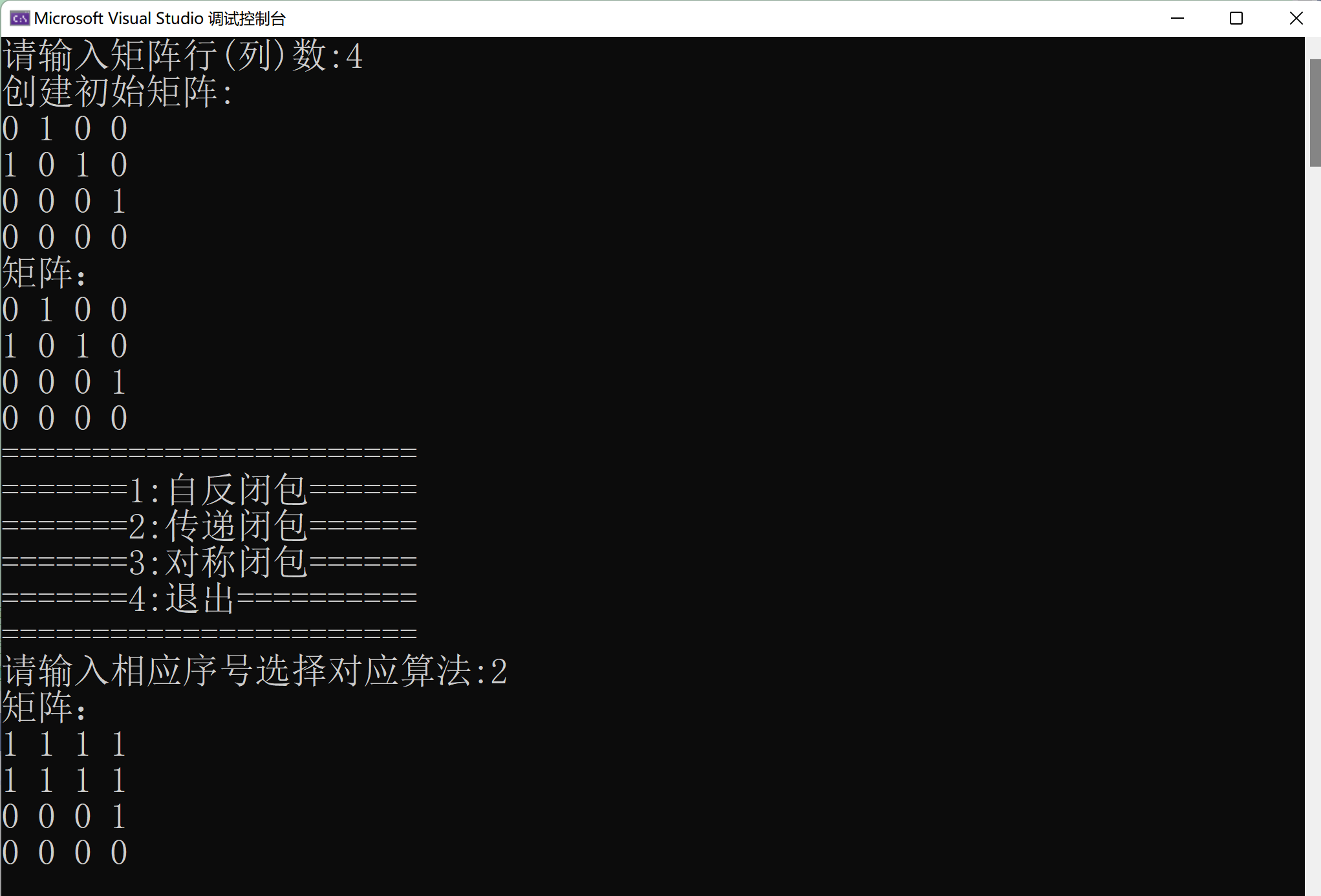
# 3 项目测试

以教材p89例4.10的矩阵为例：

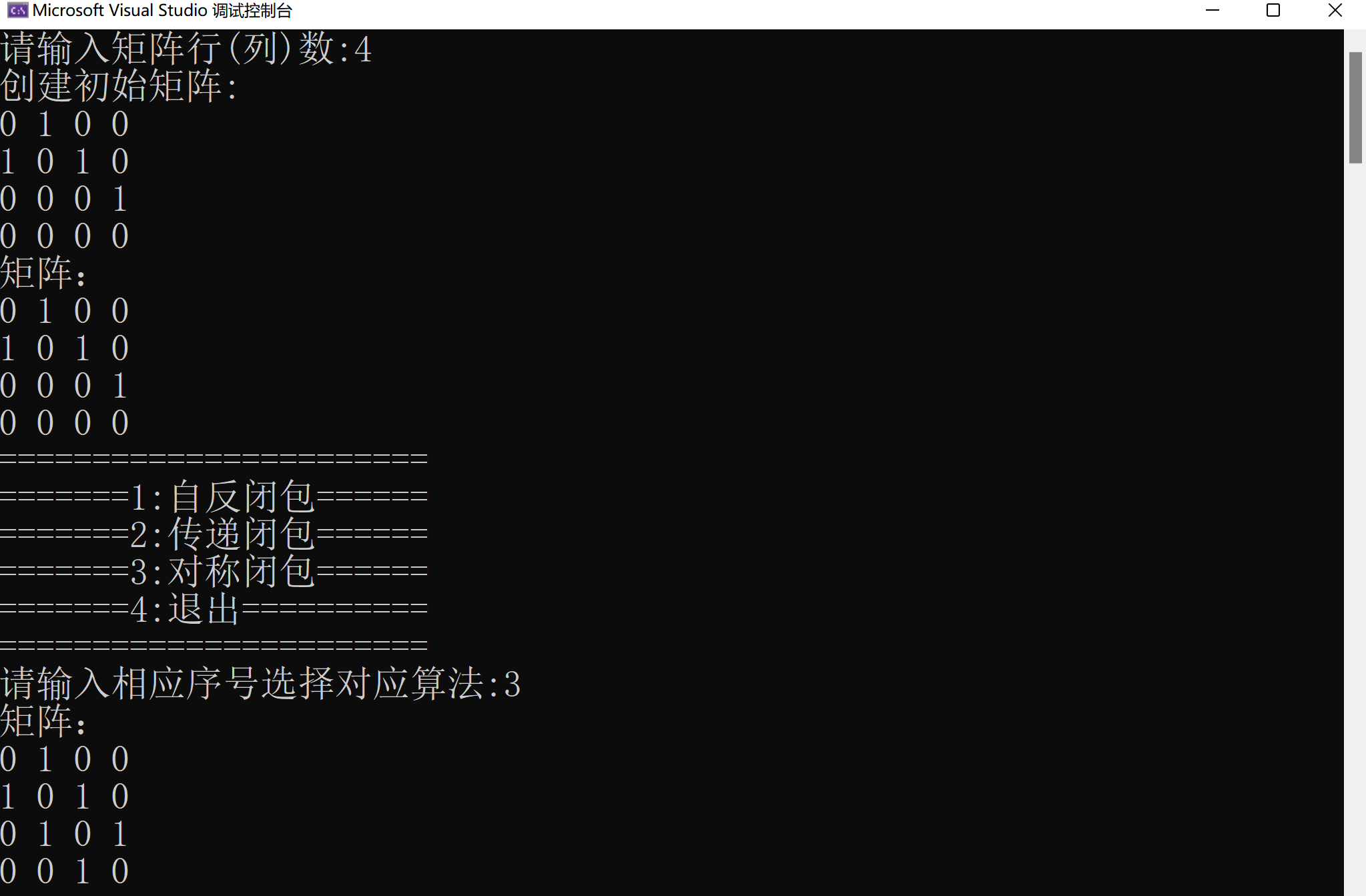
1. 自反闭包：



1. 传递闭包：



1. 对称闭包：



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，由以上版块证明本项目与教材p89例题4.11例题结果一致，并且经过大量数据验证，满足正确性。

## 4.2 可使用性

本算法可以很方便地使用，求传递、对称、自反闭包的功能已封装在一个函数中。并且该算法有良好的界面和完备的用户文档。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

时间复杂度：

* 自反闭包：O(n)
* 对称闭包：O(n2)
* 传递闭包：O(n4)

## 4.5 健壮性

本算法对于边界条件，诸如：选择对应算法的操作、输入行/列数非法都有相应的错误提示和重新输入提示。

# 5 实验感想

关系的闭包是二元关系中非常重要的概念，它使得人们在研究一个二元关系的自反、对称、传递性质的时候不会研究使得研究范围太大，更不会研究范围太小，使得这个闭包关系对于原本的二元关系来说恰当好处。

本实验基于纸面上的矩阵表示二元关系和实现，把程序实现融入其中，使得我们不仅能在离散数学的课堂上学习到专业的数学知识，还能将离散数学的知识应用于程序设计。在计算自反闭包和对称闭包的时候需要用到矩阵加法和转置，而计算传递闭包的时候需要用到矩阵的乘法，这都是离散数学和线性代数结合的体现，这也启示着我学习是一个持续的过程，一环扣一环，不能在任何一个结点落下，后续知识点的学习往往需要前一个阶段知识的铺垫。

在计算传递闭包的思想是将一个个矩阵通过相乘计算出n种不同次方并相加，这种算法的时间复杂度非常高：需要n个矩阵相加O(n)，每个nXn的矩阵需要通过矩阵相乘得到O(n2)，每个矩阵中的每个项又需要某行和某列各个对应数的相乘之和O(n)，各个操作之间是嵌套关系，所以总的时间复杂度为O(n4)，非常消耗时间，并且在计算过程中需要若干个辅助矩阵和变量，也非常消耗空间。

在后续的实验设计中还将实现一种时间复杂度和空间复杂度更低的Warshall算法，不断更新迭代已有的算法。