**《离散数学》求关系的自反、对称和传递闭包项目文档**

1. **题目简介**

使用传统的方法分别求解关系的自反、对称和传递闭包；其中闭包的构造方法用矩阵表示，关系矩阵是方阵，需要输入行数(或列数）一个参数，然后按行分别输入矩阵的元素，元素之间以空格分割开；

给出4个选项，包括求自反、传递、对称闭包和退出，通过输入对应序号选择算法，最后输出所求关系矩阵。

1. **解题思路**

**2.1实验原理**

**2.1.1 闭包的定义**

设R是非空集合A上的关系, R 的自反 (对称或传递) 闭包是A上的关系R’，

使得 R’满足以下条件：

(1) R’是自反的（对称的或传递的）

(2) R 包含于R’

(3) 对A上任何包含R 的自反（对称或传递）关系R”有R’包含于R’’

一般将 R 的自反闭包记作 r(R), 对称闭包记作 s(R), 传递闭包记作 t(R).

**2.1.2 闭包的构造方法——矩阵表示**

设关系R, r(R), s(R), t(R)的关系矩阵分别为M, Mr, Ms和 Mt , 则   
 Mr =M+E

Ms =M+M’

Mt =M+M2+M3+…

其中E 是和 M 同阶的单位矩阵, M’是 M 的转置矩阵.

对以矩阵表示的关系，其自反闭包只要将矩阵的主对角线全部置为1，对称闭包则由关系矩阵加上其转置矩阵得到（逻辑加）。

注意：在上述等式中矩阵的元素相加时使用逻辑加。

**2.2 设计程序**

根据功能要求，先设计main函数具体实现，其中包括健壮性处理，输入判断；还包括输入提示等。

主要部分为根据选项调用相应的函数；

vector<vector<int>>matrix(n,vector<int>(n));//用来存放关系矩阵的二维vector

int choice;//选择序号

cin >> choice;

switch (choice)

{

case 1:

zifan(matrix,n);

break;

case 2:

chuandi(matrix, n);

break;

case 3:

duichen(matrix, n);

break;

case 4:

break;

default:

break;

}

1. **核心算法**

**3.1 求自反闭包算法实现**

//主对角线元素均为1

void zifan(vector<vector<int>>matrix,int n)

{

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

matrix[i][i] = 1;

output(matrix,n);

}

**3.1求自反闭包算法描述**

Matrix直接用来存放自反关系矩阵；通过循环赋值，主对角线元素为1，其余位置为原矩阵的值不变；最后调用output输出函数输出matrix矩阵。

**3.2 求传递闭包算法实现**

void chuandi(vector<vector<int>>matrix,int n)

{

vector<vector<int>>temp(n,vector<int>(n));//临时矩阵，先初始化为0

vector<vector<int>>result(n, vector<int>(n));//结果矩阵

vector<vector<int>>mi(n, vector<int>(n));//用来求幂矩阵

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

{

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

{

result[i][j]=matrix[i][j];//结果矩阵为M+M^2+M^3......+M^n

mi[i][j]=matrix[i][j];//幂矩阵初始化为1次幂，即为原矩阵

}

}

//根据有穷集合，最多不超多R^n

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

{

/\*求幂矩阵\*/

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

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

if (mi[i][j] == 1) //如果当前幂矩阵该位置为1

{

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

if (matrix[j][k] == 1)//且对应的原矩阵的第j行有位置为1

temp[i][k] = 1; //则临时矩阵存放新的幂次阵的该位置为1

}

/\*求逻辑加\*/

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

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

{

mi[i][j] = temp[i][j];

result[i][j] =result[i][j]||temp[i][j];

temp[i][j] = 0;//临时矩阵清零，便于下次存放数据

}

}

output(result,n);

}

**3.2 求传递闭包算法描述**

设变量temp存放临时矩阵，此初始化方法将其都赋值为0；result为结果矩阵，mi为求幂矩阵，初始化为原矩阵，即1次幂矩阵；

结果矩阵为各次幂矩阵进行逻辑加，通过外层循环最多加到n次幂；

第一步求幂矩阵，用上次的求出的幂矩阵再次和原矩阵相乘，增加一次幂，如果当前幂矩阵该位置i行j列为1，遍历原矩阵的第j行的所有元素，如果有为1的，则下一次幂矩阵的i行k列即为1，将结果先存放在临时矩阵里；该步将线性代数里求解矩阵相乘的过程做了一些变化，体现了传递加的思想；

第二步将求出的幂矩阵直接加进结果矩阵里，先将临时矩阵的值赋给mi矩阵，便于下次循环求下一次幂，然后进行逻辑加运算，最后将临时矩阵清零，便于下次循环使用。

循环相加n次幂之后，调用output输出函数，输出result结果矩阵。

此传统算法的时间复杂度为O（n^4）。

**3.3 求对称闭包算法实现**

void duichen(vector<vector<int>>matrix,int n)

{

vector<vector<int>>T(n,vector<int>(n));//转置矩阵

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

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

T[j][i] = matrix[i][j]; //行列交换

//矩阵相加（逻辑加）

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

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

matrix[i][j] = T[i][j] || matrix[i][j];//逻辑加，以matrix存放结果

output(matrix,n);

}

**3.3 求对称闭包算法描述**

将转置矩阵初始化为n行n列，求出原矩阵对应的转置矩阵，将行和列交换进行赋值；

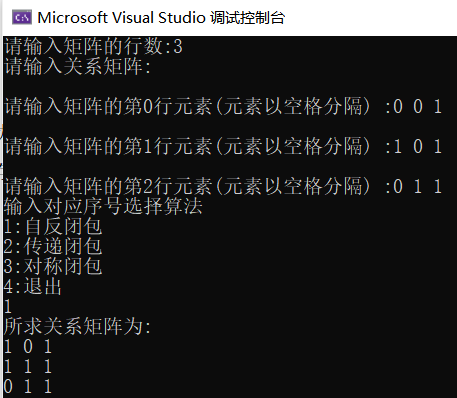
用循环遍历，将转置矩阵和原矩阵对应位置的元素进行逻辑加，用逻辑运算符||，只要其中有一个是1，得出的结果即为1，若都为0，则结果为0，将原矩阵直接作为对称闭包关系矩阵，最后调用output输出函数，输出matrix矩阵。

1. **所用数据结构**

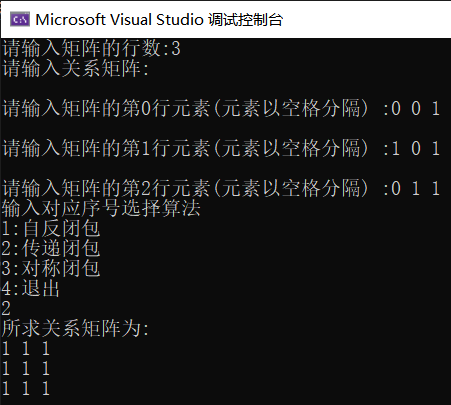
Vector数组：相当于动态开辟数组，不需要考虑数组溢出问题，不会造成空间浪费，使用方便；用来存放关系矩阵。

1. **实验结果**

**5.1 求关系的自反闭包**



**5.2 求关系的传递闭包**



**5.3 求关系的对称闭包**



1. **心得体会**

因为实验报告内的源代码存在很多问题，其中大量使用全局变量，没有注释，阅读起来比较困难，而且在求闭包的问题里，关系矩阵是方阵，行和列应该是相同的，源代码中设了两个不同的变量，这是不对的。并且源代码使用的是给定大小的数组，可能会出现溢出的问题，也存在浪费空间的问题；

使用二维vector可以避免上述数组存在的问题，也可以减少循环的次数。于是自己改写了一套C++程序来实现求闭包的过程，这样也加深了自己的理解和应用能力。在理解传递闭包的求解方法过程中，因为在纸上演算使用线性代数求矩阵相乘的方法，但是程序中稍加转变，刚开始没有看明白，最后根据给的源码去一步步分析传递加的规律。

在对闭包的定义和构造方法原理的理解的基础上，使用程序实现；除此之外，增强了程序的健壮性，加入一些输入错误处理。最后找多个输入，先在纸上进行一遍运算，然后用程序进行检验对比。