项目说明文档

《离散数学》课程实验报告

——求关系的自反、对称和传递闭包

作 者 姓 名： 杨滕超

学 号： 2151298

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 题目简介](#_Toc495668153) 4

[1.1 题目原理背景](#_Toc495668154) 4

[1.1.1 关系](#_Toc495668163) 4

[1.1.2 关系的矩阵表示](#_Toc495668163) 4

[1.1.3 闭包](#_Toc495668163) 4

[1.1.4 闭包的性质与求解](#_Toc495668163) 5

[1.2 题目概述](#_Toc495668155) 6

[1.3 程序功能](#_Toc495668155) 6

[2 解题思路](#_Toc495668156) 7

[2.1 自反闭包](#_Toc495668155) 7

[2.2 对称闭包](#_Toc495668155) 7

[2.3 传递闭包](#_Toc495668155) 7

[3 数据结构设计](#_Toc495668161) 7

[3.1 关系类](#_Toc495668162) 7

[3.2 关系类UML图 8](#_Toc495668166)

[4 核心算法](#_Toc495668161) 8

[4.1 求自反闭包](#_Toc495668174) 8

[4.1.1 主要代码](#_Toc495668163) 8

[4.1.2 算法说明](#_Toc495668163) 8

[4.2 求对称闭包](#_Toc495668174) 9

[4.2.1 主要代码](#_Toc495668163) 9

[4.2.2 算法说明](#_Toc495668163) 9

[4.3 求传递闭包](#_Toc495668174) 10

[4.3.1 主要代码](#_Toc495668163) 10

[4.3.2 算法说明](#_Toc495668163) 12

[4.4 获得矩阵的行列数](#_Toc495668174) 13

[4.4.1 主要代码](#_Toc495668163) 13

[4.4.2 算法说明](#_Toc495668163) 14

[4.5 获得矩阵内部元素的值](#_Toc495668174) 14

[4.5.1 主要代码](#_Toc495668163) 14

[4.5.2 算法说明](#_Toc495668163) 15

[4.6 选择对应算法](#_Toc495668174) 15

[4.6.1 主要代码](#_Toc495668163) 15

[4.6.2 算法说明](#_Toc495668163) 16

[5 实验结果](#_Toc495668161) 17

[5.1 样例测试](#_Toc495668174) 17

[5.2 普通测试](#_Toc495668174) 18

[5.3 仅有一个元素的集合上的关系](#_Toc495668174) 19

[5.3.1 不是空关系](#_Toc495668163) 19

[5.3.2 空关系](#_Toc495668163) 20

[5.4 特殊的关系](#_Toc495668174) 21

[5.4.1 空关系](#_Toc495668163) 21

[5.4.2 恒等关系](#_Toc495668163) 22

[5.4.3 全域关系](#_Toc495668163) 23

[5.5 行列数输入错误](#_Toc495668174) 24

[5.6 矩阵元素输入错误](#_Toc495668174) 24

[5.7 算法选择输入错误](#_Toc495668174) 24

[6 心得体会](#_Toc495668161) 25

1 题目简介

* 1. 题目原理背景

1.1.1关系

笛卡尔积A×B的任意一个子集R称为是由A到B的一个二元关系，R中任意序偶<x, y>，可记作<x, y>∈R或xRy。不在R中的任意序偶<x, y>，可以记作<x, y>R。当A = B时，我们可以称R是集合A上的二元关系。这里我们需要注意的是关系同样作为一个集合，其中的元素为序偶，即<x, y>和<y, x>是不同的。

通常集合A上的不同关系的数目依赖于A的基数，如果|A| = n，那么|A×A| = nn,A×A的子集有2n^2个，每一个子集代表A上的关系，因此A上有2n^2个不同的二元关系。

其中有几种特殊的关系，空集作为任何A×A的子集，也是A上的关系，称为空关系；全域关系A×A；恒等关系IA = {<x, x>|x∈A}。

1.1.2 关系的矩阵表示

对于由A到B的一个二元关系R来说，可以用矩阵对其进行表示。其中由A的基数|A|决定矩阵的行数，B的基数|B|决定矩阵的列数，并且我们认为地给关系中的元素编号，若关系R中存在A中的第i个元素和B中的第j个元素组成的有序对<x, y>，则相应的在矩阵中第i行第j列的位置为1，否则为0。我们需要在这里注意的是，关系的矩阵表示中只能出现0或1，这一点类似图论中的可达矩阵。

通过关系的矩阵表示，我们可以很直观地看出矩阵的相关性质。通过观察对角线是否全部为1，我们可以知道该关系是否具有自反性；通过观察该矩阵的转置是否等于其本身，我们可以知道该关系是否具有对称性；但同时，是否具有传递性可能没有办法一眼看出，但通过矩阵的量化标准，可以利用计算机中的数据结构进行表达，通过计算机求解验证是否满足传递性。从这里也可以看出，通过关系的矩阵表示，我们可以将关系很容易地在计算机中表示出来，由此对关系进行各种性质的检查与运算。

1.1.3 闭包

设R是A上的二元关系，我们希望R具有某些有用的性质，例如自反性，如果R不具由自反性，则可以通过在R中添加一部分有序对来改造R，得到性的关系R’使得R’具有自反性。但同时，我们又不希望R’和R相差太多。换句话中，我们添加的有序对要尽可能的少，满足这些要求的R’就称为R的自反闭包。对于对称闭包和传递闭包的定义类似。

对立《离散数学》课本上的定义表述为设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（对称或传递的）
2. RR’
3. 对于A上任何包含R的自反（对称或传递）关系R’’有R’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)。

1.1.4 闭包的性质与求解

每种闭包具有一些性质。

1. 自反闭包：IAr(R)
2. 对称闭包：s(R) = s-1(R)
3. 传递闭包：t(R)∘t(R)t(R)

因而对于求解闭包有以下公式

1. r(R) = RR0
2. s(R) = RR-1
3. t(R) = RR2R3…

对于Rn必然是有限的，因此存在自然数s和t，其中s≠t，满足Rs = Rt,对于计算机中，选择的算法应该是具有有穷性的，因此可以将求解传递闭包的公式进一步化简t(R) = RR2R3…Rn，以便在计算机中的表述和计算。

* 1. 题目概述

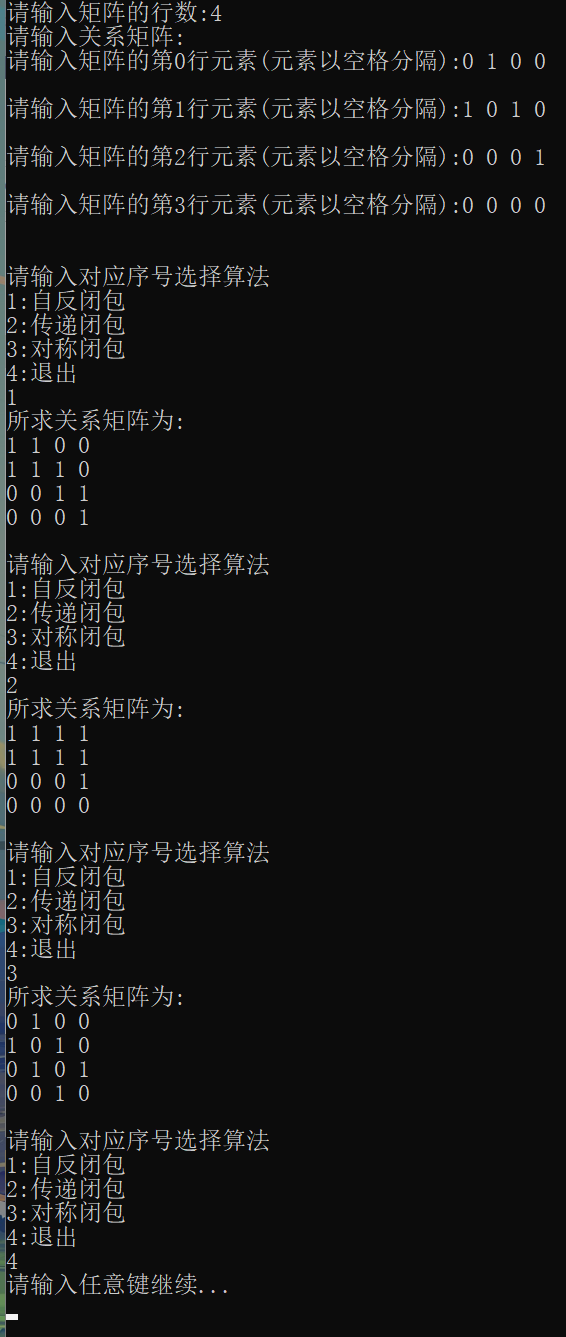
对于用户所输入的矩阵，分别进行自反闭包、对称闭包和传递闭包的求解。

* 1. 程序功能

首先请用户输入关系矩阵的行列数，然后依次输入每行的元素的值，完成关系矩阵的生成。

然后请用户选择对应的算法，以求解不同的闭包或者退出程序。

示例如下：



2 解题思路

2.1 自反闭包

根据自反闭包的求解公式r(R) = RR0，我们只需要将矩阵的主对角线全部置为1，即可以完成对上述公式的模拟。其中的复杂度为O(n)，其中n表示矩阵的行列数。

2.2 对称闭包

根据对称闭包的求解公式r(R) = RR-1，我们需要遍历整个矩阵，当查询到矩阵的第i行第j列的元素为1的时候，需要检查对称位置，即第j行第i列的元素是否也为1，若不是，需要将其改变为1，模拟添加有序对的过程。其中的时间复杂度为O(n2)，其中n表示矩阵的行列数。

2.3 传递闭包

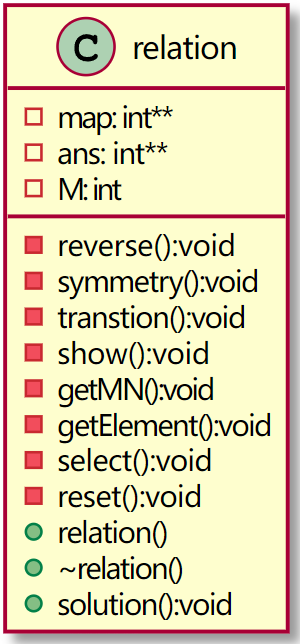
根据传递闭包的求解公式t(R) = RR2R3…Rn，我们需要对矩阵进行n次的矩阵乘法，需要注意的一点是：这里为了保证矩阵中只能出现0或1两种元素，乘法过程中使用的是逻辑加。具体过程表现为，假设两个矩阵A和B相乘得到矩阵C，C的第i行第j列的值为A的第i行的所有元素和对应的B的第j列的素有元素相乘后进行逻辑加的结果。

3 数据结构设计

3.1 关系类

根据题目要求，这里设置一个关系类，其中包括二重指针表示map和ans记录申请的二维数组的空间的首地址，分别表示用户输入的矩阵和求解得到的答案；利用一个整形变量M记录矩阵的行列数，这里需要说明，因为对于不同的A×B所表示的关系矩阵的行列数可能不相同，但是这里特指对A上的关系，只有这样子才能求解传递闭包，因此只用一个参数就可以限制。成员函数包括求解自反闭包、对称闭包、传递闭包和于用户交互的API接口，其中包括获得行列数，获得矩阵的元素的值等。

3.2 关系类UML图



4 核心算法

4.1 求自反闭包

4.1.1 主要代码

void reverse() {

//ans矩阵重置

reset();

//对角线都置为1即满足要求

//使得任意x∈A都有<x, x>

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

ans[i][i] = 1;

}

4.1.2 算法说明

将主对角线上的元素全部置为1，对所有i，使ans[i][i] = 1，相当于在原有的关系中并上IA，对所有x∈A，添加有序对<x, x>，即求得自反闭包。

4.2 求对称闭包

4.2.1 主要代码

void symmetry() {

//ans矩阵重置

reset();

//遍历整个矩阵，若对称位置为1，则该位置也为1

//使得矩阵的转置等于本身

/\*

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

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

if (map[i][j])

map[j][i] = 1;

\*/

//优化版本只遍历下三角，不考虑对角线

//减少一半的遍历，进行常数的优化

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

{

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

{

if (ans[i][j])

ans[j][i] = 1;

if (ans[j][i])

ans[i][j] = 1;

}

}

}

4.2.2 算法说明

对整个二维数组遍历，若遍历到的第i行第j列的元素值为1，则需要检查第j行第i列的值是否也为1，不为1则需要置为1，使得在关系R中存在有序对<x, y>，有序对<y, x>也应该存在，由此求解得出对称闭包。

我们对算法进行优化，只遍历一半的矩阵(沿着主对角线分隔的矩阵的上三角部分或者下三角部分)，通过对二维数组ans[i][j]和ans[j][i]分别检查是否为1，然后分别访问相应的对称位置的元素是否也为1，不为1则置为1，由此求得对称闭包。这里需要注意的是，没有必要遍历到主对角线上的元素。对于主对角线上的元素而言，若<x, x>∈R则<x, x>∈R，因此可以减少遍历的次数。由上述操作，可以有效进行常数级的优化。对于行列数较大的关系矩阵，可以在一定程度上提高时间上的效率。但仍然需要注意到的是，该算法的时间复杂度仍然是O(n2)级别。

4.3 求传递闭包

4.3.1 主要代码

void transition() {

//ans矩阵重置

reset();

//矩阵相乘需要两个临时矩阵，保存当前的R^k，和结果R^(k + 1)

//cur数组保存当前R^k

int\*\* cur = new(nothrow) int\*[M];

if (cur == NULL)

{

cerr << "申请空间失败" << endl;

exit(1);

}

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

cur[i] = new(nothrow) int[M];

if (cur[i] == NULL) {

cerr << "申请空间失败" << endl;

exit(1);

}

}

//res数组保存运算结果R^(k + 1)

int\*\* res = new(nothrow) int\* [M];

if (res == NULL)

{

cerr << "申请空间失败" << endl;

exit(1);

}

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

res[i] = new(nothrow) int[M];

if (res[i] == NULL) {

cerr << "申请空间失败" << endl;

exit(1);

}

}

//将cur赋上初始值，当前cur表示R^1

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

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

cur[i][j] = map[i][j];

//做M次矩阵相乘

for (int cnt = 0; cnt < M; ++cnt)

{

//行

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

{

//列

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

{

//临时变量保存对i行j列的运算结果

int temp = 0;

//cur[i][k]于map[k][j]与运算然后逻辑加

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

{

temp = cur[i][k] && map[k][j];

//逻辑加表现为，只要有一个是1，结果就是1

//可以提前退出

if (temp)

break;

}

res[i][j] = temp;

}

}

//更新cur

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

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

{

cur[i][j] = res[i][j];

//更新ans数组，相当于并上R^k

if (res[i][j])

ans[i][j] = 1;

}

}

//释放空间

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

delete[] cur[i];

delete[] res[i];

}

delete[] cur;

delete[] res;

}

4.3.2 算法说明

这里采用的是时间复杂度为O(n4)级别的求传递闭包的算法，其主要思想为对矩阵进行n次的乘法运算。由于进行依次矩阵乘法求得的结果矩阵中一个元素的时间复杂度为O(n)，结果数组中一共需要求n2个元素，因此进行一次矩阵乘法的时间复杂度为O(n3)，由此得出总体上的时间复杂度为O(n4)。

对于该算法而言，需要两个二维数组空间辅助，因此所需要的空间复杂度为O(n2)级别。其中一个辅助数组为cur，暂时存储计算到Rk的关系；以及另一个辅助数组res，暂时存储矩阵计算的结果，其原因在于利用原矩阵map和cur矩阵计算的时候，并不能改变这两个矩阵，其中的元素的值仍需要利用，因此在一次乘法计算完成之前，需要另一个二维数组暂时存储计算的结果。

该算法的主要过程为：将原矩阵的元素值赋给cur数组，作为R，然后进入n次循环，每次循环计算一次cur数组代表的Rk与map数组代表的R的矩阵乘法。

矩阵乘法具体过程如下：利用两层循环嵌套，i、j确定当前计算结果矩阵res[i][j]，即第i行第j列的元素。为了计算该元素的值，我们需要利用一层循环遍历cur数组第i行的每个元素和map数组第j行的每个元素。代码如下：

//临时变量保存对i行j列的运算结果

int temp = 0;

//cur[i][k]于map[k][j]与运算然后逻辑加

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

temp = cur[i][k] && map[k][j];

//逻辑加表现为，只要有一个是1，结果就是1

//可以提前退出

if (temp)

break;

}

res[i][j] = temp;

这里需要说明的是，因为这里进行的是逻辑加运算，我们没有必要将第i行第j列所有元素都遍历完全，只要有一个相乘的结果为1，逻辑加的结果就是1，由此可以直接确定res[i][j]的值，可以提前跳出循环。这是我对矩阵乘法编写程序时候的思考与优化。

每轮矩阵乘法完成后，对cur数组进行更新，将res数组的值赋给cur数组，作为下一次的Rk，同时，我们也将res数组代表的矩阵对ans数据代表的矩阵进行并运算，代码如下：

//更新cur

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

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

cur[i][j] = res[i][j];

//更新ans数组，相当于并上R^k

if (res[i][j])

ans[i][j] = 1;

}

}

4.4 获得矩阵的行列数

4.4.1 主要代码

void getMN() {

cout << "请输入矩阵的行数:";

while (1) {

cin >> M;

if (cin.fail() || M <= 0) {

cin.clear();

cin.ignore(65536, '\n');

cout << "输入有误，请重新输入正整数" << endl;

}

else

break;

}

//根据行列数为矩阵申请相应大小的空间

map = new(nothrow) int\* [M];

if (map == NULL)

{

cerr << "申请空间失败" << endl;

exit(1);

}

ans = new(nothrow) int\* [M];

if (ans == NULL)

{

cerr << "申请空间失败" << endl;

exit(1);

}

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

map[i] = new(nothrow) int[M];

if (map[i] == NULL) {

cerr << "申请空间失败" << endl;

exit(1);

}

}

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

ans[i] = new(nothrow) int[M];

if (ans[i] == NULL) {

cerr << "申请空间失败" << endl;

exit(1);

}

}

}

4.4.2 算法说明

这里由用户输入矩阵的行列数，为了使得错误输入后程序仍然能正确运行避免程序崩溃，这里需要进行错误处理。利用永真循环，判断所输入的字符不是非法输入，并且是正整数，才能跳出循环，否则输出输入错误提示用户，“输入错误，请重新输入正整数”。

利用得到的矩阵行列数，我们可以对类内数据成员的二重指针变量申请二维数组的空间。这里仍然需要对申请失败的情况做出判断，因为电脑内存等一些列因素可能导致动态申请空间不成功，因此需要在此做出判断。否则在之后对二维数组的访问过程中将访问的非法空间，导致错误。

相应地，对于动态申请的内存，我们一定不能忘记在类的析构函数中释放申请的空间，否则在程序运行结束后，将产生没有指针指向的空间，由此产生电脑的内存碎片，造成内存泄露。

4.5 获得矩阵内部元素的值

4.5.1 主要代码

void getElement() {

cout << "请输入关系矩阵:" << endl;

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

cout << "请输入矩阵的第" << i << "行元素(元素以空格分隔):";

for (int j = 0; j < M;)

{

cin >> map[i][j];

if (cin.fail() || !(map[i][j] == 0 || map[i][j] == 1))

{

cin.clear();

cin.ignore(65536, '\n');

cout << "输入应为0或1" << endl;

}

else

{

++j;

}

}

cout << endl;

}

}

4.5.2 算法说明

通过输出提示，提示用户一次输入每行矩阵元素的值。这里注意我们需要进行输入的判断，其中一点是非法输入的判断，例如这里cin输入的对象是整型变量int，但输入了字母，我们通过cin.fail()函数进行识别；其二是对于矩阵中元素值的限定，作为可达矩阵，其中的元素的值应该只能为0或1，因此我们也需要对除了上述两个数字的其他数字做出判断与提示。发现错误输入之后，利用cin.clear()和cin.ignore()两个函数进行清除错误标记与缓冲区的跳过，以便下一次输入的成功运行。

对矩阵的每一行的输入，我们逐列进行，若输入正确，列号增加；若输入错误，列号不变化，下一次进行对该位置的重新输入。

4.6 选择对应算法

4.6.1 主要代码

void select() {

char op = '\0';

bool loop = true;

while (loop)

{

cout << endl;

cout << "请输入对应序号选择算法" << 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;

cin >> op;

switch (op) {

case '1':

this->reverse();

this->show();

break;

case '2':

this->transition();

this->show();

break;

case '3':

this->symmetry();

this->show();

break;

default:

loop = false;

cin.clear();

cin.ignore(65536, '\n');

break;

}

}

}

4.6.2 算法说明

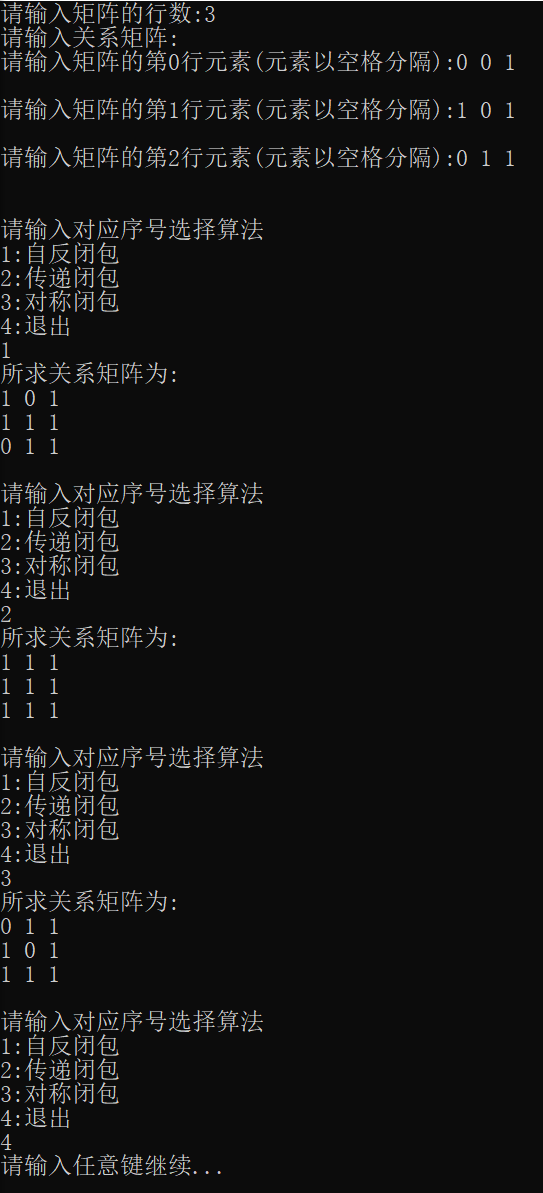
这里的算法选择函数完成了由用户输入相应的算法编号进行相应的函数的执行。

有用户输入的地方应当出现输入错误的判断。输入错误的处理与上述类似，若出现了不是预期的输入，则进行相关提示的输出以及清除错误标记和缓冲区的跳过，以方便下一次的输入。

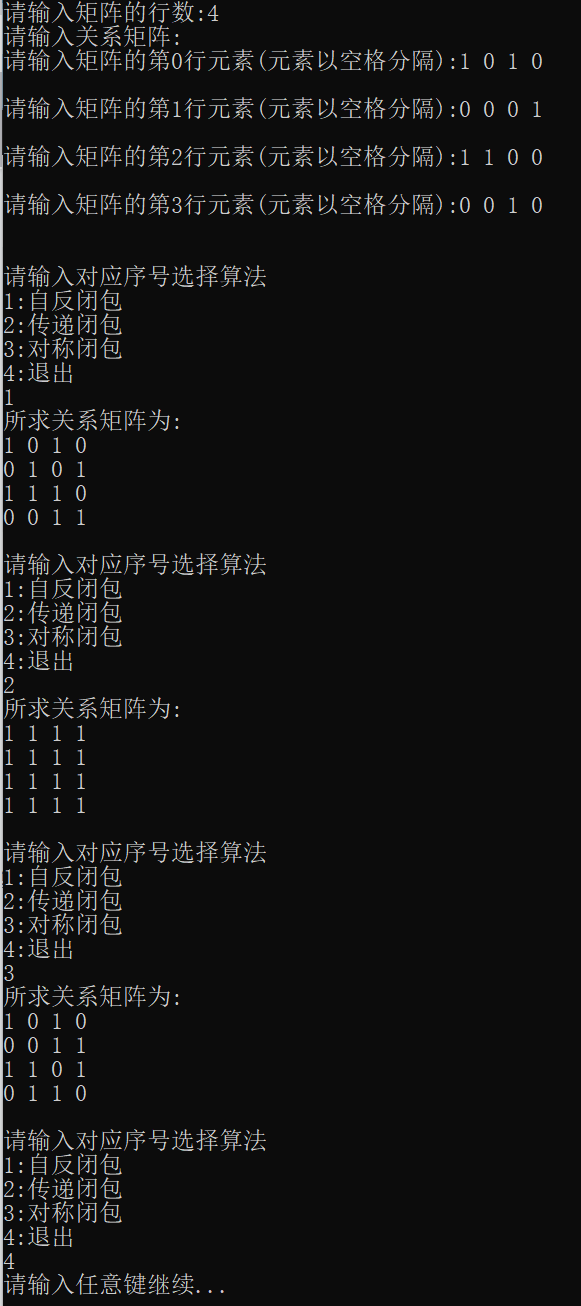
利用switch分支结果形成输入的字符与对应的算法的映射关系，执行相应的函数。需要注意的是，在程序运行的过程中，我们可以进行多次算法的选择，这里就使得我们不能改变最开始输入的矩阵的信息，因此我们需要另一个矩阵也就是二维数组暂时存储并输出进行闭包运算的结果，使得多次算法的选择可以挖成。另一个需要注意的地方是，每次进行闭包运算求解之前，都需要将临时存储结构的二维数组进行初始化，将其初始化为用户所输入的原矩阵，否则之前的闭包求解结果可能对当前闭包求解结果造成影响，导致答案的错误。

5 实验结果

5.1 样例测试

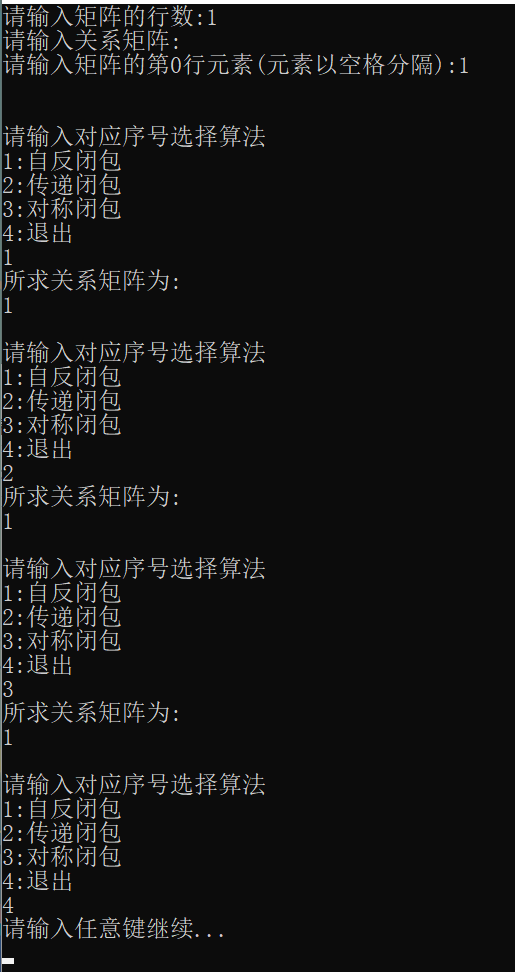


5.2 普通测试

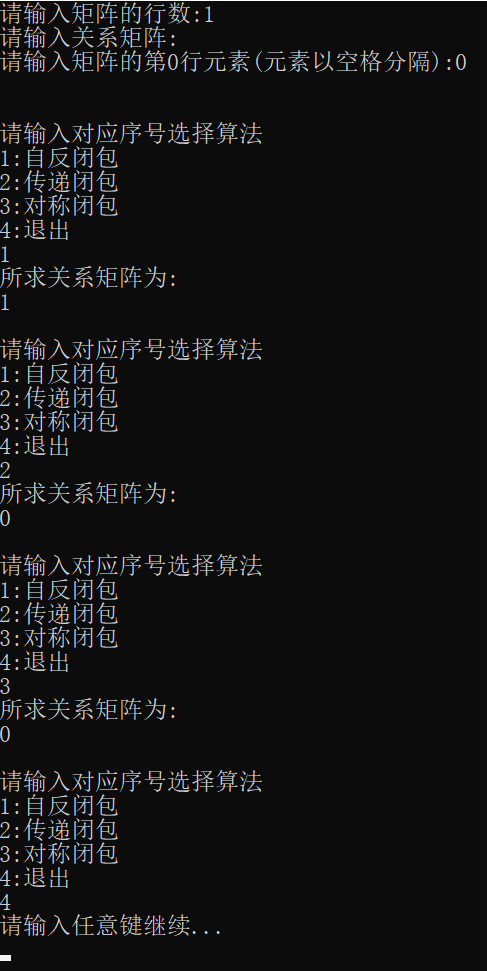


5.3 仅有一个元素的集合上的关系

5.3.1 不是空关系

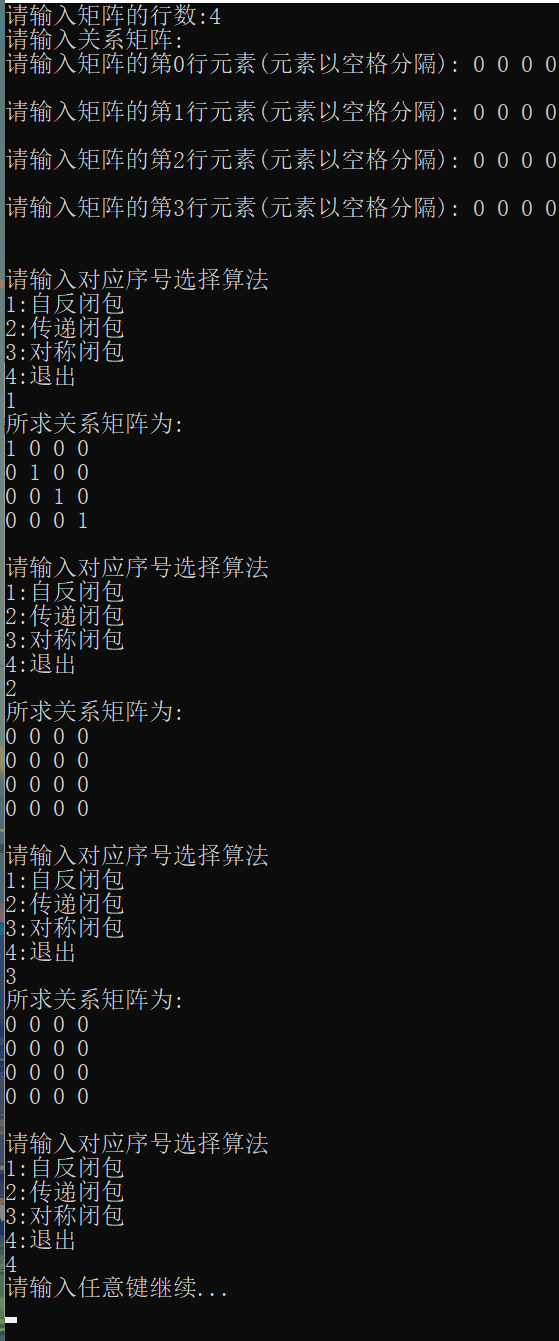


5.3.2 空关系

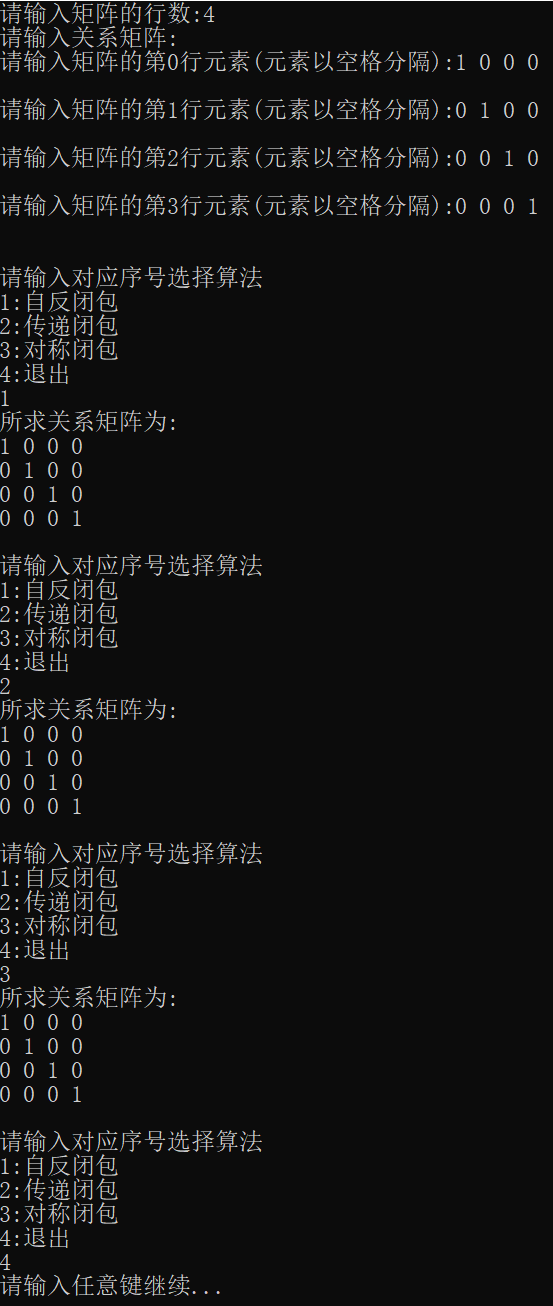


5.4 特殊的关系

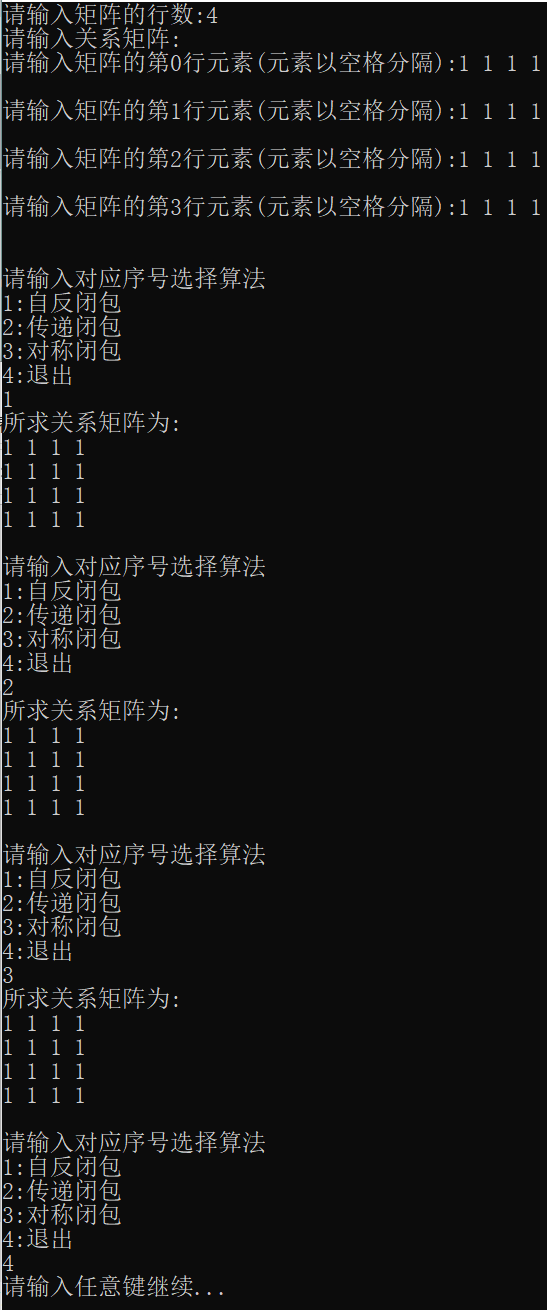
5.4.1 空关系



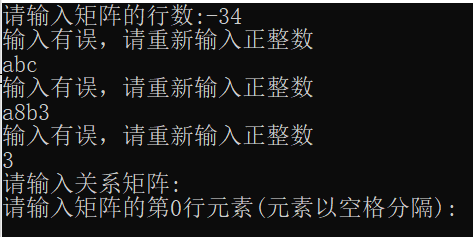
5.4.2 恒等关系



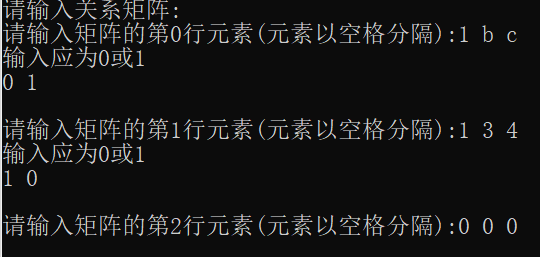
5.4.3 全域关系



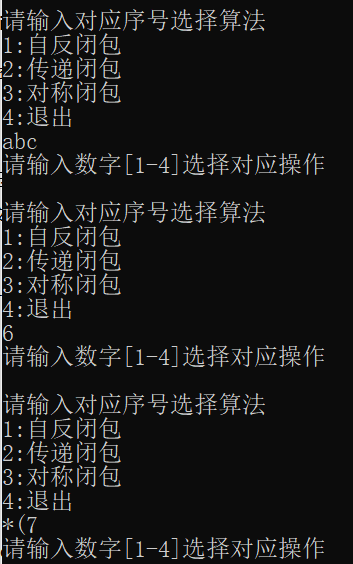
5.5 行列数输入错误



5.6 矩阵元素输入错误



5.7 算法选择输入错误



6 心得体会

通过利用程序求解关系的自反闭包、对称闭包和传递闭包，我进一步加深了对关系闭包的理解，和三个闭包所具有性质。对于一个关系，我们需要添加最少的有序对使得该关系具有自反性、对称性或传递性，因此我们在此考虑它们的充分必要条件。

对于自反闭包，有恒等关系属于其中，因此我们只需要关注还有再补充哪些有序对可以使得恒等关系属于其中即可，在矩阵中表示为主对角线上全部为１，即包含了恒等关系。

对于对称闭包，若<x, y>∈R，则应有<y, x>∈R，若发现关系矩阵中第i行第j列的元素值为1，则应检查第j行第i列的元素是否也为1，若不为1，说明不存在该关系对，不满足对称性，由此将该位置置为1，添加有序对。遍历整个矩阵，由此得到添加有序对最少的一个对称矩阵。

对于传递闭包，利用求解传递闭包的公式，进行n次矩阵乘法并求矩阵，并且对每次结果进行逻辑加，最终得到的即为添加最少有序对的具有传递性质的关系。

在编写程序的过程中，我发现对于关系的量化是基于矩阵这一个数学表现形式的，由此我们才能将《离散数学》中的关系这一个有序对的集合抽象为C++程序中的二维数组。正是通过矩阵良好的性质，才能将闭包具有个性质和求解闭包的数学方法应用于编程当中。类似矩阵的各种运算，例如：初等变换，解方程组、求行列式、特征值等等，都可以按照一定的算法去实现。由此我能明显地体会到离散数学与编程的紧密联系。

此外，对于求解闭包的程序编写的同时，我们也应该不断对算法的优化进行思考，以最大程度地提高程序的时间效率与空间效率。例如求对称闭包时候可以只遍历关系矩阵沿主对角线的下三角或者上三角，这样子可以减少一半的遍历时间，虽然时间复杂度仍然是O(n2)，但进行常数级别的优化也是一大进步。再例如求传递闭包时候的矩阵乘法，因为进行的是逻辑加，求结果矩阵中第i行第j列的元素的值时候，对表示Rk的矩阵的第i行的所有元素与表示R的矩阵的第j列的所有元素相乘然后逻辑加，其实只要相乘的结果出现了1，结果矩阵相应位置就是1，后面的元素就没有必要遍历了。由此进行常数级别的优化。

因此在编写程序的时候，我们要把生活中的具体示例通过数学模型清晰地量化表示出来，辅之以详细的分析，例如程序的时间复杂度、空间复杂度、可复用性等。通过这次实验本人的思维得到了极大的提升，脱离出离散数学的课堂，我再往后的学习科研中也将更专注于程序本身属性的提高，而不仅仅将视野局限于完成一个项目。