同濟大學
TONGJI UNIVERSITY

《离散数学》课程实验报告

题目 关系的自反、对称、传递闭包

姓　名 林继申

学　号 2250758

学　院 软件学院

专　业 软件工程

教　师 李　冰

二〇二三 年 十二 月 二十二 日

目录

1 实验目的

本实验旨在深化对离散数学中关系属性理论的理解，通过实践操作加强对自反性、对称性和传递性这三个核心关系属性的认识。实验的主要目的包括：

(1)理论与实践结合：将离散数学的理论知识与计算机编程实践相结合，以增强理论概念的应用能力；

(2)编程技能提升：通过使用 C++编程语言，提高编写高效、可读性强的代码的能力，特别是在处理数学问题方面；

(3)关系属性的理解：深入理解自反、对称和传递这三种关系属性的定义及其在数学和计算机科学中的应用；

(4)算法实现和分析：实现用于计算关系的自反闭包、对称闭包和传递闭包的算法，并对算法的效率和实用性进行分析；

(5)数据结构应用：学习如何使用适当的数据结构（如矩阵）来有效地表示和处理关系数据。

通过本实验，学生不仅能够巩固离散数学的基本概念，还能提高编程解决问题的能力，为日后在更复杂的计算问题中应用这些知识打下坚实的基础。

2 实验内容

本实验的内容是通过编程实践深入探究和实现关系的自反、对称、传递闭包的计算，这包括理解这三种关系属性的理论基础，使用矩阵作为数据结构来表示关系，设计算法来计算每种闭包，并通过一系列的测试案例来验证算法的正确性和效率，从而加深对离散数学关系属性的理解，并提升通过编程解决数学问题的能力。

3 实验环境

程序开发语言：C++

集成开发环境：Microsoft Visual Studio 2022 (Release 模式)

编译运行环境：本项目适用于 x86 架构和 x64 架构

4 实验原理

4.1 关系的自反性

自反性是一种基本的关系属性。如果集合中的每个元素都与自身存在这种关系，则一个关系被认为是自反的。假设我们有一个集合 A 和一个定义在 A 上的关系 R，如果对于 A 中的每个元素 a，(a,a)总是在关系 R 中，那么这个关系就被称为自反的。自反性的一个典型例子是大于等于关系，因为对于任何数 a，都有 a ≥a 成立。在计算机科学和数学的应用中，自反性常用于确保算法或理论中的稳定性和一致性，特别是在处理等价关系或定义序关系时。实现自反闭包的过程涉及确保关系矩阵的主对角线上的所有元素都是 1，表示每个元素与自身的关系。

4.2 关系的对称性

对称性是另一种重要的关系属性。如果一个关系 R 在集合 A 上是对称的，那么对于 A 中的任意两个元素 a 和 b，如果(a,b)在 R 中，则(b,a)也在 R 中。换句话说，如果一个元素与另一个元素有关系，则反过来也成立。在算法设计中，对称性经常用于简化问题，减少需要处理的数据量。例如，在无向图中，边的关系是对称的。计算对称闭包的过程包括检查关系矩阵，并确保对于每个元素(i,j)，其反向元素(j,i)也存在。

4.3 关系的传递性

传递性是关系的第三种核心属性。一个关系 R 是传递的，如果对于任意三个元素 a、b 和 c，只要(a,b)和(b,c)都在 R 中，那么(a,c)也必须在 R 中。传递性在许多数学和计算领域中都非常重要，特别是在定义序关系和等价关系时。例如，如果我们知道 A 大于 B，B 大于 C，那么根据传递性，我们可以得出 A 大于 C。在计算关系的传递闭包时，目标是确保如果有一个通过中间步骤可以到达的路径，则直接路径也应该存在，这通常涉及到复杂的算法计算，如使用 Warshall 算法，以确保关系矩阵能够完全反映传递性。

5 实验过程

5.1 实验思路

本实验的主要思路是设计并实现一个程序，用于计算任意给定关系的自反闭

包、对称闭包和传递闭包。通过这个过程，实验旨在结合离散数学的理论知识和实际编程技能。首先，利用矩阵这一数据结构来表示和存储关系。接着，分别实现三个函数来计算关系的各种闭包。自反闭包确保矩阵的主对角线上所有元素为1，对称闭包在矩阵中为每个存在的关系创建其对称对应关系，而传递闭包通过矩阵的幂运算确保关系的传递性。最后，程序提供了用户友好的界面来输入关系矩阵，并展示闭包计算的结果。

5.2 实验设计

5.2.1 数据结构设计

由于这个程序主要涉及基本的逻辑运算，因此不需要复杂的数据结构。主要使用 Matrix 类型（一个二维整数向量）来表示关系矩阵。

```
typedef std::vector<std::vector<int>> Matrix;
```

5.2.2 结构体与类设计

在这个实验中，没有使用结构体或类，因为程序的规模和复杂性不需要这样的封装。

5.2.3 程序主体架构设计

程序主体架构设计为：

⑴初始化和界面提示：main 函数首先设置了一个用户界面提示，使用 system("cls") 清屏，然后显示程序标题和说明，为用户提供清晰的开始界面；

⑵输入关系矩阵：程序通过 inputInteger 函数询问用户关系矩阵的大小，确保用户输入有效的矩阵尺寸。使用 inputLogicalValue 函数循环接收用户输入的关系矩阵元素，构建矩阵。这一步涉及用户与程序的直接交互，考虑到了易用性和输入效率；

⑶闭包计算与展示：分别调用 reflexiveClosure、symmetricClosure 和 transitiveClosure 函数计算对应的闭包。这些函数独立于 main 函数，体现了程序的模块化设计。计算完成后，使用重载的 operator<<函数打印出原始矩阵和各闭包矩阵的结果，使输出格式化和易于理解；

⑷交互循环与退出选项：在一个 do-while 循环中，main 函数控制着整个程序的运行流程。在每次循环结束时，询问用户是否继续或退出程序。这种设计使用户能够在完成一次闭包计算后选择继续使用程序或退出，增强了程序的交互性和用户体验。

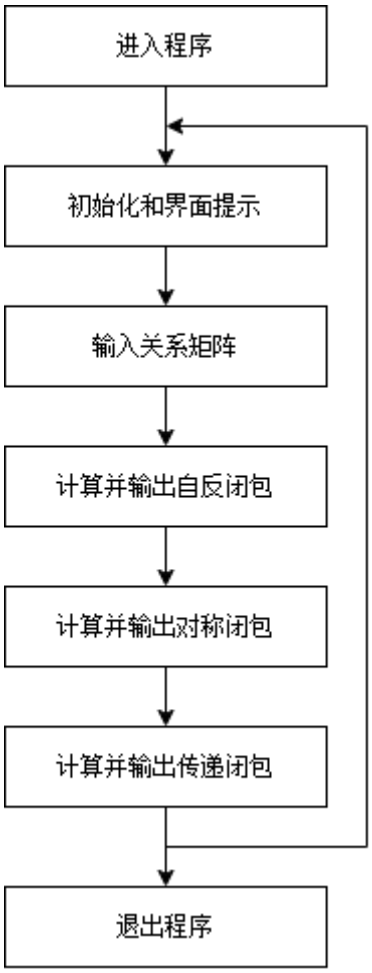

图 5.2.3.1 程序主体架构设计流程图

5.3 程序功能实现

5.3.1 逻辑值输入功能的实现

inputLogicalValue 函数用于输入逻辑值，专门用于处理二进制（真/假）值的用户输入，在这里分别用'0'和'1'或者通过参数 falseValue 和 trueValue 指定的其他字符来表示。

程序功能实现思路：

(1)falseValue 表示假的值，默认为'0'。trueValue 表示真的值，默认为'1'；

(2)该函数使用一个无限循环（while(true)），等待用户输入。使用_getch()函数从键盘获取一个字符，而不回显到控制台。首先判断是否是特殊键（如方向键），这些键的 ASCII 值通常为 0 或-32。如果是，则再次调用_getch()来获取实际的键值；

(3)接下来判断输入的字符是否与 falseValue 或 trueValue 相匹配。如果

匹配，将输入的字符打印到控制台，并根据输入的值返回 false 或 true。

5.3.2 输入关系矩阵大小功能的实现

程序通过调用 inputInteger 函数输入关系矩阵大小。inputInteger 函数用于获取用户输入的整数，同时限制输入必须在指定的范围内。inputInteger 函数对输入非法的情况进行了处理，代码具体执行逻辑如下：

⑴进入一个无限循环，它会一直运行直到用户提供有效的输入；

⑵用户的输入被读取到 tempInput 变量中，这里采用 double 类型来接收输入以便后续检查；

⑶进行输入验证：std::cin.good()检查输入流的状态是否正常，确保没有发生数据类型输入错误，tempInput==static_cast<int>(tempInput)检查用户输入是否为整数，通过将其转换为整数再比较，tempInput>=lowerLimit 和 tempInput<=upperLimit 确保输入在指定的范围内；

⑷合法输入处理：如果用户提供了合法的输入，函数会清除输入流的错误状态，丢弃输入缓冲区中的任何剩余内容，然后返回转换后的整数值；

⑸非法输入处理：如果用户提供的输入不合法，函数会输出错误消息，清除输入流的错误状态，丢弃输入缓冲区中的内容，并继续循环以等待用户提供合法的输入。

5.3.3 输出关系矩阵与关系集合功能的实现

输出关系矩阵与关系集合功能是通过重载 std::ostream& operator<<函数实现的。程序功能实现思路为：

⑴输出关系矩阵

①函数首先获取矩阵的大小（size），这是矩阵的行数和列数；

②使用两个嵌套的 for 循环遍历矩阵的每个元素；

③在外层循环的开始，判断是否是第一行来决定输出"Matrix"或者缩进，以格式化矩阵的显示；

④内层循环输出矩阵的每个元素，元素之间用逗号分隔；

⑤每行输出完成后，根据是否是最后一行输出换行和闭括号，完成矩阵的格式化显示。

⑵输出关系集合

①在矩阵输出完毕后，函数输出一个换行，接着开始输出关系集合；

②使用一个布尔变量 first 来追踪是否是集合中的第一个元素。这是为了格式化输出，确保在元素之间正确地添加逗号分隔符；

③再次遍历矩阵，对于每个值为 1 的元素，将其坐标转换为关系对。例如，如果 matrix[i][j]为 1，则输出为<a+i, a+j>形式的关系对；

④使用字符转换 char('a'+i)和 char('a'+j)来将矩阵的索引转换为字母标签，增加了输出的可读性。

5.3.4 退出程序功能的实现

退出程序的功能是通过在 main 函数内部的一个循环结构实现的。此功能允许用户在完成操作后选择是否退出程序。

程序功能实现思路：

⑴程序使用 do-while 循环来反复执行程序和询问用户是否退出程序。在每次循环的开始，使用 system("cls")清除屏幕，以提供清晰的界面；

⑵在完成一轮操作后，程序会输出询问用户是否退出程序的提示：“是否退出程序［y/n]：”。此时，程序再次调用 inputLogicalValue 函数，这次用于接收用户的退出决定。参数被设置为'n'（不退出）和'y'（退出）；

⑶如果用户输入'y'（对应真值），则 inputLogicalValue 函数返回 true，导致 do-while 循环结束，程序随之退出。如果用户输入'n'（对应假值），inputLogicalValue 函数返回 false，do-while 循环继续，用户可以进行新一轮操作。

5.4 核心算法实现

5.4.1 自反闭包计算算法的实现

自反闭包的计算是通过 reflexiveClosure 函数实现的。该函数的核心目的是确保在给定的关系矩阵中，每个元素都与自己存在关系。具体实现步骤如下：

⑴初始化矩阵：函数创建一个与输入矩阵相同大小的新矩阵 mat，用于存储闭包结果；

⑵遍历矩阵主对角线：函数遍历矩阵的主对角线（即索引为(i,i)的元素）；

⑶设置自反性：对于每个遍历到的元素，将其值设为 1，从而确保每个元素与自己存在关系；

⑷返回结果：函数返回修改后的矩阵，即关系的自反闭包。

5.4.2 对称闭包计算算法的实现

对称闭包的计算通过 symmetricClosure 函数完成。此函数保证如果元素(i,j)在关系中，则(j,i)也在关系中。具体实现步骤如下：

⑴初始化矩阵：函数复制原始矩阵到新矩阵 mat；

⑵遍历矩阵元素：使用双层循环遍历矩阵的每个元素；

⑶设置对称性：对于每个值为 1 的元素(i,j)，设置其对称位置(j,i)的值也为 1；

(4)返回结果：最终返回修改后的矩阵，即关系的对称闭包。

5.4.3 传递闭包计算算法的实现

传递闭包的计算通过传统算法实现，由 `transitiveClosure` 函数和辅助的 `multiplyMatrices` 函数共同完成，使用矩阵的幂运算来确保传递性。具体实现步骤如下：

(1)初始化矩阵：函数创建一个与输入矩阵相同大小的新矩阵 `mat`，用于存储传递闭包结果；

(2)计算传递闭包：通过计算矩阵从第 2 次幂到第 n 次幂（n 是矩阵的大小）来构建传递闭包。每计算一次矩阵幂，就将其结果与当前的传递闭包矩阵进行逻辑加运算。这里的逻辑加运算是指如果闭包矩阵或矩阵幂中的相应元素之一为 1，则结果矩阵中的相应元素设为 1；

(3)返回结果：函数返回修改后的矩阵，即关系的传递闭包。

6 实验数据分析

```
+------------------------------------------------------------+
|                关系的自反、对称、传递闭包                  |
|  Reflexive, Symmetric and Transitive Closures of a Relation |
+------------------------------------------------------------+

请输入关系矩阵大小 [整数范围: 2~26]: 1
>>> 关系矩阵大小输入不合法，请重新输入关系矩阵大小!
请输入关系矩阵大小 [整数范围: 2~26]: 27
>>> 关系矩阵大小输入不合法，请重新输入关系矩阵大小!
请输入关系矩阵大小 [整数范围: 2~26]: 2.5
>>> 关系矩阵大小输入不合法，请重新输入关系矩阵大小!
请输入关系矩阵大小 [整数范围: 2~26]: a
>>> 关系矩阵大小输入不合法，请重新输入关系矩阵大小!
请输入关系矩阵大小 [整数范围: 2~26]: abc
>>> 关系矩阵大小输入不合法，请重新输入关系矩阵大小!
请输入关系矩阵大小 [整数范围: 2~26]: 中文输入测试
>>> 关系矩阵大小输入不合法，请重新输入关系矩阵大小!
```

图 6.1 实验数据 1

实验数据分析 1：分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理，并要求用户重新输入关系矩阵大小。

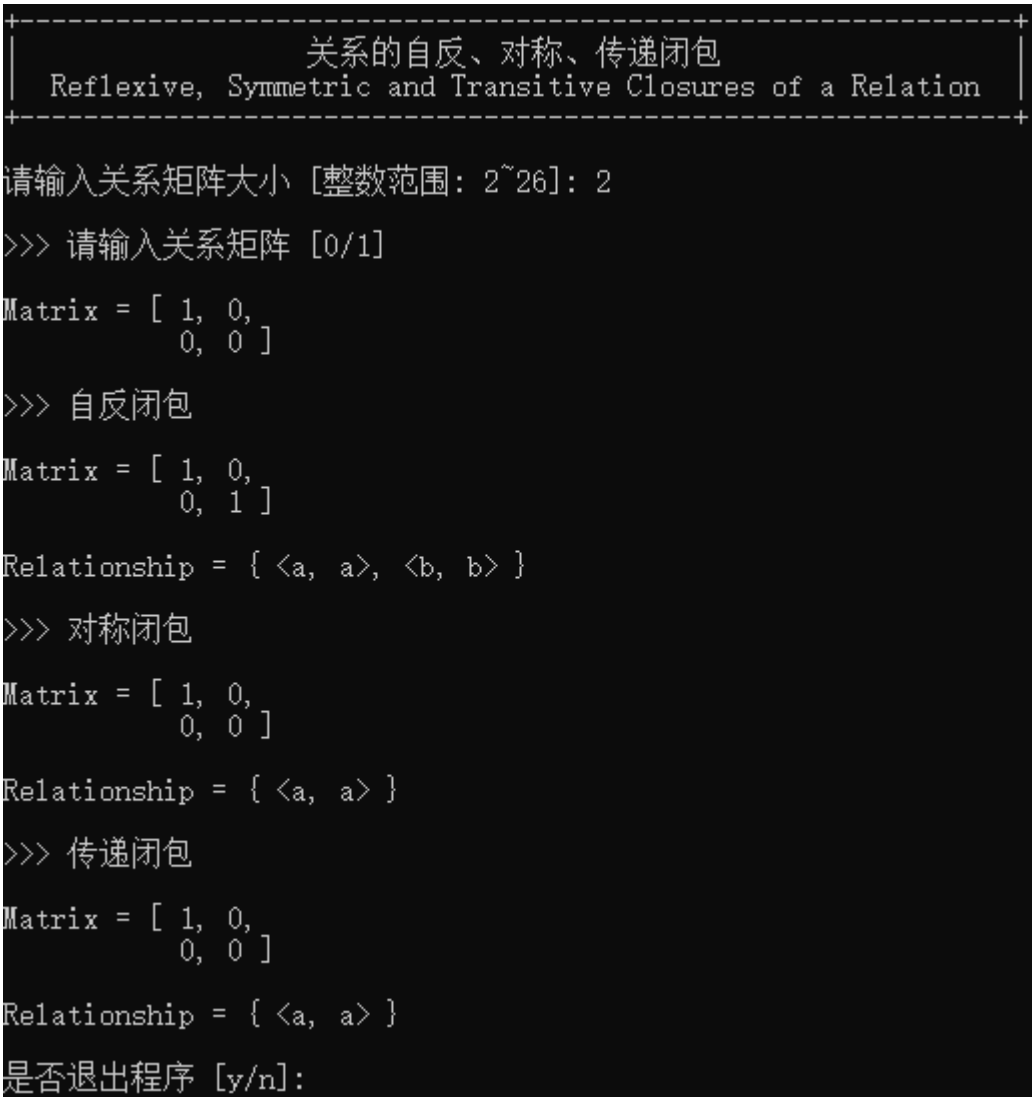

图 6.2 实验数据 2

实验数据分析 2：关系矩阵大小为 2，程序输出正确的关系矩阵和关系集合。

关系矩阵：$\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}$

自反闭包：$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$

对称闭包：$\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}$

传递闭包：$\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}$

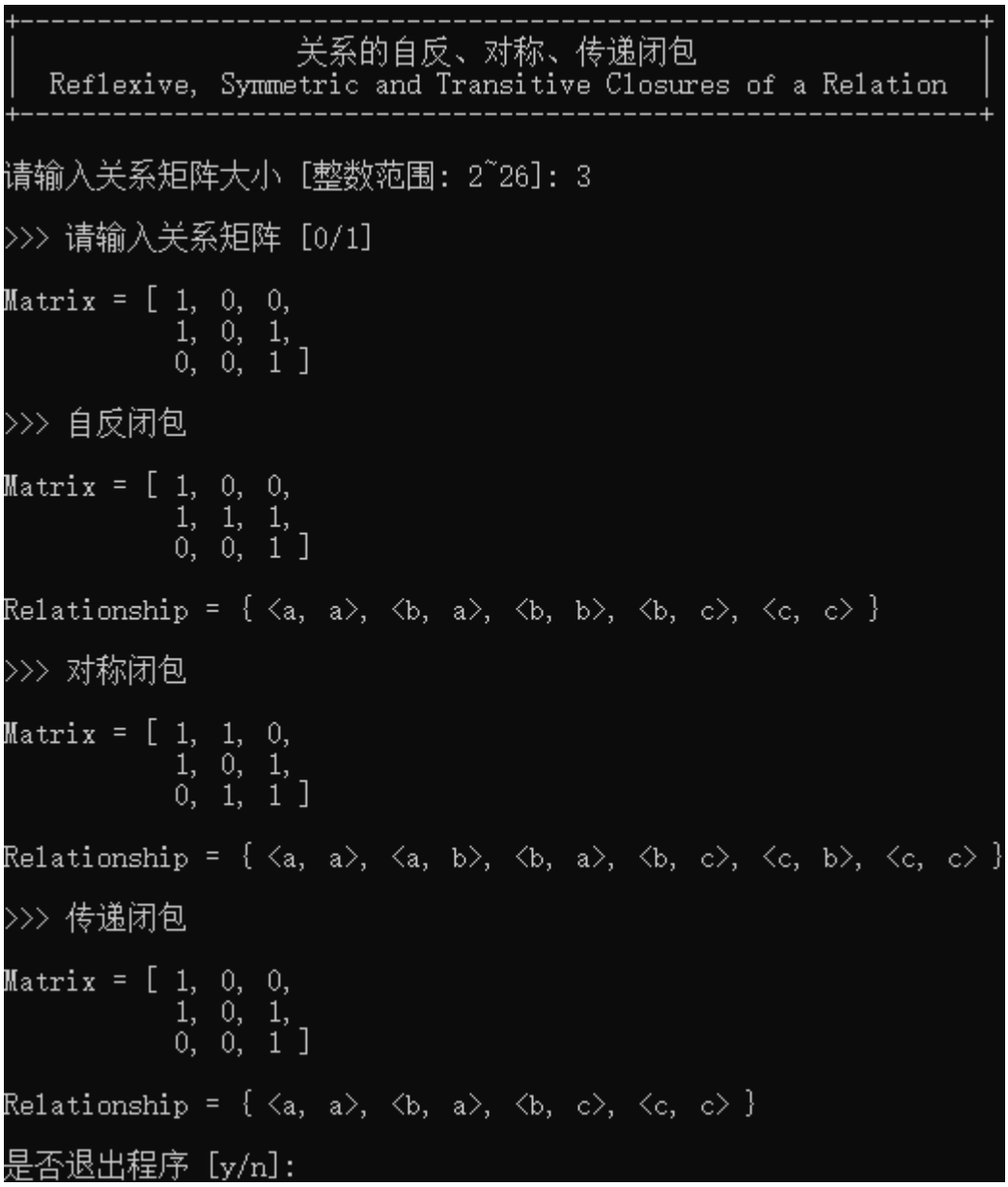

图 6.3 实验数据 3

实验数据分析 3：关系矩阵大小为 3，程序输出正确的关系矩阵和关系集合。

关系矩阵：$\begin{bmatrix}1 & 0 & 0\\1 & 0 & 1\\0 & 0 & 1\end{bmatrix}$

自反闭包：$\begin{bmatrix}1 & 0 & 0\\1 & 1 & 1\\0 & 0 & 1\end{bmatrix}$

对称闭包：$\begin{bmatrix}1 & 1 & 0\\1 & 0 & 1\\0 & 1 & 1\end{bmatrix}$

传递闭包：$\begin{bmatrix}1 & 0 & 0\\1 & 0 & 1\\0 & 0 & 1\end{bmatrix}$

```
请输入关系矩阵大小 [整数范围: 2~26]: 4

>>> 请输入关系矩阵 [0/1]

Matrix = [ 1, 0, 1, 0,
           1, 0, 0, 0,
           0, 0, 1, 0,
           0, 0, 0, 0 ]

>>> 自反闭包

Matrix = [ 1, 0, 1, 0,
           1, 1, 0, 0,
           0, 0, 1, 0,
           0, 0, 0, 1 ]

Relationship = { <a, a>, <a, c>, <b, a>, <b, b>, <c, c>, <d, d> }

>>> 对称闭包

Matrix = [ 1, 1, 1, 0,
           1, 0, 0, 0,
           1, 0, 1, 0,
           0, 0, 0, 0 ]

Relationship = { <a, a>, <a, b>, <a, c>, <b, a>, <c, a>, <c, c> }

>>> 传递闭包

Matrix = [ 1, 0, 1, 0,
           1, 0, 1, 0,
           0, 0, 1, 0,
           0, 0, 0, 0 ]

Relationship = { <a, a>, <a, c>, <b, a>, <b, c>, <c, c> }

是否退出程序 [y/n]:
```

图 6.4 实验数据 4

实验数据分析 4：关系矩阵大小为 4，程序输出正确的关系矩阵和关系集合。

关系矩阵：$\begin{bmatrix}1 & 0 & 1 & 0\\1 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 0\end{bmatrix}$

自反闭包：$\begin{bmatrix}1 & 0 & 1 & 0\\1 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$

对称闭包：$\begin{bmatrix}1 & 1 & 1 & 0\\1 & 0 & 0 & 0\\1 & 0 & 1 & 0\\0 & 0 & 0 & 0\end{bmatrix}$

传递闭包：$\begin{bmatrix}1 & 0 & 1 & 0\\1 & 0 & 1 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 0\end{bmatrix}$

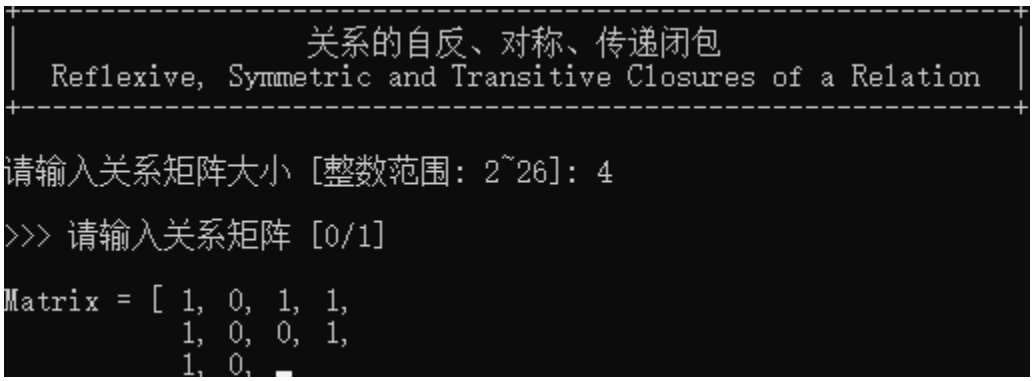

图 6.5 实验数据 5

实验数据分析 5：在输入关系矩阵的逻辑值时只能输入字符 0 或 1，若输入其他字符则不回显，光标闪烁直到输入字符 0 或 1。

图 6.6 实验数据 6

实验数据分析 6：当输入 n 时，程序清空屏幕并执行下一次关系的自反、对称、传递闭包运算。

图 6.7 实验数据 7

实验数据分析 7：当输入 y 时，程序退出。

7 实验心得

这次实验围绕关系的自反性、对称性和传递性闭包，让我深刻体会到了理论与实践相结合的重要性。通过编写和测试程序，我不仅加深了对自反性、对称性和传递性这三个关系属性的理解，还提升了我的编程技能，尤其是在处理数学问题方面的能力。

通过实现矩阵的逻辑输入功能和矩阵大小输入功能，我学习到了如何在程序中处理用户的输入，并确保输入数据的有效性和准确性。这不仅锻炼了我的编程技巧，也提高了我的逻辑思考能力。

我在实现自反闭包、对称闭包和传递闭包的计算中，深刻体会到了算法设计

的重要性。特别是在实现传递闭包时，我通过矩阵的幂运算来确保关系的传递性，这个过程让我意识到了算法在解决实际问题中的重要作用。

通过这次实验，我也了解到了如何使用 C++编程语言中的数据结构和算法来解决实际问题。我学习到了如何有效地使用矩阵来表示和处理关系数据，这在未来的编程和数学问题中将非常有用。

总的来说，这次实验不仅加深了我对离散数学理论知识的理解，还提升了我的编程能力。通过实践中的学习和解决问题，我更加深刻地认识到理论知识的重要性和实际应用的能力。我期待在未来能将这些知识和技能应用到更复杂的问题中，以进一步提高自己的能力。

8 程序源文件

```
/*****************************************************************
 * Project Name:  Assignment_3
 * File Name:     assignment_3.cpp
 * File Function: 关系的自反、对称、传递闭包
 * Author:        Jishen Lin (林继申)
 * Update Date:   2023/12/22
 *****************************************************************/

#include <iostream>
#include <vector>
#include <conio.h>
#include <limits>

/* Define Matrix type */
typedef std::vector<std::vector<int>> Matrix;

/*
 * Function Name:    reflexiveClosure
 * Function:         Calculate reflexive closure
 * Input Parameters: const Matrix matrix
 * Return Value:     reflexive closure
 */
Matrix reflexiveClosure(const Matrix matrix)
{
    Matrix mat(matrix);
    size_t size = mat.size();
    for (size_t i = 0; i < size; i++)
        mat[i][i] = 1;
```

```
    return mat;
}

/*
 * Function Name:    symmetricClosure
 * Function:         Calculate symmetric closure
 * Input Parameters: const Matrix matrix
 * Return Value:     symmetric closure
 */
Matrix symmetricClosure(const Matrix matrix)
{
    Matrix mat(matrix);
    size_t size = mat.size();
    for (size_t i = 0; i < size; i++)
        for (size_t j = 0; j < size; j++)
            if (mat[i][j])
                mat[j][i] = 1;
    return mat;
}

/*
 * Function Name:    multiplyMatrices
 * Function:         Multiply matrices
 * Input Parameters: const Matrix& a
 *                   const Matrix& b
 * Return Value:     result
 */
Matrix multiplyMatrices(const Matrix& a, const Matrix& b)
{
    size_t n = a.size();
    Matrix result(n, std::vector<int>(n, 0));
    for (size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < n; j++) {
            for (size_t k = 0; k < n; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return result;
}

/*
 * Function Name:    transitiveClosure
 * Function:         Calculate transitive closure
```

```
 * Input Parameters: const Matrix matrix
 * Return Value:     transitive closure
 */
Matrix transitiveClosure(const Matrix matrix)
{
    Matrix mat(matrix);
    size_t n = matrix.size();
    for (size_t m = 2; m <= n; m++) {
        Matrix matPower = multiplyMatrices(mat, matrix);
        for (size_t i = 0; i < n; i++) {
            for (size_t j = 0; j < n; j++) {
                mat[i][j] = static_cast<int>(matPower[i][j] > 0);
            }
        }
    }
    return mat;
}

/*
 * Function Name:    operator<<
 * Function:         Overload operator <<
 * Input Parameters: std::ostream& out
 *                   const Matrix& matrix
 * Return Value:     out
 */
std::ostream& operator<<(std::ostream& out, const Matrix& matrix)
{
    size_t size = matrix.size();
    for (size_t i = 0; i < size; i++) {
        out << (i == 0 ? "Matrix = [" : "         ");
        for (size_t j = 0; j < size; j++) {
            out << " " << matrix[i][j];
            if (j < size - 1)
                out << ",";
        }
        out << (i < size - 1 ? ",\n " : " ]\n");
    }
    out << std::endl << "Relationship = { ";
    bool first = true;
    for (size_t i = 0; i < size; i++)
        for (size_t j = 0; j < size; j++)
            if (matrix[i][j]) {
                if (first)
                    first = false;
```

```
                else
                    out << ", ";
                out << "<" << char('a' + i) << ", " << char('a' + j) <<
">";
            }
    out << " }" << std::endl;
    return out;
}

/*
 * Function Name:    inputLogicalValue
 * Function:         Input logical value
 * Input Parameters: char falseValue
 *                   char trueValue
 * Return Value:     true / false
 */
bool inputLogicalValue(char falseValue = '0', char trueValue = '1')
{
    while (true) {
        char optn = _getch();
        if (optn == 0 || optn == -32)
            optn = _getch();
        else if (optn == falseValue || optn == trueValue) {
            std::cout << optn;
            return optn == falseValue ? false : true;
        }
    }
}

/*
 * Function Name:    inputInteger
 * Function:         Input an integer
 * Input Parameters: int lowerLimit
 *                   int upperLimit
 *                   const char* prompt
 * Return Value:     an integer
 */
int inputInteger(int lowerLimit, int upperLimit, const char* prompt)
{
    while (true) {
        std::cout << "请输入" << prompt << " [整数范围: " << lowerLimit
<< "~" << upperLimit << "]: ";
        double tempInput;
        std::cin >> tempInput;
```

```
        if (std::cin.good() && tempInput == static_cast<int>(tempInput)
&& tempInput >= lowerLimit && tempInput <= upperLimit) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            return static_cast<int>(tempInput);
        }
        else {
            std::cerr << std::endl << ">>> " << prompt << "输入不合法，请
重新输入" << prompt << "! " << std::endl << std::endl;
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
        }
    }
}

/*
 * Function Name:    main
 * Function:         Main function
 * Return Value:     0
 */
int main()
{
    do {
        /* System entry prompt */
        system("cls");
        std::cout << "+-------------------------------------------------
-------------+" << std::endl;
        std::cout << "|                    关系的自反、对称、传递闭
包                  |" << std::endl;
        std::cout << "|  Reflexive, Symmetric and Transitive Closures of
a Relation  |" << std::endl;
        std::cout << "+-------------------------------------------------
-------------+" << std::endl << std::endl;

        /* Input matrix */
        int size = inputInteger(2, 26, "关系矩阵大小");
        Matrix matrix(size, std::vector<int>(size));
        std::cout << std::endl << ">>> 请输入关系矩阵 [0/1]" << std::endl
<< std::endl;
        for (int i = 0; i < size; i++) {
            std::cout << (i == 0 ? "Matrix = [" : "         ");
            for (int j = 0; j < size; j++) {
```

```
                std::cout << " ";
                matrix[i][j] = (inputLogicalValue() ? 1 : 0);
                if (j < size - 1)
                    std::cout << ",";
            }
            std::cout << (i < size - 1 ? ",\n " : " ]\n");
        }

        /* Reflexive closure */
        Matrix reflexive = reflexiveClosure(matrix);
        std::cout << std::endl << ">>> 自反闭包" << std::endl <<
std::endl << reflexive << std::endl;

        /* Symmetric closure */
        Matrix symmetric = symmetricClosure(matrix);
        std::cout << ">>> 对称闭包" << std::endl << std::endl <<
symmetric << std::endl;

        /* Transitive closure */
        Matrix transitive = transitiveClosure(matrix);
        std::cout << ">>> 传递闭包" << std::endl << std::endl <<
transitive << std::endl;

        /* Whether to exit the program */
        std::cout << "是否退出程序 [y/n]: ";
    } while (!inputLogicalValue('n', 'y'));
    return 0;
}
```