同济大学
TONGJI UNIVERSITY

《离散数学》课程实验报告

题目 Warshall 算法求解传递闭包

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传递性的认识，并实现通过 Warshall 算法求解传递闭包。实验的主要目的包括：

(1)理解和掌握传递闭包的概念

通过 Warshall 算法，深入了解传递闭包的理论基础，学习如何在实际问题中识别并应用传递闭包；

(2)掌握 Warshall 算法的原理和应用

研究 Warshall 算法的算法结构和逻辑流程，学习如何将算法应用于计算给定关系的传递闭包；

(3)提高编程能力和逻辑思维

通过实际编码实践，提高 C++编程技能，培养解决复杂问题的逻辑思维和分析能力；

(4)理解数据结构在算法中的应用

学习如何有效地使用数据结构来存储和处理信息，理解数据结构选择对算法性能的影响。

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传递闭包的计算，这包括理解传递性的理论基础，使用矩阵作为数据结构来表示关系，使用 C++实现 Warshall 算法来计算传递闭包，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 关系的传递性

关系的传递性是关系的核心属性之一。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。

4.2 Warshall 算法

Warshall 算法，由 Stephen Warshall 在 1962 年提出，是一种高效算法，用于计算有向图的传递闭包。它是理解和实现计算机科学中的关系闭包概念的关键方法，尤其在数据分析、图论和算法设计领域具有重要应用。

Warshall 算法基于动态规划思想。它逐步构建传递闭包，通过不断迭代更新矩阵来达到最终目标。算法的核心在于检查所有顶点对，判断是否存在通过中间顶点连接的路径。

Warshall 算法优势：

(1)简洁性：Warshall 算法以其简洁的逻辑和易于实现的特点著称；

(2)高效性：算法的时间复杂度为 $O(n^3)$，其中 n 是图中顶点的数量；

(3)适用性：适用于任何大小的有向图。

Warshall 算法应用场景：

(1)数据库管理系统：在处理复杂的查询优化问题时，Warshall 算法被用于分析和优化数据关系；

(2)网络路由：在计算从一个网络节点到另一个节点的所有可能路径时使用；

(3)社交网络分析：用于分析用户之间的连接路径，如朋友推荐系统；

(4)编译原理：在编译器设计中，用于分析程序中变量的使用和定义之间的关系。

Warshall 算法局限性：

虽然 Warshall 算法在多种场景下非常有效，但它在处理大规模数据时可能会受到内存使用和运算时间的限制。对于大型图，算法可能需要大量的内存和计算资源。

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，使用 C++实现 Warshall 算法，实现函数来计算关系的传递闭包。最后，程序提供了用户友好的界面来输入关系矩阵，并展示传递闭包计算的结果。

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：

(1)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；

(2)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；

(3)传递闭包计算与展示：调用 transitiveClosure 函数计算对应传递闭包。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传递闭包矩阵的结果，使输出格式化和易于理解；

(4)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。

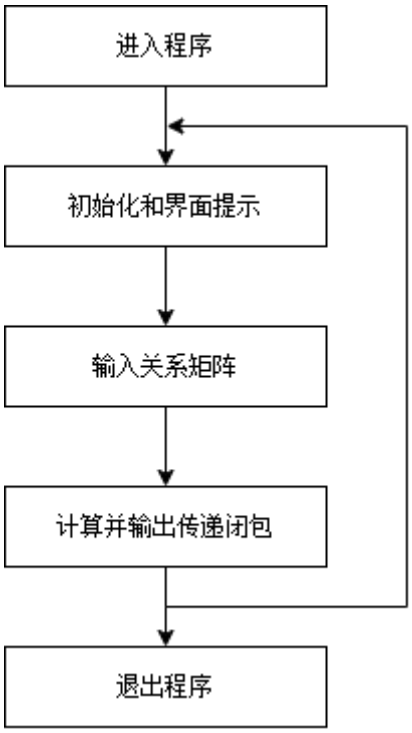

图 5.2.3.1 程序主体架构设计流程图

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：

(1)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；

(2)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）；

(3)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

传递闭包的计算通过 Warshall 算法实现，由 transitiveClosure 函数完成，具体实现步骤如下：

(1)初始化和拷贝原始矩阵：这一步创建了一个名为 mat 的新矩阵，它是输入矩阵的副本。这样做是为了不修改原始矩阵，同时在新矩阵上进行操作；

(2)获取矩阵大小：获取矩阵的大小（即顶点的数量），用于控制循环迭代的次数；

(3)三层循环迭代更新：这个三重循环是 Warshall 算法的核心。它遍历了矩阵中的每个元素，并进行如下更新：

①外层循环（变量 k）：遍历所有可能的中间顶点；

②中层循环（变量 i）：遍历所有可能的起始顶点；

③内层循环（变量 j）：遍历所有可能的结束顶点；

④矩阵更新：如果 mat[i][j]是真（1），或者存在一个通过顶点 k 的路径从 i 到 j（即 mat[i][k]和 mat[k][j]都是真），则将 mat[i][j]设置为真。这确保了传递性；

(4)返回计算结果：函数返回修改后的矩阵 mat，这就是图的传递闭包。

算法复杂度分析：

(1)时间复杂度：$O(n^3)$，其中 n 是图中顶点的数量。这是因为算法包含三个嵌套循环，每个循环都遍历 n 个元素；

(2)空间复杂度：O(n²)，需要额外的空间来存储传递闭包矩阵。

6 实验数据分析

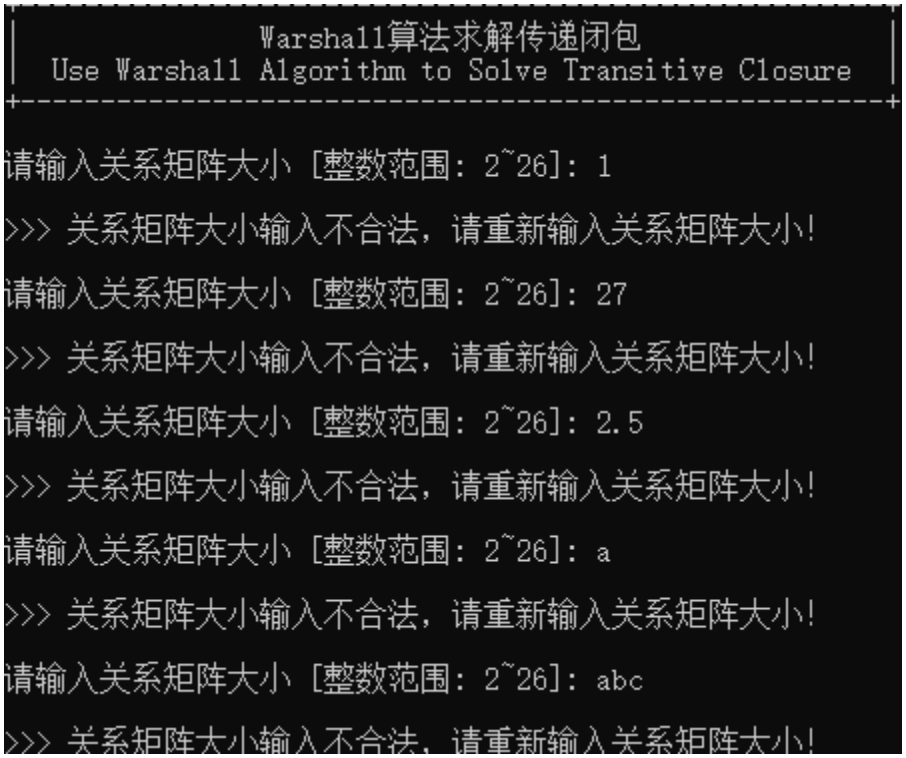

图 6.1 实验数据 1

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。

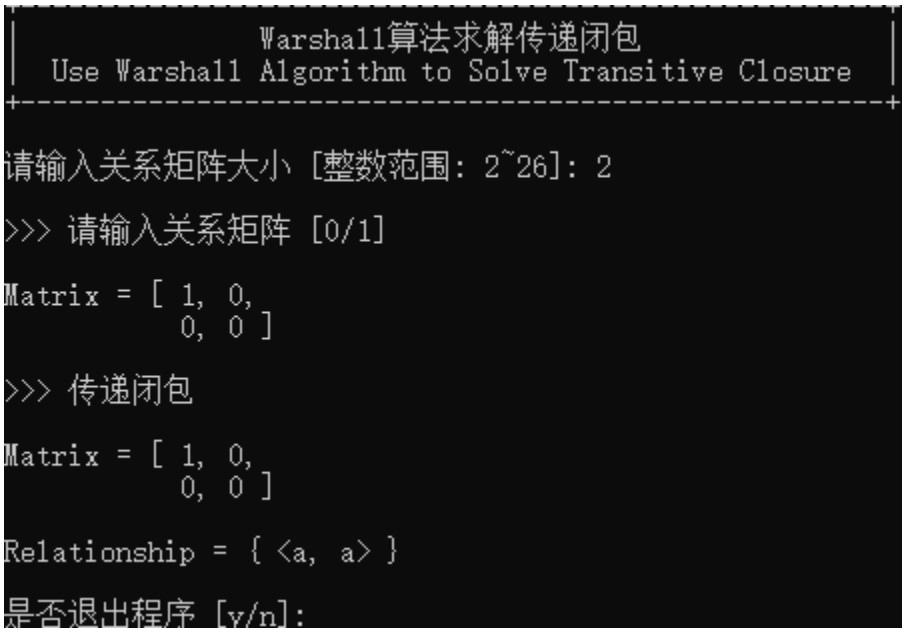

图 6.2 实验数据 2

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

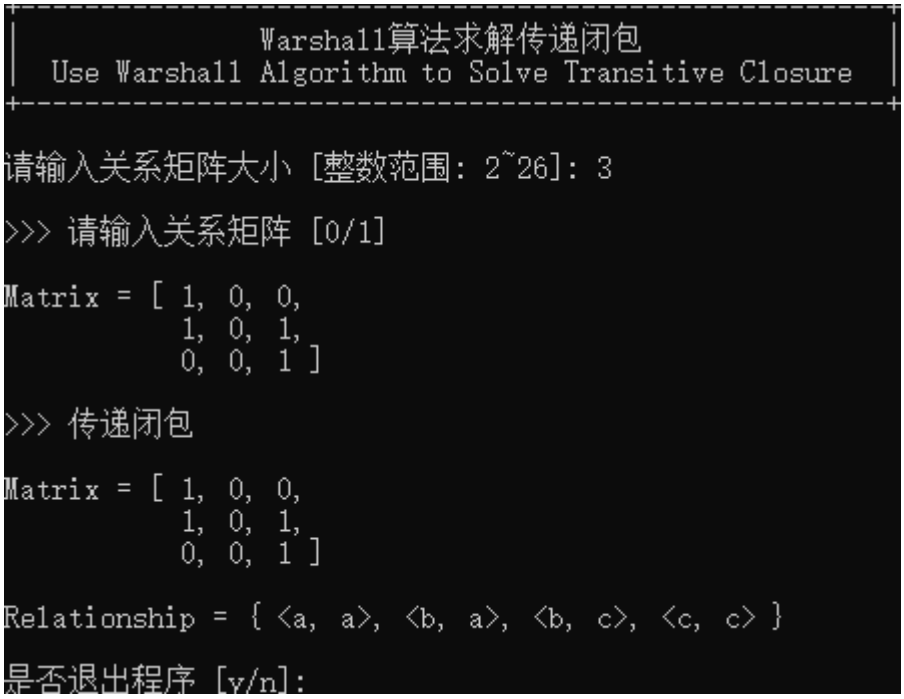

图 6.3 实验数据 3

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

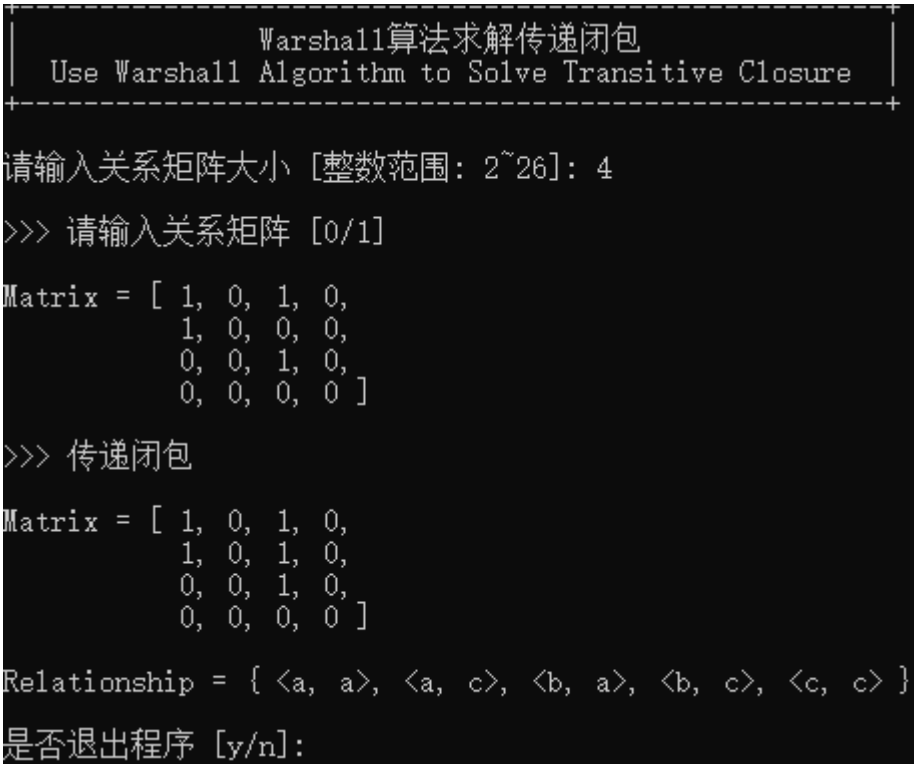

图 6.4 实验数据 4

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

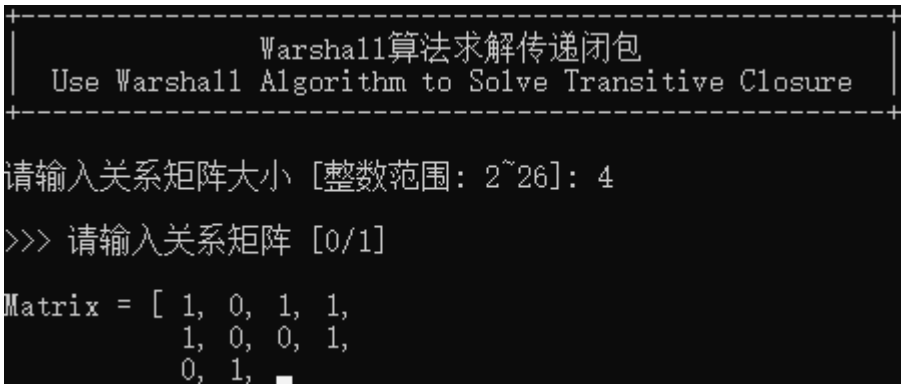

图 6.5 实验数据 5

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。

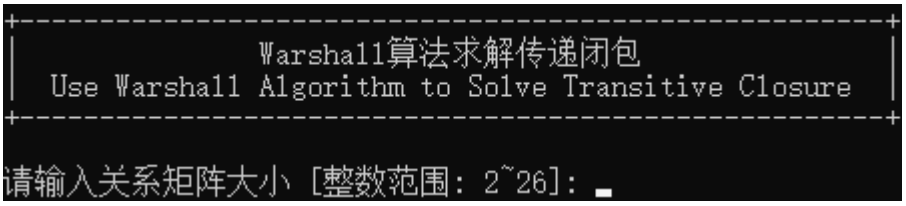

图 6.6 实验数据 6

实验数据分析 6：当输入 n 时，程序清空屏幕并执行下一次 Warshall 算法求解传递闭包运算。

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

在完成关于 Warshall 算法求解传递闭包的实验后，我更深入地理解了关系的传递性和传递闭包的概念。在数学和计算领域,这些概念是基本且非常重要的。通过编写程序来实现 Warshall 算法，我不仅加深了对这些概念的理解，而且也增强了我的逻辑思维和问题解决能力。

这个实验提升了我的编程技能，尤其是在使用 C++这一编程语言方面。在实

现算法的过程中，我学会了如何更有效地使用数据结构，如矩阵，来存储和处理信息。这一点对我未来在计算机科学领域的学习和工作将非常有益。

通过设计用户友好的界面来输入关系矩阵并展示传递闭包的计算结果，我认识到了程序设计中用户体验的重要性。这让我意识到，编写程序不仅仅是实现功能，还需要考虑用户如何与程序交互。

这个实验强化了我对实践学习的重视。虽然理论学习是基础，但通过实际动手实践，我能更加深刻地理解和掌握知识。

总之，这次实验不仅增强了我的专业知识和技能，而且也激发了我对离散数学和计算机科学的更大兴趣。我期待在未来的学习和研究中继续应用和学习这些知识。

8 程序源文件

```
/*****************************************************************
 * Project Name:  Assignment_6
 * File Name:     assignment_6.cpp
 * File Function: Warshall 算法求解传递闭包
 * Author:        Jishen Lin (林继申)
 * Update Date:   2023/12/13
 *****************************************************************/

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

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

/*
 * 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 size = mat.size();
    for (size_t k = 0; k < size; k++)
```

```
        for (size_t i = 0; i < size; i++)
            for (size_t j = 0; j < size; j++)
                mat[i][j] = ((mat[i][j] || (mat[i][k] && mat[k][j])) ?
1 : 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 << "|              Warshall 算法求解传递闭
包              |" << std::endl;
        std::cout << "|  Use Warshall Algorithm to Solve Transitive
Closure  |" << 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");
        }

        /* Transitive closure */
        Matrix transitive = transitiveClosure(matrix);
```

```
        std::cout << std::endl << ">>> 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;
}
```