

## 

**《离散数学》课程实验报告**

# 题目 WarShall算法求解传递闭包

## 姓 名 尹诚成

学 号 2351279

学 院 计算机科学与技术学院

专 业 软件工程

教 师 唐剑锋

二〇二四 年 十二 月 二十 日

目录

1 实验目的 1

2 实验内容 1

3 实验环境 1

4 实验原理 1

4.1 关系的传递性 1

4.2 WarShall算法 2

5 实验过程 3

5.1 实验思路 3

5.2 实验设计 3

5.2.1 数据结构设计 3

5.2.2 结构体与类设计 3

5.2.3 程序主题架构设计 3

5.3 程序功能实现 4

5.3.1 逻辑值输入功能的实现 5

5.3.2 输入关系矩阵大小功能的实现 5

5.3.3 输出关系矩阵与关系集合功能的实现 6

5.3.4 退出程序功能的实现 6

5.4 核心算法实现 7

6 实验数据分析7

7 实验心得10

8 程序源文件11

1. **实验目的**

本实验主要目的是加深对离散数学中关系理论属性的理解，通过实践操作加强对关系的传递性的认识，并通过 WarShall 算法求解传递闭包，具体内容包括：

(1)理解和掌握传递闭包的概念

(2)掌握 WarShall 算法的原理和应用

(3)提高编程能力和逻辑思维

(4)理解数据结构在算法中的应用

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。

1. **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。

1. **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

1. **实验原理**
   1. 关系的传递性

如果一个关系R对于任意三个元素a、b、c，(a,b)和(b,c)都在R中，并且(a,c)也在R中，则这个关系被认为是传递的。。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。

* 1. Warshall 算法

Warshall 算法，由 Stephen Warshall 在 1962 年提出，是一种专门用于计算有向图传递闭包的算法。该算法在计算机科学中的关系闭包概念理解和实现方面起着关键作用，尤其在数据分析、图论和算法设计等领域具有广泛的应用。

算法基础：

Warshall 算法基于动态规划的思想，通过逐步构建传递闭包来不断更新矩阵，直到达到最终目标。其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。

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而著称。

(2)高效性：算法的时间复杂度为 O(n³)，其中 n 是图中顶点的数量。虽然看似复杂度较高，但在处理中等规模的有向图时，其性能仍然是可以接受的。

(3)适用性：Warshall 算法适用于任何大小的有向图，不受图的结构或顶点数量的限制。

算法局限性：

尽管 Warshall 算法在多种场景下非常有效，但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，导致性能下降。因此，在选择是否使用 Warshall 算法时，需要根据具体的应用场景和数据规模进行权衡。

1. **实验过程**
   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。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。

* 1. 实验设计
     1. 数据结构设计

由于本程序利用矩阵来表示存储关系，故使用一个二位整数向量Matrix 类型来表示关系矩阵。

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

* + 1. 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。

* + 1. 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 inputMatrixCount 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 WarShall 函数计算对应传递闭包。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 outputMatrix 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。

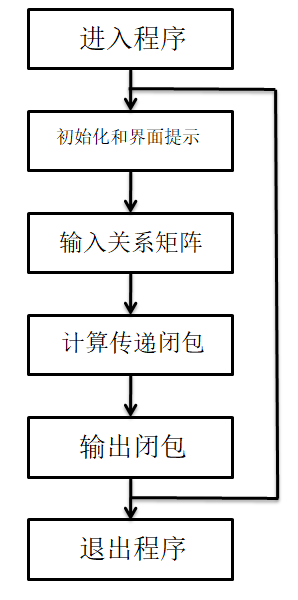


图 5.2.3 程序主体架构设计流程图

* 1. 程序功能实现
     1. 逻辑值输入功能的实现

函数 inputLogicalValue 用于输入逻辑值，处理二进制（真/假）值的用户输入，这里分别用'0'和‘1’或者通过参数 FalseValue 和 TrueValue 指定的其他字符表示。

程序功能实现思路：

(1)FalseValue 表示假的值，默认值为'0'。TrueValue表示真的值，默认值为 '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（如回车键），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。

* + 1. 输入关系矩阵大小功能的实现

程序通过调用inputMatrixCount函数输入关系矩阵大小。inputMatrixCount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。inputMatrixCount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：

(1)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；

(2)用户的输入被读取到count变量中

(3)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>=minCount 和tempInput<=maxCount 确保输入在指定的范围内；

(4)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；

(5)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。

* + 1. 输入关系矩阵和关系集合功能的实现

输出关系矩阵与关系集合功能通过outputMatrix函数和outputLogicalRelationship函数实现，程序功能实现思路为：

(1)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。

(2)输出关系集合

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在内层循环中，对于每一个值为 1 的元素，将其坐标转化为关系对。例如，如果 martix[i][j] 为 1 ，则输出 <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。

* + 1. 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。

* 1. 核心算法实现

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

（1）接受矩阵：输入参数 Matrix& matrix是一个引用类型的矩阵

（2）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row）：遍历所有可能的中间顶点；

②中层循环（变量col）：遍历所有可能的起始顶点；

③内层循环（变量i）：遍历所有可能的结束顶点；

④矩阵更新：若 matrix[row][col] 是真(1)，则对matrix[row][i]进行更新，更新后的值为matrix[row][i] |（位运算） matrix[col][i]的值。这确保了传递性。

（3）返回计算结果

算法复杂度分析：

(1)时间复杂度：O(n³)，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)，函数直接在输入的邻接矩阵上进行操作，没有额外的存储空间消耗，因此空间复杂度为 O(1)，但考虑原始邻接矩阵的存储，矩阵的总空间复杂度为 O(n²)。

1. **实验数据分析**

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。

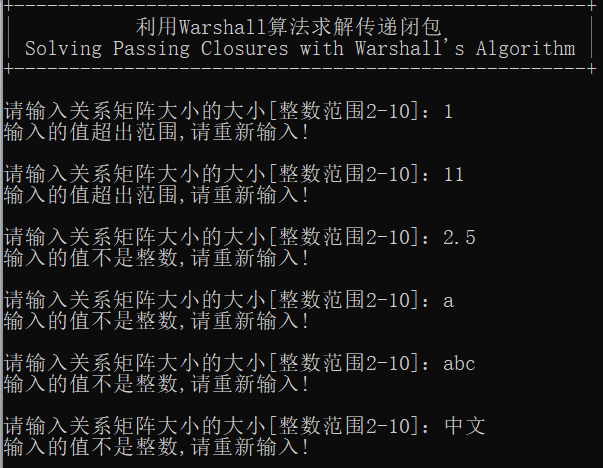


图 6.1 实验数据 1

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： 传递闭包：

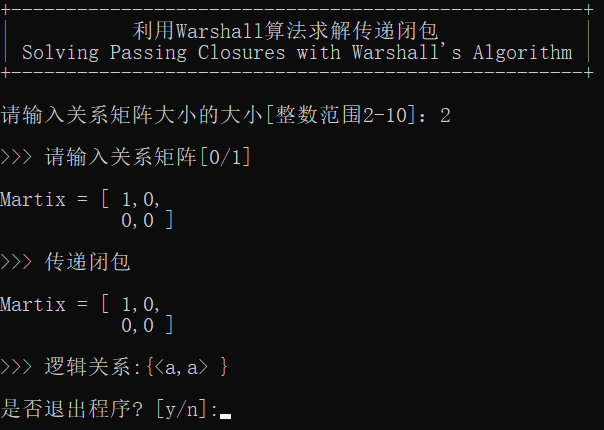


图 6.2 实验数据 2

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： 传递闭包：

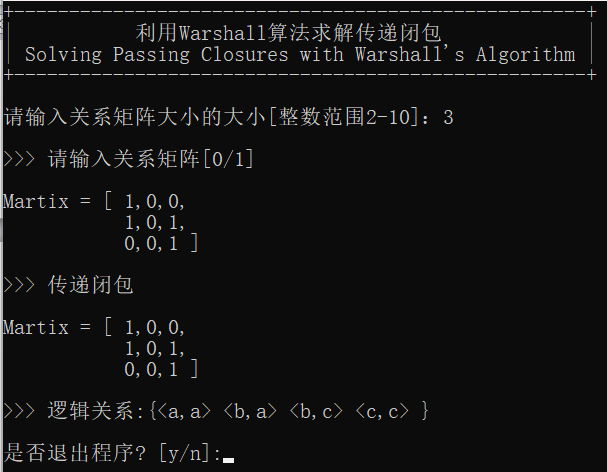


图6.3 实验数据 3

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： 传递闭包：

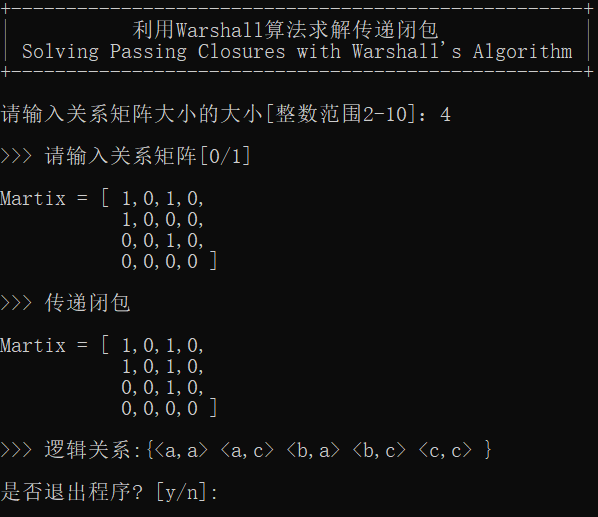


图 6.4 实验数据 4

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。

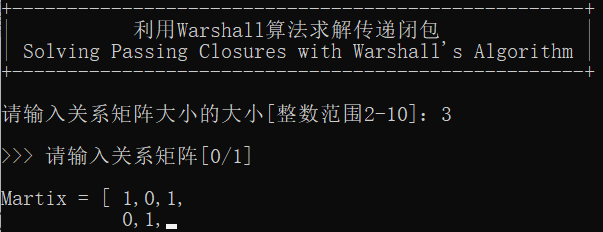


图 6.5 实验数据 5

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传递闭包运算。

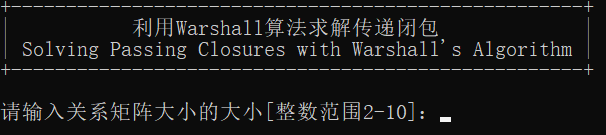


图 6.6 实验数据 6

实验数据分析7：当输入y时，程序退出。

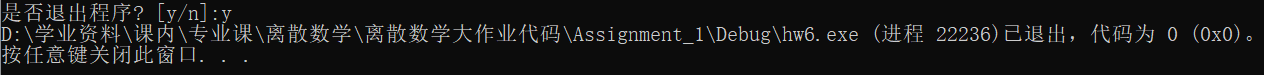


图 6.7 实验数据 7

1. **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。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.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。

1. **程序源文件**

#include <iostream>

#include <conio.h>

#include <limits>

#include <vector>

/\*define the martix\*/

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

/\*

\* Function Name: inputLogicalValue

\* Function: input a logical value from user

\* Input Parameters:char FalseValue = '0'

\* char TrueValue = '1'

\* Return Value: int: 0 for False, 1 for True

\*/

int inputLogicalValue(char FalseValue = '0', char TrueValue = '1')

{

while (true) {

char InputChar = \_getch();

if (InputChar == 0 || InputChar == -32)

InputChar = \_getch();

else if (InputChar == FalseValue || InputChar == TrueValue) {

std::cout << InputChar;

return InputChar == TrueValue;

}

}

}

/\*

\* Function Name: logicalPlus

\* Function: calculate the logical OR operation

\* Input Parameters:const int a

\* const int b

\* Return Value: int: 0 for False, 1 for True

\*/

int logicalPlus(const int relation1, const int relation2) {

return static\_cast<int>(relation1 || relation2);

}

/\*

\* Function Name: inputMartixInfo

\* Function: input a logical matrix from user

\* Input Parameters:Matrix &matrix

\* Return Value: void

\*/

void inputMatrixInfo(Matrix& matrix) {

std::cout << std::endl;

std::cout << ">>> 请输入关系矩阵[0/1]" << std::endl << std::endl;

std::cout << "Martix = [ ";

for (size\_t row = 0; row < matrix.size(); row++) {

for (size\_t col = 0; col < matrix[row].size(); col++) {

/\*input a logical value\*/

matrix[row][col] = inputLogicalValue();

if (row != matrix.size() - 1 || col != matrix[row].size() - 1)

std::cout << ",";

}

if (row == matrix.size() - 1) {

std::cout << " ]" << std::endl;

break;

}

std::cout << std::endl << " ";

}

std::cout << std::endl;

}

/\*

\* Function Name: inputMatrixCount

\* Function: input a matrix count from user

\* Input Parameters:const int minCount

\* const int maxCount

\* const char\* prompt

\* Return Value: int: the matrix count

\*/

int inputMatrixCount(const int minCount, const int maxCount, const char\* prompt) {

double count;

while (true) {

std::cout << "请输入" << prompt << "的大小[整数范围" << minCount << "-" << maxCount << "]：";

std::cin >> count;

if (std::cin.fail() || count != static\_cast<int>(count)) {

std::cout << "输入的值不是整数,请重新输入!" << std::endl << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

continue;

}

else if (count < minCount || count > maxCount) {

std::cout << "输入的值超出范围,请重新输入!" << std::endl << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

continue;

}

else

break;

}

return count;

}

/\*

\* Function Name: warShall

\* Function: calculate the logical AND operation

\* Input Parameters:Matrix &matrix

\* Return Value: void

\*/

void WarShall(Matrix& martix) {

for (size\_t col = 0; col < martix.size(); col++) {

for (size\_t row = 0; row < martix.size(); row++) {

if (martix[row][col] == 1) {

for (size\_t i = 0; i < martix.size(); i++) {

martix[row][i] = martix[row][i] | martix[col][i];

}

}

}

}

}

/\*

\* Function Name: outputMatrix

\* Function: output a logical matrix

\* Input Parameters:const Matrix matrix

\* const char\* prompt

\* Return Value: void

\*/

void outputMatrix(const Matrix matrix, const char\* prompt) {

std::cout << ">>> " << prompt << std::endl << std::endl;

std::cout << "Martix = [ ";

for (size\_t row = 0; row < matrix.size(); row++) {

for (size\_t col = 0; col < matrix[row].size(); col++) {

std::cout << matrix[row][col];

if (row != matrix.size() - 1 || col != matrix[row].size() - 1)

std::cout << ",";

}

if (row == matrix.size() - 1) {

std::cout << " ]" << std::endl;

break;

}

std::cout << std::endl << " ";

}

std::cout << std::endl;

}

/\*

\* Function Name: outputLogicalRelationship

\* Function: output the logical relationship of the matrix

\* Input Parameters:const Matrix matrix

\* Return Value: void

\*/

void outputLogicalRelationship(const Matrix matrix) {

std::cout << ">>> 逻辑关系:{";

for (size\_t row = 0; row < matrix.size(); row++) {

for (size\_t col = 0; col < matrix[row].size(); col++) {

if (matrix[row][col] == 1) {

std::cout << "<" << static\_cast<char>(row + 'a') << "," << static\_cast<char>(col + 'a') << "> ";

}

}

}

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

}

/\*

\* Function Name: main

\* Function: main function

\* Return Value: 0

\*/

int main() {

do {

system("cls");

/\*print the title\*/

std::cout << "+----------------------------------------------------+" << std::endl;

std::cout << "| 利用Warshall算法求解传递闭包 |" << std::endl;

std::cout << "| Solving Passing Closures with Warshall's Algorithm |" << std::endl;

std::cout << "+----------------------------------------------------+" << std::endl << std::endl;

/\*input the information of the matrix\*/

unsigned int row = inputMatrixCount(2, 10, "关系矩阵大小");

unsigned int col = row;

Matrix matrix(row, std::vector<int>(col, 0));

inputMatrixInfo(matrix);

/\*calculate the passing closure\*/

WarShall(matrix);

/\*output the result\*/

outputMatrix(matrix, "传递闭包");

outputLogicalRelationship(matrix);

std::cout << "是否退出程序? [y/n]:";

} while (!inputLogicalValue('n', 'y'));

return 0;

}