**项目说明文档**

**离散数学课程实验**

**——Warshall算法**

**求传递闭包**

作 者 姓 名 刘淑仪

学 号 2251730

指 导 教 师 唐剑锋

学 院 专 业 软件学院 软件工程



二〇二三 年 十二 月 二十二 日

目录

1 项目分析 1

1.1 实验目的分析 1

1.2 实验内容概述 1

2 项目设计与实现 1

2.1 实验原理 1

2.1.1 题目涉及知识原理 1

2.1.2 解题原理 2

2.2 实验过程设计 3

2.2.1 实验实现思路 3

2.1.2 实验实现流程图 4

2.3 项目具体实现 5

3 项目实现与测试 5

4 源代码展示 7

5 集成开发环境与编译运行环境 11

# 项目分析

## 实验目的分析

该实验课程的目标是让学生了解什么是关系矩阵，如何用二维数组来表示和存储一个关系矩阵；了解什么是自反闭包、传递闭包和对称闭包，以及如何用矩阵运算来求解这些闭包。

## 实验内容概述

用c++实现输入一个关系矩阵（要求关系矩阵的各个位置的元素均为0或1），用Warshall算法求该关系矩阵的传递闭包所对应的关系矩阵。

Ⅰ、键盘输入关系矩阵的行列数（两者相等且大于0）

Ⅱ、键盘依次输出关系矩阵各行的元素（要求元素为0或1）

Ⅲ、用Warshall算法求该关系矩阵的传递闭包并将其打印出来。

Ⅳ、输入1/0，选择是否继续

Ⅴ、若输入为1，则返回第Ⅰ步；若输入为0，则结束程序。

# 项目设计与实现

## 实验原理

### 题目涉及知识原理

1.关系矩阵

在图论中，关系矩阵是一种表示图中顶点之间关系的方式。对于有向图来说，这种矩阵通常称为邻接矩阵。在邻接矩阵中：

矩阵的行和列代表图中的顶点。如果顶点 i 到顶点 j 有一条边，那么矩阵中的元素 (i, j) 为 1；否则为 0。对于无权图，矩阵元素是二元的（0 或 1），表示是否存在边；对于有权图，矩阵元素表示边的权重。

2.传递闭包

传递闭包是图的一个性质，它描述了图中顶点间的可达性。对于有向图的传递闭包：

如果从顶点 A 到顶点 B 存在一条路径（直接或间接），那么在 A 和 B 之间存在传递关系。传递闭包的邻接矩阵中，如果从顶点 i 到顶点 j 存在路径，那么元素 (i, j) 为 1；否则为 0。传递闭包是原始图的一个扩展，它显示了从任何一个顶点到另一个顶点是否存在路径。

3.Warshall算法

Warshall 算法是一个用于计算有向图的传递闭包的算法。传递闭包是一个图的概念，它描述了从图中任意节点到另一个节点是否存在一条路径。在有向图中，如果从节点 A 到节点 B 存在一条路径，那么在 A 和 B 之间就有一个传递关系。Warshall 算法的目的是找出图中所有这样的传递关系。Warshall 算法的关键在于它利用了动态规划的思想，通过逐步构建中间结果来避免重复计算，从而高效地找出所有的传递关系。由于算法的时间复杂度为 O(n^3)，其中 n 是图中节点的数量，它适用于中等规模的图。

### 2.1.2 解题原理

Warshall 算法是一种时间复杂度为 O(n^3) 的算法，用于计算有向图的传递闭包。这个算法通过三层嵌套循环来实现，具体步骤如下：

1. 两层外循环：前两层循环负责遍历矩阵的每一个元素。这里，矩阵代表图中的顶点间关系，其中行和列分别对应不同的顶点。
2. 检查并更新路径：在遍历过程中，如果在矩阵的第 i 列和第 j 行的元素为 1（即存在从顶点 j 到顶点 i 的直接路径），则进入第三层循环。在这个循环中，第 i 行的所有元素会被“逻辑相加”到第 j 行的对应元素上。这里的“逻辑相加”指的是：如果从顶点 i 到另一顶点 k 有路径（矩阵的第 i 行和第 k 列的元素为 1），那么从顶点 j 到顶点 k 也应该存在路径。因此，矩阵的第 j 行和第 k 列的元素也会被设置为 1。
3. 逻辑运算的含义：这种操作的本质在于：如果存在从顶点 j 到顶点 i 的路径，并且从顶点 i 到顶点 k 的路径，根据传递性，从顶点 j 到顶点 k 也必然存在路径。因此，矩阵的相应元素更新为 1。如果从顶点 i 到顶点 k 没有路径，那么元素保持原值。
4. 循环控制与终止条件：外层循环变量 i 从 1 开始递增。对于每个 i，内层循环遍历所有的 j 和 k，按照上述规则更新矩阵。当 i 的值超过顶点数量 n 时，算法停止。

## 实验过程设计

### 2.2.1 实验实现思路

首先，定义一个Relation类，用来表示一个二元关系，该类有一个私有成员变量graph，用来存储关系矩阵，以及一些公有成员函数，用来实现不同的闭包运算、打印矩阵和选择算法等功能。

然后，定义一个构造函数，接受一个关系矩阵作为参数，并用它来初始化graph，这样就可以根据用户输入的矩阵创建一个Relation对象。接着，定义三个闭包运算的函数，分别是reflexiveClosure、symmetricClosure和transitiveClosure，它们的功能是对graph进行相应的修改，使其具有自反性、对称性和传递性。具体的算法如下：

reflexiveClosure：遍历graph的对角线元素，并将其设置为1，以实现自反性。symmetricClosure：遍历graph的所有元素，如果(i, j)为1，则设置(j, i)也为1，以实现对称性。transitiveClosure：warshallTransitiveClosure 方法实现 Warshall 算法，通过三重循环更新矩阵，以确保传递性。如果存在从 i 到 k 的路径和从 k 到 j 的路径，则确保从 i 到 j 的路径也存在。

定义一个print函数，用来打印graph的内容，使用setw和left来控制输出格式，使其对齐。之后，定义一个option函数，用来让用户选择不同的闭包运算，使用cout和cin来输出和输入信息，使用switch语句来根据用户输入的序号执行相应的操作，如果用户输入1-3，就调用相应的闭包运算函数，并打印结果，如果用户输入4，就退出程序，返回0，否则继续循环，返回1。

然后，定义两个辅助函数，分别是inputNum和generateMatrix，它们的功能是用来输入数字和生成矩阵，具体的算法如下：

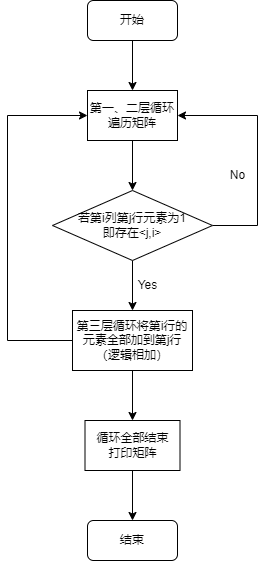
inputNum：接受一个布尔值作为参数，表示是否是输入矩阵的大小，如果是，就限制输入的数字在0-9之间，如果不是，就限制输入的数字在0-4之间，使用cin和getch来获取用户输入的数字，使用cin.fail和cin.clear来检测和处理输入错误，使用cin.ignore来清空输入缓冲区，使用cout来输出提示信息，返回输入的数字。generateMatrix：使用inputNum函数来输入矩阵的行数和列数，创建一个二维向量graph来存储矩阵，使用getch和cout来获取和输出矩阵的元素，返回graph。

在main函数中，定义一个布尔变量loop来控制程序的循环，定义一个二维向量matrix来存储用户输入的矩阵，调用menu函数来输出菜单信息，调用generateMatrix函数来生成矩阵，创建一个Relation对象relation，并用matrix来初始化，调用print函数来打印原始矩阵，然后使用while循环，调用option函数来让用户选择不同的闭包运算，直到用户选择退出，返回0，结束循环，返回0。

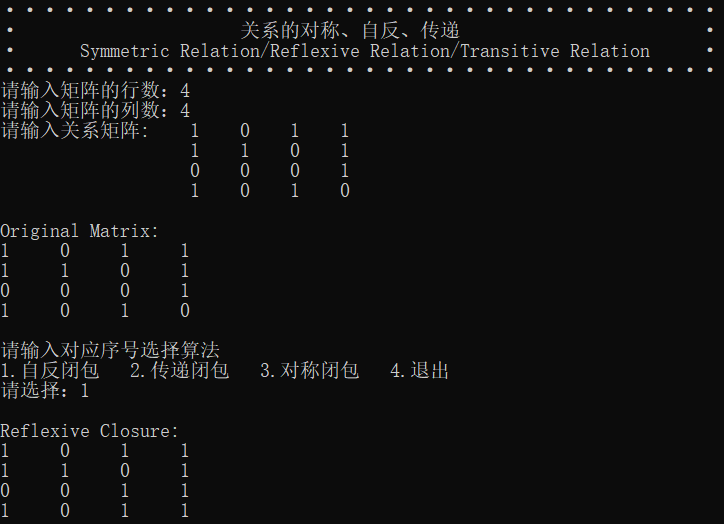
### 2.1.2 实验实现流程图

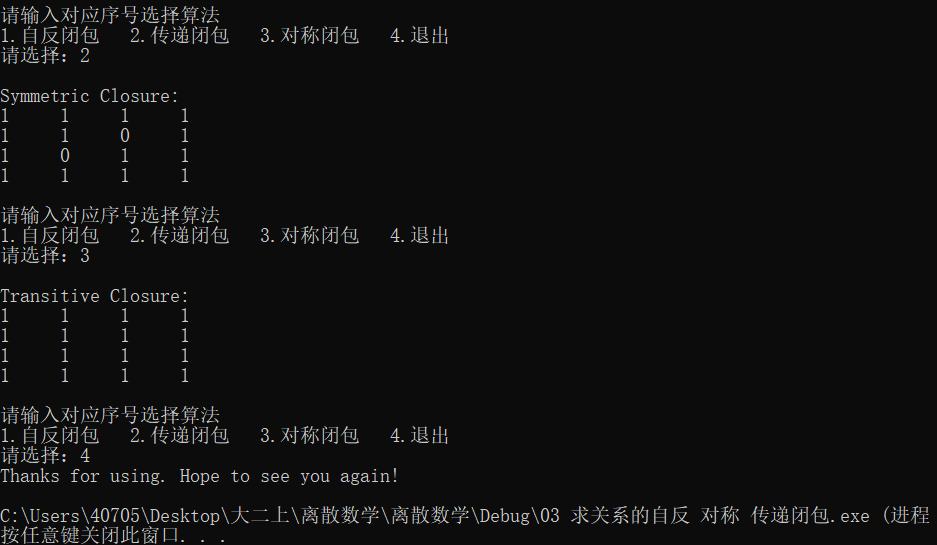
详见第三篇报告，此处只展示Warshall算法。

求传递闭包：



## 项目具体实现





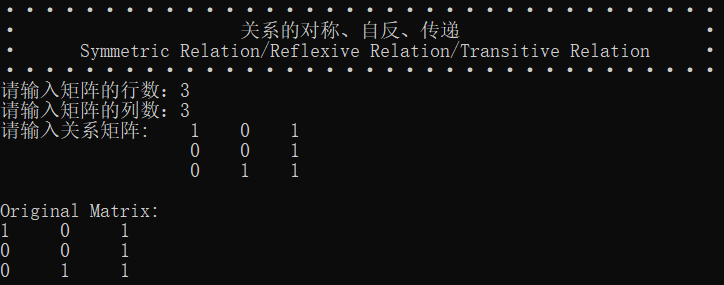
# 项目实现与测试

结果分析：

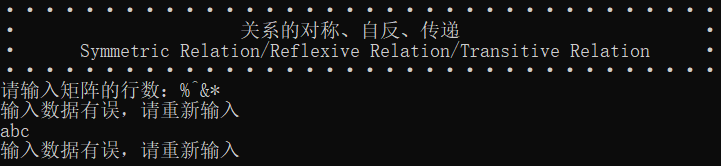
1. 欢迎界面



1. 输入界面
   1. 基本输入



* 1. 错误输入

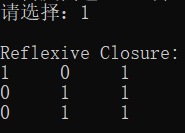


关系矩阵的值只能为0或1，否则不予显示也不予存储。

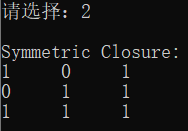
1. 选择操作界面



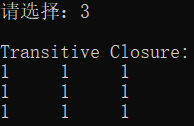
1. 显示自反闭包



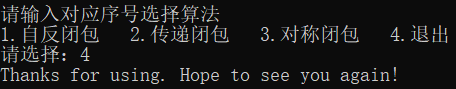
1. 显示传递闭包



1. 显示对称闭包



1. 退出界面



# 源代码展示

#include <iostream>

#include <vector>

#include <conio.h>

#include <iomanip>

using namespace std;

int inputNum(bool opt);

vector<vector<int>> generateMatrix();

class Relation {

private:

    vector<vector<int>> graph;

public:

    // 构造函数 接受一个关系矩阵作为参数并初始化关系对象

    Relation(const vector<vector<int>>& matrix) :graph(matrix) {}

    void reflexiveClosure();                        // 自反闭包

    void symmetricClosure();                        // 对称闭包

    void warshallTransitiveClosure();               // 传递闭包

    void print();

    bool option();

};

bool Relation::option()

{

    cout << "\n请输入对应序号选择算法\n";

    cout << "1.自反闭包   2.传递闭包   3.对称闭包   4.退出\n";

    cout << "请选择：";

    int opt = inputNum(0);

    switch (opt) {

    case 1:

        reflexiveClosure();

        cout << "\nReflexive Closure:\n";

        print();

        break;

    case 2:

        symmetricClosure();

        cout << "\nSymmetric Closure:\n";

        print();

        break;

    case 3:

        warshallTransitiveClosure();

        cout << "\nTransitive Closure:\n";

        print();

        break;

    case 4:

        cout << "Thanks for using. Hope to see you again!\n";

        return 0;

    }

    return 1;

}

void Relation::reflexiveClosure()

{

    // 遍历关系矩阵的对角线元素，并将其设置为1，以实现自反性

    for (int i = 0; i < graph.size(); i++) { graph[i][i] = 1; }

}

void Relation::symmetricClosure()

{

    // 遍历关系矩阵的所有元素，如果(i, j)为1，则设置(j, i)也为1，以实现对称性

    for (int i = 0; i < graph.size(); i++) {

        for (int j = 0; j < graph.size(); j++) {

            if (graph[i][j] == 1) {

                graph[j][i] = 1;

            }

        }

    }

}

void Relation::warshallTransitiveClosure()

{

    for (int k = 0; k < graph.size(); k++) {

        for (int i = 0; i < graph.size(); i++) {

            for (int j = 0; j < graph.size(); j++) {

                // 更新 graph[i][j] 为 1 如果 i 到 j 的路径存在

                graph[i][j] = graph[i][j] || (graph[i][k] && graph[k][j]);

            }

        }

    }

}

// 打印关系矩阵

void Relation::print()

{

    for (const auto& row : graph) {

        for (int val : row) {

            cout << setw(5) << left << val << " ";

        }

        cout << endl;

    }

}

// 输入数字

int inputNum(bool opt)

{

    int maxSize;

    if (opt)

        maxSize = 9;

    else

        maxSize = 4;

    int inputNum;

    while (true) {

        std::cin >> inputNum;

        if (std::cin.fail() || inputNum < 0 || inputNum > maxSize) {

            std::cin.clear();

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

            std::cout << "输入数据有误，请重新输入\n";

        }

        else {

            if (std::cin.get() == '\n')

                break;

            else {

                std::cin.clear();

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

                std::cout << "输入数据有误，请重新输入\n";

            }

        }

    }

    return inputNum;

}

// 生成矩阵

vector<vector<int>> generateMatrix()

{

    int matRow = 0, matCol = 0;

    cout << "请输入矩阵的行数：";

    matRow = inputNum(1);

    cout << "请输入矩阵的列数：";

    matCol = inputNum(1);

    vector<vector<int>> graph(matRow, vector<int>(matRow, 0));

    cout << "请输入关系矩阵:";

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

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

            graph[i][j] = \_getch() - '0';

            cout << setw(5) << graph[i][j];

        }

        cout << endl << "               ";

    }

    return graph;

}

void menu()

{

    cout << "·················\n";

    cout << "·        关系的传递闭包        ·\n";

    cout << "·      Transitive Relation     ·\n";

    cout << "·················\n";

}

int main()

{

    bool loop = 1;

    vector<vector<int>> matrix;

    menu();

    matrix = generateMatrix();

    Relation relation(matrix);

    cout << "\nOriginal Matrix:\n";

    relation.print();

    while (loop) {

        loop = relation.option();

    }

    return 0;

}

# 集成开发环境与编译运行环境

Windows系统：Windows 11 x64

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

Windows编译运行环境：本项目适用于x86架构和x64架构