# 《离散数学》课程实验报告 6 Warshall 算法求关系矩阵的传递闭包

2254321 陆诚彬

## 1. 实验内容

本实验的目的是实现一个 C++程序，使用算法来计算给定关系矩阵的传递闭包。传递闭包是一个图论概念，用于确定在一个有向图中，顶点间是否存在一条路径。在本实验中，我们通过用户输入的矩阵来代表这样一个有向图，其中矩阵的行和列代表图的顶点，元素值表示顶点间的直接关系。

## 2. 解题思路

- **Warshall 算法理解**

首先需要理解 Warshall 算法的原理，这是一个动态规划算法，用于计算图的传递闭包。算法通过逐步构建路径信息，最终得出哪些顶点间存在路径。

- **输入处理**

由于算法需要用户输入的矩阵数据，因此编写代码以接受用户输入的矩阵，并对输入进行有效性检查。

- **算法实现**

根据 Warshall 算法的步骤，使用三重循环对矩阵进行处理，得到传递闭包。

- **结果输出**

最后，打印出计算得到的传递闭包矩阵。

## 3. 数据结构设计

**矩阵存储：** 使用 std::vector<std::vector<int>>来存储用户输入的矩阵。这种数据结构方便动态地处理不同大小的矩阵。

**输入验证：** 为了确保输入的有效性，使用标准输入流 std::cin 的状态标志进行检查，并在发现错误输入时清除错误状态并忽略错误输入直到下一个换行符。

# 4. 项目实现

**代码结构：**

**warshallAlgorithm 函数：**

**项目核心函数，**实现 Warshall 算法，计算并打印传递闭包。

**参数：**

matrix：整型二维向量的引用。此参数代表图的邻接矩阵。使用引用（&）至关重要，因为它允许在不复制整个矩阵的情况下进行原地修改。

const int n：矩阵的大小（行数/列数）。声明为 const 以确保函数不更改矩阵的大小。

**函数体：**

```
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                matrix[i][j] = matrix[i][j] || (matrix[i][k] && matrix[k][j]);
            }
        }
    }
```

这组嵌套循环是 Warshall 算法的核心。最外层循环遍历所有顶点（k），考虑每个顶点作为潜在路径中的中间顶点。

中间和内层循环遍历所有顶点对（i 和 j）。对于每对顶点，算法检查是否存在从 i 到 j 的直接路径（matrix[i][j]）或通过 k 的路径（matrix[i][k] && matrix[k][j]）。

逻辑或运算符（||）用于更新矩阵的[i][j]项。如果存在直接路径（matrix[i][j]）或通过 k 的间接路径（matrix[i][k] && matrix[k][j]），则[i][j]项设置为 true（或 1）。

**Warshall 算法的概念概述：**

1. 该算法基于动态规划，逐步更新邻接矩阵。
2. 如果顶点 k 可以作为连接两个顶点 i 和 j 的中间点，则在更新的矩阵中考虑通过 k 的 i 到 j 的路径。
3. 经过所有迭代后，最终的矩阵表示所有顶点对之间所有可能的路径，有效地计算了传递闭包。

**复杂度：**

Warshall 算法的时间复杂度为 $O(n^3)$，其中 n 是图中的顶点数。这是由于三重嵌套循环结构遍历了所有可能的顶点对，每个中间顶点。空间复杂度为 $O(1)$，因为算法在原地修改输入矩阵，不需要额外空间。

**validateInput 函数：**检查并处理无效的输入。

**selectOption 函数：**处理用户输入，调用 warshallAlgorithm 函数。

**main 函数：**程序入口，调用 selectOption 函数。

**错误处理：**

在用户输入矩阵大小和元素时，如果输入非法（如非数字或负数），程序会提示用户重新输入，直到输入有效。

**实现细节：**

使用 C++11 标准，确保了更好的兼容性和现代 C++特性的使用。在处理用户输入时，对输入流进行了严格的错误检查和处理，增加了程序的健壮性。

# 5. 设计小结

在本次实验中，我们成功实现了一个基于 Warshall 算法的程序，用于计算任意有向图的传递闭包。整个设计过程不仅涉及算法的实现，还包括了对用户输入的有效性验证和错误处理，以及合理的数据结构设计来存储和处理图的邻接矩阵。

**算法实现的核心：**Warshall 算法的实现是本项目的核心，它展示了如何应用动态规划的思想来解决复杂的图论问题。通过逐步更新邻接矩阵，算法能够有效地确定图中所有顶点对之间是否存在路径。

**健壮的输入处理：**在实验过程中，对用户输入进行有效性检查和错误处理是至关重要的。这不仅提高了程序的用户友好性，还增加了程序的健壮性，防止了非法输入导致的潜在问题。

**数据结构的选择：**使用 std::vector<std::vector<int>>作为邻接矩阵的存储方式，灵活地适应了不同大小矩阵的需求，同时保证了程序的高效性和易用性。

**代码结构和清晰度：**整个项目的代码结构清晰，逻辑分明，易于理解和维护。通过合理的函数划分和详细的注释，提高了代码的可读性。

# 6. 实验心得

通过这次实验，我不仅学习和掌握了 Warshall 算法的原理和实现，还深入理解了动态规划在解决实际问题中的应用。我也体会到了健壮的输入验证和错误处理在软件开发中的重要性，它们是确保程序稳定运行的关键。

此外，这次实验也强化了我的 C++编程技能，特别是在使用现代 C++特性和数据结构方面。我学会了如何更有效地管理和处理数据，以及如何编写结构清晰、易于维护的代码。

最重要的是，这次实验让我意识到理论知识和实践应用之间的联系。通过亲自编写程序实现理论算法，我不仅加深了对算法的理解，还提高了解决实际问题

的能力。这种从理论到实践的转换对我的学习和未来的职业生涯都是非常宝贵的经验。

## 7. 项目源代码

```
#include <iostream>
#include <vector>
#include <limits>

using namespace std;

// 使用 Warshall 算法计算传递闭包
// @param matrix 引用传递的二维向量，代表关系矩阵
// @param n 矩阵的大小（行数和列数）
void warshallAlgorithm(vector<vector<int>>& matrix, const int n) {
    // 三重循环遍历矩阵中的每个元素
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                // 更新矩阵元素以反映传递性
                matrix[i][j] = matrix[i][j] || (matrix[i][k] && matrix[k][j])
;
            }
        }
    }

    // 打印传递闭包矩阵
    cout << "传递闭包为:\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << "\n";
    }
}

// 检查输入是否有效，并在必要时清除输入缓冲区
bool validateInput() {
    if (cin.fail()) {
        cin.clear(); // 清除输入缓冲区的错误标志
         cin.ignore(numeric_limits<streamsize>::max(), '\n'); // 忽略缓冲区直
到下一个换行符
        return false;
    }
    return true;
}

void selectOption() {
    int n, d;
    cout << "请输入矩阵的行数和列数: ";
    while (!(cin >> n >> d) || n <= 0 || d <= 0) {
        cout << "无效输入，请重新输入矩阵的行数和列数: ";
        validateInput();
    }

```

```
49.         vector<vector<int>> matrix(n, vector<int>(d));
50.         cout << "请输入关系矩阵:\n";
51.         for (int i = 0; i < n; i++) {
52.             cout << "请输入矩阵的第" << i << "行元素(元素以空格分隔): ";
53.             for (int j = 0; j < d; j++) {
54.                 while (!(cin >> matrix[i][j])) {
55.                     cout << "无效输入，请重新输入第" << i << "行的第" << j << "个元
    素: ";
56.                     validateInput();
57.                 }
58.             }
59.         }
60. 
61.         warshallAlgorithm(matrix, n);
62.     }
63. 
64.     int main() {
65.         selectOption(); // 运行主要功能
66.         return 0;
67.     }
```