同濟大學
TONGJI UNIVERSITY

离散数学

课程实验报告

Warshall 算法求解传递闭包

姓　　名：　　马小龙

学　　号：　　2353814

学　　院：计算机科学与技术学院（软件学院）

专　　业：　　软件工程

指导教师：　　李冰

二〇二四年十一月三十日

目录

一 实验目的

1. 旨在帮助学生理解传递闭包的定义，同时通过编程实现其算法，提升对集合论和关系代数的直观感知。

2. Warshall 算法是计算传递闭包的经典算法，它基于动态规划的思想，利用逐步更新矩阵的方式来合并间接关系。实验的核心目标是帮助学生掌握该算法的工作原理，理解其如何通过三个嵌套的循环将每对元素之间的间接关系加入到矩阵中。

2. 实验要求学生能够在理解算法的基础上，清晰地分解问题并实现解决方案，将理论中的传递闭包计算转化为实际的编程实现，运用多种编程技巧，从而提升编程能力和逻辑思维能力。

3. 本实验是对传统求解传递闭包算法的改进；通过该实验，培养学生的算法优化意识、通过改进数据结构或者算法设计来提高效率的意识。

4. 通过这个实验，学生能够更好地理解传递闭包算法在实际问题中的潜在应用，激发其对算法在实际应用中如何解决复杂问题的兴趣和探索精神。

通过本实验学生学习并实现 Warshall 算法，在实践中加深对离散数学中“关系”和“闭包”这些抽象概念的理解，提升编程能力，锻炼分析问题和解决问题的能力，并为进一步学习图论、路径算法等内容打下坚实的基础。

二 实验内容

本实验的主要内容是使用 C++程序实现 Warshall 算法求解传递闭包。实验要求学生编写程序，接受用户输入的关系矩阵，正确、合理地实现 Warshall 算法实现传递闭包计算。同时，实现过程中应该包含基本的错误处理逻辑，提高程序的鲁棒性。实验的关键目标是使学生熟练掌握传递闭包的计算方法，加深对离散数学相关概念的理解，并提升其在实际编程中的应用能力。

三 实验环境

编程开发语言：C++

文 件 编 码：UTF-8

集成开发环境：Clion 2024

工　　具　　集：MinGW 11.0 w64

四 实验原理

4.1 关系的传递性

关系的传递性是集合论和数学中一个重要的概念，它描述了集合中的元素之间的某种关系。当一个关系是传递时，如果某些元素之间存在关系，那么通过这些元素之间的关系，可以推导出其他元素之间的关系。

具体来说，对于一个集合 A 和其上的二元关系 $R \subseteq A \times A$，如果满足以下条件：如果 $(a,b) \in R$ 且 $(b,c) \in R(b, c)$，那么必定有 $(a,c) \in R(a, c)$，那么就可以说关系 R 具有传递性。

4.2 Warshall 算法

Warshall 算法是一种用于计算图的**传递闭包**的经典算法，最早由美国数学家 Robert Warshall 在 1962 年提出。传递闭包在图论、关系代数、数据库理论等领域具有广泛的应用，其核心目的是通过计算给定图（或关系矩阵）中的所有间接路径，从而找到每对节点之间是否存在传递关系。

Warshall 算法通过在一个关系矩阵中逐步更新元素的值来计算传递闭包。它利用了动态规划的思想，逐渐通过引入中间节点来确定两节点之间是否存在路径。通过迭代更新矩阵，最终得到一个反映所有传递关系的矩阵。

Warshall 算法的核心原理是“如果从节点 i 到节点 k 和从节点 k 到节点 j 存在路径，则从节点 i 到节点 j 也应该存在路径”。通过这种方式，算法通过不断迭代更新矩阵元素，直到所有可能的传递路径都被计算出来。算法的基本思路是依次考虑每个节点是否能够作为中间节点连接其他两个节点。

Warshall 优点：

Warshall 算法非常直观，理解起来较为简单；对于较小规模的图，Warshall 算法能够快速计算传递闭包；同时该算法适用于任意的关系矩阵。

Warshall 缺点：

Warshall 算法的时间复杂度是 O(n^3)，对于大规模图（尤其是节点数非常大的情况下）效率较低。同时，该算法假设关系矩阵是静态的，不适用于实时更新关系或动态变化的图。

五 实验过程

5.1 实验思路

本实验的思路主要是通过输入一个关系矩阵，应用 Warshall 算法计算传递闭包。首先，用户输入关系矩阵，然后程序通过 Warshall 算法逐步更新矩阵，判断每对元素间是否存在关系，从而计算得到传递闭包矩阵。最终，输出计算得到的传递闭包矩阵，展示所有元素对之间的关系。实验过程中，重点是理解 Warshall 算法的原理，并通过实际操作验证算法的正确性和有效性。

5.2 数据结构设计

本实验主要使用动态二维数组 std::vector<std::vector<bool>> matrix 来存储关系矩阵。

5.2 程序框架设计

1.程序首先进入一个 while 无限循环，调用清屏指令，然后调用 TransitiveClosure 函数。

2. TransitiveClosure 函数主要用来进行矩阵相关的输入工作，调用不同函数实现计算和输出关系矩阵。主要逻辑如下：

首先，程序会提示用户输入集合 A 的元素个数（即矩阵的维度），然后通过循环读取每行输入关系矩阵的元素。每次输入时，程序都会验证用户输入的合法性。

然后，根据用户提供的关系矩阵，调用 Transitive 计算关系矩阵的传递闭包。

最后，输出计算关系矩阵的传递闭包，退出 TransitiveClosure 函数。

3. TransitiveClosure 函数执行完成后，程序会询问用户是否继续计算，当用户输入正确的指令后，程序进行下一步计算或退出。

5.3 程序功能实现

5.3.1 关系矩阵输入功能

在 TransitiveClosure 函数中，用户首先需要输入集合 A 的元素个数（即关系矩阵的维度）要求输入一个正整数。程序会使用一个双精度浮点数来存储用户输入的数据然后检查用户输入的有效性。如果用户输入的值无效（例如负数、浮点数、零或者非整数），程序会提示重新输入。这个过程保证了矩阵的维度是合理的。

接着，程序会要求用户逐行输入关系矩阵的元素。关系矩阵是一个二维矩阵，矩阵的元素为 0 或 1，其中 1 表示关系成立，0 表示关系不成立。矩阵元素每次输入时可以输入一行；输入时，程序会检查当前输入元素的合法性（是否为 0 或 1），当前元素合法时，将当前元素赋给举证对应位置元素；当输入不合法时会提示输入错误信息，清除缓冲区和输入状态，并要求重新输入当前行。

5.3.2 矩阵输出功能

每当输入举证完成后或者计算出新的闭包矩阵后，程序会调用 OutputMatrix 函数将计算结果输出到屏幕上。

通过格式化输出矩阵，使得用户能够清晰地查看每个闭包的结果。

5.3.3 程序退出功能

在每次计算完成后，程序都会将询问用户是否继续进行计算，进入一个 while 循环，使用_getch（）来读取字符，当用户输入的字符为 N 或 n，退出主循环，程序结束；如果输入 Y 或 n 继续执行下一次循环，清屏后执行下一步计算。当用户输入的字符不是这四个之一时，继续读入字符，直到读入正确的字符。

5.4 核心算法实现

首先，用输入的矩阵来创建一个副本 t_matrix，它将用来存储计算得到的自反闭包，调用 Transitive 函数，通过引用的方式传递矩阵 t_matrix，并传递它的维度。

然后，进行三层嵌套循环：

外层循环 for (int k = 0; k < n; k++)：该循环用来遍历每一个可能的中间节点 k，每次选择一个节点 k 作为中间点，通过它来连接其他的节点。

中间循环 for (int i = 0; i < n; i++)：这个循环遍历每个起始节点 i，表示从节点 i 出发。

内层循环 for (int j = 0; j < n; j++)：这个循环遍历每个目标节点 j。

在循环的过程中更新矩阵，具体判断逻辑为：

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

如果 i -> j 已经有路径（t_matrix[i][j] == 1），那么就保持不变。

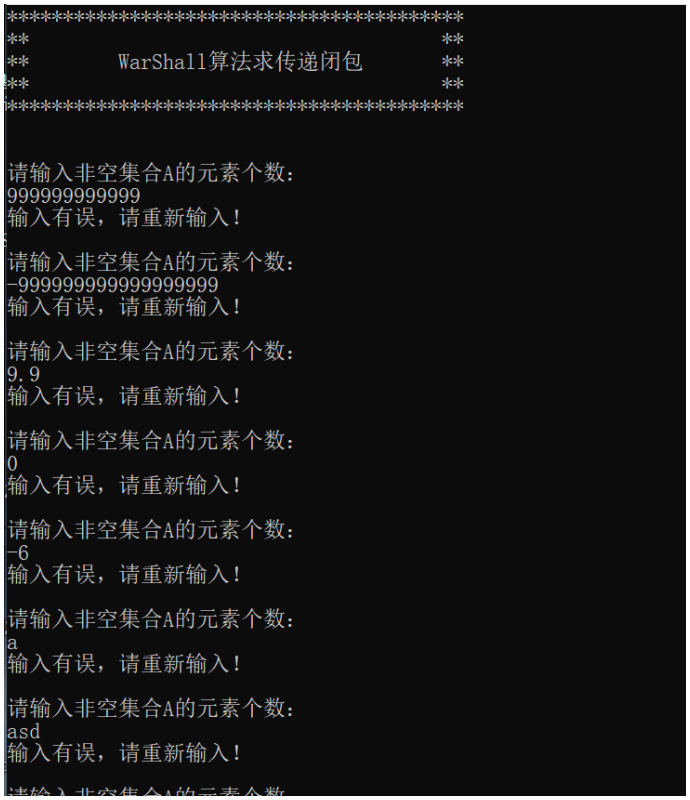
如果通过中间节点 k，即 i -> k 和 k -> j 都有路径（t_matrix[i][k] == 1 && t_matrix[k][j] == 1），那么 i -> j 也有路径，设置 t_matrix[i][j] = 1。

通过以上逻辑，可以成功将 t_matrix 修改成传递闭包。

由于通过引用传递参数，TransitiveClosure 函数中的 t_matrix 也发生了变化，即为所求关系矩阵。

六 实验结果测试

输入 A 集合中元素个数，即关系矩阵维度时，分别输入超过类型上下限的整数、浮点数、负数、零、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。



输入关系矩阵每一行时，分别输入超过类型上下限的整数、非 0 或非 1 的数、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。

```
****************************************
**                                    **
**        WarShall算法求传递闭包      **
**                                    **
****************************************


请输入非空集合A的元素个数：
3
请输入矩阵的第0行元素（元素以空格分开,只能输入0或1）：
99999999999 1 0
第0行输入有误，请重新输入！
0 -9999999999 0
第0行输入有误，请重新输入！
0 0 6
第0行输入有误，请重新输入！
1 0 1
请输入矩阵的第1行元素（元素以空格分开,只能输入0或1）：
a 5 9
第1行输入有误，请重新输入！
1 asd 1
第1行输入有误，请重新输入！
1 1.1 2
第1行输入有误，请重新输入！
1 0 1.1
第1行输入有误，请重新输入！
```

当输入正确数据时，程序可以正确计算关系矩阵结果。

```
****************************************
**                                    **
**        WarShall算法求传递闭包      **
**                                    **
****************************************


请输入非空集合A的元素个数：
2
请输入矩阵的第0行元素（元素以空格分开,只能输入0或1）：
1 1
请输入矩阵的第1行元素（元素以空格分开,只能输入0或1）：
0 0

输入的关系矩阵为：
1 1
0 0

所求传递闭包为：
1 1
0 0
```

```
****************************************
**                                    **
**        WarShall算法求传递闭包      **
**                                    **
****************************************


请输入非空集合A的元素个数：
3
请输入矩阵的第0行元素（元素以空格分开,只能输入0或1）：
0 0 1
请输入矩阵的第1行元素（元素以空格分开,只能输入0或1）：
1 0 1
请输入矩阵的第2行元素（元素以空格分开,只能输入0或1）：
0 1 1

输入的关系矩阵为：
0 0 1
1 0 1
0 1 1

所求传递闭包为：
1 1 1
1 1 1
1 1 1
```

八 实验心得

通过这次实验，我的能力和思维方式在多个方面得到了提升，同时也加深了对一些关键知识的理解。

首先，算法思维和问题分解能力得到了提升。通过实现 Warshall 算法，我学会了如何将一个问题分解为多个小问题，通过逐步迭代更新矩阵，逐渐获得最终解。这种通过中间结果逐步逼近目标的过程，不仅帮助我理解了动态规划的思想，也让我更加熟练地运用算法思想来解决问题。

其次，我的程序设计与调试能力得到了提高。在实现过程中，如何正确处理输入、如何组织代码、如何避免常见的错误（如输入无效数据时程序崩溃）等问题，都要求我注重程序的健壮性和可维护性。这让我更加注重代码的细节，理解到编写高质量代码的重要性，不仅要保证算法的正确性，还要确保程序在各种情况下都能稳定运行。

在实验中，我还深入理解了关系的计算与应用。Warshall 算法本质上是通过矩阵操作来计算关系的传递闭包，而关系在计算机科学中有广泛的应用，特别是在数据库、网络和系统分析等领域。通过这个实验，我更加清晰地理解了如何通过矩阵表示关系，以及如何通过更新关系矩阵来计算元素之间的可达性。这为我后续学习图论奠定了基础。

另外，时间复杂度和算法效率的意识也得到了加强。在实验过程中，我意识到 Warshall 算法的时间复杂度是 O(n^3)，在处理更大规模数据时，可能较为低效，促使我思考在处理大规模数据时如何选择更合适的算法。实验使我认识到算法的效率对程序的性能至关重要，尤其是在处理大规模数据时，算法的时间复杂度往往决定了程序的实际可用性。

总的来说，完成这次实验不仅加深了我对传递闭包等知识的理解，还提升了我在算法设计、程序实现和优化方面的能力。同时，它也让我意识到编写高效、健壮程序的重要性，培养了我在实际问题中如何选择和应用合适算法的能力。

附录 · 源码

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

/*************************************************************
* Function Name:   ClearBuffer
* Function:        清除输入流中的缓冲区，防止非法输入后影响后续输入
* Input Parameter: 无
* Returned Value:  无
*************************************************************/
void ClearBuffer()
{
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

/*************************************************************
* Function Name:   OutputMatrix
* Function:        输出当前的关系矩阵
* Input Parameter: matrix - 关系矩阵, n - 矩阵的维度
* Returned Value:  无
*************************************************************/
void OutputMatrix(const std::vector<std::vector<bool>>& matrix, const int n,const
char *prompt)
{
    std::cout << prompt;
    for (int i = 0; i < n; i++) {
       for (int j = 0; j < n; j++) {
          if (j!=0)
             std::cout << ' ';
          std::cout << matrix[i][j];
       }

       std::cout << "\n";
    }
}

/*************************************************************
* Function Name:   Transitive
```

```
* Function:         实现 Warshall 算法，计算关系的传递闭包
* Input Parameter: t_matrix - 输入的关系矩阵，n - 矩阵的维度
* Returned Value:  无
*************************************************************/
void Transitive(std::vector<std::vector<bool>>& t_matrix, const int n)
{
    // 通过 Warshall 算法进行传递闭包计算
    for (int k = 0; k < n; k++)          // 遍历所有中间点 k
        for (int i = 0; i < n; i++)      // 遍历起点 i
            for (int j = 0; j < n; j++) // 遍历终点 j
                // 如果 i->k 和 k->j 存在路径，则 i->j 也存在路径
                t_matrix[i][j] = ((t_matrix[i][j] || (t_matrix[i][k] &&
t_matrix[k][j])) ? 1 : 0);
}

/*************************************************************
* Function Name:    TransitiveClosure
* Function:         负责输入关系矩阵并调用传递闭包算法
* Input Parameter: 无
* Returned Value:  无
*************************************************************/
void TransitiveClosure()
{
    // 输出程序标语
    std::cout << ("*****************************************\n") //标语
              << ("**                                     **\n")
              << ("**        WarShall 算法求传递闭包       **\n")
              << ("**                                     **\n")
              << ("*****************************************\n\n\n");
    // 输入关系矩阵的维度
    double dN;
    while (true) {
        std::cout << "请输入非空集合 A 的元素个数: \n";
        std::cin >> dN;
        if (std::cin.fail() || dN <= 0||dN!=static_cast<int>(dN)) {
            std::cout << "输入有误，请重新输入！\n\n";
            ClearBuffer();
            continue;
        }
        ClearBuffer();
        break;
    }
    const int n=static_cast<int>(dN);
```

```
    // 输入关系矩阵的初始元素
    std::vector<std::vector<bool>> matrix(n, std::vector<bool>(n));//利用动态内存储
存关系矩阵
    for (int i = 0; i < n; i++) {
        std::cout << "请输入矩阵的第" << i << "行元素（元素以空格分开,只能输入 0 或
1）: \n";
        for (int j = 0; j < n; j++) {
            double temp;
            std::cin >> temp;
            if (std::cin.fail() || temp !=1 &&temp != 0) {
                std::cout << "第" << i << "行输入有误，请重新输入！\n";
                ClearBuffer();
                j = -1;
                continue;
            }
            matrix[i][j] = static_cast<bool>(temp);
        }
        ClearBuffer();
    }

    OutputMatrix(matrix, n, "\n 输入的关系矩阵为: \n");

    // 计算传递闭包
    std::vector<std::vector<bool>> t_matrix = matrix;
    Transitive(t_matrix, n);

    // 输出传递闭包矩阵
    OutputMatrix(t_matrix, n,"\n 所求传递闭包为: \n");
}

int main()
{
    while (true) {
        system("cls");//清屏
        TransitiveClosure();
        // 提示用户是否继续执行程序
        std::cout << "\n 是否继续执行程序？[Y/n] ";
        char ch;
        do {
            ch = static_cast<char>(_getch());
        } while (ch != 'Y' && ch != 'y' && ch != 'N' && ch != 'n');
        std::cout << ch << '\n';
```

```
        // 如果用户输入 N 或 n，则退出循环，结束程序
        if (ch == 'N' || ch == 'n')
            break;
    }

    std::cout << "\n 欢迎下次使用！ \n";

    return 0;
}
```