同济大学
TONGJI UNIVERSITY

离散数学

课程实验报告

关系的自反、对称、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

本实验旨在通过 C++编程实现关系的自反闭包、对称闭包和传递闭包，帮助学生深入理解关系的基本性质和闭包运算。通过本实验，学生将能够：

1.学会定义和表示集合之间的关系，理解关系的自反性、对称性、传递性等基本性质。理解如何通过编程实现对集合间关系的操作，如自反性、对称性和传递性；

2.掌握和实现自反、对称、传递关系闭包的构建方法，深入理解自反、对称和传递这三种关系属性的定义及其在数学和计算机科学中的应用；

3.深入理解关系闭包在实际问题中的应用，例如数据库完整性约束、图的路径计算等。为以后在掌握图论、数据库管理系统等领域中应用关系的闭包运算奠定基础；

4.通过使用 C++语言实现关系的闭包计算，锻炼学生的编程能力，通过编写程序来解决复杂的数学问题，提高学生在算法设计和实现方面的能力，为后续学习复杂算法打基础；

5.增强学生的数学逻辑思维能力，学习如何将离散数学中的理论应用到实际的编程问题中，培养学生解决实际问题的能力。

二 实验内容

本实验的主要内容是使用 C++程序实现关系的自反闭包、对称闭包和传递闭包。关系是离散数学中的一个重要概念，涉及集合之间的关系以及如何操作这些关系。实验中需要使用矩阵存储集合元素之间的关系，其中元素的值为 true 表示有关系，为 false 表示没有关系。学生需要设计对应算法，来计算每种闭包，并通过一系列数据检测程序正确性。学生将通过编写 C++代码，深入理解关系的基本性质和闭包操作。

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 关系的自反性

自反性是关系的基本性质之一，通常用来描述某些类型的“自我关联”现象。关系的自反性是指集合中的每个元素都与自身有关系。换句话说，对于一个集合 A 及其上的二元关系 R，如果集合 A 中的任意元素 x，都满足$(x, x) \in R$，则说明元素 x 与自身有关系。这种性质使得在任何情况下，关系 R 都会包括集合 A 中的每个元素与它自己之间的关系。

4.2 关系的对称性

关系的对称性是指如果集合中的某一元素与另一元素存在关系，那么反过来，第二个元素与第一个元素也应该存在这种关系。更正式地说，若对于集合 A 上的一个二元关系 R，如果对于任意的 $a,b \in A$，若 $(a,b) \in R(a, b)$，那么必定有 $(b,a) \in R$，那么关系 R 就被称为对称关系。对称关系在许多实际问题中都很重要。例如，在社交网络中，朋友关系显然是对称的；在计算机科学中，很多问题涉及到对称性检查，如图的对称性、网络连接性等。在数学中，对称性常用于描述等价关系、对称图等。

4.3 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。

五 实验过程

5.1 实验思路

本程序的实现核心是计算集合关系的三种闭包：自反闭包、对称闭包和传递闭包。程序通过二维动态数组来存储关系矩阵，然后分别实现三种闭包的计算。

首先，自反闭包的实现思路是，通过遍历关系矩阵的对角线元素，将所有对角线位置的元素设置为 1，确保每个元素与自身存在关系。其次，对称闭包的计算方法是，首先生成矩阵的转置矩阵，然后将原矩阵与转置矩阵进行并集操作。具体来说，若矩阵的某个元素（i，j）为 1，那么转置矩阵的（j，i）位置也一定为 1，通过对两个矩阵对应位置元素进行或运算，可以实现将原矩阵转换为对称矩阵。最后，传递闭包的实现稍显复杂，它基于传递性的定义，即如果元素 i 与元素 j 有关系，且元素 j 与元素 k 有关系，那么元素 i 与元素 k 也必须有关系。为了实现这一点，程序采用三重循环的方法，首先遍历矩阵的每一行和列，检查是否存在间接关系。若发现存在从 i 到 j 的关系和从 j 到 k 的关系，就更新矩阵中从 i 到 k 的关系。

这三个闭包计算过程依赖于原矩阵的元素逐步更新，从而最终得到满足自反性、对称性和传递性的闭包矩阵。通过这种逐步更新的方法，能够高效地生成所需的闭包结果。

5.2 程序框架设计

1.程序首先进入一个 do-while 循环，调用清屏指令，然后调用 Relations 函数，根据其返回值判断是否继续执行循环。

2. Relations 函数主要用来进行矩阵相关的输入工作，并通过用户选择来调用不同函数。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，程序都会验证用户用户输入的合法性。

然后，提供用户选择不同算法的菜单，允许用户选择计算自反闭包、对称闭包、传递闭包或退出程序。

最后，根据用户的选择，程序选择执行 Reflexive、Symmetric、Transitive 函数来分别计算自反闭包、对称闭包、传递闭包，计算完成后，通过 OutputMatrix 输出计算后的闭包矩阵，函数返回；当用户选择重新定义关系矩阵时，函数返回 true，继续执行新的 main 函数中的循环循环；当用户选择退出时，函数返回 false，退出循环。

5.3 程序功能实现

5.3.1 关系矩阵输入功能

在 Relations 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 用户操作选择功能

在用户输入矩阵后，程序将进入一个循环，等待用户选择所需执行的操作。Select 函数提供了一个菜单，用户可以选择以下几种操作：

- 计算自反闭包
- 计算对称闭包
- 计算传递闭包
- 重新定义关系矩阵
- 退出程序

每次用户输入一个有效的操作编号，程序会执行相应的闭包计算。若用户选择“重新定义关系矩阵”，程序会退出当前函数并返回 true，通过主函数 do-while 循环重新清屏、调用 Relations 函数；若用户选择“退出程序”，程序会退出当前函数并返回 false，从而退出主函数循环，继而退出程序。

5.3.3 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。

5.4 核心算法实现

5.4.1 自反闭包的计算

自反闭包的核心目标是确保关系矩阵中的每个元素都与自己存在关系。这意味着，对于矩阵中的每个元素 (i, i)，该元素的值必须设置为 1。即每个元素必须自反。

首先，用输入的矩阵来创建一个副本 r_matrix，它将用来存储计算得到的自反闭包，调用 Reflexive 函数，通过引用的方式传递矩阵 r_matrix，并传递它的维度。然后，通过遍历矩阵的主对角线（即所有 (i, i) 的元素），在遍历过程中，将所有主对角线上的元素的值设置为 1，确保了自反性。

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

5.4.2 对称闭包的计算

对称闭包的目标是确保关系矩阵中每对存在关系的元素 (i, j) 和 (j, i) 都被设置为 1，即如果 i 与 j 有关系，那么 j 与 i 也必须有关系。

首先，用输入的矩阵来创建一个副本 s_matrix，它将用来存储计算得到的自反闭包，调用 Symmetric 函数，通过引用的方式传递矩阵 s_matrix，并传递它的维度。然后，创建 s_matrix 矩阵的转置 transpose，即 transpose[i][j] = matrix[j][i]。接着，对于原矩阵 s_matrix 和转置矩阵 transpose，将它们对应位置上的元素做并集操作，将计算结果存储在 s_matrix 矩阵中。

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

5.4.3 传递闭包的计算

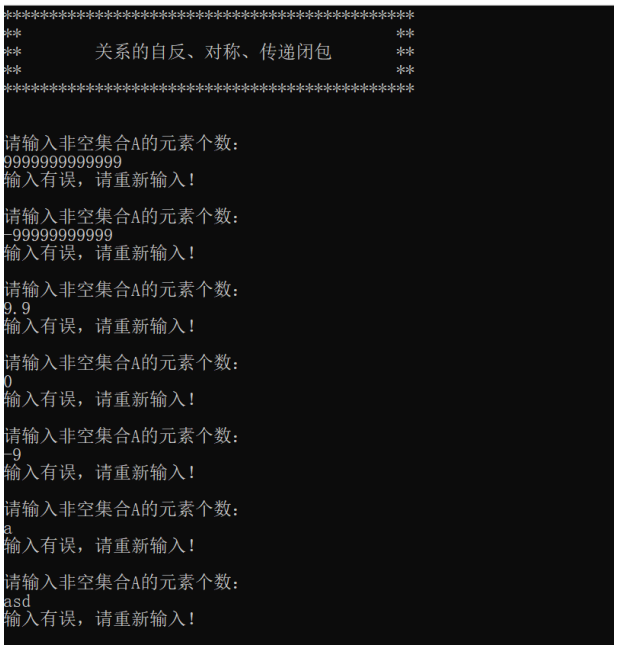
传递闭包的核心目标是确保矩阵中所有间接关系也被直接表达出来。如果 a 与 b 有关系，且 b 与 c 有关系，那么 a 必须与 c 也有关系。

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，基于矩阵的三个索引：i、j 和 k。我们遍历所有可能的三元组 (i, j, k)，其中 i、j 和 k 都是矩阵的行列索引。具体来说，如果 t_matrix[j][i] != 0，即存在从 j 到 i 的关系，我们检查是否有从 i 到 k 的关系。如果存在，即 t_matrix[i][k]=1，那么 t_matrix[j][k] 也应该被设置为 1，表示从 j 到 k 也存在间接关系。

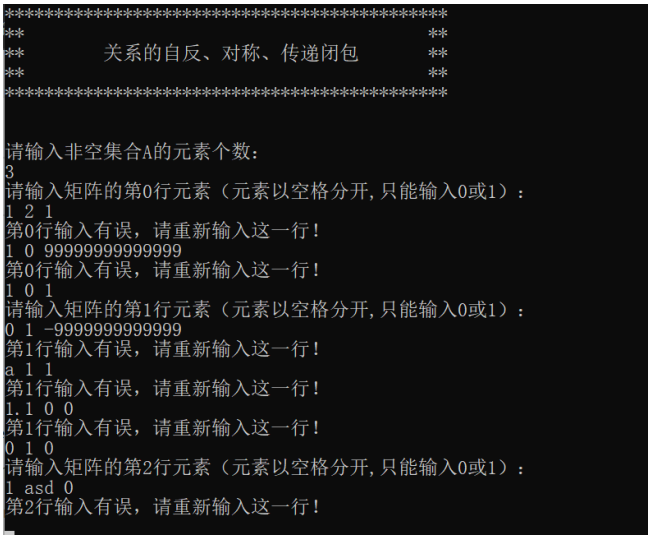
由于通过引用传递参数，Relations 函数中的 t_matrix 也发生了变化，即为所求关系矩阵。

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。



当输入正确数据时，程序可以正确计算关系矩阵结果。

```
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

输入对应序号选择算法 [0]:退出程序 [1]:自反闭包 [2]:传递闭包 [3]:对称闭包 [4]:重新定义关系矩阵
1

所求自反闭包为：
1 1
0 1

输入对应序号选择算法 [0]:退出程序 [1]:自反闭包 [2]:传递闭包 [3]:对称闭包 [4]:重新定义关系矩阵
2

所求传递闭包为：
1 1
0 0

输入对应序号选择算法 [0]:退出程序 [1]:自反闭包 [2]:传递闭包 [3]:对称闭包 [4]:重新定义关系矩阵
3

所求对称闭包为：
1 1
1 0
```

```
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

输入对应序号选择算法 [0]:退出程序 [1]:自反闭包 [2]:传递闭包 [3]:对称闭包 [4]:重新定义关系矩阵
1

所求自反闭包为：
1 0 1
1 1 1
0 1 1

输入对应序号选择算法 [0]:退出程序 [1]:自反闭包 [2]:传递闭包 [3]:对称闭包 [4]:重新定义关系矩阵
2

所求传递闭包为：
1 1 1
1 1 1
1 1 1

输入对应序号选择算法 [0]:退出程序 [1]:自反闭包 [2]:传递闭包 [3]:对称闭包 [4]:重新定义关系矩阵
3

所求对称闭包为：
0 1 1
1 0 1
1 1 1
```

八 实验心得

在进行这次实验的过程中，我深入理解了关系的三种基本闭包：自反闭包、对称闭包和传递闭包，并通过程序实现了它们的计算。这个实验不仅让我更加熟悉了离散数学中关于关系和闭包的概念，还加深了我对矩阵操作和算法实现的理解。

首先，编写程序时，我需要对关系矩阵进行细致的处理：在自反闭包的计算中，通过确保每个元素与自己有关系来完成闭包，对称闭包的计算要求我对矩阵进行转置并求并集，传递闭包则涉及到了更复杂的三重循环，这些操作都需要我细心地处理矩阵的每个元素，确保每次操作都符合数学定义。

在实现过程中，我还遇到了一些实际的编程挑战。例如，如何有效地处理用户输入，确保输入合法，并根据用户的选择执行不同的闭包操作。通过调试程序，我学到了如何利用条件判断和循环结构来提高程序的鲁棒性，减少因用户输入错误的导致程序崩溃的概率。

通过这次实验，我不仅在数学上对闭包的理解更加深入，而且在编程方面也收获了很多。实现这些算法让我对如何将数学问题转化为编程问题有了更多的思考，特别是如何优化算法，减少不必要的计算。此外，实验中的一些小细节，如清屏操作、缓冲区清理等，也让我更加注重程序的细节和用户体验。

总体来说，这次实验是一次非常有价值的学习经历，它不仅增强了我的算法设计能力，也提高了我的编程技巧和解决实际问题的能力。在今后的学习中，我将继续深入研究离散数学中的其他内容，并努力将这些理论与实际编程技能相结合，解决更复杂的问题。

附录 · 源码

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

/*************************************************************
* Function Name:   ClearBuffer
* Function:        清除输入缓冲区，防止无效输入影响后续操作
* Input Parameter: None
* Returned Value:  None
*************************************************************/
void ClearBuffer()
{
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

/*************************************************************
* Function Name:   Select
* Function:        提供用户选择操作的选项，输入序号选择不同算法
* Input Parameter: None
* Returned Value:  用户选择的序号
*************************************************************/
int Select()
{
    double select;
    while (true) {
       std::cout << "\n 输入对应序号选择算法 [0]:退出程序 [1]:自反闭包 [2]:传递闭包
[3]:对称闭包 [4]:重新定义关系矩阵\n";
       std::cin >> select;
       if (std::cin.fail() || select > 4 || select <
0||select!=static_cast<int>(select)) {
          std::cout << "输入有误，请重新输入！\n\n";
          ClearBuffer();
          continue;
       }
       ClearBuffer();
       break;
    }
    return static_cast<int>(select);
```

```
}

/*************************************************************
* Function Name:   OutputMatrix
* Function:        输出矩阵结果
* Input Parameter: const std::vector<std::vector<int>>& matrix: 关系矩阵
*                  const int n: 矩阵维数
* Returned Value:  None
*************************************************************/
void OutputMatrix(const std::vector<std::vector<int>>& 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:   Reflexive
* Function:        计算并生成自反闭包
* Input Parameter: std::vector<std::vector<int>>& r_matrix: 自反闭包
*                  const int n: 矩阵维数
* Returned Value:  None
*************************************************************/
void Reflexive(std::vector<std::vector<int>>& r_matrix, const int n)
{
    for (int i = 0; i < n; i++)
       r_matrix[i][i] = 1;
}

/*************************************************************
* Function Name:   Symmetric
* Function:        计算并生成对称闭包
* Input Parameter: std::vector<std::vector<int>>& s_matrix: 对称闭包
*                  const int n: 矩阵维数
* Returned Value:  None
*************************************************************/
```

```
void Symmetric(std::vector<std::vector<int>>& s_matrix, const int n)
{
    // 生成矩阵的转置矩阵
    std::vector<std::vector<int>> transpose(n, std::vector<int>(n));
    for (int i = 0; i < n; i++)
       for (int j = 0; j < n; j++)
          transpose[j][i] = s_matrix[i][j];
    // 将原矩阵与转置矩阵求并集
    for (int i = 0; i < n; i++)
       for (int j = 0; j < n; j++)
          s_matrix[i][j] = s_matrix[i][j] + transpose[i][j] >= 1 ? 1 : 0;
}

/*************************************************************
* Function Name:   Transitive
* Function:        计算并生成传递闭包
* Input Parameter: std::vector<std::vector<int>>& t_matrix: 传递闭包
*                  const int n: 矩阵维数
* Returned Value:  None
*************************************************************/
void Transitive(std::vector<std::vector<int>>& t_matrix, const int n)
{
    for (int i = 0; i < n; i++)
       for (int j = 0; j < n; j++)
          if (t_matrix[j][i] != 0)
             for (int k = 0; k < n; k++)
                t_matrix[j][k] = t_matrix[j][k] + t_matrix[i][k] >= 1 ? 1 : 0;

}

/*************************************************************
* Function Name:   Relations
* Function:        用户界面和矩阵输入及操作选择流程
* Input Parameter: None
* Returned Value:  None
*************************************************************/
bool Relations()
{
    // 输出程序标语
    std::cout << ("*********************************************\n") //标语
              << ("**                                         **\n")
              << ("**        关系的自反、对称、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<int>> matrix(n, std::vector<int>(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, 请重新输入这一行! \n";
                ClearBuffer();
                j = -1;
                continue;
            }
            matrix[i][j] = static_cast<int>(temp);
        }
        ClearBuffer();
    }

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

    // 根据用户选择执行不同的闭包操作
    while (true) {
        std::vector<std::vector<int>> c_matrix = matrix;
        int select = Select();
        switch (select) {
            case 1:
                Reflexive(c_matrix, n);
```

```
            OutputMatrix(c_matrix,n,"\n 所求自反闭包为: \n");
        break;
        case 2:
           Transitive(c_matrix, n);
            OutputMatrix(c_matrix, n,"\n 所求传递闭包为: \n");
        break;
        case 3:
           Symmetric(c_matrix, n);
            OutputMatrix(c_matrix, n,"\n 所求对称闭包为: \n");
        break;
        case 4:
           return true;
        case 0:
           default:
              std::cout << "欢迎下次再次使用!\n";
        return false;
      }
   }

}

/*************************************************************
* Function Name:   main
* Function:        通过调用 Relations 函数, 实现与用户的交互
* Input Parameter: None
* Returned Value:  0
*************************************************************/
int main()
{
   do {
      system("cls"); //清屏
   }while (Relations());

   return 0;
}
```