《离散数学》课程实验报告 6 用 Warshall 算法求关系的传递闭包

2253893 苗君文 软件工程

1. 题目简介

1.1. 背景与目的

在离散数学领域中，研究集合和关系的性质是非常重要的。关系是集合之间的一种对应关系，而传递闭包是关系中的一种特殊性质。传递闭包反映了关系中元素之间的传递性，对于分析集合之间的依赖关系、路径关系等具有广泛的应用。Warshall 算法是一种经典的计算传递闭包的算法，通过其高效的时间复杂度，能够在实际问题中得到广泛应用。

本实验旨在通过编程，深入理解 Warshall 算法在求解关系的传递闭包中的应用。通过输入集合 A 和关系 R，利用 Warshall 算法计算 R 的传递闭包，最终以集合形式输出传递闭包。通过这个实验，可以更好地理解关系的传递性质，并掌握 Warshall 算法。

1.2. 问题描述

设计一个程序，通过 Warshall 算法，求解给定关系的传递闭包。程序需要处理集合和关系的输入，并输出传递闭包的结果。

1.3. 程序输入输出

1.3.1. 输入

用户通过控制台输入集合 A 的元素个数、集合 A 的具体元素、关系 R 的元素个数和关系 R 的具体元素。输入时程序能保证输入错误时不崩溃，且可以做出相应的错误处理。

1.3.2. 输出

程序将输出传递闭包的结果，以集合形式表示。

2. 解题思路

2.1. 明确传递闭包的定义

设 R 是非空集合 A 上的关系， R 的传递闭包是 A 上的关系 R′， 使得 R′满足以下条件：

(1) R ′是传递的

(2) R ⊆ R ′

(3) 对 A 上任何包含 R 的传递关系 R ″ 有 R ′⊆ R ″.

一般将 R 的传递闭包记作 t (R).

2.2. 传递闭包的数学构造方法

2.2.1. 集合表示

设 R 为 A 上的关系，则有 $t(R)=R\cup R^2\cup R^3\cup\cdots$，对于有穷集合 A (|A|=n) 上的关系，并最多不超过 R^n。若 R 是传递的，则 t(R)=R.

2.2.2. 矩阵表示

设关系 R, t(R)的关系矩阵分别为 M, Mt ，则 $Mt=M+M^2+M^3+\cdots$，注意：在上述等式中矩阵的元素相加和相乘时都使用逻辑加。

2.3. 理解 Warshall 算法的原理

Warshall 算法是一种经典的图算法，用于计算图中所有顶点对之间的最短路径或检测图中的传递性闭包。在关系代数和集合论中，Warshall 算法经常被用于计算关系的传递闭包。

2.3.1. Warshall 算法的基本思想

给定一个图（或关系矩阵），Warshall 算法通过逐步改进矩阵来找到所有节点对之间的最短路径或计算传递闭包。其基本思想如下：

（1）初始化矩阵：将矩阵的初始值设置为图的邻接矩阵。如果存在直接的边连接两个顶点，则相应的矩阵元素为 1，否则为 0。

（2）迭代更新矩阵：对于每一个节点对(i, j)，检查是否存在一个中间节点 k，使得从 i 到 k 和从 k 到 j 都存在路径。如果存在这样的 k，将 i 和 j 之间的路径标记为存在。迭代执行这个过程，逐步更新矩阵。

（3）传递闭包的计算：对于关系矩阵，Warshall 算法最终得到的矩阵描述了图中所有节点对之间的可达性。如果矩阵的元素 t[i][j]为 1，表示存在一条从顶点 i 到顶点 j 的路径。

2.3.2. 对 Warshall 算法的举例说明

已知 A={a,b,c,d},R={(a,b),(b,d),(d,a),(d,c)}，求 R 的传递闭包。

先写出 R 的关系矩阵：

$$R^{(0)}=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

选出第 1 行与第 1 列，用 R_{i1} 与 R_{1j} 做或运算，如果运算结果为 1，则对应的 R_{ij} 需要更新为 1：

$$R^{(0)}=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

可以看出（d,a）对应的值变为 1，而后类似地对第 2 行与第 2 列，第 3 行与第 3 列，第 4 行与第 4 列做类似的运算，得出相应的 $R^{(1)}$，$R^{(2)}$,$R^{(3)}$,$R^{(4)}$：

$$R^{(1)}=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 1 & 0 \end{bmatrix} \end{array}$$

$$R^{(2)}=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & \mathbf{1} \end{bmatrix} \end{array}$$

$$R^{(3)}=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array} \qquad R^{(4)}=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} \mathbf{1} & 1 & \mathbf{1} & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$T=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

最后得出传递闭包对应的矩阵为：

2.4. Warshall 算法的伪代码描述

（1）置新矩阵 M: = AR（AR 表示集合 A 的二元关系 R 的集合）

（2）置 j: = 1

（3）对所有 i，如果 M[i,j] = 1，则对 k = 1,2，…，n，置

M[i,k]: = M[i,k] + M[j,k]

（4）j: = j + 1

（5）如果 j <= n，则转到步骤（3），否则停止

这段伪代码描述了 Warshall 算法的基本步骤。在迭代过程中，算法通过检查是否存在中间节点来逐步更新传递闭包矩阵。

2.5. 界面设计

本程序为方便使用者调试与使用，增设了让用户判断是否需要继续运行程序的环节。使用循环与判断语句，并做了详尽的输入错误处理。

3. 所用数据结构

以下两个数据结构分别用于存储输入的关系和计算传递闭包的中间结果。结构体 BR 方便表示和存储二元关系，而二维数组 bool** tR 则用于表示关系的传递闭包矩阵。

3.1. 结构体 BR（Binary Relation）

该结构体是用于表示二元关系的结构体。其中含有两个成员变量：其中 a 表示关系中的第一个元素。b 表示关系中的第二个元素。

```
//用结构体来表示二元关系
typedef struct{
    char a;
    char b;
}BR;
```

3.2. 二维动态数组 bool** tR

该二维动态数组用于表示关系的传递闭包矩阵。动态分配内存以创建一个布尔类型的二维数组。其中的元素 tR[i][j] 代表关系的传递闭包中，元素 A[i] 和 A[j] 之间是否存在传递关系。

4. 核心算法

4.1. 初始化集合 A 的函数

该函数 init_aggregation 的功能是初始化集合 A。以下是对该函数的详细描述：

首先，用户需要输入集合 A 中的元素个数，要求输入一个正整数，并按回车键确认。这一步通过调用 dealInputError 函数来确保用户输入的是合法的正整数。

接下来，根据用户输入的元素个数，动态分配内存创建一个字符数组 A，用于存储集合 A 的元素。用户被要求依次输入集合 A 中的元素，每个元素之间以空格分隔，按回车键确认。同样确保输入正确，否则重新输入。

通过这个函数，用户可以方便地初始化集合 A，并在控制台上逐个输入集合元素，确保输入的是合法的字母。

4.2. 初始化二元关系 R 的集合的函数

该函数 init_BinaryRelation 的功能是创建集合 A 的二元关系 R 的集合并初始化。以下是对该函数的详细描述：

首先，用户需要输入二元关系 R 中的元素个数，要求输入一个正整数，并按回车键确认。同样通过调用 dealInputError 函数来确保输入合法。

然后，根据用户输入的元素个数，动态分配内存创建一个 BR 结构体数组 R，用于存储二元关系 R 的元素。

接着，用户需要依次输入二元关系 R 中的元素，每行一个元素对，每个元素对由两个字母表示,用空格分隔,按回车键确认,也可以完成相应的错误处理。

4.3. Warshall 算法的核心部分

首先，通过函数参数传递的 bool**& tR，表示关系 R 的传递闭包矩阵。初始化传递闭包矩阵，将其所有元素置为 0。

其次，对于输入的二元关系 R 中的每个元素，将其转化为传递闭包矩阵 tR 中的 1。通过 fun 函数找到元素对应的索引。

而后，利用 Warshall 算法的思想，通过检索行、列，逐步更新传递闭包矩阵。如果矩阵中某一位置为 1，表示存在一条边，就将该行的所有元素与对应列的元素相加，更新传递闭包矩阵。最终得到关系的传递闭包。

```
//Warshall 算法的核心部分
void Warshall(char*& A, BR*& R, bool**& tR)
{
    int i, j, k;
    int x, y;
    //用关系矩阵表示二元关系 R
    for (i = 0; i < m; i++){
        x = fun(R[i].a, A);
        y = fun(R[i].b, A);
        tR[x][y] = 1;
    }
    //计算传递闭包的过程
    for (i = 0; i < n; i++){ //检索列
        for (j = 0; j < n; j++){ //检索行
            if (tR[j][i] == 1){
                for (k = 0; k < n; k++){
                    tR[j][k] = tR[j][k] + tR[i][k];
                }
            }
        }
    }
}
```

4.4. 传递闭包的关系矩阵转化为集合表示

该函数 translation_output 的功能是将传递闭包 t(R) 的关系矩阵表示转化为集合表示，并输出。

首先，通过遍历传递闭包矩阵 tR，统计闭包中的元素个数（即矩阵中值为 1 的元素个数），以确保输出集合时逗号显示正确。

再利用两重循环遍历传递闭包矩阵，输出集合表示的元素。对于值为 1 的元素，输出 <A[i], A[j]> 形式的关系对，表示集合中的一个元素。

4.5. 输入错误处理相关函数

4.5.1. 单个参数的输入错误处理

函数 dealInputError 使用 while 循环不断尝试获取用户输入，直到输入满足要求。使用 cin.fail() 及 min、max 来判断输入是否出错，检查输入是否在有效范围内。如果输入无效，则输出错误信息，清除输入缓冲区并忽略之后的字符。通过一个字符来获取输入一个数之后的字符以检查输入字符的个数，若不正确，也输出错误信息，执行相应清除操作。如果输入有效，则跳出循环。

```
//处理输入错误
void dealInputError(int& n, int min = 1, int max = 1024)
{
    while (1) {
        char c;  //用来获取输入一个数之后的一个字符，若获取一个字符且非换行符则视为输入错误。
        cin >> n;
        if (cin.fail() || n<min || n>max) {
            cout << "输入错误，请重新输入。" << endl;
            cin.clear();
            cin.ignore(9999, '\n');
            continue;
        }  //内容错误
        else if (cin.get(c) && c != '\n') {
            cout << "输入错误，请重新输入。" << endl;
            cin.clear();
            cin.ignore(9999, '\n');
            continue;
        }  //个数错误
        else
            break;
    }
    return;
}
```

此函数可以用于判断输入的集合 A 中元素的个数的正误，二元关系 R 中的元素个数的正误。

4.5.2. 集合 A 中元素输入的错误处理

根据已知的 A 中元素的个数使用循环，输入每个元素，如果输入的字符不是字母，则视为输入内容错误；如果输入的个数超过已知的 A 中元素的个数，则视为输入的个数错误，二者需要重新输入，直到输入正确。

4.5.3. 二元关系 R 的集合输入的错误处理

使用循环嵌套，让用户逐行输 R 中的元素。此处不仅要保证每次输入的两个元素都是字母，还要保证输入的字母是集合 A 中的元素，否则输入错误；同样地，如果输入的元素个数超过两个，也是为输入错误，都需要重新输入，直到输入正确。

其中，判断输入的字符为集合 A 中的元素，用自定义的函数 strchr 来完成。该函数返回值类型为 bool，找到则为 true，否则返回 false。

```
//判断字符c是否在字符串str中
bool my_strchr(const char* str, int c)
{
    while (*str != '\0') {
        if (*str == c) {
            return true;
        }
        ++str;
    }
    return false;
}
```

5. 心得体会

5.1. Warshall 算法的理解

通过实现 Warshall 算法求解传递闭包，我对这一算法有了更深刻的理解。Warshall 算法通过构建关系矩阵，迭代计算传递闭包的过程，是一种高效的方法，尤其适用于有向图的传递性计算。Warshall 算法的时间复杂度为 O(n^3)，相较于其他传递闭包计算方法表现出较好的性能。

Warshall 算法在处理密集图时效果显著。密集图是图中的边相对较多，即关系矩阵中的大部分元素都是非零。在这种情况下，Warshall 算法的计算效率往往明显优于其他算法。而且算法实现相对简单，降低了程序的复杂性，提高了代码的可读性和可维护性。

5.2. 错误处理的提升

在用户输入部分，我实现了一些错误处理机制，例如对输入的元素类型进行检查、对输入范围进行限制、对输入的元素的个数的限定等。这使得程序更加健

壮，对用户更加友好。使我对错误处理的方法有了更精进的认识，这在实际项目中也尤为重要。

6. 测试结果

6.1. 输入集合 A 中的元素个数

（1）输入的值错误

```
**********************************************
**                                          **
**          欢迎进入用Warshalls算法         **
**          求解关系的传递闭包程序          **
**                                          **
**********************************************

请输入集合 A 中的元素个数(正整数)，按回车键输入下一项:
-3
输入错误，请重新输入。
```

（2）输入的个数错误

```
5 2
输入错误，请重新输入。
```

（3）输入内容错误

```
d
输入错误，请重新输入。
```

（4）输入内容及个数错误

```
9 k
输入错误，请重新输入。
```

（5）输入正确则可以进入下一环节

```
5
请依次输入集合 A 中的5个元素(形如：a b c d ......这样的格式)，按回车键输入下一项:
```

6.2. 输入集合 A 中的元素

（1）输入的内容不是字母

```
请依次输入集合 A 中的5个元素(形如：a b c d ......这样的格式)，按回车键输入下一项:
6 a b
输入错误，请重新输入。
```

（2）输入的个数超过 A 中本应有的个数

```
请依次输入集合 A 中的5个元素(形如：a b c d ......这样的格式)，按回车键输入下一项：
6 a b
输入错误，请重新输入。
a b c d e f
输入错误，请重新输入。
```

（3）输入正确则可以进入下一环节

```
请依次输入集合 A 中的5个元素(形如：a b c d ......这样的格式)，按回车键输入下一项：
a b c d e
请输入二元关系 R 中的元素个数(正整数)，按回车键输入下一项：
```

6.3. 输入二元关系 R 中的元素个数

（1）输入内容错误

```
请输入二元关系 R 中的元素个数(正整数)，按回车键输入下一项：
a b c
输入错误，请重新输入。
```

（2）输入的值不在 1-n*n 之间（n 为 A 中的元素个数）

```
请输入二元关系 R 中的元素个数(正整数)，按回车键输入下一项：
a b c
输入错误，请重新输入。
26
输入错误，请重新输入。
```

（3）输入的个数错误

```
请输入二元关系 R 中的元素个数(正整数)，按回车键输入下一项：
a b c
输入错误，请重新输入。
26
输入错误，请重新输入。
6 4
输入错误，请重新输入。
```

（4）输入正确则可以进入下一环节

```
6
请依次输入 R 中的6个元素，一行是一个元素
(形如：
a b
b c
c d
......
这样的格式)，按回车键输入下一项：
请输入第1个元素：
```

6.4. 依次输入二元关系 R 中的元素

（1）输入的内容不是集合 A 中的元素（如：输入的为非字母字符，或是字母但不在 A 中，输入的个数超过两个等情况）

```
请依次输入 R 中的6个元素，一行是一个元素
(形如:
a b
b c
c d
......
这样的格式)，按回车键输入下一项:
请输入第1个元素: 3 6
输入错误，请重新输入正确的元素。
请输入第1个元素: a f
输入错误，请重新输入正确的元素。
请输入第1个元素: a b c
输入错误，请重新输入正确的元素。
请输入第1个元素: a 3
输入错误，请重新输入正确的元素。
请输入第1个元素:
```

（2）输入正确即可进入下一环节

```
请输入第1个元素: a b
请输入第2个元素: b c
请输入第3个元素: c c
请输入第4个元素: c d
请输入第5个元素: e a
请输入第6个元素: e d
```

6.5. 输出 R 的传递闭包（集合形式）

```
R 的传递闭包(集合形式)为:
t(R) = {<a,b>,<a,c>,<a,d>,<b,c>,<b,d>,<c,c>,<c,d>,<e,a>,<e,b>,<e,c>,<e,d>}
```

6.6. 选择是否继续运行程序

（1）输入的内容错误（多种情况，具体如下）

```
是否继续运行该程序？（y/n）：3
输入无效，请重新输入。

是否继续运行该程序？（y/n）：36
输入无效，请重新输入。

是否继续运行该程序？（y/n）：j6
输入无效，请重新输入。

是否继续运行该程序？（y/n）：a
输入无效，请重新输入。

是否继续运行该程序？（y/n）：
```

（2）输入 y/Y/n/N 均输入正确（下图以 y 为例）

```
是否继续运行该程序？（y/n）：y

请输入集合 A 中的元素个数(正整数)，按回车键输入下一项：
```

6.7. 输入正确时的总览

```
**********************************************
**                                          **
**          欢迎进入用Warshalls算法         **
**          求解关系的传递闭包程序          **
**                                          **
**********************************************

请输入集合 A 中的元素个数(正整数)，按回车键输入下一项：
5
请依次输入集合 A 中的5个元素(形如：a b c d ......这样的格式)，按回车键输入
a b c d e
请输入二元关系 R 中的元素个数(正整数)，按回车键输入下一项：
6
请依次输入 R 中的6个元素，一行是一个元素
(形如：
a b
b c
c d
......
这样的格式)，按回车键输入下一项：
请输入第1个元素：a b
请输入第2个元素：b c
请输入第3个元素：c c
请输入第4个元素：c d
请输入第5个元素：e a
请输入第6个元素：e d

R 的传递闭包(集合形式)为：
t(R) = {<a,b>,<a,c>,<a,d>,<b,c>,<b,d>,<c,c>,<c,d>,<e,a>,<e,b>,<e,c>,<e,d>}

是否继续运行该程序？（y/n）：y
```

```
请输入集合 A 中的元素个数(正整数)，按回车键输入下一项：
3
请依次输入集合 A 中的3个元素(形如：a b c d ......这样的格式)，按回车键输入下一项：
a b c
请输入二元关系 R 中的元素个数(正整数)，按回车键输入下一项：
4
请依次输入 R 中的4个元素，一行是一个元素
(形如：
a b
b c
c d
......
这样的格式)，按回车键输入下一项：
请输入第1个元素：a a
请输入第2个元素：a c
请输入第3个元素：b c
请输入第4个元素：c b

R 的传递闭包(集合形式)为：
t(R) = {<a,a>,<a,b>,<a,c>,<b,b>,<b,c>,<c,b>,<c,c>}

是否继续运行该程序？（y/n）：n


D:\同济\学习\大二（上）\离散数学\我的dm大作业\hw6\Debug\hw6.exe (进程 39260)已退出，代码为 0。
按任意键关闭此窗口. . .
```