同濟大學

《离散数学课程项目文档》

——Warshall 算法求关系的传递闭包

作 者 姓 名： 胡峻玮

学 号： 2153393

指 导 教 师： 唐剑锋

学院、 专业： 软件学院 软件工程

目　录

1.项目分析

1.1 项目背景

假设集合 A 上存在一个非空关系 R，人们常常期望这个关系 R 具备某些有益的特性，比如自反性（或者对称性、传递性）。为了赋予 R 这些特性，需要向 R 中加入若干有序对，构造出一个新的关系 R'。目的是让 R'拥有所需的属性，同时又不希望 R'过于庞大。也就是说，希望能够加入尽可能少的有序对。满足这一条件的 R'被称为 R 的自反（或对称、传递）闭包。

1.2 项目要求

手动输入矩阵阶数（元素个数），然后手动输入关系矩阵，程序需要能使用 Warshall 算法求解该关系的传递闭包。

2.项目设计

2.1 数据结构设计

由项目分析可以得出，该项目需要完成关系的闭包求解。由于求解需要频繁涉及元素的直接访问、赋值等，因此可以使用二维数组存储关系矩阵，从而表示二元关系。由于一般需要推理时使用到的命题数量较少，一般的数组可以做到，故本题不再额外使用新的数据结构。

2.2 算法设计

2.2.1 算法思路

通过项目分析得知，计算关系的传递闭包等同于求解关系图的可达矩阵，也就是说，如果点 A 可以到达点 B，则对应的矩阵元素设为 1，反之则为 0。假定一个矩阵中第 i 行第 j 列的元素表示为 M[i,j]，如果 M[i,k]和 M[k,j]都是 1，那么根据传递性，M[i,j]也应当是 1。基于这个原理，我们可以设计以下算法步骤：

1. 标出矩阵的第 k 行和第 k 列，这相当于以元素 M[k,k]为核心绘制一个十字形。

2. 检查十字形之外的所有元素 M[i,j]，如果 M[i,j]=0 且 M[i,k]=1 以及 M[k,j]=1，那么就将 M[i,j]更新为 1；如果 M[i,j]已经是 1，表示 i 到 j 的路径已经是可达的，无需改动。

3. 递增 k 的值，并重复步骤 1 和 2，直至 k 从 0 增加到 n-1，此时矩阵变为所求的传递闭包矩阵。

以求解如下关系（$R^{(0)}$就是原矩阵）的传递闭包为例：

$$R^{(0)}=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&0&1&0\end{bmatrix}\quad \dashrightarrow \quad R^{(1)}=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&\boxed{1}&1&0\end{bmatrix}$$

标记第 0 行第 0 列，检查所有非十字上且为 0 元素。

M[3,1]=0 && M[3,0]=1 && M[0,1]=1，因此把 M[3,1]置为 1

$$R^{(1)}=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&1&1&0\end{bmatrix}\quad \dashrightarrow \quad R^{(2)}=\begin{bmatrix}0&1&0&\boxed{1}\\0&0&0&1\\0&0&0&0\\1&1&1&\boxed{1}\end{bmatrix}$$

标记第 1 行第 1 列，检查所有非十字上且为 0 元素。

M[0,4]=0 && M[0,1]=1 && M[1,4]=1，因此把 M[0,4]置为 1

M[4,4]=0 && M[4,1]=1 && M[1,4]=1，因此把 M[4,4]置为 1

$$R^{(2)}=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix} \longrightarrow R^{(3)}=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

标记第 2 行第 2 列，检查所有非十字上且为 0 元素。

没有符合要求的元素

$$R^{(3)}=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix} \longrightarrow R^{(4)}=\begin{bmatrix}\boxed{1}&1&\boxed{1}&1\\\boxed{1}&\boxed{1}&\boxed{1}&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

标记第 3 行第 3 列，检查所有非十字上且为 0 元素。

M[0,0]=0 && M[0,3]=1 && M[3,0]=1，因此把 M[0,0]置为 1

M[0,2]=0 && M[0,3]=1 && M[3,2]=1，因此把 M[0,2]置为 1

M[1,0]=0 && M[1,3]=1 && M[3,0]=1，因此把 M[1,0]置为 1

M[1,1]=0 && M[1,3]=1 && M[3,1]=1，因此把 M[1,1]置为 1

M[1,2]=0 && M[1,3]=1 && M[3,2]=1，因此把 M[1,2]置为 1

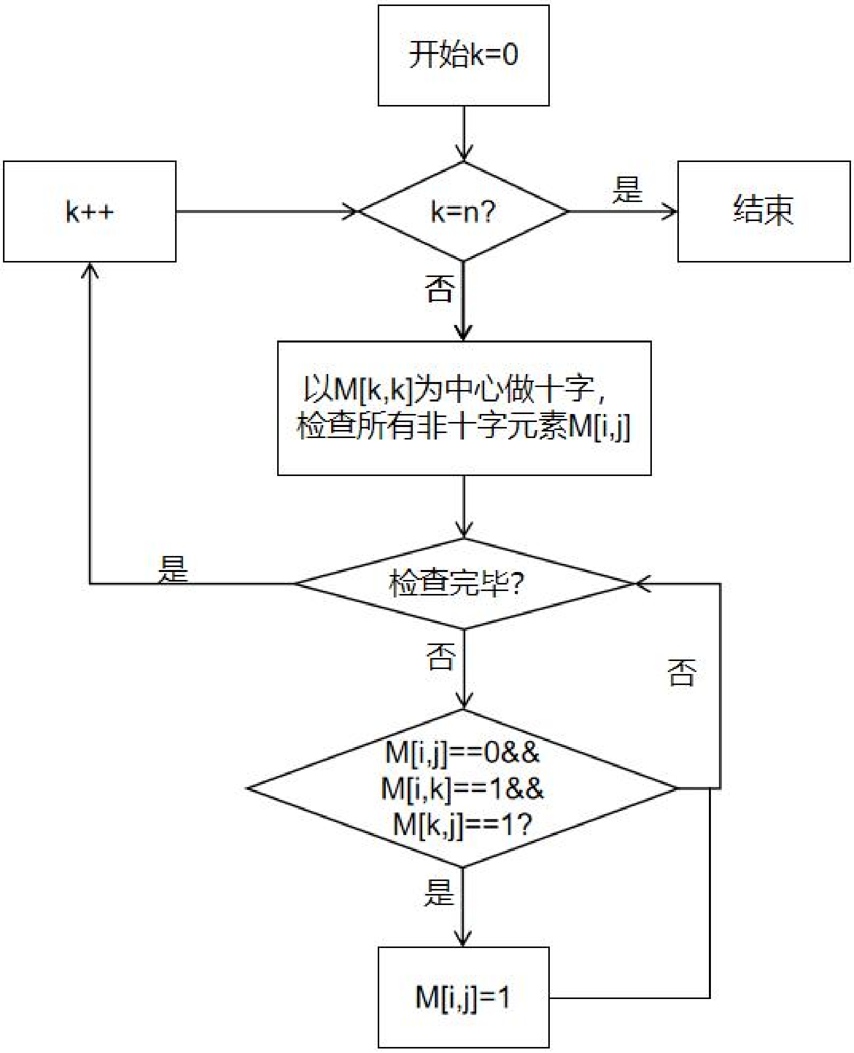
所得 $R^{(4)}$就是所求传递闭包的矩阵

2.2.2 性能评估

设 R 是集合 A 上的二元关系，A 中有 n 个元素。

每趟检查非十字上的元素有$(n-1)^2$个，需要 $O(n^2)$，而 k 从 0 到 n-1 循环 n 次，故总的时间复杂度为 $O(n^3)$。

2.2.3 流程图表示



2.2.4 代码实现

· 求传递闭包

```
// Warshall 算法求传递闭包
void TransitiveClosureWarshall(bool** matrix, int size)
{
    for (int k = 0; k < size; ++k)
    {
        for (int i = 0; i < size; ++i)
        {
            for (int j = 0; j < size; ++j)
            {
                if (i != k && j != k && !matrix[i][j])
                {
                    matrix[i][j] = matrix[i][k] && matrix[k][j];
                }
            }
        }
    }
}
```

· 项目主体部分

```
int main()
{
    int num = 0;
    bool flag = true;

    while (flag)
    {
        while (true)
        {
            cout << "请输入矩阵阶数：";
            cin >> num;
            if (cin.good() && num > 0 && num <= INT_MAX)
            {
                break;
            }

            cin.clear();
            cin.ignore(INT_MAX, '\n');
            cout << "输入错误！请重新输入" << endl;
        }
```

```
        bool** matrix;
        CreateMatrix(matrix, num);

        for (int i = 0; i < num; ++i)
        {
            cout << "请输入矩阵的第" << i << "行元素(元素以空格分隔):";
            for (int j = 0; j < num; ++j)
            {
                while (true)
                {
                    cin >> matrix[i][j];
                    if (cin.good())
                    {
                        break;
                    }

                    cin.clear();
                    cin.ignore(INT_MAX, '\n');
                    cout << "矩阵的第" << i << "行，第" << j << "列元
素输入错误，请继续输入:";
                }
            }
        }

        cout << "关系的传递闭包：" << endl;
        TransitiveClosureWarshall(matrix, num);
        ShowMatrix(matrix, num);
        DeleteMatrix(matrix, num);
        cout << "是否继续执行程序?(Y/N,Y/y 继续，N/n 退出)";
        char ch = '0';
        cin >> ch;
        if (ch == 'Y' || ch == 'y')
            continue;
        else if (ch == 'N' || ch == 'n')
            break;

    }

    return EXIT_SUCCESS;
}
```

3. 项目测试

3.1 一般测试

测试结果：

```
C:\Users\10728\Desktop\离散数学大作业\6\6\x64\Debug\6.exe
请输入矩阵阶数: 4
请输入矩阵的第0行元素(元素以空格分隔):0 1 0 0
请输入矩阵的第1行元素(元素以空格分隔):0 0 0 1
请输入矩阵的第2行元素(元素以空格分隔):0 0 0 0
请输入矩阵的第3行元素(元素以空格分隔):1 0 1 0
关系的传递闭包:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

3.2 集合上恒等关系情况

测试结果：

```
请输入矩阵阶数: 4
请输入矩阵的第0行元素(元素以空格分隔):1 0 0 0
请输入矩阵的第1行元素(元素以空格分隔):0 1 0 0
请输入矩阵的第2行元素(元素以空格分隔):0 0 1 0
请输入矩阵的第3行元素(元素以空格分隔):0 0 0 1
关系的传递闭包:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

3.3 集合只有一个元素情况（无环）

测试结果：

```
请输入矩阵阶数: 1
请输入矩阵的第0行元素(元素以空格分隔):0
关系的传递闭包:
0
```

3.4 集合只有一个元素情况（有环）

测试结果：

```
请输入矩阵阶数：1
请输入矩阵的第0行元素(元素以空格分隔):1
关系的传递闭包：
1
```

3.5 输入错误处理

测试结果：

```
C:\Users\10728\Desktop\离散数学大作业\6\6\x64\Debug\6.exe
请输入矩阵阶数：a
输入错误！请重新输入
请输入矩阵阶数：4
请输入矩阵的第0行元素(元素以空格分隔):0 1 a 0
矩阵的第0行，第2列元素输入错误，请继续输入:1 0
请输入矩阵的第1行元素(元素以空格分隔):1 1 0 0
请输入矩阵的第2行元素(元素以空格分隔):0 1 1 0
请输入矩阵的第3行元素(元素以空格分隔):0 0 0 0
关系的传递闭包：
1 1 1 0
1 1 1 0
1 1 1 0
0 0 0 0
```

4. 心得体会

在算法的时间效率上，Warshall 算法实现了从 $O(n^4)$ 到 $O(n^3)$ 的显著跃升，这不仅是计算效率上的革命性提升，更是在理论与实践之间架起了一座桥梁。算法本身不依赖额外的存储空间来暂存中间结果，这种空间复杂度上的优化，体现了算法设计中的巧妙与精炼。它以较低的编码复杂度实现高效的运算，无疑是在多个维度上的优化典范。

然而，能够将 Warshall 算法有效地应用于实际问题，根本在于对其算法逻辑的深刻理解。它巧妙地将传递闭包的构建过程转化为可达矩阵的构建，本质上是一种从局部到整体的思考模式。它从最基本的可达关系出发——如果 a 可达 b， b 可达 c，那么 a 必然可达 c，通过这样的逻辑单元逐步迭代，最终涵盖所有可能的路径，绘制出完整的传递闭包图景。这样的算法不仅仅是对计算过程的简化，更是对问题本质的深刻洞察。