离散数学课程实验说明文档

——Warshall算法求关系的传递闭包

作 者 姓 名： 马威

学 号： 2151294

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](#_Toc495668153)

[1.1 项目背景 1](#_Toc495668154)

[1.2 项目要求 1](#_Toc495668155)

[2 项目设计 1](#_Toc495668156)

[2.1 数据结构设计 1](#_Toc495668157)

2.2 算法设计 2

2.2.1 算法思路 2

2.2.2 性能评估 3

2.2.3 流程图表示 4

2.2.4 代码实现 4

·求传递闭包 4

·项目主体部分 5

[3 项目测试 7](#_Toc495668161)

[3.1 一般测试 7](#_Toc495668157)

[3.2 集合上恒等关系情况 7](#_Toc495668157)

[3.3 集合只有一个元素情况（无环） 8](#_Toc495668157)

[3.4 集合只有一个元素情况（有环） 8](#_Toc495668157)

[3.5 输入错误处理 8](#_Toc495668157)

4 心得体会 9

**1.项目分析**

1.1 项目背景

设R是非空集合A上的关系，有时候人们希望R具有一些有用的性质，例如，自反性（对称性或传递性）。为此，需要在R中添加一些有序对而构成新的关系R`，使得R`具有所需要的性质，但又不希望R`变得“太大”。换句话说，希望添加的有序对尽可能少。满足这些要求的R`就是R的自反（对称或传递）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**

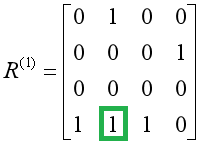
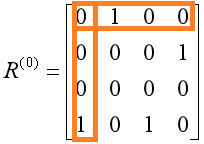
由项目分析可以得出，求关系的传递闭包就是求关系图的可达矩阵，即两点可达则对应元素为1，否则为0。设矩阵i行j列元素为M[i,j]，若M[i,k]、M[k,j]都为1，则根据传递闭包具有传递性，M[i,j]也应为1。由此可设计如下算法：

（1）标记矩阵的第k行第k列（相当于以M[k,k]为中心画一个十字）

（2）检查除十字外的其余所有元素M[i,j]，若M[i,j]=0 && M[i,k]=1 && M[k,j]=1，则将M[i,j]置为1；若M[i,j]本就是1，说明i到j本就可达，不做任何操作

（3）k自增，重复（1）（2）直到k完成了从0到n-1的一次循环，矩阵就是所求传递闭包的矩阵了。

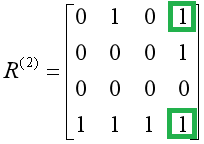
以求解如下关系（R**(0)**就是原矩阵）的传递闭包为例：

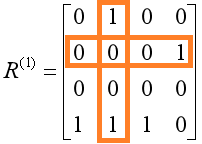


---------->

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

---------------------------------------------------------------------



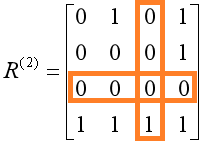
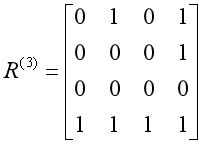
---------->

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

---------------------------------------------------------------------

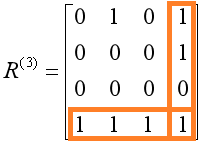
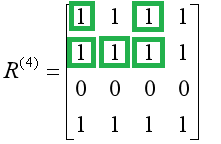


---------->

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

---------------------------------------------------------------------



---------->

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

---------------------------------------------------------------------

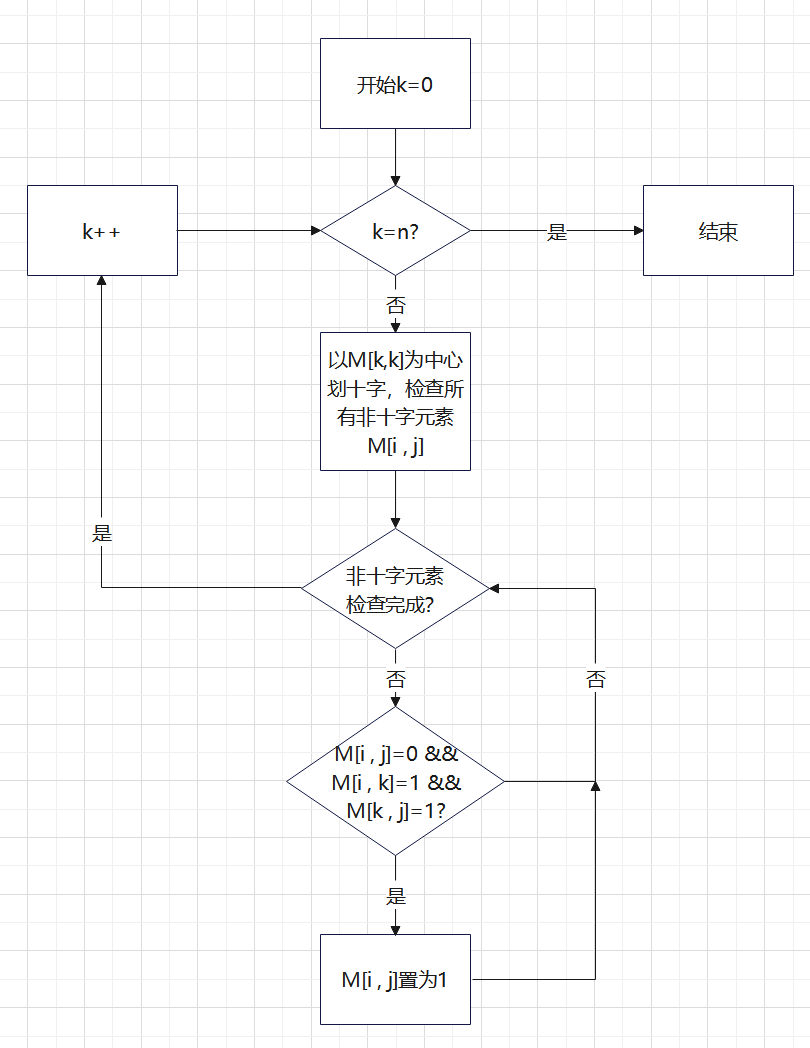
所得R**(4)**就是所求传递闭包的矩阵

**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

void transitive\_Warshall(bool\*\* matrix, const int num)

{

    for (int k = 0; k < num; k++) {

        for (int i = 0; i < num; i++) {

            for (int j = 0; j < num; j++) {

                if (i != k && j != k && !matrix[i][j]) {  /\*非十字且为零的元素M[i,j]\*/

                    matrix[i][j] = matrix[i][k] && matrix[k][j];  /\*若M[i,k]=1 && M[k,j]=1，则M[i,j]置为1\*/

                }

            }

        }

    }

}

·项目主体部分

int main()

{

    int num = 0;

    bool if\_on = true;

    while (if\_on) {

        while (1) {  /\*输入元素个数\*/

            cout << "请输入元素个数：";

            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\*\* data;

        create(data, num);

        for (int i = 0; i < num; i++) {  /\*输入每个元素\*/

            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 (1) {

                    cin >> data[i][j];

                    if (cin.good() && (data[i][j] == true || data[i][j] == false))

                        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;

        transitive\_Warshall(data, num);  /\*Warshall算法求解\*/

        show(data, num);

        clear(data, num);

    }

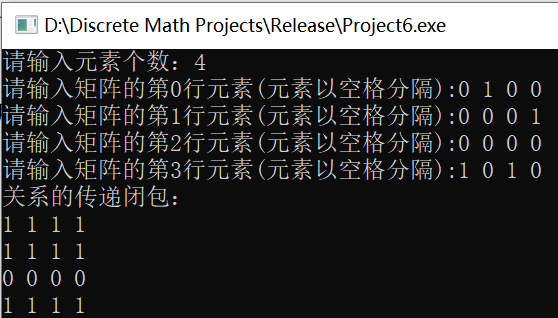
    return 0;

}

**3.项目测试**

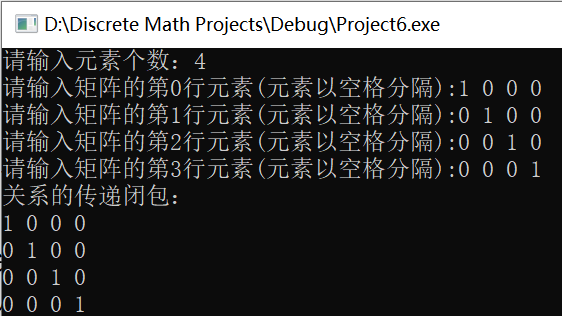
3.1 一般测试

测试结果：



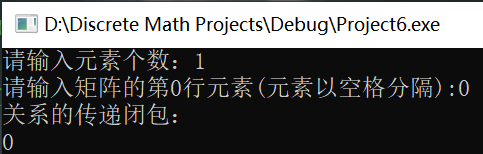
3.2 集合上恒等关系情况

测试结果：



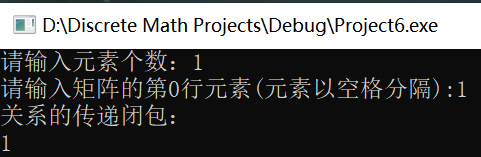
3.3 集合只有一个元素情况（无环）

测试结果：



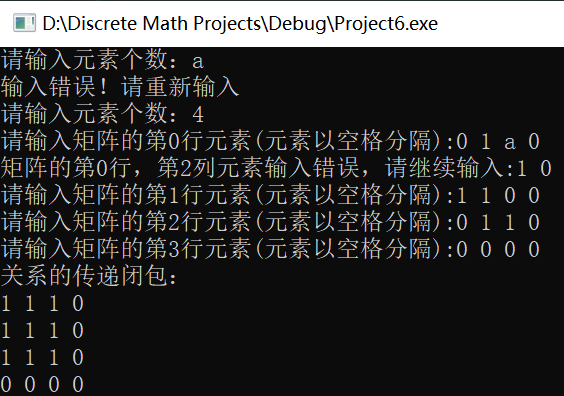
3.4 集合只有一个元素情况（有环）

测试结果：



3.5 输入错误处理

测试结果：



**4.心得体会**

时间复杂度上，Warshall算法相较于之前的一般求法从O(n**4**)降到了O(n**3**)，也不需要额外的空间对结果进行暂存，代码的复杂性也大大降低，可以说是再各个方面来说都比较好的算法。

当然，将该算法付诸实际的基础是对算法原理的清楚了解。其思路是把传递闭包的求解转换为可达矩阵的求解，从简单情况入手（a到b可达，b到c可达，则a到c可达），再进行循环，综合考虑每一个可达的情况，最终求出传递闭包。