离散数学课程实验说明文档

——求关系的自反、对称和传递闭包

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 2

2.2.3 流程图表示 3

2.2.4 代码实现 3

·求自反闭包 3

·求对称闭包 3

·求传递闭包 4

·项目主体部分 5

[3 项目测试 8](#_Toc495668161)

[3.1 求关系的自反闭包 8](#_Toc495668157)

[3.2 求关系的对称闭包 8](#_Toc495668157)

[3.3 求关系的传递闭包 9](#_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，程序需要能求解该关系的自反、对称和传递闭包。

**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**

由各类闭包的性质可以推得：



转换成关系矩阵，有以下求解方法：



**2.2.2 性能评估**

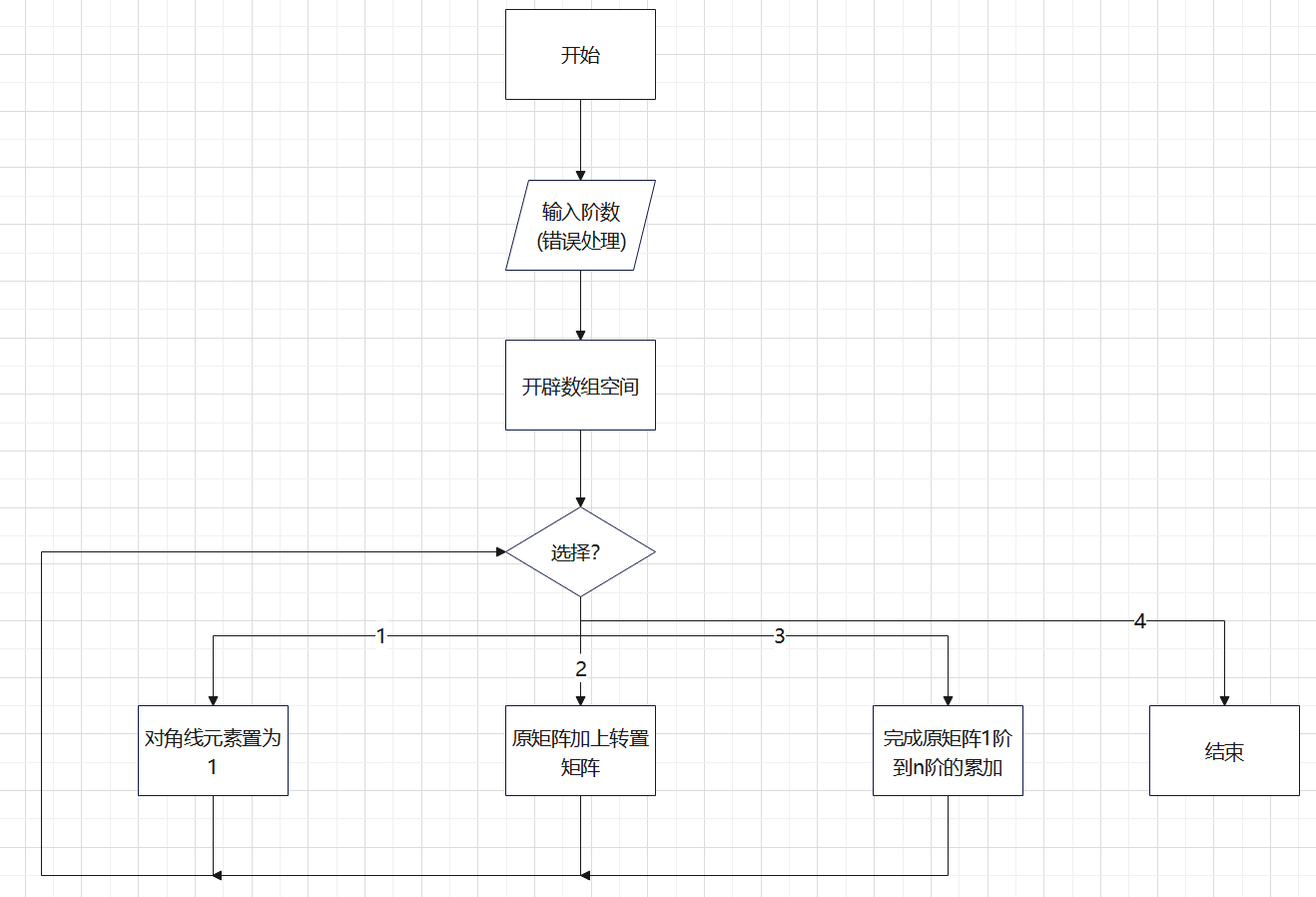
设R是集合A上的二元关系，A中有n个元素。

自反闭包只需让对角线元素都为1，仅需n次，时间复杂度为O(n)。

对称闭包是原矩阵加上原矩阵的转置矩阵，转置需要O(n**2**)，相加需要O(n**2**)，总时间复杂度仍为O(n**2**)。

传递闭包需要完成原矩阵1次到n次方的累加，第k趟中已经得到原矩阵k-1次方，与原矩阵相乘得原矩阵k次方，需要O(n**3**)，累加到结果上需要O(n**2**)，故一趟仍需O(n**3**)。总时间复杂度为O(n**4**)。

**2.2.3 流程图表示**

****

**2.2.4 代码实现**

·求自反闭包

void reflexive(bool\*\* matrix, const int num)

{

    //对角线上元素都置为1

    for (int i = 0; i < num; i++)

        matrix[i][i] = true;

}

·求对称闭包

void symmetric(bool\*\* matrix, const int num)

{

    bool\*\* rev;

    create(rev, num);

    //取原矩阵的转置

    for (int i = 0; i < num; i++) {

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

            rev[i][j] = matrix[j][i];

        }

    }

    //原矩阵与转置相加

    for (int i = 0; i < num; i++) {

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

            matrix[i][j] = matrix[i][j] || rev[i][j];

        }

    }

    clear(rev, num);

}

·求传递闭包

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

{

    //原矩阵

    bool\*\* Default;

    create(Default, num);

    //原矩阵的n次方

    bool\*\* Current;

    create(Current, num);

    //运算原矩阵n次方时，存放临时结果

    bool\*\* Result;

    create(Result, num);

    //给Default和Current赋值，使得Default是原矩阵，Current是单位阵

    for (int i = 0; i < num; i++) {

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

            Default[i][j] = matrix[i][j];

            if (i == j)

                Current[i][j] = true;

            else

                Current[i][j] = false;

        }

    }

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

        //Current和Default相乘，得到M的n次方

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

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

                bool\* primes = new bool[num];

                for (int k = 0; k < num; k++)

                    primes[k] = Current[i][k] && Default[k][j];

                Result[i][j] = false;

                for (int k = 0; k < num; k++)

                    Result[i][j] = Result[i][j] || primes[k];

                delete[] primes;

            }

        }

        //更新Current的值

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

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

                Current[i][j] = Result[i][j];

            }

        }

        //matrix和Current相加，得到结果

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

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

                matrix[i][j] = matrix[i][j] || Current[i][j];

            }

        }

    }

    clear(Default, num);

    clear(Current, num);

    clear(Result, num);

}

·项目主体部分

int main()

{

    int num = 0;

    bool if\_on = true;

    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:";

            }

        }

    }

    while (if\_on) {

        cout << "输入对应序号选择算法\n"

            << "1.自反闭包\n"

            << "2.对称闭包\n"

            << "3.传递闭包\n"

            << "4.退出\n";

        char selection = '\0';

        while (1) {

            selection = \_getch();

            if (selection >= '1' && selection <= '4') {

                cout << selection << endl;

                break;

            }

        }

        bool\*\* matrix;

        create(matrix, data, num);

        switch (selection) {

        case '1':

            reflexive(matrix, num);

            show(matrix, num);

            clear(matrix, num);

            break;

        case '2':

            symmetric(matrix, num);

            show(matrix, num);

            clear(matrix, num);

            break;

        case '3':

            transitive(matrix, num);

            show(matrix, num);

            clear(matrix, num);

            break;

        case '4':

            if\_on = false;

            clear(matrix, num);

            break;

        }

    }

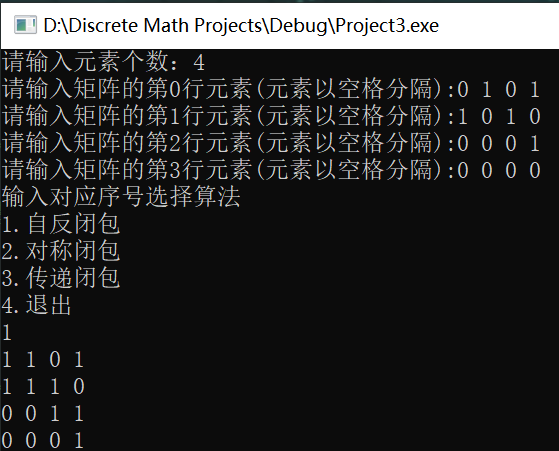
    return 0;

}

**3.项目测试**

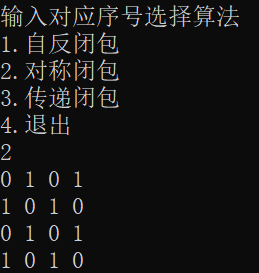
3.1 求关系的自反闭包

测试结果：



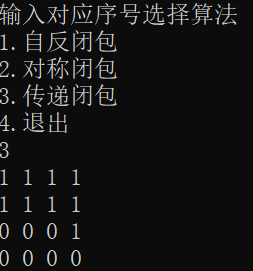
3.2 求关系的对称闭包

测试结果：



3.3 求关系的传递闭包

测试结果：



**4.心得体会**

在信息的存储上，本项目中由于需要二维数组存储，需要的连续空间比较大，因此采用了动态开辟的方式进行存储。

在问题的求解上，自反闭包和对称闭包没有问题，到传递闭包时屡屡出现错误，后来检查时发现，由于传递闭包要完成原矩阵的1次到n次累加，因此第k趟时，原矩阵、得到的原矩阵k次方、1次到k次累加结果都要进行暂存，以准备下一趟的运算。在刚开始编码时并没有注意到这些问题，因此错漏百出。所以得出教训：若循环运算时某一趟要使用上一趟的结果，则需要每次都将上一趟的结果暂存起来备用，否则丢失了上一趟的结果，这一趟的运算也毫无意义。