# 《离散数学》课程实验报告3 求关系的自反、对称和传递闭包

# 题目简介；

本题目实现关于关系矩阵的自反闭包、传递闭包和对称闭包的计算。用户可以输入一个关系矩阵，并选择进行自反闭包、传递闭包或对称闭包中的一种操作。程序根据用户的选择进行相应的计算，并输出闭包后的关系矩阵。

# 二.解题思路；：

# 1. 用户输入矩阵的行数和列数，以及矩阵元素，构建关系矩阵。

# 2. 用户选择进行自反闭包、传递闭包或对称闭包中的一种操作。

# 3. 根据用户的选择，程序执行相应的闭包计算：

# - 对于自反闭包，将矩阵对角线元素置为1。

# - 对于传递闭包，通过迭代计算，更新矩阵元素的值，直到不再改变为止。

# - 对于对称闭包，将矩阵转置并与原矩阵求并集，得到对称闭包后的矩阵。

# 4. 输出闭包后的关系矩阵，以供用户查看。

# 三.数据结构；

# 1. `int s[100][100]`：二维数组，用来存储关系矩阵，其中`s[i][j]`代表矩阵中第`i`行第`j`列的元素值。

# 2. `int d, n, i, j`：变量用来表示矩阵的行数、列数以及循环中使用的计数器。

# 四. 实验原理和实现过程（算法描述）；

## 1.实验原理

对以矩阵表示的关系，其自反闭包只要将矩阵的主对角线全部置为1，对称闭包则由关系矩阵加上其转置矩阵得到（逻辑加）。

## 2.实验过程

# 1. 自反闭包算法（zifan）：

# - 首先，用户输入一个关系矩阵，然后程序将对角线上的元素设置为1，以满足自反性。

# - 最后，输出处理后的关系矩阵。

# 2. 对称闭包算法（duichen）：

# - 用户输入一个关系矩阵，程序将获取该关系矩阵的转置矩阵。

# - 然后，程序将原关系矩阵和其转置矩阵的对应元素相加，超过1的元素设为1，以满足对称性。

# - 最后，输出处理后的关系矩阵。

# 3. 传递闭包算法（chuandi）：

# - 用户输入一个关系矩阵，程序将根据传递闭包的定义，对关系矩阵进行处理。

# - 程序通过遍历关系矩阵，若存在元素 s[i][j] 和 s[j][k] 均为1，则将 s[i][k] 设为1，以满足传递性。

# - 最后，输出处理后的关系矩阵。

# 4. 退出程序（exitProgram）：

# - 当用户选择退出时，程序输出欢迎信息，并正常退出。

# 整个程序通过递归调用select函数实现了用户交互和算法选择，用户可以根据需要选择不同的算法对输入的关系矩阵进行处理。

# 五.部分核心代码；

#include <iostream>

#include <cstdlib>

using namespace std;

void output(int s[][100]); // 输出矩阵的函数

void zifan(int s2[][100]); // 自反闭包算法

void duichen(int s2[][100]); // 对称闭包算法

void chuandi(int s2[][100]); // 传递闭包算法

void select(); // 选择操作的函数

void exitProgram(); // 退出程序的函数

int s[100][100], z; // 全局变量：关系矩阵和选择操作的变量

int d, n, i, j; // 行数、列数、循环变量

int main()

{

select(); // 调用选择操作的函数

return 0;

}

void select()

{

cout << "请输入矩阵的行数:";

cin >> n; // 输入矩阵的行数

cout << "请输入矩阵的列数:";

cin >> d; // 输入矩阵的列数

cout << "请输入关系矩阵:\n";

for (i = 0; i < n; i++)

{

cout << "\n";

cout << "请输入矩阵的第" << i << "行元素(元素以空格分隔):";

for (j = 0; j < d; j++)

cin >> s[i][j]; // 输入关系矩阵

}

cout << "输入对应序号选择算法\n1:自反闭包\n2:传递闭包\n3:对称闭包\n4:退出\n";

cin >> z; // 输入选择操作的序号

switch (z)

{

case 1:

zifan(s); // 调用自反闭包算法

break;

case 2:

chuandi(s); // 调用传递闭包算法

break;

case 3:

duichen(s); // 调用对称闭包算法

break;

case 4:

exitProgram(); // 退出程序

break;

default:

cout << "无效的选择！" << endl; // 输入错误时输出提示

select(); // 重新选择操作

}

}

void output(int s[][100])

{

cout << "所求关系矩阵为:\n"; // 输出提示

for (i = 0; i < n; i++)

{

for (j = 0; j < d; j++)

cout << s[i][j]; // 输出关系矩阵

cout << endl;

}

}

void zifan(int s2[][100])

{

for (i = 0; i < n; i++)

s2[i][i] = 1; // 自反闭包：对角线上的元素设为1

output(s2); // 输出处理后的关系矩阵

select(); // 重新选择操作

}

void duichen(int s2[][100])

{

int s1[100][100];

for (i = 0; i < n; i++)

for (j = 0; j < d; j++)

s1[j][i] = s2[i][j]; // 获取关系矩阵的转置矩阵

for (i = 0; i < n; i++)

for (j = 0; j < d; j++)

{

s2[i][j] = s2[i][j] + s1[i][j]; // 对称闭包：矩阵元素与其转置元素相加

if (s2[i][j] > 1)

s2[i][j] = 1; // 超过1的元素设为1

}

output(s2); // 输出处理后的关系矩阵

select(); // 重新选择操作

}

void chuandi(int s2[][100])

{

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

for (i = 0; i < n; i++)

for (j = 0; j < d; j++)

if (s2[i][j] == 1 && s2[j][k] == 1)

s2[i][k] = 1; // 传递闭包：若存在元素 s[i][j] 和 s[j][k] 均为1，则将 s[i][k] 设为1

output(s2); // 输出处理后的关系矩阵

select(); // 重新选择操作

}

void exitProgram()

{

cout << "欢迎下次再次使用!" << endl; // 输出欢迎提示

exit(0); // 退出程序

}