/\*\* @author group

\*

\* 开发环境：

\* Windows 10 家庭版 x64, Intel i7 8750H, 8GB RAM, 128 GB SSD

\* IDE: IntelliJ IDEA

\* JDK: OpenJDK 11.0.1

\*

\* 运行环境：

\* 推荐使用 JDK 11.0.1 环境下编译运行

\*

\* void testExp\_04() 对应离散数学-实验题4：

\* 输入 关系矩阵M的大小 n

\* 输入 矩阵M

\* 输出 矩阵M

\* 输出 矩阵M对应的关系

\* 输出 M是否为偏序关系

\* 输出 M的极小、极大元

\* 输出 M最小、最大元

\*

\*/

/\*

Sample Input:

Input Size of Matrix:

7

Input the Matrix:

1 0 0 0 1 0 0

0 1 0 0 0 1 1

0 0 1 0 0 1 0

0 0 0 1 1 0 1

0 0 0 0 1 0 0

0 0 0 0 0 1 0

0 0 0 0 0 0 1

Sample Output:

M:

[1, 0, 0, 0, 1, 0, 0]

[0, 1, 0, 0, 0, 1, 1]

[0, 0, 1, 0, 0, 1, 0]

[0, 0, 0, 1, 1, 0, 1]

[0, 0, 0, 0, 1, 0, 0]

[0, 0, 0, 0, 0, 1, 0]

[0, 0, 0, 0, 0, 0, 1]

{ <1,1>, <1,5>, <2,2>, <2,6>, <2,7>, <3,3>, <3,6>, <4,4>, <4,5>, <4,7>, <5,5>, <6,6>, <7,7>, }

集合M是偏序关系

1 是极小元

2 是极小元

3 是极小元

4 是极小元

5 是极大元

6 是极大元

7 是极大元

无最小元

无最大元

\*/

Scanner scanner=new Scanner(System.in);

System.out.println("Input Size Of Matrix: ");

/\*

n从键盘中输入，用于确定关系矩阵的大小

\*/

int n=scanner.nextInt();

/\*

定义一个二维数组，用于保存关系矩阵，0为关系不存在，非零为关系存在

下面的循环用于输入矩阵

\*/

int[][] M=new int[n][n];

System.out.println("Input the Matrix: ");

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

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

M[i][j]=scanner.nextInt()==0?0:1;

}

}

/\*

输出矩阵M的值

\*/

System.out.println("M: ");

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

System.out.println(Arrays.toString(M[i]));

}

System.out.print("{");

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

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

if(M[i][j]!=0){

System.out.print(" <"+(i+1)+ ","+(j+1)+">, ");

}

}

}

System.out.println("}");

//region 对矩阵关系的判断

/\*

验证自反性

检查对角线是否都存在，如果不存在则将属性@isReverse设置为false

\*/

boolean isReverse=true;

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

if(M[i][i]==0){

isReverse=false;

break;

}

}

/\*

验证对称性，如果@M[i][j]!=0且@M[j][i]==0则判断没有对称性

(对称性对偏序性不产生影响)

\*/

// boolean isSymmetrical=true;

// for (int i = 0; i < n; i++) {

// for (int j = 0; j < n; j++) {

// if(M[i][j]!=0&&M[j][i]==0){

// isSymmetrical=false;

// }

// }

// }

/\*

验证反对称性，如果i!=j且@M[i][j]!=0且@M[j][i]！=0则判断没有反对称性

\*/

boolean isUnSymmetrical=true;

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

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

if(i!=j&&M[i][j]!=0&&M[j][i]!=0){

isUnSymmetrical=false;

break;

}

}

}

/\*

验证传递性，对矩阵进行二次矩阵乘法并相加，最后与原矩阵进行比较，如果M不是M\*M的子集，则矩阵没有传递性

\*/

boolean isTransitivity=true;

int[][] tempM=new int[n][n];

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

tempM[i]=M[i].clone();

}

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

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

tempM[i][j] = getLineMultiply(M, M, i, j);

}

}

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

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

if(tempM[i][j]!=0&&M[i][j]==0){

isTransitivity=false;

break;

}

}

}

//endregion

/\*

偏序关系满足自反性、反对称性、传递性 : 如果满足偏序关系，则进一步计算关系的极大极小元、最大最小元

\*/

if (isReverse&&isUnSymmetrical&&isTransitivity){

System.out.println("集合M是偏序关系");

int[] in = new int[n];

int[] out = new int[n];

/\*

计算极大元，极小元

通过计算每个元素的入度和出度来判断极大元和极小元，入度为1则为极小元，出度为1则为极大元

如果有多个入度为1的元素，则极小元有多个，故没有最小元，否则唯一的极小元为最小元

如果有多个出度为1的元素，则极大元有多个，故没有极大元，否则唯一的极大元为最大元

\*/

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

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

if (M[i][j]!=0){

out[i]++;

}

if (M[j][i]!=0){

in[i]++;

}

}

}

int numIn=0;

int numOut=0;

int tMax = 0,tMin = 0;

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

if(in[i]==1){

System.out.println((i+1)+" 是极小元");

tMin=i+1;

numIn++;

}

if (out[i]==1){

System.out.println((i+1)+" 是极大元");

tMax=i+1;

numOut++;

}

}

if(numIn==1){

System.out.println("最小元是 "+tMin);

} else {

System.out.println("无最小元");

}

if(numOut==1){

System.out.println("最大元是 "+tMax);

} else {

System.out.println("无最大元");

}

} else {

System.out.println("集合M不是偏序关系");

}

scanner.close();

}

/\*\*

\* 该方法实现对矩阵A的i行与矩阵B的j列进行矩阵乘法并返回结果

\* @param a 关系矩阵A

\* @param b 关系矩阵B

\* @param i 要进行相乘的矩阵A的第i行

\* @param j 要进行相乘的矩阵B的第j列

\* @return 0: 矩阵乘法结果为0, 1: 矩阵乘法结果不为0

\*/

private static int getLineMultiply(int[][] a, int[][] b, int i, int j){

int t=0;

for(int k=0;k<a.length;k++){

t+=a[i][k]\*b[k][j];

}

return t==0?0:1;

}