|  |
| --- |
| 离散数学实验报告 |
| 学号：1120221704 姓名： 戴尚轩 |

目录

[1 求命题的主范式 1](#_Toc20898)

[1.1 概述 1](#_Toc29224)

[1.2 步骤流程 1](#_Toc9174)

[1.3 程序实现 1](#_Toc15741)

[1.3.1 PriorityofOperator 2](#_Toc27426)

[1.3.2 evalRPN 2](#_Toc7764)

[1.3.3 calculate 3](#_Toc3210)

[1.3.4 Print 4](#_Toc13811)

[2 消解算法 4](#_Toc31558)

[2.1 概述 4](#_Toc19862)

[2.2 步骤流程 5](#_Toc31483)

[2.3 程序实现 5](#_Toc9278)

[2.3.1 alphacount 6](#_Toc30930)

[2.3.2 Init 6](#_Toc7154)

[2.3.3 connect 7](#_Toc23082)

[2.3.4 dispel 7](#_Toc18981)

[3 求关系的传递闭包 8](#_Toc22011)

[3.1 概述 8](#_Toc19478)

[3.2 步骤流程 8](#_Toc22047)

[3.3 程序实现 8](#_Toc1894)

[3.3.1 Get 9](#_Toc2078)

[3.3.2 Walshell 9](#_Toc61)

[3.3.3 Output 9](#_Toc16696)

[4 求偏序集中的极大元与极小元 10](#_Toc31155)

[4.1 概述 10](#_Toc14258)

[4.2 步骤流程 10](#_Toc31315)

[4.3 程序实现 10](#_Toc25536)

[4.3.1 search 11](#_Toc24058)

[4.3.2 find 11](#_Toc11388)

[4.3.3 input 12](#_Toc10074)

[4.3.4 output 12](#_Toc27075)

[5 代数系统算律的判断 12](#_Toc5865)

[5.1 概述 12](#_Toc18945)

[5.2 步骤流程 13](#_Toc31244)

[5.3 程序实现 13](#_Toc25377)

[5.3.1 find 14](#_Toc31922)

[5.3.2 Input 14](#_Toc12353)

[5.3.3 Commutativecal 14](#_Toc22722)

[5.3.4 Associativecal 14](#_Toc2530)

[5.3.5 Idemponentcal 14](#_Toc23051)

[5.3.6 Output 14](#_Toc21666)

[5.3.7 Indentityfind 14](#_Toc8852)

[5.3.8 Zerofind() 15](#_Toc17097)

[6 求Zn群中元素的阶 15](#_Toc14969)

[6.1 概述 15](#_Toc10027)

[6.2 步骤流程 15](#_Toc4034)

[6.3 程序实现 15](#_Toc3116)

[6.3.1 Rank 16](#_Toc28111)

[7 二部图的判定 16](#_Toc6130)

[7.1 概述 16](#_Toc29011)

[7.2 步骤流程 17](#_Toc20227)

[7.3 程序实现 17](#_Toc25841)

[7.3.1 Input 17](#_Toc4312)

[7.3.2 DFS 17](#_Toc5115)

[7.3.3 find 18](#_Toc17982)

[7.3.4 Output 19](#_Toc19422)

[8 有向图连通性的判定 19](#_Toc14195)

[8.1 概述 19](#_Toc19010)

[8.2 步骤流程 20](#_Toc28764)

[8.3 程序实现 20](#_Toc1262)

[8.3.1 Input 20](#_Toc2892)

[8.3.2 justify 20](#_Toc25059)

[8.3.3 Output 21](#_Toc16185)

**表目录**

[表 1 求命题的主范式函数说明表 2](#_Toc2162)

[表 2 消解算法函数说明表 6](#_Toc29430)

[表 3 求关系的传递闭包函数说明表 8](#_Toc25410)

[表 4 求偏序集中的极大元与极小元函数说明表 10](#_Toc13133)

[表 5 代数系统算律函数说明表 13](#_Toc24198)

[表 6 求Zn群中元素的阶函数说明表 15](#_Toc15304)

[表 7 二部图的判定函数说明表 17](#_Toc30646)

[表 8 有向图连通性的判定函数说明表 20](#_Toc23364)

**图目录**

[图 1 求命题的主范式流程图 1](#_Toc28981)

[图 2 evalRPN 函数执行流程 3](#_Toc14292)

[图 3 calculate 函数执行流程 4](#_Toc23072)

[图 4 消解算法流程图 5](#_Toc12141)

[图 5 Init函数执行流程 6](#_Toc15728)

[图 6 dispel函数执行流程 7](#_Toc26842)

[图 7 求关系的传递闭包流程图 8](#_Toc6100)

[图 8 Walshall函数执行流程 9](#_Toc15718)

[图 9 求偏序集中的极大元与极小元流程图 10](#_Toc15632)

[图 10 search函数执行流程 11](#_Toc20500)

[图 11 find函数执行流程 12](#_Toc8784)

[图 12 代数系统算律的判断流程图 13](#_Toc19087)

[图 13 求Zn群中的节程序流程图 15](#_Toc8891)

[图 14 Rank函数执行流程 16](#_Toc26208)

[图 15 二部图的判定程序流程图 17](#_Toc29260)

[图 16 DFS函数执行流程 18](#_Toc5731)

[图 17 find函数执行流程 19](#_Toc13021)

[图 18 有向图连通性的判定程序流程图 20](#_Toc25724)

[图 19 Walshell函数执行流程 21](#_Toc24453)

# 求命题的主范式

## **概述**

输入命题公式的合式公式，求出公式的真值表，并输出该公式的主合取范式和主析取范式。

## **步骤流程**

程序运行主逻辑如图1所示。

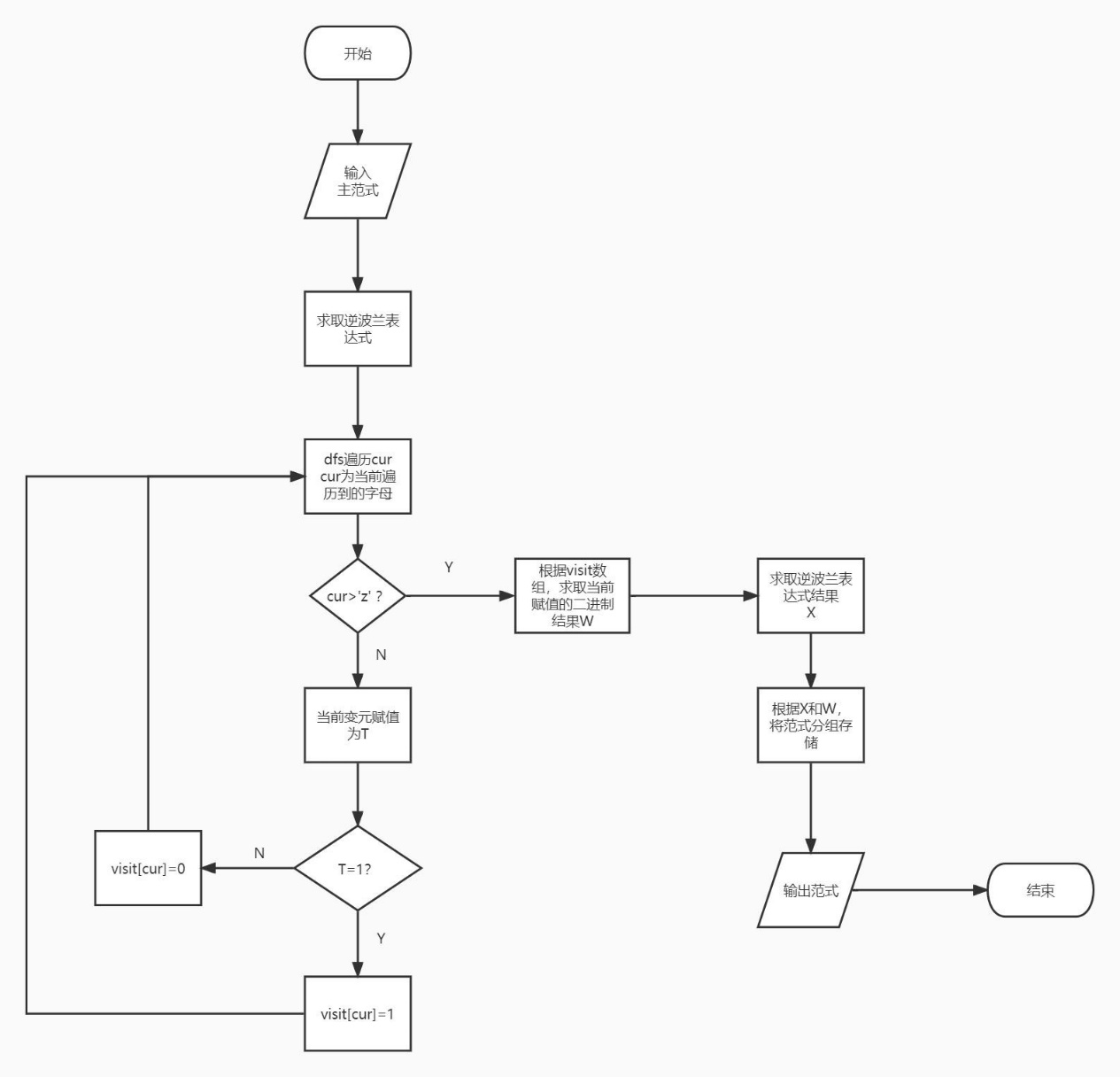


图 1 求命题的主范式流程图

## **程序实现**

表 1 求命题的主范式函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
|  | PriorityofOperator | 用于归定和处理运算符的优先级 |
|  | evalRPN | 用于将输入的字符串转成逆波兰表达式 |
|  | calculate | 用于计算逆某一赋值下的结果 |
|  | Print | 用于输出最终结果 |

### **PriorityofOperator**

用于归定和处理运算符的优先级，其中 ！ 优先级最高，其次是&、|、-、+、（）、最后 # 为 哨兵运算符，用于强制栈内运算符弹出，维护计算结果的正确性。

### **evalRPN**

用于将输入的字符串转成逆波兰表达式，其中就要使用函数一作为辅助。 依次遍历整个字符串，如果为字母，则直接输出到存储结果的数组中，如果为运算符&、|、+、-，则需要与栈顶运算符的优先级进行比较，处理类似 ！插队（优先级更高）的现象，如果为（ 则直接入栈，如果为），则一直弹出栈顶元素直到找到与之匹配的左括号。

求取最终表达式过程中将读入并维护出现不同变元的总种类数，同时更新SumofVar，用于统计最终范式和迭代的数量。 关于本函数的执行流程，如所示。

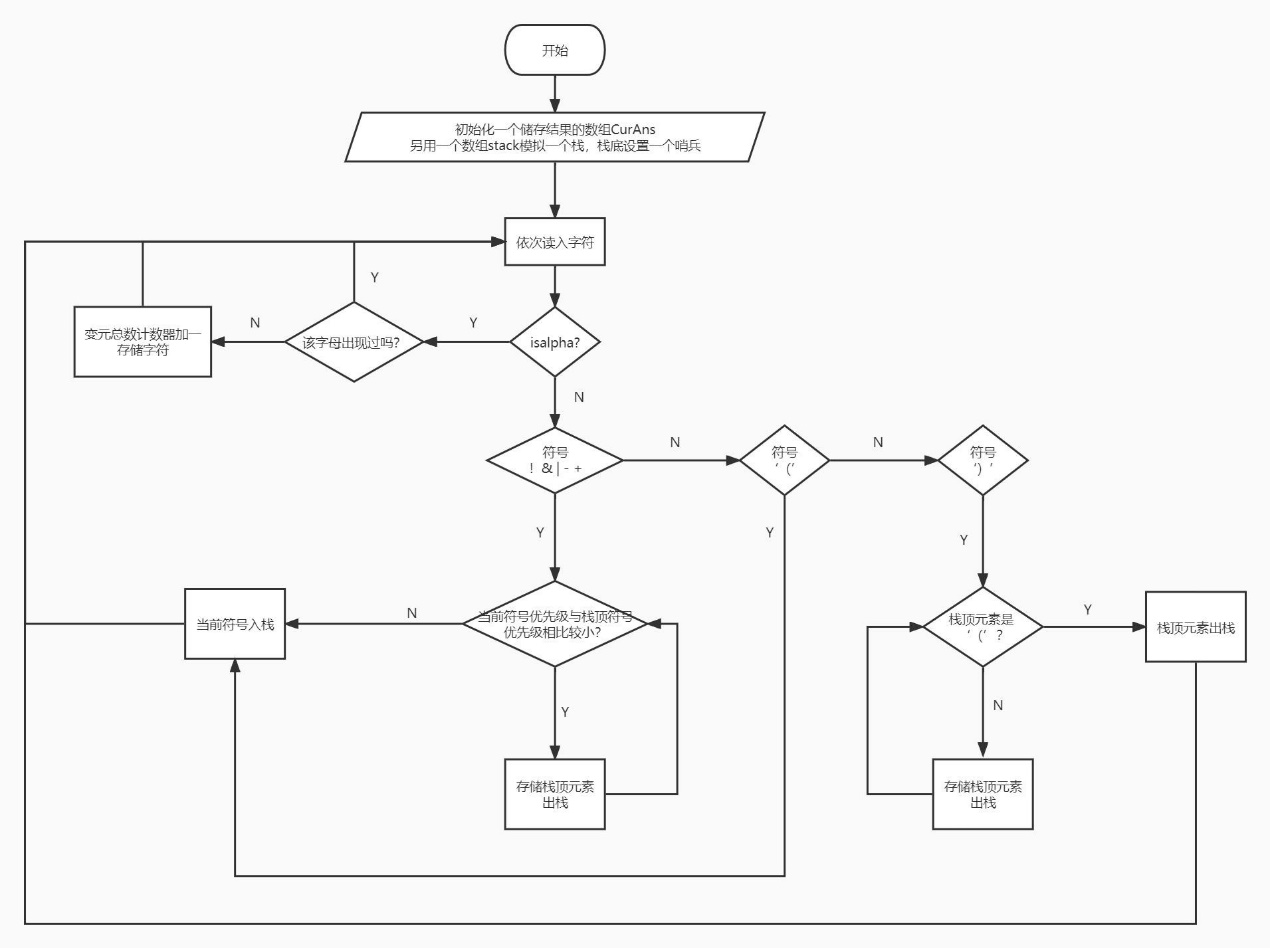


图 2 evalRPN 函数执行流程

### **calculate**

用于计算逆波兰表达式在某一赋值下的结果。函数参数传入目前每个字母对应的赋值的数组和逆波兰表达式的字符串。设置一个数值栈，读到字母的时候推入字母对应的值，读到!则将栈顶值取反，读到&、|、-、+取两个栈顶元素计算。最后将栈顶元素（此时栈内仅一个元素）作为函数返回值，即是本次计算的结果。

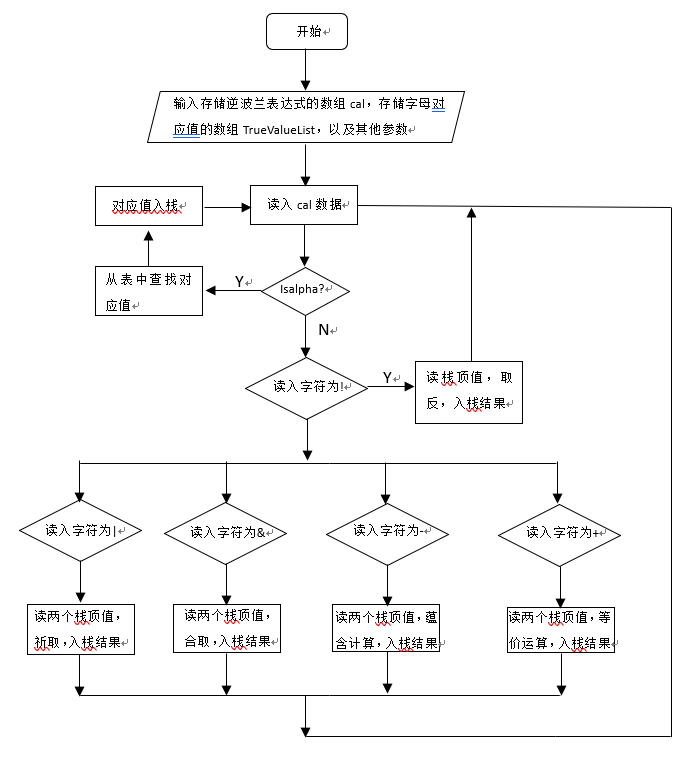


图 3 calculate 函数执行流程

### **Print**

每次计算得到的结果，为真存到m数组中，为假存到M数组中，记录两个数组存入的数据数；如果length\_m为0为矛盾式，length\_M为0为永真式，最后按照顺序输出即可。

# 消解算法

## 概述

输入：合式公式 A 的合取范式，当 A 是可满足时，回答“YES ”；否则回答“NO”。

## 步骤流程

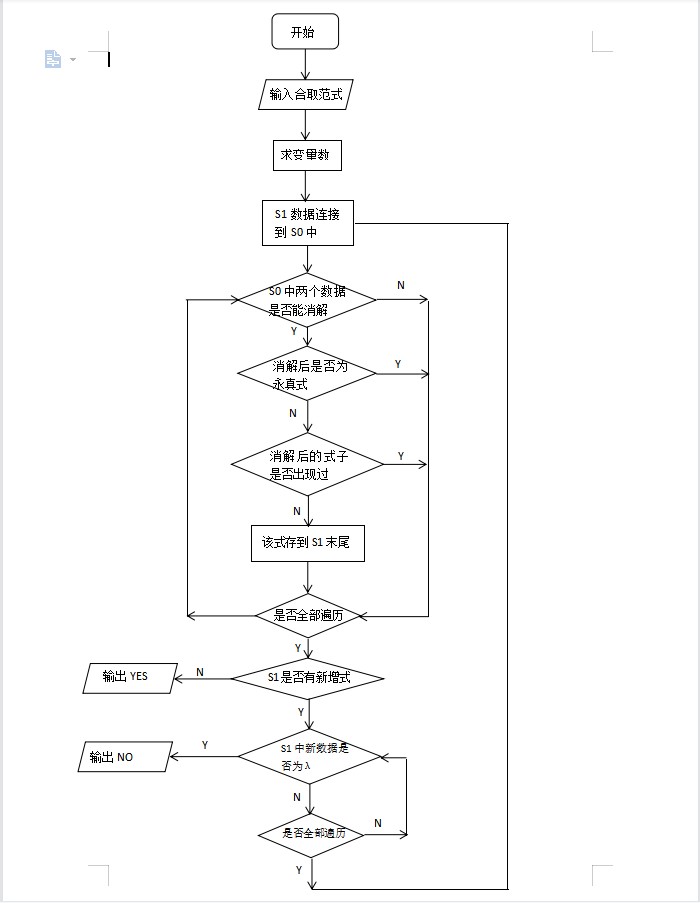


图 4 消解算法流程图

## 程序实现

表 2 消解算法函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
| 1. | alphacount | 用于计算不同的字母个数 |
| 2. | Init | 用于初始化S0 |
| 3. | connect | 用于将S1（新生成式）连接到S0中 |
| 4. | dispel | 用于实现具体消解和输出 |

### **alphacount**

用于计算不同的字母个数，读取整个需要消解的式子，遇到字母则与已有的字母表alphalist进行对比，如果没有出现过该字母则将其顺次存到alphalist中，最后并返回alphalist的串长。

### **Init**

用于初始化S0。读取输入的主和取范式，将每一个祈取式存入S0中。如果该式是永真式，则忽略。

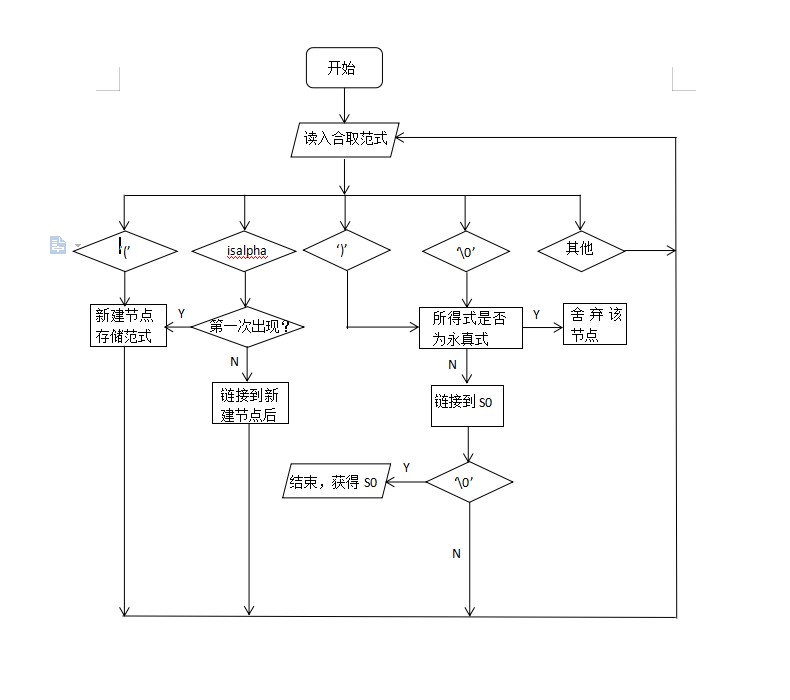


图 5 Init函数执行流程

### **connect**

用于将S1（新生成式）连接到S0中。由于采用的是链表的链接，直接将S1的数据接到S0后面，并将S1置NULL即可

### **dispel**

用于实现具体消解和输出。每次将S0内所得合取式依次互相比较，能消解且消解所得非永真式的即存到S1中。对于S1，本次没有获得新的消解式即输出YES，否则读取所有S1内式子，有λ的输出NULL，没有的进行connect并再次消解.

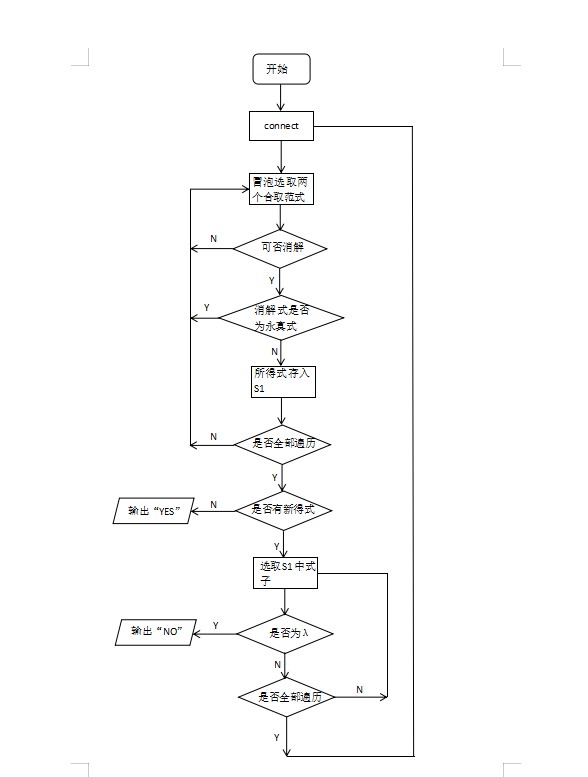


图 6 dispel函数执行流程

# 求关系的传递闭包

## **概述**

输入：一次输入一个关系矩阵，每一行两个相邻元素之间用一个空格隔开，输入元素的行与列分别对应关系矩阵的航宇列。关系的基数小于12。输出：给关系的传递闭包所对应的关系矩阵。

## **步骤流程**

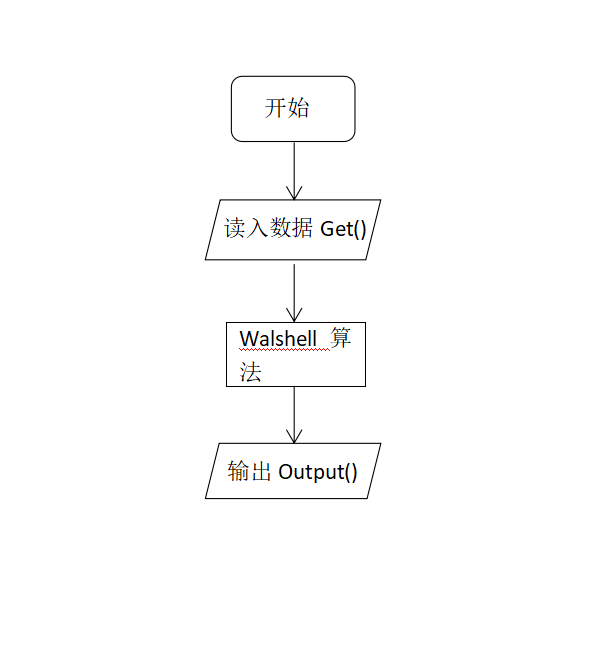


图 7 求关系的传递闭包流程图

## **程序实现**

表 3 求关系的传递闭包函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
| 1. | Get | 用于输入数据 |
| 2. | Walshell | 用于进行关系闭包实际计算 |
| 3. | Output | 用于输出关系闭包矩阵 |

### **Get**

用于读入输入的数据。将所有的数据存入一个一维数组，记录总数据量；总数据量开根号即关系矩阵的阶数；最后再重新存入矩阵中。

实际上可以使用getline()函数读取一行数据来记录矩阵阶数。

### **Walshell**

用于实现具体的计算关系闭包。首先建立图的关系矩阵。对于每一个点对(i,j)进行n次判定，第k步判定时，如果已经联通（M[i,j]=1）则忽略，未联通则考虑经过xk的通路，即M[i,k]=M[k,j]=1时，M[i,j]改写为1。

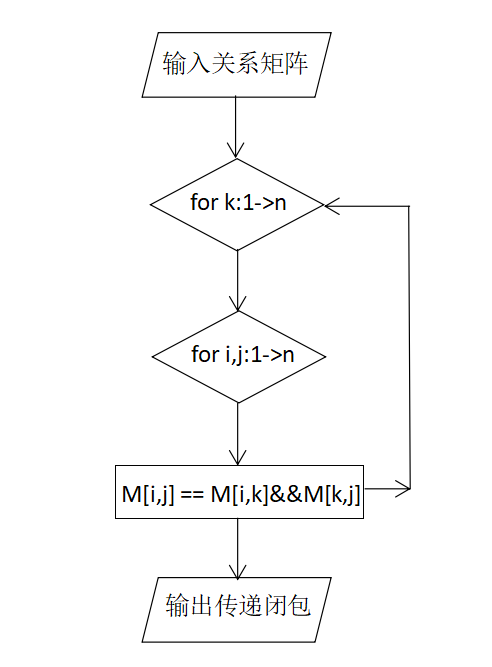


图 8 Walshall函数执行流程

### **Output**

按照Warshell算法得到的传递闭包，直接按照顺序输出新的关系矩阵即可。

# 求偏序集中的极大元与极小元

## 概述

输入偏序集<A, £>，A中的元素数不超过20个，分别用单个小写的英文字母表示。输入的第一行给出A中的各个元素，两个相邻的元素之间用逗号隔开，第二行给出偏序关系£，用有序对的形式给出(只给出哈斯图中的满足覆盖的两个元素形成的有序对)，如<a,b>,<c,a>等等，两个相邻的有序对之间用逗号隔开。

输出A的极小元与极大元。输出的第一行给出各个极小元，两个相邻元素之间用逗号隔开，输出的元素要求按照英文字母的自然顺序排列输出，第二行给出各个极大元，两个相邻元素之间用逗号隔开，输出的元素要求按照英文字母的自然顺序排列输出。

## 步骤流程

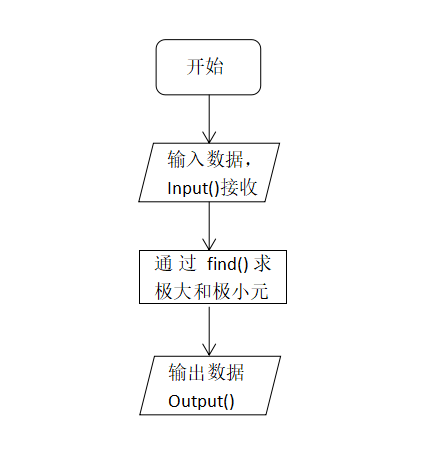


图 9 求偏序集中的极大元与极小元流程图

## 程序实现

表 4 求偏序集中的极大元与极小元函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
| 1. | search | 用于将数字和字母转换,字母转数字需考虑输入顺序问题 |
| 2. | find | 用于进行极大和极小的实际计算 |
| 3. | input | 用于在输入字母表和偏序关系时转换 |
| 4. | output | 用于输出极大和极小字母序列 |

### search

考虑到首个输入字母不一定为a，转换时不能简单的用ch- ‘0’。search用于数字和字母之间相互转换。使用时设置判定符type, type = 1为字母转数字，type = 2 为数字转字母。输入时用vector<char> list动态存储字母表。数字转字母时直接输出list[num]即可；字母转数字时需要在list中寻找字母所在位置对应的数字。

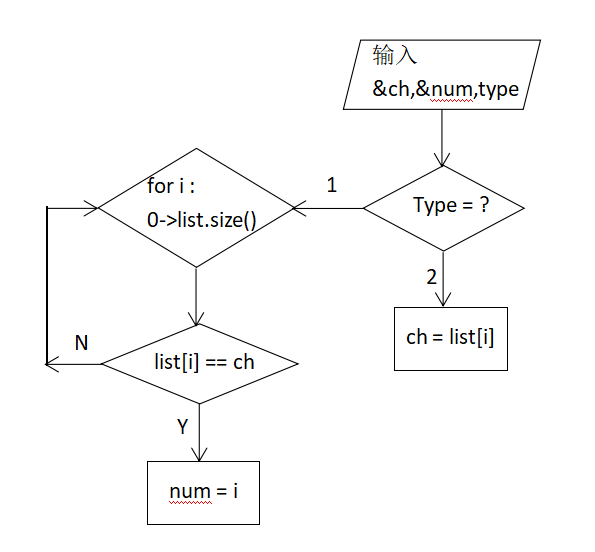


图 10 search函数执行流程

### find

对于每个字母，如果关系矩阵中该字母所在行没有1，则为极大；所在列没有1，则为极小。

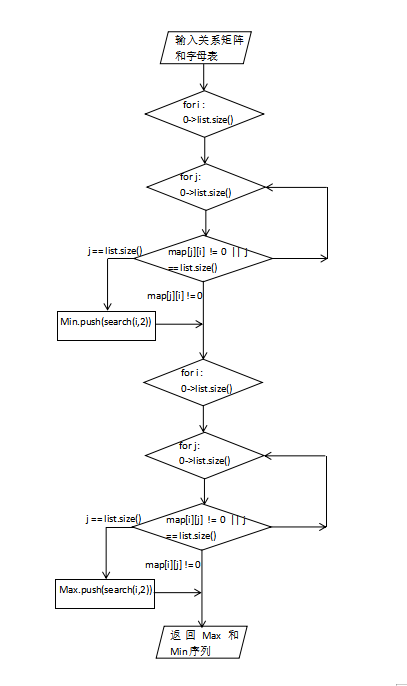


图 11 find函数执行流程

### input

输入的字母表存到list当中，输入的偏序关系存入到map当中。其中将用到search()函数。

### output

按照find()函数得到的Min和Max序列，顺序输出字母。

# 代数系统算律的判断

## 概述

假设代数系统<A, \*>中，A为有限集合，\*为二元运算。请判断A中\*运算是否满足交换律、结合律、幂等律；是否有幺元和零元，若有请输出，若无请输出n。

## 步骤流程

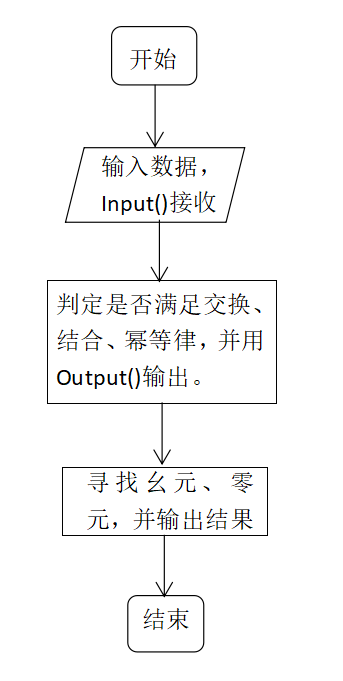


图 12 代数系统算律的判断流程图

## 程序实现

表 5 代数系统算律函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
| 1. | find | 用于将字母转换为对应位置的数字 |
| 2. | Input | 用于输入关系列表 |
| 3. | Commutativecal | 用于判定是否满足交换律 |
| 4. | Associatecal | 用于判定是否满足结合律 |
| 5. | Idemponentcal | 用于判定是否满足幂等律 |
| 6. | Output | 用于输出前三种判定的结果 |
| 7. | Identityfind | 用于寻找和输出关系的单位元 |
| 8. | Zerofind | 用于寻找和输出关系的零元 |

### find

用于将字母转换为位置数字。传入参数ch, 遍历记录了输入字符的数组list,若list[i]==ch，返回i。对于非法值返回-1。

### Input

用于将输入的字符存入list,将运算关系矩阵存入map。

### Commutativecal

用于进行交换律的验证。对于关系矩阵任意非对角元素i，j，若map[i][j] != map[j][i]，则返回false。若能够完成全部循环，返回true。

### Associativecal

用于进行结合律的验证。对于关系矩阵任意三个不同元素i,j,k,若map[find(map[i][j])][k] != map[i][find(map[j][k])]，则返回false。若能完成全部循环则返回true。

### Idemponentcal

用于进行幂等律的验证。对于关系矩阵任意元素i,若list[i] != map[i][i]（即a != a\*a），则返回false。若能完成全部循环，则返回true。

### Output

用于将上述三种性质验证的结果输出。若验证结果为false打印n，true则打印y。

### Indentityfind

用于寻找单位元。定义判断元素int flag = 0。对于每个元素i，如果其对任意j，都满足map[i][j] == list[j]，则输出该元素，并将flag置为1。最后如果flag为0则输出n表示无单位元。

### Zerofind()

用于寻找零元。定义判断元素int flag = 0。对于每个元素i，如果其对任意j，都满足map[i][j] == list[i]，则输出该元素，并将flag置为1。最后如果flag为0则输出n表示无零元。

# 求Zn群中元素的阶

## 概述

设Zn为模n整数加群(n>1)，求Zn中元素的阶。

输入：n，x（x是Zn中的元素）

输出：x的阶

## 步骤流程

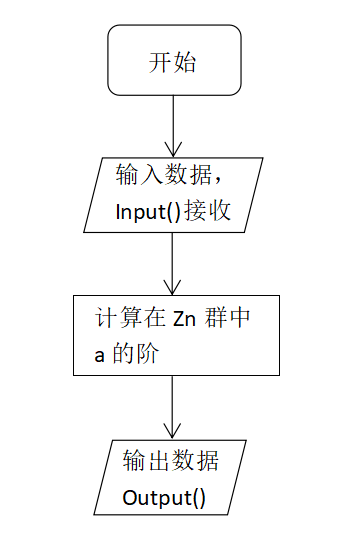


图 13 求Zn群中的节程序流程图

## 程序实现

表 6 求Zn群中元素的阶函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
| 1. | Rank | 用于计算a的阶 |

### Rank

用于计算a的阶并输出。对于给定的a和Z，用init记录a的初始值，设置list[Z]数组用于辅助计算，初始化为0值。然后进入循环：如果list[a]==0，则令list[a] = 1, a = ( a + init ) % Z，同时用ans记录进行的循环次数。循环退出后打印ans的值。

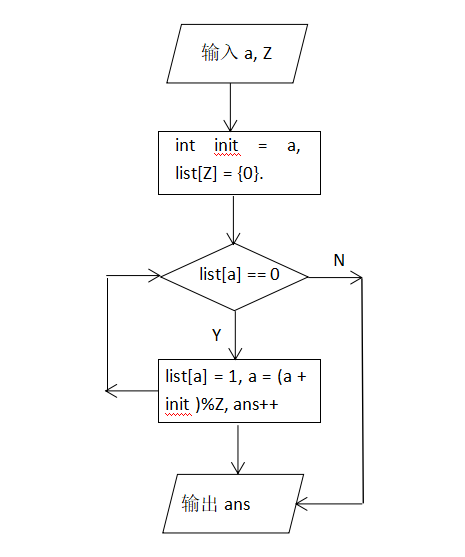


图 14 Rank函数执行流程

# 二部图的判定

## 概述

输入：

正整数n，代表无向图G的阶数；随后的n行代表G的邻接矩阵，每行有n个数据，每个数据以空格分隔。其中每个数据表示顶点vi邻接顶点vj边的条数。

输出：

若为二部图，输出yes；否则，输出no。

## 步骤流程

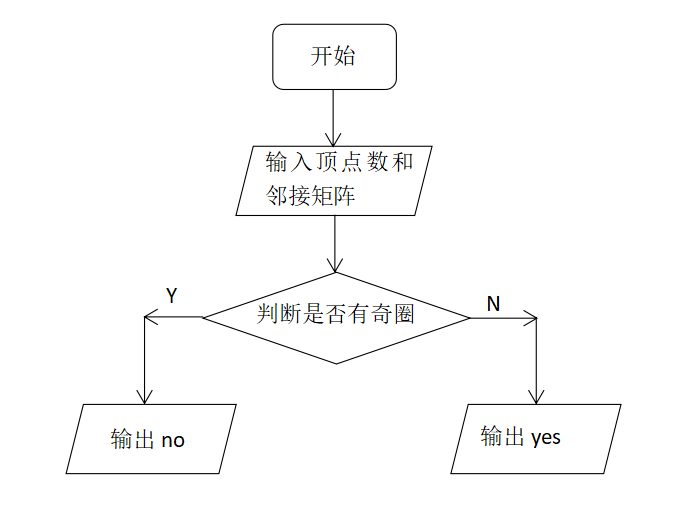


图 15 二部图的判定程序流程图

## 程序实现

表 7 二部图的判定函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
| 1. | Input | 用于输入数据的存储 |
| 2. | DFS | 用于遍历从某一点开始的所有回路并判定是否为奇圈 |
| 3. | find | 用于执行具体的判定流程 |
| 4. | Output | 用于执行输出 |

### Input

用于输入数据的存储。将输入的顶点数存入n，输入的邻接矩阵存入map二维数组。

### DFS

用于遍历从某一点开始的所有回路并判定是否为奇圈。对于传入的顶点编号（设为i），DFS寻找能够回到i点的圈。每次遍历时，用传入的参数length记录已经走过的距离，用vertex数组记录已经走过的点，用begin记录本次应该寻找的终点。如果存在不经过重复顶点且回到了i点的偶圈，则直接返回false。如果遍历完整个图都没有偶圈返回true。

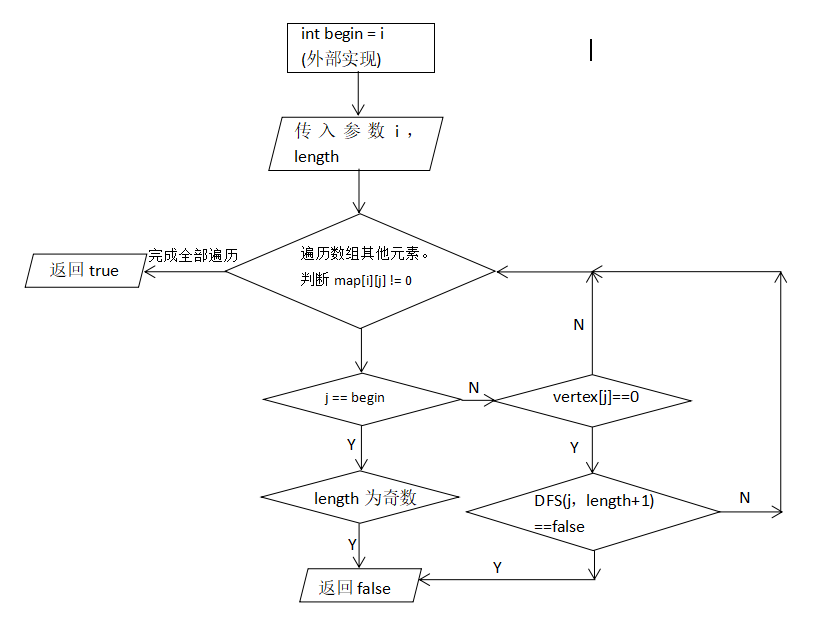


图 16 DFS函数执行流程

### find

用于执行具体的判定流程。对于函数是否有奇圈的判定，首先检查是否存在长度为1的圈，即map[i][i]==0的情况，有则直接返回false；而后遍历顶点，用begin记录每次判定时的初始起点，进行DFS判定，若有一个顶点返回值为false，则返回false。如能完成全部遍历，返回true。

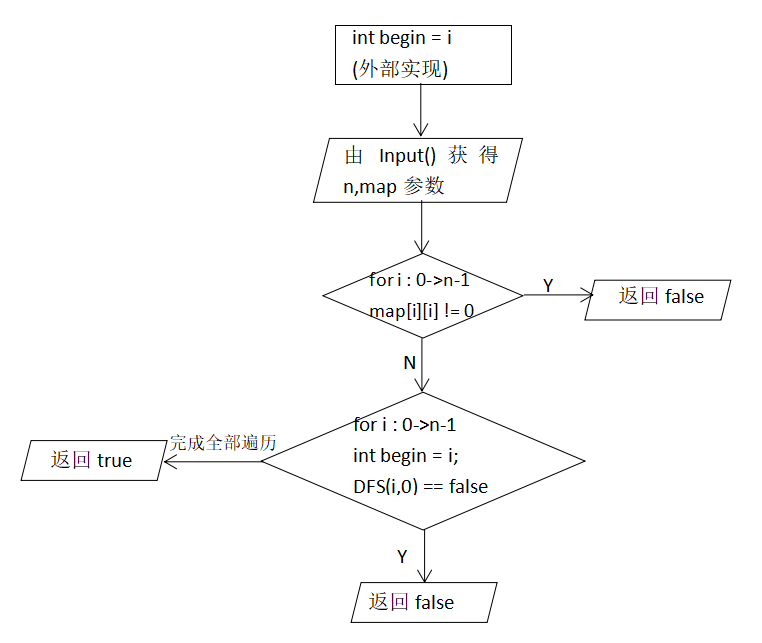


图 17 find函数执行流程

### Output

用于执行输出。根据find 函数的返回值，输出yes或no。

# 有向图连通性的判定

## 概述

题目：判断一个图是否为强连通图、单向连通图、弱连通图。输入为有向图的邻接矩阵。

输入

第一行为正整数N（0<N<=100），代表图中点的个数。接下来N行，每行有N个数据，每个数据以空格分隔，代表邻接矩阵。注意：输入的都是连通图。

输出

输出有一行，字母A，B，C。A代表强连通图，B代表单向连通图，C代表弱连通图。

## 步骤流程

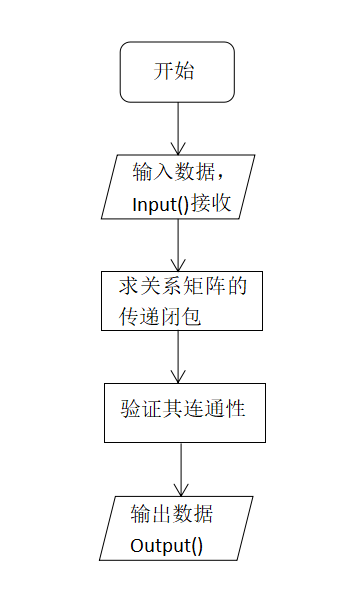


图 18 有向图连通性的判定程序流程图

## 程序实现

表 8 有向图连通性的判定函数说明表

| 序号 | 名称 | 说明 |
| --- | --- | --- |
| 1. | Input | 用于输入数据的存储 |
| 2. | justify | 用于计算传递闭包并验证连通性 |
| 3. | Output | 用于执行输出 |

### Input

用于输入数据的存储。将顶点数存入n中，邻接矩阵存入map中。

### justify

用于计算传递闭包并验证连通性，该函数返回值为1时是强连通图，返回2时是单向连通图，返回3时是弱连通图。利用Walshell算法计算矩阵的传递闭包以计算每个顶点之间的连通性。然后遍历每个顶点并设置变量flag = 1。若存在两个点i，j之间完全不连通，直接返回3；i，j之间单向联通，flag改写为2；如果没有在循环过程就直接返回，函数返回flag的值。

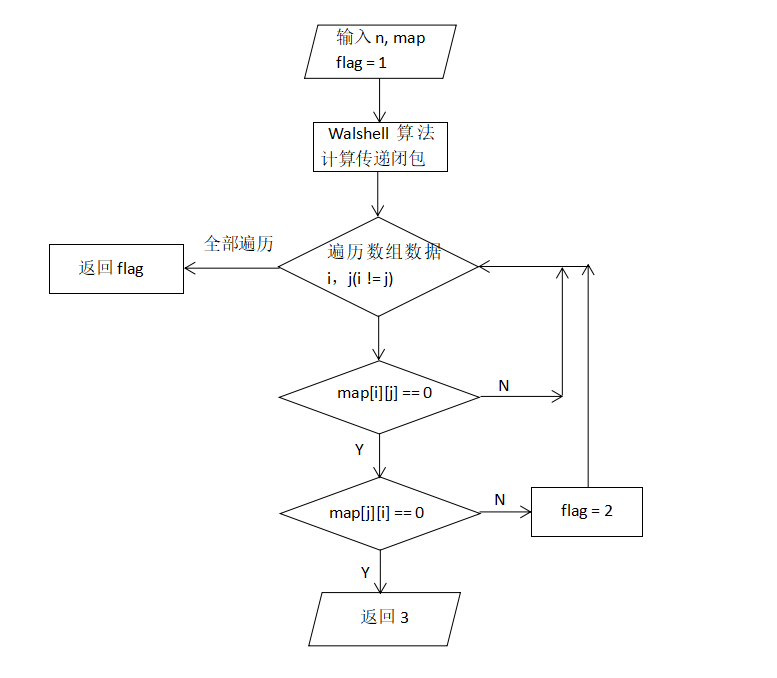


图 19 Walshell函数执行流程

### Output

用于执行输出：接收justify函数返回值，返回1输出 “A”，返回2输出 “B”，返回3输出 “C”。