```
 1  /* Problem 4--Bewitching Logic
 2     The crux was understanding the deductions you can make in
 3     propositional logic.
 4     If P==>Q is a rule, then !Q==>!P is a rule.
 5     P==>P and !P==>!P are always rules.
 6     If P==>Q and Q==>R are rules, then P==>R is a rule.
 7     If P==>Q is a rule and P is true, then Q is true.
 8     If P==>!P is a rule, then !P is true.
 9     If P==>Q and !P==>Q, then Q must be true */
10
11  import java.io.*;
12  import java.util.*;
13
14  public class prob4 {
15
16   private static Scanner in;
17   private static PrintWriter out;
18   private static int vars, stmts, start, cs;
19   private static boolean[][] ImpArr;
20
21   public static void main (String[] args) throws Exception {
22
23    in = new Scanner (new File ("prob4.in"));
24    out = new PrintWriter ("prob4.out");
25    cs = 1;
26    while (true) {
27     vars = in.nextInt(); //Read in the number of vars and statements
28     stmts = in.nextInt();
29     in.nextLine();
30     if (vars==0 && stmts==0) break;
31     ReadIn(); //Read in the propositions
32     Process ();
33    }
34    in.close ();
35    out.close ();
36   }
37
38   /* We store our implications in a 2-dimensional array of boolean.  Row
39      is antecedent, col is consequent.  Even rows/cols are affirmative,
40      odd are negated. */
41   public static void ReadIn () throws Exception {
42
43    ImpArr = new boolean[2*vars][2*vars]; //Load up tautologies
44    for (int i=0; i < 2*vars; i++) ImpArr[i][i] = true;
45    for (int i=0; i < stmts; i++) {
46     String line = in.nextLine ();
47     int ant, cons;
48     boolean negant=false, negcons=false; //Check for !
49     if (line.charAt(0)=='!') {negant=true; line = line.substring(1);}
50     int pos = 0;
51     for (; line.charAt(pos)!='='; pos++); //Check for ==>
52     ant = Integer.parseInt (line.substring (0,pos));
53     line = line.substring (pos+3);
54     if (line.charAt(0)=='!') {negcons=true; line = line.substring(1);}
55     cons = Integer.parseInt (line); //Check for !
56     int antloc = 2*ant;
57     if (negant) antloc++;
58     int consloc = 2*cons;
59     if (negcons) consloc++;
60     ImpArr[antloc][consloc] = true;  //Load up statement
61     antloc = 2*cons;
62     if (!negcons) antloc++;
63     consloc = 2*ant;
64     if (!negant) consloc++;
```

```
 65     ImpArr[antloc][consloc] = true;  //Load up contrapositive
 66    }
 67    start = Integer.parseInt (in.nextLine());
 68   }
 69
 70   /* We go through our matrix and apply the chain rule until we can apply
 71      it no more.  If we find a contradiction, we're done!
 72      The vals array stores a 1 for known true variables, a 0 for known
 73      false ones, and a -1 for unknown ones. */
 74   public static void Process () throws Exception {
 75
 76    int [] vals = new int[vars]; //Initialize all variables to -1
 77    for (int i=0; i < vars; i++) vals[i]=-1;
 78    vals[start] = 1; //And we know one true one
 79    boolean changes = true, contradiction = false;
 80    while (changes) { //Keep applying the chain rule until no more changes
 81     changes = false; //Can be made
 82     for (int i=0; i < 2*vars; i++)
 83      for (int j=0; j < 2*vars; j++)
 84       for (int k=0; k < 2*vars; k++)
 85        if (ImpArr[i][j] && ImpArr[j][k]) //CHAIN RULE
 86         if (!ImpArr[i][k]) {
 87          ImpArr[i][k] = true;
 88          changes = true;
 89         }
 90    }
 91    for (int i=0; i < vars && !contradiction; i++) {
 92     if (ImpArr[2*i][2*i+1]) //Apply P=>!P deduce !P
 93      if (vals[i]==1) contradiction = true;
 94      else vals[i]=0;
 95     if (ImpArr[2*i+1][2*i]) //Apply !P=>P deduce P
 96      if (vals[i]==0) contradiction = true;
 97      else vals[i]=1;
 98    }
 99    for (int i=0; i < vars && !contradiction; i++)
100     for (int j=0; j < vars && !contradiction; j++) {
101      if (ImpArr[2*i][2*j] && ImpArr[2*i+1][2*j])
102       if (vals[j]==0) contradiction = true;
103       else vals[j]=1; //Apply P==>Q !P==>Q to deduce Q
104      if (ImpArr[2*i][2*j+1] && ImpArr[2*i+1][2*j+1])
105       if (vals[j]==1) contradiction = true;
106       else vals[j]=0;
107     }
108    changes = true;
109    while (changes && !contradiction) {
110              //Using our known variables, derive information
111     changes = false; //until we can't derive anymore.
112     for (int j=0; j < vars && !contradiction; j++)
113      if (vals[j]!=-1) {
114       int st = 2*j;
115       if (vals[j]==0) st++;
116       for (int i=0; i < vars && !contradiction; i++) {
117        if (ImpArr[st][2*i])
118         if (vals[i]==0) contradiction = true;
119         else if (vals[i]==-1) {
120          vals[i] = 1;
121          changes = true;
122         }
123        if (ImpArr[st][2*i+1])
124         if (vals[i]==1) contradiction = true;
125         else if (vals[i]==-1) {
126          vals[i]=0;
127          changes = true;
128         }
```

```
129       }
130      }
131    } //Print answer
132    out.printf ("Case %d:\r\n\r\n",cs++);
133    if (contradiction) out.printf ("CONTRADICTION\r\n\r\n");
134    else {
135     out.printf ("TRUE\r\n");
136     for (int i=0; i < vars; i++)
137      if (vals[i]==1) out.printf ("%d ",i);
138     out.printf ("\r\n\r\n");
139     out.printf ("FALSE\r\n");
140     for (int i=0; i < vars; i++)
141      if (vals[i]==0) out.printf ("%d ",i);
142     out.printf ("\r\n\r\n");
143     out.printf ("UNDEFINED\r\n");
144     for (int i=0; i < vars; i++)
145      if (vals[i]==-1) out.printf ("%d ",i);
146     out.printf ("\r\n\r\n");
147    }
148   }
149  }
150
```