```
/* Problem 1--Pegs
   This may be the trickiest bit of programming in the contest.  This
   solution essentially attempts all possible moves, memoizing the
   results. */

import java.io.*;
import java.util.*;

public class prob1 {

 private static Scanner in;
 private static PrintWriter out;
 private static HashMap<String,String> hm; //Hash Table
 //Key is the board configuration, Data is the list of moves solving it
 //from there.
 private static int cs;

 public static void main (String[] args) throws Exception {

  in = new Scanner (new File ("prob1.in"));
  out = new PrintWriter ("prob1.out");
  cs = 1;
  hm = new HashMap ();
  int pegs;
  while ((pegs=in.nextInt()) > 0) { //Read in the input
   int pi = in.nextInt(), pj = in.nextInt();
   Board board = new Board (pegs,pi,pj);
   String s = Solvable (board); //Solve the board
   out.print ("Case "+(cs++)+"\r\n");
   if (s.equals ("")) out.print ("UNSOLVABLE\r\n\r\n");
   else {
    int sz = board.getsz(); //Print the board
    int size = s.length()/(sz*(sz+1)/2);
    int pos = 0;
    for (int ct=0; ct < size; ct++) {
     for (int i=0; i < sz; i++) {
      for (int j=0; j <= i; j++)
       out.print (s.charAt(pos++)=='1'?'O':'|');
      out.print ("\r\n");
     }
     out.print ("\r\n");
    }
   }
  }
  in.close ();
  out.close ();
 }

 /* accessors */
 public static Scanner getin() throws Exception {return in;}

 public static HashMap<String,String> gethm () throws Exception {
  return hm;
 }

 /* Solvable attempts to solve the board. */
 public static String Solvable (Board board) throws Exception {

  if (hm.containsKey (board.getstr())) //if we've already computed it,
   return hm.get (board.getstr());     //just return it
  if (board.getpct()==1) return board.getstr(); //one peg left, DONE
  String s;
  for (int i=0; i < board.getsz(); i++)
   for (int j=0; j <= i; j++) {
    s = Process (board,i,j,i+1,j); //try all possible moves
    if (!s.equals("")) return board.getstr()+s;
    s = Process (board,i,j,i-1,j);
    if (!s.equals("")) return board.getstr()+s;
```

```
 69      s = Process (board,i,j,i,j+1);
 70      if (!s.equals("")) return board.getstr()+s;
 71      s = Process (board,i,j,i,j-1);
 72      if (!s.equals("")) return board.getstr()+s;
 73      s = Process (board,i,j,i+1,j+1);
 74      if (!s.equals("")) return board.getstr()+s;
 75      s = Process (board,i,j,i-1,j-1);
 76      if (!s.equals("")) return board.getstr()+s;
 77     }
 78    return "";
 79   }
 80
 81   /* Process makes one move on the board and tries to solve it from
 82      there. */
 83   public static String Process (Board board, int pi, int pj,
 84                                 int hi, int hj) throws Exception {
 85
 86    if (!board.ValidHole (pi,pj) && !board.ValidHole (hi,hj) &&
 87         board.ValidHole (pi+2*(hi-pi),pj+2*(hj-pj))) {
 88     Board newboard = new Board (board);
 89
 90     newboard.setb (pi,pj,true);
 91     newboard.setb (hi,hj,true);
 92     newboard.setb (pi+2*(hi-pi),pj+2*(hj-pj),false);
 93     newboard.setpct (board.getpct()-1);
 94     String s = Solvable (newboard);
 95     hm.put (newboard.getstr(),s);
 96     return s;
 97    }
 98    return "";
 99   }
100  }
101
102  /* Board represents a possible board configuration */
103  class Board implements Cloneable {
104
105   private int sz, pct;
106   private boolean [][] b;
107
108   /* Accessors and Mutators */
109   public int getsz () throws Exception {return sz;}
110   public boolean getb (int i, int j) throws Exception {return b[i][j];}
111   public boolean[][] getarray () throws Exception {return b;}
112   public void setsz (int s) throws Exception {sz = s;}
113   public void setb (int i, int j, boolean B) throws Exception {
114    b[i][j] = B;
115   }
116   public int getpct () throws Exception {return pct;}
117   public void setpct (int p) throws Exception {pct = p;}
118
119   public Board (int s, int i, int j) throws Exception {
120    b = new boolean[s][s];
121    sz =s; b[i][j] = true; pct = sz*(sz+1)/2-1;
122   }
123
124   public Board (Board board) throws Exception {
125
126    sz = board.getsz ();
127    pct = board.getpct ();
128    b = new boolean[sz][sz];
129    for (int i=0; i < sz; i++)
130     for (int j=0; j <= i; j++)
131      b[i][j] = board.getb (i,j);
132   }
133
134   /* Checks whether a hole is valid in the game */
135   public boolean ValidHole (int i, int j) throws Exception {
136
```

```
137    return i >= 0 && i < sz && j >= 0 && j < sz && i >= j && b[i][j];
138   }
139
140   /* converts the board to a string for hashing */
141   public String getstr () throws Exception {
142
143    String s = "";
144    for (int i=0; i < sz; i++)
145     for (int j=0; j<=i; j++)
146      s += b[i][j]?'1':'0';
147    return s;
148   }
149  }
150
```