

1. This problem concerns a restricted form of the *predicate calculus* (logic with “for all” and “there exists”). In this restricted language, formulas are written in postfix form: operands first, followed by an operator. All operands and operators are separated by whitespace. Each formula is a string of

- *Variables*, written as lower-case letters a–z.
- The *relational operators* ‘>’ (greater than), ‘<’ (less than), and ‘=’ (equals), having their usual meanings on the set of integers.
- The *logical operators* ‘&’ (and), ‘|’ (or), ‘_’ (implies), and ‘~’ (not).
- The *constants*, optionally signed integer numerals.

Variables are implicitly *universally quantified*; that is, a formula is true iff it is true for all values of any variables in it.

For example the formula

$$i\ j\ >\ j\ i\ <\ _$$

denotes the true assertion “ $i > j$ implies $j < i$ for all i and j .” (In logic, A implies B iff A is false or B is true). The relational operators must be applied to variables and integer constants only; they yield true or false. The logical operators must be applied to logical values only. The formula must yield a logical value. You may assume all formulas obey this rule.

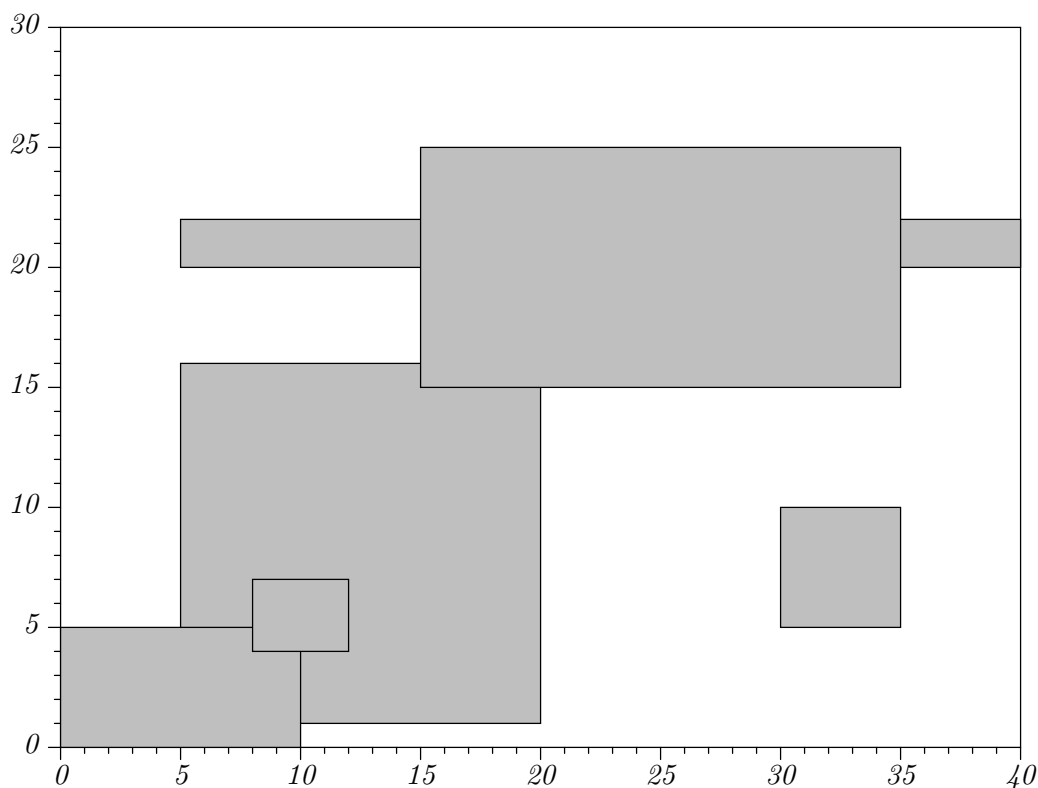
You are to write a program to read formulas and print whether each is true or false. Each formula is on a separate line, preceded by two integer constants, L and U , which indicate the lower and upper bounds over which the variables may range. You may assume $L \leq U$ and $U - L \leq 2^{20}$. The formula follows L and U , with its individual operators and operands separated by one or more blanks.

The output should echo the input and then say whether the formula is true, in the format shown in the examples below.

Example.

Input	Output
-1 1 i j > k j < & k i < _	i j > k j < & k i < _ is true in [-1..1]
0 1 i j = ~ k i = ~ k j = ~ & & 0 1 = _	i j = ~ k i = ~ k j = ~ & & 0 1 = _ is true in [0..1]
0 2 i j = ~ k i = ~ k j = ~ & & 0 1 = _	i j = ~ k i = ~ k j = ~ & & 0 1 = _ is false in [0..2]

2. A set of rectangular tiles are laid upon a rectangular floor, all with one edge parallel to one of the walls. The tiles may overlap. You are to write a program that determines how much of the floor is covered, given the sizes and positions of the rectangles. The problem, naturally, is in making sure that you count the areas where tiles overlap only once. For example, in the figure below, six tiles cover 505 square units of area, while the total area of the tiles is 582 square units.



The input to your program will consist of a sequence of sets of data in free format. Each set consists of two positive integers L and W , respectively giving the length and width of the floor, followed by quadruples of non-negative integers L_i , W_i , x_i , and y_i , each giving the length and width (L_i, W_i) of a rectangle, and the position of its lower-left corner (x_i, y_i). These numbers satisfy the following constraints.

$$0 \leq x_i \leq L - L_i, \quad 0 \leq y_i \leq W - W_i, \quad L_i, W_i > 0.$$

A quadruple of all 0's marks the end of a set of data. You may assume that any individual rectangle, and the total area covered by the rectangles are less than 2^{30} square units.

For each set of data, the output is to consist of a single message of the form

“Set k covers M square units”

where k is the number of this set of data (numbering from 0), and M is the amount of floor covered. Sample input and the resulting output follow.

Example.

Input	Output
40 30	Set 0 covers 505 square units
15 15 5 1 35 2 5 20	Set 1 covers 50 square units
10 5 0 0	
4 3 8 4	
20 10 15 15	
5 5 30 5	
0 0 0 0	
10 10	
5 4 0 0	
5 6 5 0	
5 4 0 0	
5 6 5 0	
0 0 0 0	

3. [Due to E. W. Dijkstra] Consider decimal numerals containing only the digits 1–3. A numeral is considered “good” if no two adjacent non-empty substrings of it are equal; otherwise it is “bad.” Hence, the numerals ‘1’, ‘12’, and ‘1213’ are good, while ‘11’, ‘32121’, and ‘121321312’ are bad.

You are to write a program that, given $n > 0$, finds the smallest good n -digit numeral. The input consists of a sequence of positive integers. For each of these integers, n , the output is to contain a line of the form

The smallest good numeral of length n is s .

where s is the answer. For example,

Input	Output
1 4	The smallest good numeral of length 1 is 1.
7	The smallest good numeral of length 4 is 1213.
9	The smallest good numeral of length 7 is 1213121.
	The smallest good numeral of length 9 is 121312313.