Problem I. Interactive Proofs

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

This is interactive problem.

A concept of interactive proof in complexity theory was introduced by Goldwasser, Micali and Rackoff in 1985. There are two parties: *Prover* and *Verifier* that have access ot a common word x and Verifier wants to determine whether x belongs to some set L, and Prover wants to convince Verifier that this is the case. Verifier is a randomized program that must work in polynomial time, but Prover's time and memory are not limited. Prover and Verifier interact by sending each other messages, and then Verifier either accepts, or rejects.

Verifier's goal is to accept only if x indeed belongs to L. Prover's goal is to force Verifier to accept. The proof system must generally satisfy the following conditions: if $x \in L$ Prover can make Verifier accept with high probability, if $x \notin L$ Verifier accepts with negligible probability no matter how hard the Prover tries.

In this problem we investigate the properties of one important interactive proof protocol for the $\#SAT$ problem.

Let φ be a boolean formula with n boolean variables denoted as $x_1, \ldots, x_n$. The allowed operations are *logical or* denoted as |, *logical and* denoted as &, and *logical not* denoted as !. Operations are listed from lowest precedence to highest, parenthesis can be used to change operations precedence as usually. Let us consider an example $\varphi_\oplus(x_1, x_2) = x_1\&x_2|!x_1\&!x_2$.

We will use 0 to denote *false* and 1 to denote *true* further on. The boolean vector $(a_1, \ldots, a_n)$ is said to satisfy φ if after setting $x_i = a_i$ for all i the value of φ is 1. For example, vectors $(0, 0)$ and $(1, 1)$ satisfy the formula $\varphi_\oplus$ above, and vectors $(1, 0)$ and $(0, 1)$ do not. The number of satisfying vectors is called the *weight* of φ and is denoted as $w(\varphi)$, so $w(\varphi_\oplus) = 2$.

We will now consider correct interaction protocol to prove that $w(\varphi) = k$. From the formal side we define the language $\#SAT = \{\langle\varphi, k\rangle | w(\varphi) = k\}$ and prove that $\langle\varphi, k\rangle \in \#SAT$.

First both Prover and Verifier pick a big engough prime p. All further calculations are performed modulo p. For the boolean formula φ denote its *arithmetization* $A(\varphi)$ recursively in the following way.

- If y is a variable, $A(y) = y$.
- If ψ is a formula and $\varphi = !\psi$, $A(\varphi) = 1 - A(\psi)$.
- If ξ and ψ are formulas and $\varphi = \xi\&\psi$, $A(\varphi) = A(\xi) \cdot A(\psi)$.
- If ξ and ψ are formulas and $\varphi = \xi|\psi$, $A(\varphi) = 1 - (1 - A(\xi)) \cdot (1 - A(\psi))$.

So, for example, $A(\varphi_\oplus) = 1 - (1 - x_1x_2)(1 - (1 - x_1)(1 - x_2))$.

It is easy to see, that if $(a_1, \ldots, a_n)$ is a satisfying assignment for φ then $A(\varphi)(a_1, \ldots, a_n) = 1$, and if it is not then $A(\varphi)(a_1, \ldots, a_n) = 0$. Therefore

$$w(\varphi) = \sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \ldots \sum_{x_n=0}^{1} A(\varphi)(x_1, \ldots, x_n).$$

The proof that $w(\varphi) = k$ proceeds in n steps.

At the first step consider the function

$$P_1(x_1) = \sum_{x_2=0}^{1} \ldots \sum_{x_n=0}^{1} A(\varphi)(x_1, \ldots, x_n).$$

This function is a polynomial in x_1: $P_1(x_1) = c_{1,d_1}x_1^{d_1} + c_{1,d_1-1}x_1^{d_1-1} + \ldots + c_{1,1}x_1 + c_{1,0}$. Clearly, $P_1(0) + P_1(1) = k$. Prover sends Verifier P_1 by transmitting d_1 followed by $c_{1,d_1}, \ldots, c_{1,0}$. Verifier checks that $P_1(0) + P_1(1) = k$ and randomly chooses r_1 — integer from 0 to $p-1$. Verifier sends r_1 to Prover.

Now the second step starts. Consider the function

$$P_2(x_2) = \sum_{x_3=0}^{1} \ldots \sum_{x_n=0}^{1} A(\varphi)(r_1, x_2, \ldots, x_n).$$

Note that x_1 is substituted by r_1 as an argument for $A(\varphi)$. This function is again a polynomial in x_2, so Prover sends this polynomial to Verifier. Verifier checks that $P_2(0) + P_2(1) = P_1(r_1)$, and randomly chooses r_2 — integer form 0 to $p-1$. Verifier sends r_2 to Prover.

In general the i-th step for i from 2 to $n-1$ proceeds as follows. The function

$$P_i(x_i) = \sum_{x_{i+1}=0}^{1} \ldots \sum_{x_n=0}^{1} A(\varphi)(r_1, \ldots, r_{i-1}, x_i, x_{i+1}, \ldots, x_n)$$

is considered. This function is a polynomial in x_i, Prover sends this polynomial to Verifier. Verifier checks that $P_i(0) + P_i(1) = P_{i-1}(r_{i-1})$, randomly chooses $r_i \in \{0, \ldots, p-1\}$ and sends r_i to Prover.

The last step differs just a little. The function

$$P_n(x_n) = A(\varphi)(r_1, \ldots, r_{n-1}, x_n)$$

is sent to Verifier. Verifier checks that $P_n(0) + P_n(1) = P_{n-1}(r_{n-1})$, chooses random $r_n \in \{0, \ldots, p-1\}$ and checks whether $A(\varphi)(r_1, \ldots, r_n) = P_n(r_n)$.

Please note again, that all calculations are performed modulo p.

If all tests preformed by Verifier were successful, the proof is accepted. If at least one test failed, the proof is rejected. It can be shown that if the degree of $A(\varphi)$ is d, the probability that the proof is accepted but $w(\varphi) \neq k$ is at most $1 - (1 - d/p)^n$, so choosing p big enough this value can be made negligible.

The core idea of this proof protocol that allows to decrease the probability of Prover forcing Verifier to accept the wrong statement is substituting random integers modulo p instead of just 0-s and 1-s in arithmetization of φ. If Verifier only chooses 0 or 1 as r_i Prover can easily cheat Verifier and force it to accept with high probability even if $w(\varphi) \neq k$. This is exactly what you have to do in this problem.

Your program will act as Prover and try to force jury's Verifier to accept the incorrect statement $w(\varphi) = k$. Verifier will follow the protocol described above with one exception: it will choose $r_i \in \{0, 1\}$. We will use $p = 239$.

Interaction Protocol

Your program will have to run several proofs in each run. Your program will be accepted if it successfully forces Verifier to accept in at least 1/2 cases.

Each proof starts with your program reading n — the number of variables in φ, from standard input ($2 \le n \le 7$). If $n = 0$ that means that test run is over and your program must terminate. If $n > 0$ the proof must start.

The following line of standard input contains formula for φ. Formula uses the following characters: '`&`', '`|`', '`!`', '`(`', '`)`', and '`x`' followed by its 1-based index from 1 to n. Spaces can separate formula parts freely, but '`x`' is never separated from its index. Length of formula is at most 100 characters.

The following line contains k, your program must try to prove that $w(\varphi) = k$ even if that is not the case, following the above protocol ($0 \le k \le 2^n$).

After reading n, φ and k your program must start protocol by sending P_1 to Verifier.

It must print its degree d_1 followed by $c_{1,d_1}, \ldots, c_{1,0}$ to standard output. Separate numbers by spaces or new line characters, print new line character after the last number and flush standard output.

Degrees of all polynomials must not exceed 20. All coefficients must be between 0 and 238. Constant zero polynomial has degree 0 for the purpose of this problem.

Verifier checks the condition $P_1(0) + P_1(1) = k$ and prints r_1 that you must read from standard input, r_1 will be 0 or 1.

After that you must send P_2 in the same format, and read r_2, and so on.

After sending P_n you must not wait for any more input for this test case and proceed to next test case.

Verifier doesn't terminate the protocol even if one of the tests it performed failed. Therefore your program must also run all n rounds of the protocol even if it found out that it cannot cheat Verifier. However it is not allowed to print incorrect polynomials (with incorrect degree or incorrect coefficients).

For each test your program will be asked to make at least 64 and at most 128 proofs and will be accepted if it cheats Verifier in at least 1/2 of all cases for this test case. The only exception is sample test, in which there is only 1 run and the program will be accepted any way if it follows the interaction protocol.

It is guaranteed that Verifier's answers do not depend on the polynomials sent to it by Prover.

Examples

standard input	*standard output*
2 x1 & x2 | !x1 & !x2 3 0 0	 1 1 1 1 238 1

In the example above the protocol proceeds as follows. The correct $P_1(x_1)$ is $P_1(x_1) = 1$. However, Prover cannot send it to verifier because $P_1(0)+P_1(1) = 2$, but it needs to prove that $w(\varphi) = 3$ (which is incorrect, of course). So Prover sends incorrect polynomial $P_1(x_1) = x_1+1$ which satisfies $P_1(0)+P_1(1) = 3$. Verifier checks this and chooses $r_1 = 0$.

Now the Prover sends $P_2(x_2) = -x_2 + 1$, or, since all calculations are performed modulo 239, $P_2(x_2) = 238x_2 + 1$. Verifier checks that $P_2(0) + P_2(1) = P_1(0) = 1$. Now it chooses r_2 either 0 or 1 and checks whether $A_\varphi(0, r_2) = P_2(r_2)$. Fortunately for the Prover this is the case for both $r_2 = 0$ and $r_2 = 1$, so independently of Verifier's last choice it accepts. The goal is reached, the Verifier is cheated.

The last 0 at standard input indicates the end of the test run, so Prover terminates.