Problem J. J

Input file:	`j.in`
Output file:	`j.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

> The J programming language, developed in the early 1990s by Kenneth E. Iverson and Roger Hui, is a synthesis of APL (also by Iverson) and the FP and FL function-level languages created by John Backus.
>
> Wikipedia. J (programming language)

APL language family is famous for its support of advanced operations on vectors and arrays, and J is not an exception. For example, all unary and binary numeric operations in these languages by default are applicable to vectors and arrays of different dimensions. Plus operation (‘`+`’) can add not only scalars, like in other languages, but also scalars and vectors (the scalar is added to each component of the vector), or vectors and vectors (the vectors are added component-wise).

The expressive power of J is amazing (as well as its cryptic syntax), but for this problem we need just a small subset of the language. We consider a single expression, where we may use one vector variable X, one scalar variable N — the length of the vector X, and the following operations:

- We can add (‘`+`’), subtract (‘`-`’) or multiply (‘`*`’) two vectors, vector and scalar, or two scalars.
- We can use unary minus (‘`-`’) and unary squaring operations (‘`*:`’) for scalars and vectors (component-wise).
- We can *fold* a vector with plus operation (‘`+/`’) — that is, compute the sum of a vector (unary operation).

Operations are evaluated right-to-left, natural precedence of operations is ignored in J. The order of evaluation can be altered by parentheses. More precisely the syntax is specified in the following BNF.

$$
\begin{aligned}
\langle \text{expression} \rangle &::= \langle \text{term} \rangle \mid \langle \text{term} \rangle\ (\text{`+'} \mid \text{`-'} \mid \text{`*'})\ \langle \text{expression} \rangle \mid (\text{`-'} \mid \text{`*:'} \mid \text{`+/'})\ \langle \text{expression} \rangle \\
\langle \text{term} \rangle &::= \text{`('}\ \langle \text{expression} \rangle\ \text{`)'} \mid \text{`X'} \mid \text{`N'} \mid \langle \text{number} \rangle \\
\langle \text{number} \rangle &::= (\text{`0'} \mid \text{`1'} \mid \ldots \mid \text{`9'})^{+}
\end{aligned}
$$

To correctly impose one more limitation on expression syntax, let us define *complexity* of an expression:

- complexity of scalars (numbers, ‘`N`’, and result of fold) is zero;
- complexity of ‘`X`’ is one;
- complexity of addition and subtraction is the maximum of their operands’ complexities;
- complexity of multiplication is the sum of its operands’ complexities;
- complexity of unary squaring is twice the complexity of its operand.

For example, the complexity of expression “`(3-+/*:*:X)-X**:X`” is 3, while the complexity of its subexpression “`*:*:X`” is 4.

Your program is given a scalar-valued expression and a value of the vector X, and it should compute the expression result modulo 10^9. The complexity of each subexpression in the given expression does not exceed 10.

Input

The first line contains one integer number N ($1 \le N \le 10^5$) — the length of the vector X.

The second line contains N integers — components of the vector X ($0 \le X_i < 10^9$).

The third line contains the expression to be computed, a non-empty string of not more than 10^5 symbols. Each number in the expression is less than 10^9. The fold is never applied to a scalar.

Output

Output a single integer number — the expression result modulo 10^9.

Examples

j.in	j.out
5 1 2 3 4 5 +/*:X	55
5 1 2 3 4 5 N++/X-X+1	0
3 11 56 37 +/(3-+/*:*:X)-X**:X	964602515

The first expression computes squared length of the vector X in Euclidean metrics.

The second expression result does not depend on X and always equals to zero.

The actual result of the third expression is -35397485.