Problem C. Combinator Expression

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

Combinatory logic may be thought as one of computational models allowing to express any computable function as a composition of functions from a small finite *basis*. In this problem we consider a restricted variant of BCKW basis, BCKI.

Combinator expression in BCKI basis is a string, corresponding to the following grammar:

$$\begin{aligned} \langle \text{Expression} \rangle &::= \langle \text{Expression} \rangle \langle \text{Term} \rangle \mid \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &::= \text{`('} \langle \text{Expression} \rangle \text{`)'} \mid \text{`B'} \mid \text{`C'} \mid \text{`K'} \mid \text{`I'} \end{aligned}$$

As we can see from the grammar, the expression is a tree of applications where leafs are combinators B, C, K and I. The application is left-associative. For example BIC is equivalent to $(BI)C$, but not to $B(IC)$.

For the sake of the explanation we will use lowercase English letters ($a \dots z$) to represent sub-expressions. These lowercase letters will not appear in real data. For example, BIC can be represented by BxC (that is, $B\underset{x}{I}C$), x ($\underset{x}{BIC}$), xy ($\underset{x}{BI}\underset{y}{C}$), Bxy ($B\underset{x}{I}\underset{y}{C}$), but not by Bx.

We say that in expression pq we apply p to q. We can employ our intuition by saying that p is a function and q is its parameter. However, the evaluation process is quite different from traditional computation — instead of passing values over fixed expression tree, we evaluate by altering that tree so that the result is also some combinator expression.

To evaluate an expression, we need to select some sub-expression, corresponding to one of the patterns specified in the table below — that is, there should exist such x (and maybe y and z) that the pattern from the table becomes equal to the sub-expression. Then we need to replace the sub-expression with the reduction result from the table.

Pattern	Reduction result	Description
$Bxyz$	$x(yz)$	Composition function (Zusammensetzungsfunktion)
$Cxyz$	$(xz)y$	Exchange function (Vertauschungsfunktion)
Kxy	x	Constant function (Konstanzfunktion)
Ix	x	Identity function (Identitätsfunktion)

After the replacement took place we must repeat the process, until there remains no suitable sub-expressions. This final expression is *normal form* of the original one.

For example, in expression $CIC(CB)I$ we can make the following letter assignment

$$\underset{C}{C}\,\underset{x}{I}\,\underset{y}{C}\,\underset{z}{(CB)}\,I$$

and see that $CIC(CB)I \equiv (((CI)C)(CB))I \equiv (((Cx)y)z)I$ contains C combinator pattern and thus reduces to $((xz)y)I \equiv I(CB)CI$:

$$(\underset{C}{C}\,\underset{x}{I}\,\underset{y}{C}\,\underset{z}{(CB)})I \;\to\; (\underset{x}{I}\,\underset{z}{(CB)}\,\underset{y}{C})I$$

One more example: $B((CK)I)IC$ expression. Let us first reduce combinator B:

$$(\underset{B}{B}\,\underset{x}{((CK)I)})\,\underset{y}{I}\,\underset{z}{C} \;\to\; \underset{x}{((CK)I)}(\underset{y}{I}\,\underset{z}{C})$$

Now, let's reduce the last I:

$$((CK)I)(\underset{I}{I}\,\underset{x}{C}) \to ((CK)I)C$$

And now we finish evaluation with two more reductions:

$$((\underset{C}{C}\,\underset{x}{K})\,\underset{y}{I})\,\underset{z}{C} \to (\underset{K}{K}\,\underset{x}{C})\,\underset{y}{I} \to C$$

It is possible to show that the normal form remains the same irrespectable to the order of evaluation. For example, the following evaluation order:

$$C(K(II)(\underset{I}{I}\,\underset{x}{C})) \to C(K(\underset{I}{I}\,\underset{x}{I})(C)) \to C((\underset{K}{K}\,\underset{x}{I})\,\underset{y}{C}) \to CI$$

leads to the same result as

$$C(K(\underset{I}{I}\,\underset{x}{I})(IC)) \to C((\underset{K}{K}\,\underset{x}{I})\,\underset{y}{(IC)}) \to CI$$

However, as you see, the number of reductions is different: 3 in the first case and 2 in the second. This poses an interesting problem — to find an evaluation order with the minimal number of reductions for a given formula.

Your task is to write a program which finds the minimal number of reductions required for a given combinator expression to be evaluated to its normal form.

Input

The only line of the input file contains a combinator expression corresponding to the grammar above. The length of the expression does not exceed 30 000. The expression contains no whitespaces or symbols not specified in the grammar.

Output

Output a single integer — the minimal number of reductions required for the given formula to evaluate it to normal form.

Examples

combinator.in	combinator.out
C(K(II)(IC))	2
CIBI	3
BBBBBCCCCCKKKKKIIIII	15