D. Programming Language

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Recently, Valery have come across an entirely new programming language. Most of all the language attracted him with template functions and procedures. Let us remind you that templates are tools of a language, designed to encode generic algorithms, without reference to some parameters (e.g., data types, buffer sizes, default values).

Valery decided to examine template procedures in this language in more detail. The description of a template procedure consists of the procedure name and the list of its parameter types. The generic type `T` parameters can be used as parameters of template procedures.

A procedure call consists of a procedure name and a list of variable parameters. Let's call a procedure *suitable* for this call if the following conditions are fulfilled:

- its name equals to the name of the called procedure;
- the number of its parameters equals to the number of parameters of the procedure call;
- the types of variables in the procedure call match the corresponding types of its parameters. The variable type matches the type of a parameter if the parameter has a generic type `T` or the type of the variable and the parameter are the same.

You are given a description of some set of template procedures. You are also given a list of variables used in the program, as well as direct procedure calls that use the described variables. For each call you need to count the number of procedures that are suitable for this call.

Input

The first line contains a single integer n ($1 \le n \le 1000$) — the number of template procedures. The next n lines contain the description of the procedures specified in the following format:

"`void procedureName (type_1, type_2, ..., type_t)`" ($1 \le t \le 5$), where `void` is the keyword, `procedureName` is the procedure name, `type_i` is the type of the next parameter. Types of language parameters can be "`int`", "`string`", "`double`", and the keyword "`T`", which denotes the generic type.

The next line contains a single integer m ($1 \le m \le 1000$) — the number of used variables. Next m lines specify the description of the variables in the following format:

"`type variableName`", where `type` is the type of variable that can take values "`int`", "`string`", "`double`", `variableName` — the name of the variable.

The next line contains a single integer k ($1 \le k \le 1000$) — the number of procedure calls. Next k lines specify the procedure calls in the following format:

"`procedureName (var_1, var_2, ..., var_t)`" ($1 \le t \le 5$), where `procedureName` is the name of the procedure, `var_i` is the name of a variable.

The lines describing the variables, template procedures and their calls may contain spaces at the beginning of the line and at the end of the line, before and after the brackets and commas. Spaces may be before and after keyword `void`. The length of each input line does not exceed 100 characters. The names of variables and procedures are non-empty strings of lowercase English letters and numbers with lengths of not more than 10 characters. Note that this is the only condition at the names. Only the specified variables are used in procedure calls. The names of the variables are distinct. No two procedures are the same. Two procedures are the same, if they have identical names and identical ordered sets of types of their parameters.

Output

On each of k lines print a single number, where the i-th number stands for the number of suitable template procedures for the i-th call.

Examples

input

```
4
void f(int,T)
void  f(T, T)
 void foo123   ( int,  double,  string,string  )
  void  p(T,double)
3
int a
 string    s
double x123
5
f(a,  a)
  f(s,a   )
foo   (a,s,s)
 f  (  s  ,x123)
proc(a)
```

output

```
2
1
0
1
0
```

input

```
6
void f(string,double,int)
void f(int)
   void f  ( T  )
void procedure(int,double)
void f  (T, double,int)
void f(string, T,T)
4
 int a
 int x
string  t
double  val
5
f(t, a, a)
f(t,val,a)
f(val,a, val)
 solve300(val, val)
f  (x)
```

output

```
1
3
0
0
2
```