You live on the planet *HomeSweetHome* — you are a Homer — and you are planning a holiday trip with some Homer friends. Your planet belongs to a very "populated" solar system — there are 20000 planets easily reachable with your new MegaSpeed X3 Turbo spaceship!

You have already sketched some possible paths and you are trying to choose one of them. There are some points that you find important that can make you choose one path over another. Surely everyone prefers to visit planets where they can breathe (so not to take air masks and oxygen bottles). However, this should be balanced with other characteristics; for example, whether the sights are beautiful, whether you can refuel your spaceship, whether the temperature is Homer-bearable, etc.

You have already gathered the most relevant information about the planets you are interested in visiting. You would like to have an easy way to perceive some global characteristics of every possible path, in order to make a more knowledgeable choice.

You need a program that, given a path and the characteristics of every planet along that path, tells you whether some given global properties hold in that path.

The relevant characteristics of each planet are represented by propositional symbols; for example `canBreathe`, `canSee`, `canWalk`, `canRefuel`, `hasFood`, `niceSights`, `hasWater`, `muchCold`, etc. The information about each planet is given to you as a set of propositional symbols that are known to be true for that planet; for example, `canSee, hasFood, canWalk` characterizes a planet where you are able to see what is around you, that has available food supplies, and where you can walk around freely.

A path is characterized by a number of planets and the properties of each one of them; the order by which the planets are described is important, insofar as it denotes the order by which they are going to be visited. The global characteristics of the path that you want to be able to "ask" the computer are expressed in the following language (given in BNF):

```
Expression ::= AllExpr | SomeExpr | NextExpr | proposition
AllExpr ::= 'A' integer Expression
SomeExpr ::= 'S' integer Expression
NextExpr ::= 'N' Expression
```

A proposition cannot start by a digit. Every expression is evaluated for a given planet — the **current** planet — within a path. The initial current planet is always the first planet in the path. The expression A*iprop* is true on the current planet if *prop* is true in all planets that are *i* or more positions ahead of the current one in the path. The expression S*iprop* is true on the current planet if *prop* is true in some of the planets that are *i* or more positions ahead of the current one in the path. The expression N*prop* is true on the current planet if *prop* is true in the planet that follows the current one in the path.

Let us suppose a path p_1 p_2 p_3 p_4 p_5. All expressions that we want to evaluate for the path start being evaluated in the first planet — p_1 — which is the initial current planet. Some examples follow:

`A0canBreathe` — all planets in the path are "breathable" (in the example, this expression is true for the path if it is the case that the proposition `canBreathe` is true in all p_1, p_2, p_3, p_4 and p_5 planets)

`A3hasFood` — from the third planet onwards, not including that third one, all planets in the path have available food supplies (in the example, this expression is true for the path if it is the case that the proposition `hasFood` is true in p_4 and in p_5)

`S0muchCold` — there is at least one planet in the path which is too much cold (in the example, this is true if at least one of the planets in the path has the characteristic `muchCold`)

`S2A1hasWater` — from the second planet onwards, not including that second one, there is some planet for which all the ones that follow it in the path have water (in the example, this expression is true if `A1hasWater` is true in at least one of the p_3, p_4 and p_5 planets; the expression `A1hasWater` is true in p_3, for example, if `hasWater` is true in all planets from p_4 onwards, that is, in p_4 and in p_5)

`A0S1hasWater` — in all planets we can expect that there is some planet in the rest of the path where there is water available (in the example, this expression is true for the path if `S1hasWater` is true in all planets of the path; the expression `S1hasWater` is true in p_2, for example, if `hasWater` is true in some of the p_3, p_4 and p_5 planets)

`A0NhasFood` — the successor planet of all planets in the path has food (in the example, this expression is true for the path if it is the case that the proposition `NhasFood` is true in all p_1, p_2, p_3, p_4 and p_5 planets; the expression `NhasFood` is true in p_1, for example, it the proposition `hasFood` is true in p_2)

Note that whenever it is not possible to evaluate an expression in a path, for example when the current planet does not exist, the expression evaluates to true. In the example, the expression `NhasFood` is true in p_5 because there is no Next planet following p_5 in the path.

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

The first line of input contains an integer P ($0 < P \leq 150$) that denotes the number of planets in the path.

The following P lines contain the propositions (separated by spaces) corresponding to the properties that each of the planets in the path has. Each proposition is a string of up to 10 characters (spaces are not allowed). There is a maximum of 30 propositions per planet.

The following line contains an integer Q ($0 < Q < 50$) which denotes the number of expressions that follow. The following Q lines contain an expression each. These expressions are written in the above specified language. The integers that are part of `AllExpr` and `SomeExpr` expressions may take values in the interval $[0, 150[$ (the upper limit is the maximum number of planets given above).

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The output has Q lines, each one containing the word 'yes' or 'no' depending on the validity of each of the Q expressions for the path given in the input.

Sample Input

```
4
canBreathe hasFood hasWater
canBreathe hasWater
hasWater muchCold
hasFood muchCold hasWater
24
A0canBreathe
A3hasFood
S0muchCold
S2A1hasWater
A0S1hasWater
A0NhasFood
A0hasWater
A1canBreathe
A3canBreathe
A2muchCold
A0S0muchCold
A0S3muchCold
A0S2NmuchCold
S122muchCold
NNA1canBreathe
NNA1hasFood
S2A0hasWater
S1A0muchCold
A1S1muchCold
A1S1canBreathe
A2S3muchCold
A0NS2muchCold
A0NS2canBreathe
A1S0A0NhasWater
```

Sample Output

```
no
yes
yes
yes
yes
no
yes
no
no
yes
yes
yes
yes
yes
no
yes
yes
yes
yes
no
yes
yes
no
yes
```