7256 Matching Compressed String

You are given a long string and looking for certain patterns in the string.

The string contains only lowercase letters (`a-z`), and it is represented in a compressed format. Denoting $S_1, S_2, \ldots$ as compressed strings, another compressed string S is defined recursively in one of the following ways:

- S can be any string consisting of only lowercase letters (`a-z`).
- S can be generated by repeating another string for any times. Specifically, S is represented as "$R(S_1)$", which means that the content of S_1 is repeated R times.
- S can also be the concatenation of other strings. Specifically, S is represented as "$S_1 S_2 \ldots S_L$", which means S is the concatenation of $S_1, S_2, \ldots, S_L$.
- An empty string ("") is also a valid representation.

Formally, the Backus–Naur Form (BNF) specification of the syntax is

$<compressed>$ `::=`
"" | $<lowercase-letter>$ | $<compressed><compressed>$ | $<number>$ "`(`" $<compressed>$ "`)`"

For example, the string "`baaabbaaab`" can be compressed as "`b3(a)2(b)3(a)b`". It can also be compressed as "`2(b3(a)b)`".

On the other hand, you find deterministic finite automaton (DFA) as powerful way to describe the patterns you are looking for. A DFA contains a finite set of states Q and a finite set of input symbols called the alphabet Σ. Initially, the DFA is positioned at the start state $q_0 \in Q$. Given the transition function state $\delta(q, a)$ and an input symbol a, the DFA transit to state $\delta(q, a)$ if its current state is q.

Let $w = a_1 a_2 \ldots a_n$ be a string over the alphabet Σ. According to the above definition, the DFA transits through the following sequence of states.

$$q_0, q_1 = \delta(q_0, a_1), q_2 = \delta(q_1, a_2), \ldots, q_n = \delta(q_{n-1}, a_n)$$

The DFA also contains a set of accept states $F \subseteq Q$. If the last state q_n is an accept state, we say that the DFA accepts the string w. The set of accepted strings is referred as the language that the DFA represents.

Now you are given a compressed string S and a DFA A. You want to know if A accepts the decompressed content of S.

Input

The first line of input contains a number cases T indicating the number of test ($T \leq 200$).

The first line of each test case contains a non-empty compressed string S, as described above. The length of S is not greater than 10000, and $0 \leq R \leq 10^9$. It is guaranteed that the representation of S is valid.

The description of the DFA follows.

The first line of the description contains three integers N, M, and K, indicating the number of states, the number of rules describing the transition function, and the number of accept states ($1 \leq K \leq N \leq 1000$, $0 \leq M \leq 26N$). The states are numbered from 0 to N–1.

The start state is always 0.

The second line contains K integers representing the accept states. All these numbers are distinct.

Each of the next M lines consists of two states p and q, and an input symbol a, which means that the DFA transits from p to q when it receives the symbol a. The symbol a is always a lowercase letter. It is guaranteed that, given p and a, the next state q is unique.

Output

For each test case, output a single line consisting of '`Case #`X`:` Y'. X is the test case number starting from 1. Y is '`Yes`' if the DFA accepts the string, or '`No`' otherwise.

Sample Input

```
3
2(b3(a)b)
2 3 1
0
0 1 b
1 0 b
1 1 a
b3(a)2(b)3(a)b
2 2 1
1
0 1 b
1 0 a
b3(a)2(b)3(a)b
2 4 1
0
0 1 b
0 1 a
1 0 a
1 0 b
```

Sample Output

```
Case #1: Yes
Case #2: No
Case #3: Yes
```