Bonus Problem #1:
Stacks

Imagine, that you are employed by a software development company. You work now on the famous "D++ project", which is devoted to the creation of a new generation programming language. Your particular task is quite prosaic, though. You are to develop the memory manager being able to work with a large number of stacks.

Input:

The first line of the input contains the total number of stack operations N, $0 < N \leq 100000$. Each of the next N lines contains a description of a stack operation, either in the form PUSH A B (meaning to push B into stack A), or in the form POP A (meaning to pop an element from stack A), where A is the number of stack ($1 \leq A \leq 1000$), and B is an integer ($0 \leq B \leq 109$). You may assume, that every operation is correct (i.e., before each POP operation, the respective stack is not empty).

Output:

For each POP operation, described in the input, output the value, which this POP operation gets from the top of that stack, to which it is applied. Numbers should appear according to the order of the POP operations in the input. Each number should be output in a separate line.

Sample Input:

```
7
PUSH 1 100
PUSH 1 200
PUSH 2 300
PUSH 2 400
POP 2
POP 1
POP 2
```

Sample Output:

```
400
200
300
```

Bonus Problem #2: Card Hands

Jim is writing a program for statistically analyzing card games. He needs to store many different card hands in memory efficiently. Each card has one of four suits and one of thirteen values. In his implementation, each hand is stored as a linked list of cards in a canonical order: the cards are first ordered by suit: all the clubs come first, followed by all the diamonds, then all the hearts, and finally the spades. Within each suit, the cards are ordered by value: A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K. Each hand contains at most one of any given card.

The card hands are using lots of memory. Jim therefore decides to try a more efficient representation. Whenever any two lists share a common tail, they can be updated to share one copy of the tail, and the other copy can be discarded. This process can be repeated until no two lists share a common tail.

Your job is to tell Jim how many linked list nodes he needs to store all the card hands.

Input:

The input contains several test cases followed by a line containing 0. The first line of each case contains the number of card hands. Each subsequent line describes a card hand. It starts with a number indicating the number of cards in the hand. The cards follow, separated by spaces, in the canonical order defined above. For each card, the value is given first, followed by the suit (C, D, H, or S). There are at most 100,000 cards in all hands.

Output:

For each test case, output a line containing the number of linked list nodes needed to store all the lists.

Sample Input:

```
3
3 7D AH 5S
4 9C 3D 4D 5S
2 AH 5S
0
```

Sample Output:

```
6
```

Bonus Problem #3:
Good Substrings

You have string s, consisting of lowercase English letters. Some of the English letters are good, the rest are bad. You are given a list of all the "bad" letters in the English alphabet. A substring of string s is any continuous section of the string up to the total length of the string. For instance, the string "abc" has substrigs "a", "b", "c", "ab", "bc", and "abc". A substring of s is considered "good" when the substring contains at most k "bad" letters where k is given to you (It may help to look at an example to understand this better). Your task is to find the number of "good" substrings given a string s and a list of "bad" letters

Input:

The first line of the input is the non-empty string s, consisting of lowercase English letters, the string's length is at most 1500 characters. The second line of the input is the string of characters "0" and "1", the length is exactly 26 characters. If the i-th character of this string equals "1", then the i-th English letter is good, otherwise it's bad. That is, the first character of this string corresponds to letter "a", the second one corresponds to letter "b" and so on. The third line of the input consists a single integer k which is the maximum acceptable number of bad characters in a good substring.

Output:

Print a single integer the number of distinct good substrings of string s.

Sample Input #1:

```
ababab
01000000000000000000000000
1
```

Sample Output #1:

```
5
```

Sample Input #2:

```
acbacbacaa
00000000000000000000000000
2
```

Sample Output #2:

```
8
```

Note:

In the first example there are following good substrings: "a", "ab", "b", "ba", "bab".

In the second example there are following good substrings: "a", "aa", "ac", "b", "ba", "c", "ca", "cb".