PREREQUISITES:

Observation

PROBLEM:

You have a string S, K strings $P_1, \ldots, P_K$, and some characters given in a frequency array C.

For each x from 0 to K, find out whether it's possible to make S greater than exactly x of the given strings.

EXPLANATION:

One immediate observation we can make is that appending characters to S will never make it any smaller lexicographically: it can only make it larger.

Let's try to solve for a specific value of x first.

To make things easier for us, first sort the strings P_i in increasing order, i.e, $P_1 \leq P_2 \leq \ldots \leq P_K$.
Now, notice that making S greater than exactly x strings means we want to append some characters to S to make it satisfy $P_x < S \leq P_{x+1}$.
In particular, it's enough to only look at the two strings P_x and P_{x+1}, so let's do that.

From here, we can do a bit of basic casework to throw out a few simple cases, each time making what S we have to deal with more specific.

- If $S > P_{x+1}$, obviously the answer is No.
  - Now we have $S \leq P_{x+1}$.
- If $P_x = P_{x+1}$, again the answer is No.
  - Now we have $S < P_{x+1}$.
- If $S > P_x$, we don't need to append anything: we're already done and the answer is Yes.
  - Now we have $S \leq P_x < P_{x+1}$.
- If the length of S is greater than the length of P_x, then the answer is No.
  - Now we further have $|S| \leq |P_x|$.
- If S is not a prefix of P_x, once again the answer is No.
  - Now S is a prefix of P_x.

This is the end of the 'simple' cases, which don't depend on the extra characters in C at all.
From here on, we have to check whether appending some characters to S can make it $> P_x$.

This check can be done greedily position-by-position, starting from $i = N + 1$.
When we are at position i,

- If S is a prefix of both P_x and P_{x+1}, and we can append some character that is $> P_{x,i}$ but $\leq P_{x+1,i}$, we are immediately done and the answer is `Yes`.
- If S is not a prefix of P_{x+1}, then simply check if we can append any character $> P_{x,i}$: if we can, the answer is `Yes`.
- If the above cases are not possible, note that our only hope of making $S > P_x$ in the future is to append exactly $P_{x,i}$ and maintain S as a prefix of P_x, so check if this is possible.
  - If it is possible, do so. Make sure to update whether S is still a prefix of P_{x+1} or not.
  - If it is not possible, the answer is immediately `No`.
- A little bit of care needs to be taken when i is larger than the length of P_x and/or P_{x+1}: these need to be special-cased.

Note that this algorithm runs in something like $\mathcal{O}(26 \cdot (|P_x| + |P_{x+1}|))$ time.

So, simply running this algorithm for every x is already fast enough! Every string is processed twice this way, giving us a solution in $\mathcal{O}(26 \sum |P_i|)$ which is good enough.

Depending on your implementation, you might have to take special care of $x = 0$ and $x = K$, since the strings P_0 and P_{K+1} don't exist.

TIME COMPLEXITY

$\mathcal{O}(N + K + 26 \sum |P_i|)$ per test case.

```
n = int(input())
s = input()
have = list(map(int, input().split()))
k = int(input())
strings = []
for i in range(k):
      strings.append(input())
strings.sort()
strings.append((len(strings[-1])+5)*'z')

print('Yes' if s <= strings[0] else 'No')

for i in range(k):
      if strings[i] == strings[i+1] or s > strings[i+1]:
            print('No')
            continue
      if s > strings[i]:
            print('Yes')
            continue
      if len(strings[i]) < n or strings[i][:n] != s:
            print('No')
            continue
      used = [0 for _ in range(26)]
      ans = 'Yes'
      ispref = strings[i+1][:n] == s
      for pos in range(n, 200000):
            hi = 0
            if ispref == True: hi = ord(strings[i+1][pos]) - ord('a') + 1
            else: hi = 26

            if pos == len(strings[i]):
                  ans = 'No'
                  for c in range(hi):
                        if used[c] < have[c]:
                              ans = 'Yes'
                              break
                  break

            lo = ord(strings[i][pos]) - ord('a') + 1
            ans = 'No'
            for c in range(lo, hi):
                  if used[c] < have[c]:
                        ans = 'Yes'
                        break
            if ans == 'Yes': break
            if used[lo-1] < have[lo-1]:
                  used[lo-1] += 1
                  if ispref == True and strings[i][pos] != strings[i+1][pos]:
ispref = False
                  ans = 'Yes'
                  continue
            ans = 'No'
            break
      print(ans)
```