Editorial: Did we get everything covered - 1924A

Avighna

June 2024

TL;DR: The strategy is to keep finding the entire alphabet (limited by k) using the least characters from the string. If you can successfully do this n times, the answer is yes. Otherwise, it's no: print a string with the last character of every iteration and then the character you couldn't find (until the string's length is n).

Let's define a 'valid string' as a string t such that $t_i \in S = \{c : \text{'a'} \leq c \leq \text{'a'} + k - 1\}$. For every valid string of length n to be a subsequence of s, the same needs to hold for valid strings of length $n - 1$, since they are just valid strings of length n with a character removed.

Let's look at this example test case: $n = 3, k = 3$, $s =$ aabbccabab.

Starting from the beginning of s, we need to find the shortest substring that contains every string of length 1 (with each character $\in S$) as a subsequence.

In our example, this substring would be aabbc. We need to be able to add any arbitrary letter $c \in S$ to the end of any of these subsequences. So, for example, ca, cb, and cc all need to exist as subsequences of our original string. Since we can form the string c (a length 1 subsequence), we need to check that any $c \in S$ is present in the next characters of the string: cabab.

Repeating this step n times successfully guarantees that all valid strings of length n can be formed. If we can't find the first k letters of the alphabet in the next part of the string at any point, then we know that we can't find all valid strings as substrings of s. A simple counter-example can be formed by creating a string with all the last characters of s that were iterated over in each iteration (you add one character per iteration) to find a substring that contained the entire first k letters of the alphabet. For the remaining characters, append a character that couldn't be found in the current iteration.

The code for this is as follows:

```
#include <bits/stdc++.h>

using namespace std;

#define ll long long

void solve() {
    ll n, k, m;
    string s;
    cin >> n >> k >> m >> s;
    string ans;
    for (ll j = 0, i = 0; j < n; ++j) {
    vector<bool> present(k);
    ll present_count = 0;
    char c = ' ';
    for (; i < m && present_count < k; ++i) {
        ll ch = s[i] - 'a';
        c = s[i];
        if (ch < k && !present[ch]) {
```

```
        present[ch] = true;
        present_count++;
        }
    }
    if (present_count != k) {
        cout << "NO\n"
            << ans
            << string(n - ans.length(), 'a' + (find(present.begin(), present.end(), false) - present.
        return;
    }
    if (c != ' ') {
        ans.push_back(c);
    }
    }
    cout << "YES\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    ll t;
    cin >> t;
    while (t--) {
    solve();
    }
}
```