Editorial: Unforgivable Curse (hard version) - 1800E2

Avighna

June 2024

Note that s_i denotes the i^{th} character of s, but `s.substr(x, y)` denotes the substring starting from index x (that, is, s_{x+1}) with a length of y. The former is one-indexed while the latter is zero-indexed.

We can swap s_i with s_{i+k} and with s_{i+k+1}, so when $n \leq k$., we can't perform any swaps and we need $s = t$ from the beginning.

In a substring (x) of length $k + 2$, we can perform the following swaps:

1 Swap Sequence 1

$[x_1, x_2, ..., x_{k+1}, x_{k+2}]$, swap x_1 and x_{k+1}
$\rightarrow [x_{k+1}, x_2, ..., x_1, x_{k+2}]$, swap x_{k+1} and x_{k+2} (note that these two characters **are** at a distance of $k + 1$ from each other)
$\rightarrow [x_{k+2}, x_2, ..., x_1, x_{k+1}]$, swap x_1 and x_{k+2}
$\rightarrow [x_1, x_2, ..., x_{k+2}, x_{k+1}]$

to swap s_{k+1} with s_{k+2}. To swap s_1 with s_2, we can do this:

2 Swap Sequence 2

$[x_1, x_2, ..., x_{k+1}, x_{k+2}]$, swap x_2 and x_{k+2}
$\rightarrow [x_1, x_{k+2}, ..., x_{k+1}, x_2]$, swap x_1 and x_2
$\rightarrow [x_2, x_{k+2}, ..., x_{k+1}, x_1]$, swap x_1 and x_{k+2}
$\rightarrow [x_2, x_1, ..., x_{k+1}, x_{k+2}]$

which is really the same thing as swap sequence 1, but mirrored.

Thus, in s, using swap sequence 1, we can swap s_{k+1} with s_{k+2} by considering `x = s.substr(0, k + 2)`. Similarly, we can swap s_{k+2} with s_{k+3} by considering `x = s.substr(1, k + 2)`. If we keep doing this, we arrive at the conclusion that we can swap s_i with s_{i+1} $\forall i \in [k + 1, n - 1]$. Since we can swap all adjacent characters ahead of s_i, we can shuffle these characters however we want.

By considering substrings of length $k + 2$ starting from the back (that is, $s.\text{substr}(n - k - 2, n - 1)$, etc.), we'll be able to swap s_i with s_{i+1} $\forall i \in [1, n - k - 1]$

If these two intersect, that is, if s_{i+1} for $i = n - k - 1$ overlaps with s_i for $i = k + 1$, then we'll be able to swap any two characters of the entire string. Let's calculate when this happens:

$(n - k - 1) + 1 \geq k + 1$
$\rightarrow n - k \geq k + 1$
$\rightarrow n \geq 2k + 1$ or $n > 2k$

So if $n > 2k$, we can just sort s and t and check for equality. If they're equal after sorting, we can transform s to t, otherwise we can't.

What if $n = 2k$? We can swap any two characters among the first k characters and the last k characters, but can we swap s_k and s_{k+1}? If we could find a way to do this, we'd be able to swap any two characters of our string and we could then perform the same 'sort and check for equality' maneuver.

Our string is currently $[s_1, ..., s_k, s_{k+1}, ..., s_n]$. Let's perform the following swaps:

3 Swap Sequence 3

$[s_1, s_2, ..., s_{k-1}, s_k, s_{k+1}, s_{k+2}, ..., s_n]$, swap s_1 and s_{k+1}
$\rightarrow [s_{k+1}, s_2, ..., s_{k-1}, s_k, s_1, s_{k+2}, ..., s_n]$, move s_{k+1} to be ahead s_k (we can do this since we can swap any two characters in the first half)
$\rightarrow [s_2, ..., s_{k-1}, s_k, s_{k+1}, s_1, s_{k+2}, ..., s_n]$, move s_k to the beginning of the string
$\rightarrow [s_k, s_2, ..., s_{k-1}, s_{k+1}, s_1, s_{k+2}, ..., s_n]$, swap s_k with s_1 (again, they're at a distance of $k+1$)
$\rightarrow [s_1, s_2, ..., s_{k-1}, s_{k+1}, s_k, s_{k+2}, ..., s_n]$

Using this swap sequence, we've successfully swapped s_k with s_{k+1}. Here's a concrete example that uses swap sequence 3: $n = 6, k = 3, s =$ abcdef: abc def
→ dbc aef
→ bcd aef
→ cbd aef
→ abd cef

For $n < 2k$, any character s_i $\forall i \in [n-k+1, k]$ cannot be swapped with any other character since they're at a distance of less than k from every other character (it is sufficient to check the first and last characters to prove this. Go ahead and try, you'll end up getting four inequalities). So `s.substr(n - k, 2k - n)` must equal `t.substr(n - k, 2k - n)`.

This solution has a time complexity of $\mathcal{O}(n \log n)$.

4 Code

```
std::string get(bool b) {
    return b ? "YES" : "NO";
}

void solve() {
    ll n, k;
    std::string s, t;
    std::cin >> n >> k >> s >> t;

    if (n <= k) {
        std::cout << get(s == t) << "\n";
```

```
    } else {
        bool ans = true;
        if (2 * k - n > 0) {
            ans = s.substr(n - k, 2 * k - n) == t.substr(n - k, 2 * k - n);
        }
        std::sort(s.begin(), s.end());
        std::sort(t.begin(), t.end());
        std::cout << get(ans && s == t) << "\n";
    }
}
```