Another Substring Query Problem (ASQP)

There are several different approaches to this problem that will work. Here, we describe two such techniques. Either of these approaches could be used to solve the problem:

1. The Aho-Corasick algorithm
2. Rolling hashes

ASQP: Aho-Corasick algorithm approach (1)

The Aho-Corasick Algorithm

Given m pattern strings, whose lengths sum to T, to search for in text string s of length n, the Aho-Corasick algorithm can find all matches of all m patterns in s with time complexity $O(n + T + r)$, where r is the number of matches.

We can read in all query strings in advance and take these as the patterns for input to the Aho-Corasick algorithm, after removing duplicates. The algorithm will produce the indices of all matches. By storing these indices in an array for each pattern we can determine the index of the k^{th} occurrence of each pattern in the text string, or that there are fewer than k occurrences.

But how do we know that there will not be too many matches for this approach to be efficient enough?

ASQP: Aho-Corasick algorithm approach (2)

We can show that the number of matches is bounded above by $\sqrt{T}$, the square root of the sums of the lengths of the query strings.

- In the text string, s, with $|s| = n$, there are at most n substrings of length ℓ for integer $\ell \geq 1$.
- Each of these substrings matches at most one pattern from the distinct query strings.
- So there are at most n matches of substrings of length ℓ for each integer $\ell \geq 1$.
- There are $O(\sqrt{T})$ distinct lengths of patterns. To prove this, it can be shown that $1 + \cdots + (\left\lceil \sqrt{2T} \right\rceil + 1) \geq T$.
- Therefore there are $O(n\sqrt{T})$ matches in text string s.

This gives a complexity of $O(n + t + n\sqrt{T}) = O(n\sqrt{T})$.

ASQP: Rolling Hash approach (1)

Similar to the Aho-Corasick approach, we can read in all query strings and find all matches in the text string, s, before answering any of the queries. This time instead of using the Aho-Corasick algorithm to find the matches we will use a hashing function.
For a string, r, of length u define

$$h(r) = (r[0] \cdot p^{u-1} + r[1] \cdot p^{u-2} + \cdots + r[u-2] \cdot p + r[u-1]) \bmod v,$$

where $r[i]$ is the alphabetic index of the i^{th} character of r, and p is a prime similar to the size of the alphabet (e.g., $p = 31$) and v is a large prime (e.g., $10^9 + 7$).
Compute $h(t)$ for each query string t and assume that strings with the same hash value are equal. We will then look for substrings of s that have the same hash value as those query strings. But how do we do this efficiently?

ASQP: Rolling Hash approach (2)

Observation 1

If we know the value $h(s[i,j])$ we can compute $h(s[i+1,j+1])$ in $O(1)$ operations. Specifically,

$$h(s[i+1,j+1]) = ((h(s[i,j]) - s[i] \cdot p^{j-i}) \cdot p + s[j+1]) \bmod v.$$

Observation 2

Let T be the sum of the lengths of the query strings. There are $O(\sqrt{T})$ distinct string lengths for the query strings, since it can be shown that $1 + \cdots + (\left\lceil \sqrt{2T} \right\rceil + 1) \geq T$.

ASQP: Rolling Hash approach (3)

By combining these two observations we can consider each query string length one at a time and iterate over the text to get the hash values for each substring of s of some fixed length ℓ.

For each distinct hash value obtained from all substrings of s with a length matching at least one query string's length we can store a list of indices at which this hash value appears. These lists of indices can then be used to answer each query in $O(1)$ time.

Overall this approach has complexity $O(n\sqrt{T})$.

Note that there is some probability that hash collisions will result in incorrect answers. To reduce the probability of collisions, multiple hash values can be stored for each substring by using alternative large prime moduli for each hash function.