Calculation of the hash of a string

We define the **polynomial rolling hash function** of the string s of length n as

$$\text{hash}(s)=s[0]+s[1]\cdot p+s[2]\cdot p^2+...+s[n-1]\cdot p^{n-1} \bmod m$$

$$=\sum_{i=0}^{n-1} s[i]\cdot p^i \bmod m,$$

where p and m are some chosen, positive numbers.

It is reasonable to make p, a prime number roughly equal to the number of characters in the input alphabet. For example, if the input is composed of only lowercase letters of the English alphabet, p=31 is a good choice.

Here we assume that roughly our hash function will uniformly map a string to some number in range [0, m) so the probability of collision is 1/m. Obviously, m should be a large number to minimize the collisions. A good choice for m is some large prime numbers because prime numbers have a unique property of giving modulo uniformly from [0, m). The code below will just use $m=10^9+9$.

Example :

Consider that we want to calculate the hash of the string "ali". We will take the value of p = 31 and m = $10^9 + 7$.

We convert each character of s to an integer. Here we use the conversion a→1, b→2, ..., z→26. Converting a→0 is not a good idea, because then the hashes of the strings a, aa, aaa, ... all evaluate to 0.

So,

hash("ali") = $(1 + 11*31 + 8*31^2)$ % $10^9 + 7 = 8030$

We now look at the implementation of the above algorithm.

```
/*
        function to calculate the hash of string s
*/

function compute_hash(s) {
```

```
	//   constants p and m as described above
	const p = 31;
	const m = 1e9 + 9;

	//   hash_value is the rolling hash of the string
	hash_value = 0;

	//   p_pow is a variable to store the current exponent of p
	p_pow = 1;

	for (char c : s) {
		hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
		p_pow = (p_pow * p) % m;
	}

	return hash_value;
}
```

Time Complexity: O(N), where N is the length of the given string, as we are traversing the string once.

Space Complexity: O(1) since constant space is used.

Note: Precomputing the powers of p might give a performance boost.

Fast hash calculation of substrings of a given string

Now suppose we are given a string s and indices i and j, we are interested in finding the hash of the substring s[i…j].

By definition, we have:

$$\text{hash}(s[i \dots j]) = \sum_{k=i}^{j} s[k] \cdot p^{k-i} \bmod m,$$

Multiplying by p^i gives:

$$hash(s[i \dots j]).p^i = \sum_{k=i}^{j} s[k] \cdot p^k \bmod m,$$

$$= hash(s[0 \dots j]) - hash(s[0 \dots i-1]) \bmod m$$

So we can precompute the prefix hash i.e hash(s[0..i]) for each i from 0 to n - 1. Then we can use the above formula to calculate the hash of any substring by multiplying with the modular inverse of p^i. We can pre-compute the inverse of p^i at the beginning to prevent computing it, again and again, every time we query a substring.

Applications of Hashing

- Rabin-Karp algorithm for pattern matching in a string.
- Calculating the number of different substrings of a string.
- Hashing also has applications in various cryptography techniques.