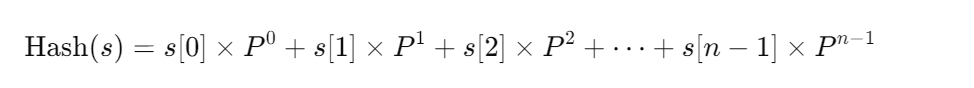
The polynomial rolling hash function is a common technique used in string matching and related algorithms to efficiently compute hash values for substrings. Here's how the concept works in the context of your code:

**1. Polynomial Hash Function:**

The core idea behind a polynomial hash function is to represent a string as a number, where each character is treated as a digit in a number system with a chosen base (denoted by P). The hash of a string of length n, s[0:n-1], can be calculated as:



Here:

* s[i] is the integer representation of the character at position i. In your code, this is computed as ord(c) - ord('a') + 1, which maps characters to integers (e.g., 'a' becomes 1, 'b' becomes 2, etc.).
* P is a base or constant (in your code, P = 31), which is often chosen as a small prime number.
* The hash value is computed modulo M (in your code, M = 10^9 + 9) to prevent overflow and to ensure that hash values remain within a manageable range.

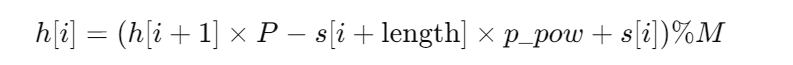
**2. Rolling Hash Technique:**

The rolling hash technique allows for efficient recalculation of hash values when sliding over a string. Instead of recomputing the entire hash from scratch for every substring, you can use the previously computed hash to derive the new hash by adding the next character and removing the previous one. This gives the algorithm its efficiency, especially when searching for common substrings.

In your code, this rolling hash technique is implemented as follows:

* For the first substring of a given length, the entire hash is computed from scratch.
* For subsequent substrings, the hash is updated by removing the contribution of the character that's sliding out of the window (using subtraction) and adding the contribution of the new character that's sliding into the window (using addition).

This is done with the following formula in your code:



* h[i+1] is the hash of the next substring (i.e., the one to the right of the current substring).
* s[i + length] is the character that's sliding out of the window.
* p\_pow is P^{\text{length}} mod M, which represents the positional weight of the character that's sliding out.
* s[i] is the new character that's sliding into the window.

**3. Why Polynomial Rolling Hash is Useful:**

* **Efficiency:** The rolling hash allows you to compute hash values for all substrings of a fixed length in linear time, O(n), where n is the length of the string. This is much faster than recomputing the hash for each substring from scratch, which would take O(n \times m) time, where m is the length of the substring.
* **Collision Resistance:** The use of a prime base P and a large modulus M helps reduce the likelihood of hash collisions, though collisions can still occur. In your code, even after finding matching hashes, the actual substrings are compared to confirm that they are truly equal.

**4. Application in Your Code:**

The function compute\_hashes computes the rolling hash for all substrings of a given length in the string s. The check\_equal\_substrings function uses these hash values to find common substrings of that length between two strings. Finally, the solve function performs a binary search to find the longest common substring between two strings by repeatedly narrowing down the possible substring lengths.

This combination of polynomial rolling hashes and binary search allows the algorithm to efficiently find the longest common substring between two strings.