**The Formula Recap:**

The polynomial hash formula for a substring of length L looks like this:

Hash=(s[0]×PL−1+s[1]×PL−2+⋯+s[L−1]×P0)%M\text{Hash} = (s[0] \times P^{L-1} + s[1] \times P^{L-2} + \dots + s[L-1] \times P^0) \% MHash=(s[0]×PL−1+s[1]×PL−2+⋯+s[L−1]×P0)%M

This means:

* The first character contributes the most (multiplied by P^{L-1}),
* The second character is multiplied by P^{L-2}, and so on,
* Until the last character, which is multiplied by P^0 (i.e., it's not multiplied at all).

**The Code Implementation:**

for j in range(length):

h[i] = (h[i] \* P + s[i + j]) % M

This code is a **loop** that computes the hash value iteratively. It does so by gradually incorporating each character from the substring, with each character's contribution being influenced by the position it's added at in the loop.

Let's break it down:

1. **Initial Setup:**
   * h[i] is initialized to 0 for the first substring. The loop will then compute the hash value for the substring starting at index i.
2. **Iterative Calculation:**
   * **First iteration (j = 0):**
     + The current hash h[i] is 0 (since it's the start).
     + We add the contribution of the first character s[i] (i.e., s[i + 0]), and then multiply by P.
     + The formula for the first step is: h[i]=(0×P+s[i])%M=s[i]%Mh[i] = (0 \times P + s[i]) \% M = s[i] \% Mh[i]=(0×P+s[i])%M=s[i]%M
     + At this point, the hash contains just the first character.
   * **Second iteration (j = 1):**
     + Now, we have already incorporated s[i] into the hash. The current hash h[i] represents the first character.
     + Next, we add the contribution of the second character s[i + 1]: h[i]=(s[i]×P+s[i+1])%Mh[i] = (s[i] \times P + s[i + 1]) \% Mh[i]=(s[i]×P+s[i+1])%M
     + This is equivalent to calculating s[i] \times P + s[i + 1] in the polynomial hash formula. At this point, we have the first two characters combined in the hash.
   * **Third iteration (j = 2):**
     + Now, the current hash h[i] includes the contribution of the first two characters.
     + We continue by adding the third character s[i + 2]: h[i]=((s[i]×P+s[i+1])×P+s[i+2])%Mh[i] = ((s[i] \times P + s[i + 1]) \times P + s[i + 2]) \% Mh[i]=((s[i]×P+s[i+1])×P+s[i+2])%M
     + This updates the hash to include the first three characters.
   * This process repeats for all length characters in the substring, with each character being added in and the hash being multiplied by P to shift the previous characters' contributions left.
3. **End Result:**
   * After the loop completes, h[i] will contain the final hash value for the substring starting at index i, calculated as: h[i]=(s[i]×PL−1+s[i+1]×PL−2+⋯+s[i+L−1]×P0)%Mh[i] = (s[i] \times P^{L-1} + s[i+1] \times P^{L-2} + \dots + s[i + L - 1] \times P^0) \% Mh[i]=(s[i]×PL−1+s[i+1]×PL−2+⋯+s[i+L−1]×P0)%M
   * This is exactly the same as the polynomial hash formula, where each character's contribution is weighted by the appropriate power of P.

**Why This Works:**

* **Multiplication by P:** Each time the hash is multiplied by P, it shifts the existing contribution to the left (i.e., increases the positional power of P), making space for the new character's contribution.
* **Adding s[i + j]:** Adding the new character to the hash incorporates its value at the current position.
* **Modulus Operation % M:** The modulus operation ensures that the hash value stays within a fixed range to prevent overflow.

In summary, this code is building the polynomial hash iteratively. Each iteration shifts the previous hash value to the left by multiplying by P and then adds the new character's contribution. This matches the polynomial hash formula, but instead of calculating everything at once, the code does it step by step through a loop.

4o