- Rolling Hash

高速に文字列が一致するかの検索を行うアルゴリズム

計算量は $O(n+m)$。

- Hashの求め方

長さ m の文字列 A について、互いに素な基数 b と mod の除数 h を用いて、

$$hash(A) = (A[0] \times b^{m-1} + A[1] \times b^{m-2} + \cdots + A[m-1] \cdot b) \bmod h$$

h が十分に大きいとき、ハッシュ値はユニーク（一意に定まる）

ため、文字列 S, t に対して、$hash(S) = hash(t)$

のとき、「高確率で」$S = t$ である。

- Rolling って何？

部分文字列の Hash をすべて計算しようとすると、

$O(nm)$ かかる。

↓

文字列 A と、A から一文字右にずらした文字列を A' として、

（A の長さを m としたとき、A' の長さは $m+1$）

$$Hash(A) = (A[0] \times b^{m-1} + \cdots + A[m-1] \times b) \bmod h$$

$$Hash(A') = (A[0] \times b^{m} + \cdots + A[m] \times b) \bmod h$$

より、

$$Hash(A') = (Hash(A) \times b + A[m] \times b) \bmod h$$

上記のように、連鎖的にHashの計算ができ、
前計算でO(N)でHashを計算しておけば、
部分文字列のHashについてはO(1)で求めることができる。

ABC 298 - F

入力例1

S = ABC, T = CBA (= reverse(S))より、
SとTをそれぞれ前から後ろに見ていくことで、
一致する最長部分文字列を見つける。(N = S.size())

1)
A B C
C B A
→ ×　　i = 0 ～ N

2)
B C
C B
→ ×　　i = 1 ～ N

3)
C
C
→ OK!　　i = 2 ～ N

すなわち、i = 2で折り返せばOK

AB [C] BA

例3

TREE

1) TREE
EERT → ×　　i=0～N

2) REE
EER → ×　　i=1～N

3) EE
EE → ok　　i=2～N

i=2で折り返せばok.

TR EE RT