# SUBSTRING SEARCH BY HASH

COURSE WORK #04

Author: prof. Yevhenii Borodavka

# PROBLEM STATEMENT

You have a sequence of numbers: $X_0 = 1$;

$X_n = (X_{n-1} * A + B) \bmod 2^{32}$; where $A = 1103515245$ and $B = 12345$.

In addition, you have a string **S** length of **N**, obtained from the sequence in the next way: $S_n$ = '**a**' + $(X_n >> 16) \bmod 26$, where $S_n$ — the ASCII code of the character.

The start of this string is "aquzzptirwetbkelbhbdqmuhpfybxseirk…".

You task is to find inside that string a substring with a length **K** and the given hash.

**Hash** = $S[0]*P^0 + S[1]*P^1 + S[2]*P^2 + \ldots + S[n-1]*P^{N-1} \bmod M$.

Use the values for **P**=1009 and for **M**=$10^9+7$.

# PROBLEM STATEMENT

Input. A single string with three numbers divided by spaces:

**N** ($1 <= N <= 10^6$) — number of characters in the string;

**K** ($1 <= K <= N$) — number of characters in the substring;

**H** ($0 <= H <= M$) — hash of the substring.

Output. The substring of length **K** that has the same hash as given or **0** if such substring is not found.

Example. **N**=1000, **K**=10, **H**=536637247

Input: 1000 10 536637247

Output: segxxmemfx

# RABIN-KARP ALGORITHM

Rabin-Karp algorithm is an algorithm used for searching/matching patterns in the text using a hash function. Unlike Naive string matching algorithm, it does not travel through every character in the initial phase rather it filters the characters that do not match and then performs the comparison.

A sequence of characters is taken and checked for the possibility of the presence of the required string. If the possibility is found then, character matching is performed.

Let us understand the algorithm with the following steps.

# RABIN-KARP ALGORITHM

1. Let the text be:


And the string to be searched in the above text be:

# RABIN-KARP ALGORITHM

2. Let us assign a numerical value(v)/weight for the characters we will be using in the problem. Here, we have taken first ten alphabets only (i.e. A to J).

# RABIN-KARP ALGORITHM

3. Let n be the length of the pattern and m be the length of the text. Here, m = 10 and n = 3.

Let d be the number of characters in the input set. Here, we have taken input set {A, B, C, ..., J}. So, d = 10. You can assume any suitable value for d.

4. Let us calculate the hash value of the pattern.

# RABIN-KARP ALGORITHM

hash value for pattern(p) $= \Sigma(v * d^{m-1}) \bmod 13$

$= ((3 * 10^2) + (4 * 10^1) + (4 * 10^0)) \bmod 13$

$= 344 \bmod 13$

$= 6$

In the calculation above, choose a prime number (here, 13) in such a way that we can perform all the calculations with single-precision arithmetic.

The reason for calculating the modulus is given below.

# RABIN-KARP ALGORITHM

5. Calculate the hash value for the text-window of size m.

For the first window **ABC**,

hash value for text(t) = $\Sigma(v * d^{n-1}) \bmod 13$

$= ((1 * 10^2) + (2 * 10^1) + (3 * 10^0)) \bmod 13$

$= 123 \bmod 13$

$= 6$

# RABIN-KARP ALGORITHM

6. Compare the hash value of the pattern with the hash value of the text. If they match then, character-matching is performed. In the above examples, the hash value of the first window (i.e. t) matches with p so, go for character matching between **ABC** and **CDD**. Since they do not match so, go for the next window.

# RABIN-KARP ALGORITHM

7. We calculate the hash value of the next window by subtracting the first term and adding the next term as shown below.

$t = ((2 * 10^2) + (3 * 10^1) + (3 * 10^0)) \bmod 13$

$= 233 \bmod 13$

$= 12$

In order to optimize this process, we make use of the previous hash value in the following way. ($h = d^{n-1} \bmod 13 = 10^{3-1} \bmod 13 = 9$)

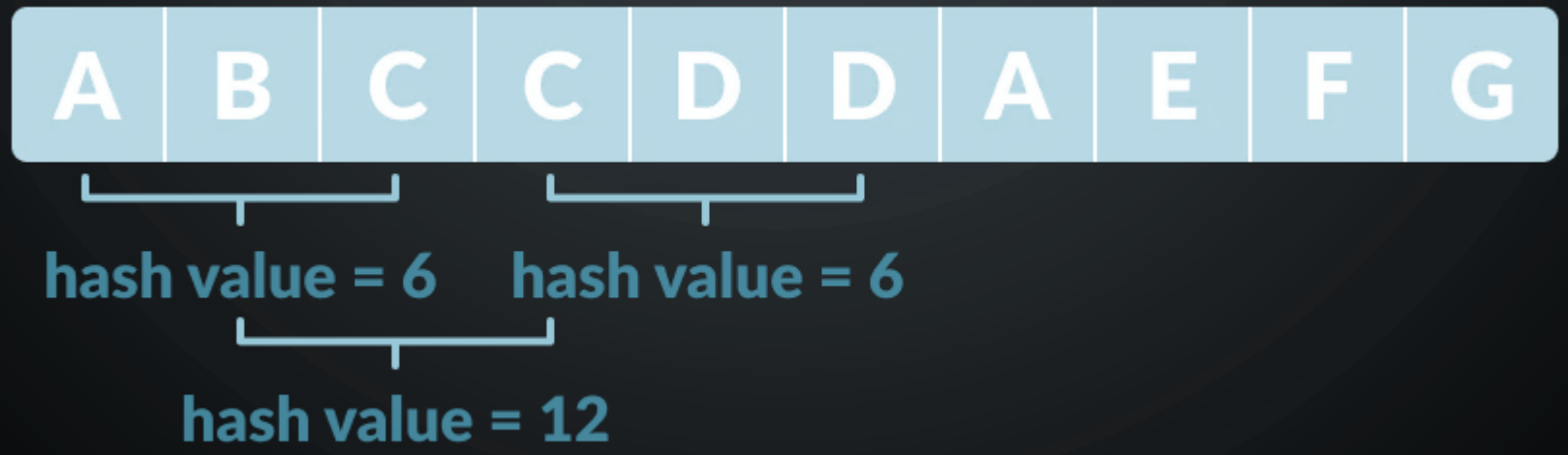
$t = ((d * (t - v[\text{removed}] * h) + v[\text{added}]) \bmod 13$

$= ((10 * (6 - 1 * 9) + 3) \bmod 13$

$= 12$

# RABIN-KARP ALGORITHM

8. For **BCC**, t = 12 (≠6). Therefore, go for the next window. After a few searches, we will get the match for the window **CDD** in the text.




The average case and best case complexity of Rabin-Karp algorithm is **O**(m + n) and the worst case complexity is **O**(m * n).

# THANK YOU!