Rabin-Karp String Matching Algorithm

Overview

In this algorithm we use the hashing technique and instead of checking for each possible position in text T (length n), we check only if the hashing of pattern P (length m) and the hashing of m characters of T at a time, give the same result. It uses a *rolling hash function* to quickly filter out positions of the text that cannot match the pattern.

Initially, it apply the hash function to the first m characters of T and check whether this result and P's hashing result is the same or not. If they are not the same, then go to the next m character of T starting at second character and again apply the hash function. If they are the same then we compare those m characters of T with P character by character.

Rolling hash function: A rolling hash (also known as recursive hashing or rolling checksum) is a hash function where the input is hashed in a window of certain length that moves through the input.

Spurious hit: If inefficient hash function is used it will return same hash code for 2 different input strings and hence pattern won't match leading to spurious hit.

Algorithm

For expository purposes, let us assume that all characters in $T \in \{0, 1, 2, \ ... \ 9\}$, so that each character is a decimal digit in given input strings. We can then view a string of k consecutive characters as representing a length-k decimal number.

```
e.g. "1234" = 1234
or "abcd" = 0123 if we take 10 alphabets [a-j] and map a -> 0, b -> 1 and so on
```

In practical applications, the number of possible characters is higher than 10 (no. of character in ASCII table is 256) and pattern length can be large. The p and t_s may be too large to work with conveniently. If P contains m characters, then we cannot reasonably assume that each arithmetic operation on p (which is m digits long) takes "constant time". We can solve this problem by choosing a suitable modulus q as a prime such that 10q just fits within one computer word.

Hash Function Selection

Given a pattern P, let p denote its corresponding decimal value (hash function output). And let t_s denote the decimal value of the length-m substring $T[s+1 \dots s+m]$.

We can compute p in time $\Theta(m)$ using Horner's rule

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \dots + 10(P[2] + 10P[1]) \dots))$$

we can compute t_0 from $T[1 \dots m]$ in time $\Theta(m)$. To compute the remaining values $t_1, t_2, \dots t_{n-m}$ in time $\Theta(n-m)$, we observe that we can compute t_{s+1} from t_s in constant time

$$t_{s+1} = 10^1(t_s - 10^{m-1}T[s+1]) + T[s+m+1] * 10^0$$

Subtracting $10^{m-1}T[s+1]$ removes the high-order digit from t_s, multiplying the result by 10 shifts the number left by one digit position, and adding $T[s+m+1]$ brings in the appropriate low-order digit.

In general, with a ***d-ary*** alphabet $\{0, 1, \dots, d-1\}$ we choose q so that dq fits within a computer word and adjust above equations to work modulo q, so that it becomes

$$t_{s+1} = (d^1(t_s - T[s+1]h) + T[s+m+1] * d^0) \mod q$$

where $h \equiv d^{m-1}(\mod q)$ is the value of the digit "1" in the high-order position of an m-digit text window. However: $t_s \equiv p(\mod q)$ does not imply that $t_s = p$. On the other hand, if $t_s \neq p(\mod q)$, then we definitely have that $ts \neq p$, so that shift s is invalid. We can thus use the test $t_s \equiv p(\mod q)$ as a fast heuristic test to rule out invalid shifts s.

```
RABIN-KARP-MATCHER($T, P, d, q$)
$n = T.length$
$m = P.length$
$h = d^{m-1} \mod q$
$p = 0$
$t_0 = 0$
for $i = 1$ to $m$                    // preprocessing
    $p = (dp + P[i]) \mod q$
    $t_0 = (dt_0 + T[i]) \mod q$
for $s = 0$ to $n - m$                // matching
    if $p == t_s$
        if $P[1..m] == T[s+1..s+m]$
            print "Pattern occurs with shift" $s$
    if $s < n - m$
        $t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \mod q$
```

Time and Space Complexity

In many applications, we expect few valid shifts—perhaps some constant c of them. In such applications, the expected matching time of the algorithm is only $O((n - m + 1) + cm) \approx O(n + m)$, plus the time required to process spurious hits.

Applications

A practical application of the algorithm is detecting plagiarism. Given source material, the algorithm can rapidly search through a paper for instances of sentences from the source material, ignoring details such as case and punctuation. Rabin-Karp is also useful for matching multiple Patterns simultaneously.

References

- Introduction to Algorithms Book
- Rabin Karp Wikipedia