String Matching

October 15, 2014

1 Introduction

In many task as

- Search in text for specific patterns.
- Pattern sequence search in DNA
- Internet Search.

2 The Problem

Assume that the text is an array $T[1..n]$ of length n, and the sought pattern is an array $P[1..m]$ with $m \leq n$. The characters in the text and the pattern are drawn from a finite alphabet Σ. The arrays are often called strings of characters.

Now, we will say that P occurs with a ***valid shift*** s if for $0 \leq s \leq n-m$ and $T[s+1..s+m] == P[1..m]$.

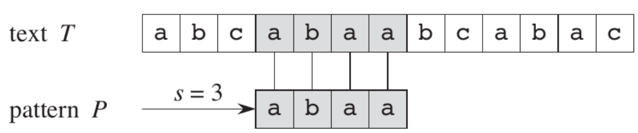

Figure 1: Valid Shift

Otherwise, it is an invalid shift. Thus, the string-matching problem is the problem of of finding all valid shifts given a patten P on a text T.

3 Notation

First, we have Σ^* as the set of all finite-length strings formed using characters from the alphabet Σ. In addition, we have the following definitions:

- The concatenation of two strings x and y is denoted by xy.

- A string w is a prefix of a string x, denoted $w \sqsubset x$, if $x = wy$ for some string $y \in \Sigma^*$.
- A string w is a suffix of a string x, denoted $w \sqsupset x$, if $x = yw$ for some string $y \in \Sigma^*$.

This notation give rise to the Overlapping-suffix lemma

Lemma. *Suppose that x, y, and z are strings such that $x \sqsupset z$ and $y \sqsupset z$.*

- *If $|x| \leq |y|$, then $x \sqsupset y$.*
- *If $|y| \leq |x|$, then $y \sqsupset x$.*
- *If $|x| = |y|$, then x=y.*

4 The naive string-matching algorithm

The naive solution uses two inner loops

Algorithm 1 Naive Algorithm

```
NAIVE-STRING-MATCHER(T, P)
n = T.length
m = P.length
for s = 0 to n − m
    if P[1..m] == T[s + 1..s + m]
        print "Pattern occurs with shift" s
```

The complexity is then $\Theta\left(\left(n-m+1\right)m\right)$which is $\Theta\left(n^2\right)$ if $m = \left\lfloor \frac{n}{2} \right\rfloor$. We need something better.

5 The Rabin-Karp algorithm

First, for explanatory purposes, assume $\Sigma = \{0, 1, 2, ..., 9\}$. Thus, each string of k consecutive characters is a k-length decimal number. Thus.

- p correspond the decimal number for pattern $P\left[1..m\right]$.
- t_s denote decimal value of m-length substring $T\left[s+1..s+m\right]$ for $s = 0, 1, ..., n-m$.

Clearly $t_s = p$ if and only if $T\left[s+1..s+m\right] = P\left[1..m\right]$, thus s is a valid shift. Now, think about this:

- If we can compute p in $\Theta\left(m\right)$.

- If we can compute all the t_s in $\Theta(n-m+1)$.

We have that $\Theta(m) + \Theta(n-m+1) = \Theta(n)$.

For p, we can use Horner's rule

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

Now, the first t_0 in time $\Theta(m)$.c:

$$t_{s+1} = 10\left(t_s - 10^{m-1}T[s+1]\right) + T[s+m+1].$$

This makes the following:

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

What happens when the numbers are quite large? We can use our friend module to handle the situation i.e.

- Compute p and t_s values modulo a suitable modulus q.

It is possible to compute $p \mod q$ in $\Theta(m)$ time and all the $t_s \mod q$ in $\Theta(n-m+1)$ time. It is more, if we select q as a prime such that $10q$ fits in a computer word. In general, for a d-ary alphabet, we choose q such that dq fits within a computer word. Thus:

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

where $h \equiv d^{m-1} (\text{mod } q)$ is the value of the digit "1" in the high-order position of an m-digit text window.

Although the solution is not perfect:

- If $t_s \equiv p \pmod q$ does not mean that $t_s = p$.
- If $t_s \neq p \pmod q$, we have that $t_s \neq p$.

To fix this problem we simply test to see if the hit is not spurious. The final algorithm can be seen in (Algorithm 2). The complexity is:

1. The algorithm takes $\Theta(m)$ preprocessing time.
2. Matching time is $\Theta((n-m+1)m)$ in the worst case.

We will not prove the complexity, but the expected matching time by Rabin-Karp algorithm is

$$O(n) + O\left(m\left(v + \frac{n}{q}\right)\right)$$

where v is the number of valid shifts, and since there are $O(n)$ positions at which the test of line 10 fails and we spend $O(m)$ time for each hit.

If $v = O(1)$ (Number of valid shifts small) and choose $q \geq m$ such that $\frac{n}{q} = O(1)$ (q to larger enough than the pattern's length), then the algorithm takes $O(m+n)$ for finding the matches. Finally, because $m \leq n$, thus the expected time is $O(n)$.

Algorithm 2 Rabin-Karp Algorithm

```
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
```