String Matching Algorithms

Topics

- Basics of Strings
- Brute-force String Matcher
- Rabin-Karp String Matching Algorithm
- KMP Algorithm
In string matching problems, it is required to find the occurrences of a pattern in a text.

These problems find applications in text processing, text-editing, computer security, and DNA sequence analysis.

*Find and Change* in word processing

Sequence of the human cyclophilin 40 gene

CCCAGTCTGG AATACAGTGG CGCGATCTCG GTTCACTGCA
ACCGCCGCCT CCCGGGTTCA AACGATTCTC CTGCCTCAGC
CGCGATCTCG : DNA binding protein GATA-1
CCCGGG : DNA binding protein Sma 1

C: Cytosine, G: Guanine, A: Adenosine, T: Thymine
Text: $T[1..n]$ of length $n$ and Pattern $P[1..m]$ of length $m$.
The elements of $P$ and $T$ are characters drawn from a finite alphabet set $\Sigma$.
For example $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b, \ldots, z\}$, or $\Sigma = \{c, g, a, t\}$.
The character arrays of $P$ and $T$ are also referred to as strings of characters.
Pattern $P$ is said to occur with shift $s$ in text $T$
if $0 \leq s \leq n-m$ and
$T[s+1..s+m] = P[1..m]$ or
$T[s+j] = P[j]$ for $1 \leq j \leq m$,
such a shift is called a valid shift.
The string-matching problem is the problem of finding all valid shifts with which a given pattern $P$ occurs in a given text $T$. 
Brute force string-matching algorithm

To find all valid shifts or possible values of $s$ so that $P[1..m] = T[s+1..s+m]$;
There are $n-m+1$ possible values of $s$.

Procedure $BF\_String\_Matcher(T,P)$

1. $n \leftarrow \text{length}[T]$;
2. $m \leftarrow \text{length}[P]$;
3. for $s \leftarrow 0$ to $n-m$
4. do if $P[1..m] = T[s+1..s+m]$
5. then shift $s$ is valid

This algorithm takes $\Theta((n-m+1)m)$ in the worst case.
a c a a b c a c a a b c

a a b

a c a a b c

a a b

a c a a b c matches

a a b
Rabin-Karp Algorithm

Let $\Sigma = \{0,1,2, \ldots,9\}$. We can view a string of $k$ consecutive characters as representing a length-$k$ decimal number. Let $p$ denote the decimal number for $P[1..m]$. Let $t_s$ denote the decimal value of the length-$m$ substring $T[s+1..s+m]$ of $T[1..n]$ for $s = 0, 1, \ldots, n-m$.

$t_s = p$ if and only if $T[s+1..s+m] = P[1..m]$, and $s$ is a valid shift.

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

We can compute $p$ in $O(m)$ time.

Similarly we can compute $t_0$ from $T[1..m]$ in $O(m)$ time.
\[ 6378 = 8 + 7 \times 10 + 3 \times 10^2 + 6 \times 10^3 \]
\[ = 8 + 10 (7 + 10 (3 + 10(6))) \]
\[ = 8 + 70 + 300 + 6000 \]

\[ m = 4 \]

\[ p = P[m] + 10(P[m-1] + 10(P[m-2] + \ldots + 10(P[2] + 10(P[1]))) \]
$t_{s+1}$ can be computed from $t_s$ in constant time.

$$t_{s+1} = 10(t_s - 10^{m-1} \pi[s+1]) + \pi[s+m+1]$$

Example: $T = 314152$
$t_s = 31415$, $s = 0$, $m = 5$ and $\pi[s+m+1] = 2$

$$t_{s+1} = 10(31415 - 10000*3) + 2 = 14152$$

Thus $p$ and $t_0, t_1, \ldots, t_{n-m}$ can all be computed in $O(n+m)$ time.
And all occurrences of the pattern $P[1..m]$ in the text $T[1..n]$ can be found in time $O(n+m)$.

However, $p$ and $t_s$ may be too large to work with conveniently.
Do we have a simple solution!!
Computation of $p$ and $t_0$ and the recurrence is done using modulus $q$.

In general, with a $d$-ary alphabet \{0,1,...,d-1\}, $q$ is chosen such that $d \times q$ fits within a computer word.

The recurrence equation can be rewritten as
$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \mod q,$$
where $h = d^{m-1}(\mod q)$ is the value of the digit “1” in the high order position of an $m$-digit text window.

Note that $t_s \equiv p \mod q$ does not imply that $t_s = p$.
However, if $t_s$ is not equivalent to $p \mod q$, then $t_s \neq p$, and the shift $s$ is invalid.

We use $t_s \equiv p \mod q$ as a fast heuristic test to rule out the invalid shifts.

Further testing is done to eliminate spurious hits.
- an explicit test to check whether
  $$P[1..m] = T[s+1..s+m]$$
\[ t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \mod q \]

\[ h = d^{m-1}(\mod q) \]

Example:

\[ T = 31415; \quad P = 26, \quad n = 5, \quad m = 2, \quad q = 11 \]

\[ p = 26 \mod 11 = 4 \]

\[ t_0 = 31 \mod 11 = 9 \]

\[ t_1 = (10(9 - 3(10) \mod 11) + 4) \mod 11 \]

\[ = (10(9 - 8) + 4) \mod 11 = 14 \mod 11 = 3 \]
Procedure RABIN-KARP-MATCHER(T,P,d,q)

Input : Text $T$, pattern $P$, radix $d$ (which is typically $\left| \Sigma \right|$), and the prime $q$.

Output : valid shifts $s$ where $P$ matches

1. $n \leftarrow \text{length}[T]$;
2. $m \leftarrow \text{length}[P]$;
3. $h \leftarrow d^{m-1} \mod q$;
4. $p \leftarrow 0$;
5. $t_0 \leftarrow 0$;
6. for $i \leftarrow 1$ to $m$
7. \hspace{1em} do $p \leftarrow (d \times p + P[i]) \mod q$;
8. \hspace{1em} $t_0 \leftarrow (d \times t_0 + T[i]) \mod q$;
9. for $s \leftarrow 0$ to $n-m$
10. \hspace{1em} do if $p = t_s$
11. \hspace{2em} then if $P[1..m] = T[s+1..s+m]$
12. \hspace{3em} then “pattern occurs with shift ‘s’”
13. \hspace{1em} if $s < n-m$
14. \hspace{2em} then $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]) \mod q$;
Comments on Rabin-Karp Algorithm

- All characters are interpreted as radix-d digits
- \( h \) is initiated to the value of high order digit position of an \( m \)-digit window
- \( p \) and \( t_0 \) are computed in \( O(m+m) \) time
- The loop of line 9 takes \( \Theta((n-m+1)m) \) time

The loop 6-8 takes \( O(m) \) time
The overall running time is \( O((n-m)m) \)
Exercises

-- Home work
- Study KMP Algorithm for String Matching
  -- Knuth Morris Pratt (KMP)
- Study Boyer-Moore Algorithm for String matching

Extend Rabin-Karp method to the problem of searching a text string for an occurrence of any one of a given set of $k$ patterns? Start by assuming that all $k$ patterns have the same length. Then generalize your solution to allow the patterns to have different lengths.

Let $P$ be set of $n$ points in the plane. We define the depth of a point in $P$ as the number of convex hulls that need to be peeled (removed) for $p$ to become a vertex of the convex hull. Design an $O(n^2)$ algorithm to find the depths of all points in $P$.

The input is two strings of characters $A = a_1, a_2, ..., a_n$ and $B = b_1, b_2, ..., b_n$. Design an $O(n)$ time algorithm to determine whether $B$ is a cyclic shift of $A$. In other words, the algorithm should determine whether there exists an index $k$, $1 \leq k \leq n$ such that $a_i = b(k+i) \mod n$, for all $i$, $1 \leq i \leq n$. 