CSc 3102: String Matching

The Knuth-Morris-Pratt Algorithm

- The String Matching Problem
- The Naive String Matching Algorithm
- The KMP String Matching Algorithm

The String Matching Problem

Definition 1. Consider text in an array $T[0 \ldots n-1]$ of length n and a pattern in array $P[0 \ldots m = 1]$ of length m. The elements of P and T are drawn from a finite alphabet Σ. P **occurs with shift s** or P **occurs beginning at position s** in text T, if $0 \leq s \leq n \leq n - m$ and $T[s \ldots s + m - 1] = P[0 \ldots m - 1]$. If P occurs with shift s in T, then we call s a **valid shift**; otherwise, we call s an **invalid shift**. The string matching problem is the problem of finding all valid shifts.

The Naive String Matching Algorithm

```
ALGORITHM: naiveStringMatcher(T[0:n-1],P[0:m-1],S[0:n-m])
  {T a string of n characters representing the host text,
   P a string of m characters representing the pattern text,
   S array with valid and invalid shifts.}
   j <- 0
   for i <- 0 to n-m
      if P[0...m-1] = T[s...s+m-1]
         S[j] <- i
         j <- j + 1
  {Note: All elements of S are initially -1}
```

It can easily be shown that the worst-case time complexity of the algorithm is $O(m(n - m + 1)) = O(mn)$.

The KMP String Matching Algorithm

KMP is a string matching algorithm due to Knuth-Morris-Pratt is based on the Naive String Matching algorithm. However, unlike the Naive String Matching algorithm, it does not necessarily shift one character at a time. It avoids shifting into a position where a match could not have possibly occur based on comparison made of the pattern with itself. By comparing the pattern with itself we compute the prefix function, Π.

Observation 1. Given that pattern characters $P[0...q-1]$ match host characters $T[s...s+q-1]$,what is the least shift s' such that $P[0...k-1] = T[s'...s'+k-1]$, where $s'+k = s+q$? Such a shift s' is the first shift greater than s that is not necessarily invalid due to our knowledge of $T[s...s+q-1]$.

Definition 2. The **prefix function**, $\Pi(q) = \max\{k : k < q \ \textit{and} \ P_k \supset P_q\}$ In words,, $\Pi[q]$ is the length of the longest prefix of P that is a proper suffix of P_q. The prefix function is computed as below:

ALGORITHM: Prefix(P)
$m \leftarrow length[P]$
$\Pi[0] \leftarrow 0$
$k \leftarrow 0$
for $q \leftarrow 1$ **to** $m-1$
  while $k > 0$ *and* $P[k] \neq P[q]$
    $k \leftarrow \Pi[k-1]$
  if $P[k] = P[q]$
    $k \leftarrow k+1$
  $\Pi[q] \leftarrow k$
return Π

Once the prefix function has been determined, we may then find the valid

shifts using the KMP String Matching algorithm as follows:

ALGORITHM: KMPStringMatcher(T,P)

$n \leftarrow length[T]$
$m \leftarrow length[P]$
$\Pi \leftarrow Prefix(P)$
$q \leftarrow 0$
for $i \leftarrow 1$ *to* n
  while $q > 0$ *and* $P[q] \neq T[i-1]$
    $q \leftarrow \pi[q-1]$
  if $P[q] = T[i-1]$
    $q \leftarrow q+1$
  if $q = m$
    print pattern occurs with shift $i-m$
    $q \leftarrow \Pi[q-1]$

Here is a sample prefix function computation:

i	0	1	2	3	4	5	6	7	8	9
P(i)	a	b	a	b	a	b	a	b	c	a
$\Pi(i)$	0	0	1	2	3	4	5	6	0	1

Table 1: $P = abababababca$

Using amortized analysis, which is beyond the scope of this course, it takes O(m) to compute the prefix function. The KMP Matching takes O(n) using the computed prefix function. Thus, the KMP String Matcher runs in time O($m+n$).

Remarks: Observe the following about the *KMPStringMatcher*:

1. Backtracking in the host string is avoided during the scan.
2. Only the mismatched characters during the scan are compared more than once during the scan.
3. Unlike the brute-force algorithm, attempts to match the pattern to a substring in the host string does not always start at the beginning of the pattern when the pattern is shifted.
4. The KMP algorithm uses the π array to avoid attempting to match a prefix when it is unnecessary.
5. Computing the prefix function of the pattern is $O(m)$, where m is the length of the pattern. Determining all the valid shifts in the host is $O(n)$, where n is the length of the host. Both algorithms are linear time. This can be proven using amortized analysis which is beyond the scope of this class. The KMP algorithm is $O(m + n) = O(n)$ since $n \geq m$.

In class we will provide the intuition behind both the *prefix* and the *KMP-StringMatcher* and an informal proof of correctness of the algorithms.

Problem 1. Given $\Sigma = \{0, 1\}$, $P = 1011011$ and $H = 10110101011011$,

a. Compute the prefix function for P. Give all the $(P[i], P[j])$ pairs that are compared in the order in which they are compared during the prefix function computation, where i and j are zero-based indices in P. How many pairs are there?

b. During the trace of each algorithm, give all the $(P[i], H[j])$ pairs that are compared in the order in which they are compared up to and including the first match, where i and j are zero-based indices in P and H. List all pairs prior to the first shift of the pattern and after every shift, with appropriate labels before each set of pairs.

   (a) Trace the action of the KMP string matcher. How many pairs are compared? How many times is the pattern shifted? What is the total number of pairs compared for the prefix function computation and the KMP algorithm?

   (b) Trace the action of the brute-force string matcher. How many pairs are compared? How many times is the pattern shifted?

   (c) Which algorithm compares fewer pairs?

Problem 2. Repeat Problem 1 but given $P = 000001$ and $H = 00000000001$.