Design and Analysis of Algorithms

Lecture-42

Dharmendra Kumar (Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research,
Prayagraj

String Matching

String Matching Problem

- We assume that the text is an array $T[1..n]$ of length n and that the pattern is an array $P[1..m]$ of length $m \leq n$.
- We further assume that the elements of P and T are characters drawn from a finite alphabet Σ.
- Pattern P ***occurs with shift*** s in text **T** if $0 \leq s \leq n-m$ and $T[s+1 .. s+m] = P[1..m]$.
- 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 with which a given pattern P occurs in a given text T .

String Matching Problem

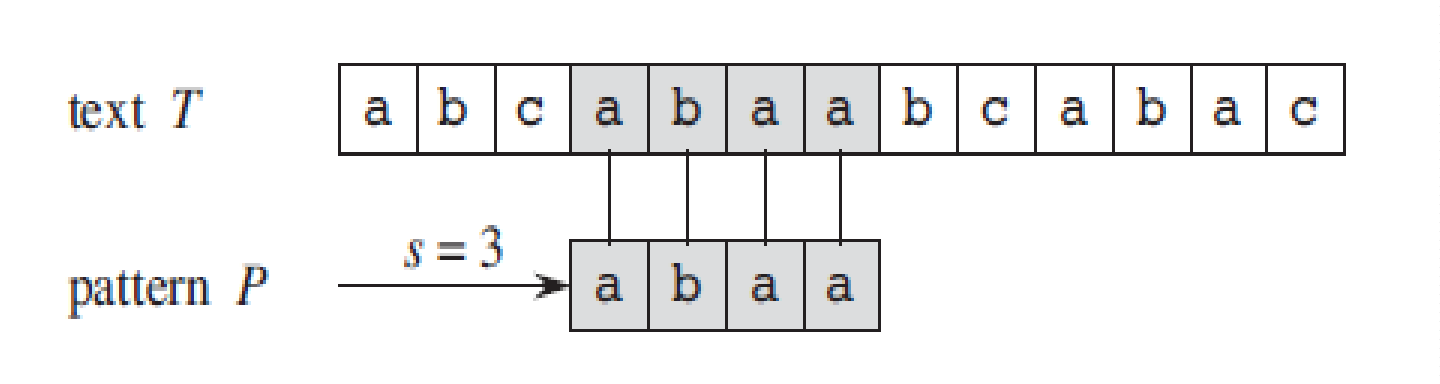
Example: Consider the text T and pattern P as following:-

T = abcabaabcabac

P= abaa

Find all valid shifts.

Solution:



Valid shift s = 3

There will be only one valid shift in this example.

Prefix and Suffix of a string

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

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

- **Example:** Clearly, $\texttt{ab} \sqsubset \texttt{abcca}$ and $\texttt{cca} \sqsupset \texttt{abcca}$.

- The empty string ε is both a suffix and a prefix of every string.

The naive string-matching algorithm

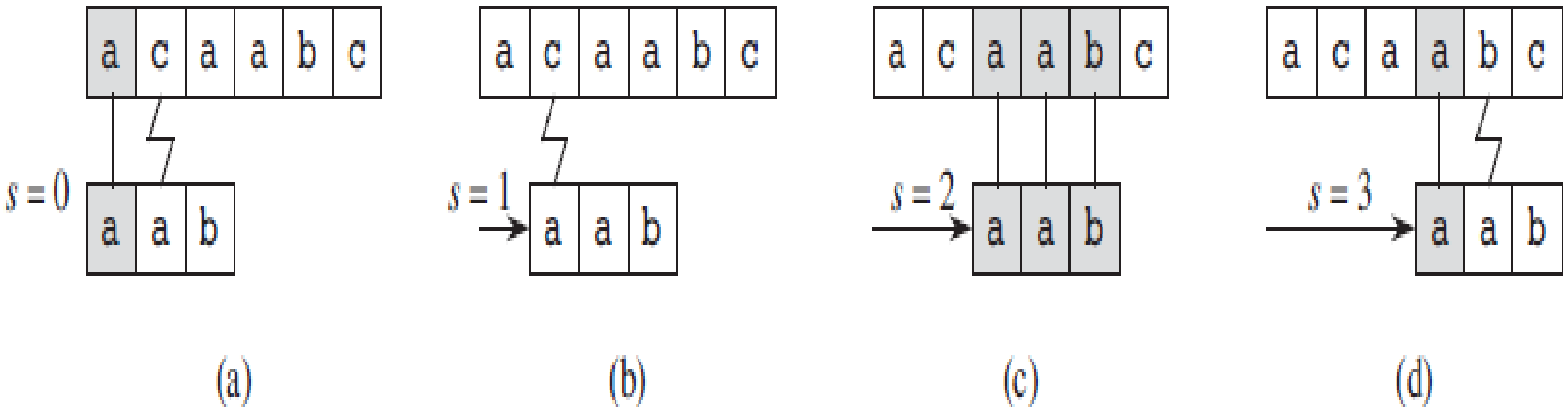
```
NAIVE-STRING-MATCHER(T, P)
n = T.length
m = P.length
for s = 0 to n − m
    if P[1..m] == T[s + 1..s + m]
        print "Pattern occurs with shift" s
```

- The worst-case running time is $\theta((n-m)m)$, which is $\theta(n^2)$ if $m = \lfloor n/2 \rfloor$.

The naive string-matching algorithm

Example: The operation of this algorithm is shown in the following:-

Here T = acaabc   and P = aab



The Rabin-Karp algorithm

- Rabin and Karp proposed a string-matching algorithm that performs well.
- Given a pattern P{1..m], let p denote its corresponding decimal value. In a similar manner, given a text T[1..n], let t_s denote the decimal value of the length-m substring T [s+1 .. s+m] , for s= 0, 1,……., n-m.
- t_s = p iff T [s+1 .. s+m] = P[1..m]
- Therefore, s is a valid shift if and only if t_s = p.

The Rabin-Karp algorithm

Computation of p and t_s using Horner's rule:

- $p = P[m]+10(P[m-1]+10(P[m-2]+10(P[m-3] + \ldots + 10(P[2]+10P[1]))))$
- The value of t_0 can be computed similarly from T[1..m].
- To compute the remaining values $t_1, t_2, t_3, \ldots, t_{n-m}$, t_{s+1} can be computed from t_s in the following way:-

$$t_{s+1} = 10\,(t_s - 10^{m-1}T[s+1]) + T[s+m+1] \quad \ldots\ldots\ldots\ldots (1)$$

- The only difficulty with this procedure is that p and t_s may be too large.

The Rabin-Karp algorithm

- To solve this problem, with d-ary alphabet {0,1,2,...., d-1}, we choose q so that dq fits with in a computer word and adjust the recurrence equation (1) to work modulo q, so that it becomes

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

  where $h = d^{m-1} \bmod q$

- The solution of working modulo q is not perfect, because:

$t_s \equiv p \bmod q$ does not imply that $t_s = p$. On the other hand, if $ts \not\equiv p \bmod q$, then we definitely have that $t_s \neq p$, so that shift s is invalid.

- Any shift s for which $t_s \equiv p \bmod q$ must be tested further to see whether s is really valid or we just have a ***spurious hit***. This additional test explicitly checks the condition

  $P[1..m] = T[s+1........s+m]$

The Rabin-Karp algorithm

Example: Consider T and P as following:-

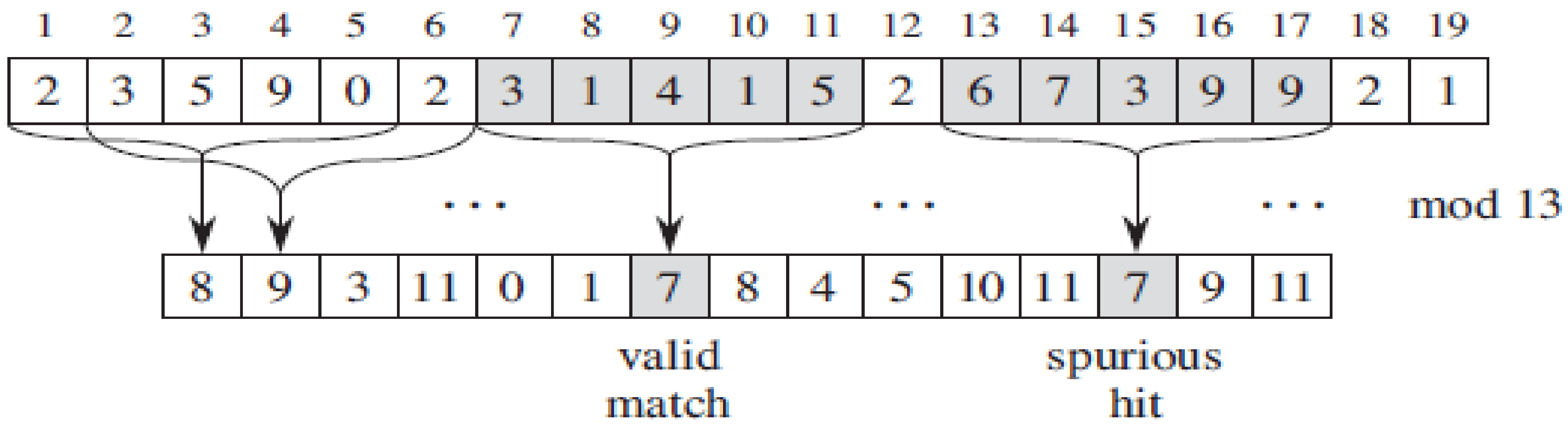
T= 2359023141526739921

P= 31415

q = 13

Find all valid shifts and spurious hit.

Solution:



The 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 = (d t_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
```

The Rabin-Karp algorithm

Time complexity

- RABIN-KARP-MATCHER takes ,θ(m) preprocessing time, and its matching time is , θ((n-m+1)m) in the worst case.

Question: For q=11, how many spurious hits does the Robin-Karp matcher encounter in the text T = 3141592653589793 when looking for the pattern P= 26?

Solution:

Knuth-Morris-Pratt(KMP) algorithm

Prefix function for a pattern

Given a pattern P[1..m], the ***prefix function*** for the pattern P is the function $\pi : \{1,2,3,\ldots\ldots,m\} \rightarrow \{0,1,2,\ldots\ldots,m-1\}$ such that

$$\pi(q) = \max\{\, k \,!\, K < q \text{ and } P_k \sqsupset P_q \}$$

$\pi(q)$ is the length of the longest prefix of P that is a proper suffix of P_q.

Knuth-Morris-Pratt(KMP) algorithm

Example: Compute the prefix function of the pattern

P = ababababca

Solution:

	1	2	3	4	5	6	7	8	9	10
i	a	b	a	b	a	b	a	b	c	a
P(i) π(i)	0	0	1	2	3	4	5	6	0	1

Knuth-Morris-Pratt(KMP) algorithm

```
KMP-MATCHER(T, P)
n = T.length
m = P.length
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0                                   // number of characters matched
for i = 1 to n                          // scan the text from left to right
    while q > 0 and P[q + 1] ≠ T[i]
        q = π[q]                        // next character does not match
    if P[q + 1] == T[i]
        q = q + 1                       // next character matches
    if q == m                           // is all of P matched?
        print "Pattern occurs with shift" i − m
        q = π[q]                        // look for the next match
```

Knuth-Morris-Pratt(KMP) algorithm

```
COMPUTE-PREFIX-FUNCTION(P)
m = P.length
let π[1..m] be a new array
π[1] = 0
k = 0
for q = 2 to m
    while k > 0 and P[k + 1] ≠ P[q]
        k = π[k]
    if P[k + 1] == P[q]
        k = k + 1
    π[q] = k
return π
```

Knuth-Morris-Pratt(KMP) algorithm

Time complexity

Running time of compute-prefix-function is θ(m).

The matching time of KMP-Matcher is θ(n).

Question: Consider text and pattern as following:-

T = bacbababaabcbab

P = aba

Find all valid shifts using KMP algo.

Question: Compute the prefix function for the pattern ababbabbabbababbabb.

AKTU Examination Questions