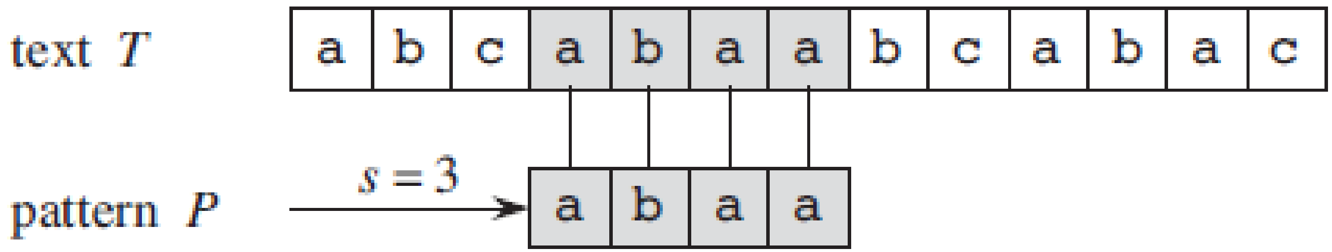
String Matching

- We formalize the string-matching problem as follows. 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. For example, we may have $\Sigma = \{0,1\}$ or $\Sigma = \{a, b, \ldots, z\}$. The character arrays P and T are often called ***strings*** of characters.



- we say that pattern P ***occurs with shift*** s in text T (or, equivalently, that pattern P ***occurs beginning at position*** s + 1 in text T) if 0 ≤ s ≤ n-m and T[s+1… s +m] = P[1…m] (that is, if T[s+j] = P[j], for 1 ≤ j ≤ 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 .

Notation and terminology

- We denote by $\sum *$ (read “sigma-star”) the set of all finite-length strings formed using characters from the alphabet $\sum$. The zero-length ***empty string***, denoted ε, also belongs to $\sum *$. The ***concatenation*** of two strings x and y, denoted xy.
- We say that a string w is a ***prefix*** of a string x, denoted $w \sqsubset x$, if x = wy for some string $y \in \sum *$. Note that if $w \sqsubset x$, then $|w| \leq |x|$. Similarly, we say that a string w is a ***suffix*** of a string x, denoted $w \sqsupset x$, if x = yw for some $y \in \sum *$.

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
```

Procedure NAIVE-STRING-MATCHER takes time O((n - m + 1)m)

The Rabin-Karp algorithm

- This algorithm makes use of elementary number-theoretic notions such as the equivalence of two numbers modulo a third number
- For expository purposes, let us assume that $\sum$ = {0, 1, 2,…, 9}, so that each character is a decimal digit. (In the general case, we can assume that each character is a digit in radix-d notation, where d =| $\sum$ |).
- 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 denotes the decimal value of the length-m substring T[s+1…s+m], for s = 0,1,…,n -m. Certainly, t_s = **p** if and only if T[s+1…s+m] = P[1…m]; thus, s is a valid shift if and only if t_s = **p**. If we could compute **p** in time θ(m) and all the t_s values in a total of θ(n-m+1) time, then we could determine all valid shifts s in time θ(m) + θ(n-m+1) = θ(n) by comparing **p** with each of the t_s values.
- T=56489005050
- P=5648

- We can compute p in time $\theta(m)$ using Horner's rule
- p = P[m] + 10(P[m-1] + 10(P[m-2]+ …+10(P[2] +10(P[1])…))
- Similarly, we can compute t_0 from T[1…m] in time $\theta(m)$.
- To compute the remaining values t_1, t_2, …, t_{n-m} in time $\theta(n-m)$, we observe that we can compute t_{s+1} from t_s in constant time, since
- $t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+ m +1]$
- 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.

- For example, if m = 5 and t_s = 31415, then we wish to remove the high-order digit T[s+1] = 3 and bring in the new low-order digit (suppose it is T[s + 5 + 1] = 2) to obtain
- t_{s+1} = 10(31415 – 10000.3) + 2
- = 14152
- 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.

- $t_{s+1} = (d(t_s - T[s + 1]h) + T[s + m + 1]) \bmod q$
- where $h = d^{m-1} \pmod q$
- In general, q is a prime number such that dq fits within a computer word where $d = |\sum|$.

RABIN-KARP-MATCHER(T, P, d, q)

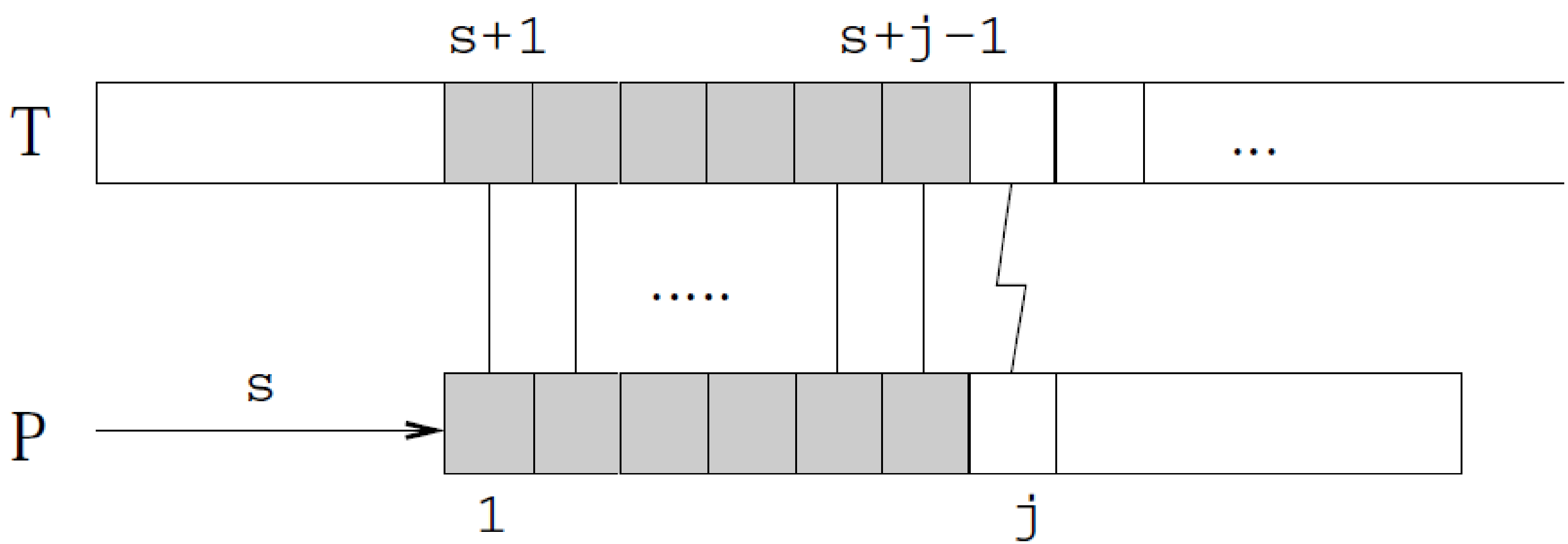
```
n = T.length
m = P.length
h = d^(m-1) mod q
p = 0
t_0 = 0
for i = 1 to m                           // preprocessing
    p = (dp + P[i]) mod q
    t_0 = (dt_0 + T[i]) mod q
for s = 0 to n - m                       // matching
    if p == t_s
        if P[1..m] == T[s + 1..s + m]
            print "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
```

Run-Time-Analysis

- RABIN-KARP-MATCHER takes $\theta(m)$ preprocessing time, and its matching time is $\theta(n-m+1)m)$ in the worst case, since (like the naive string-matching algorithm) the Rabin-Karp algorithm explicitly verifies every valid shift.
- 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
- $O((n-m+1)+ cm) = O(n + m)=O(n)$

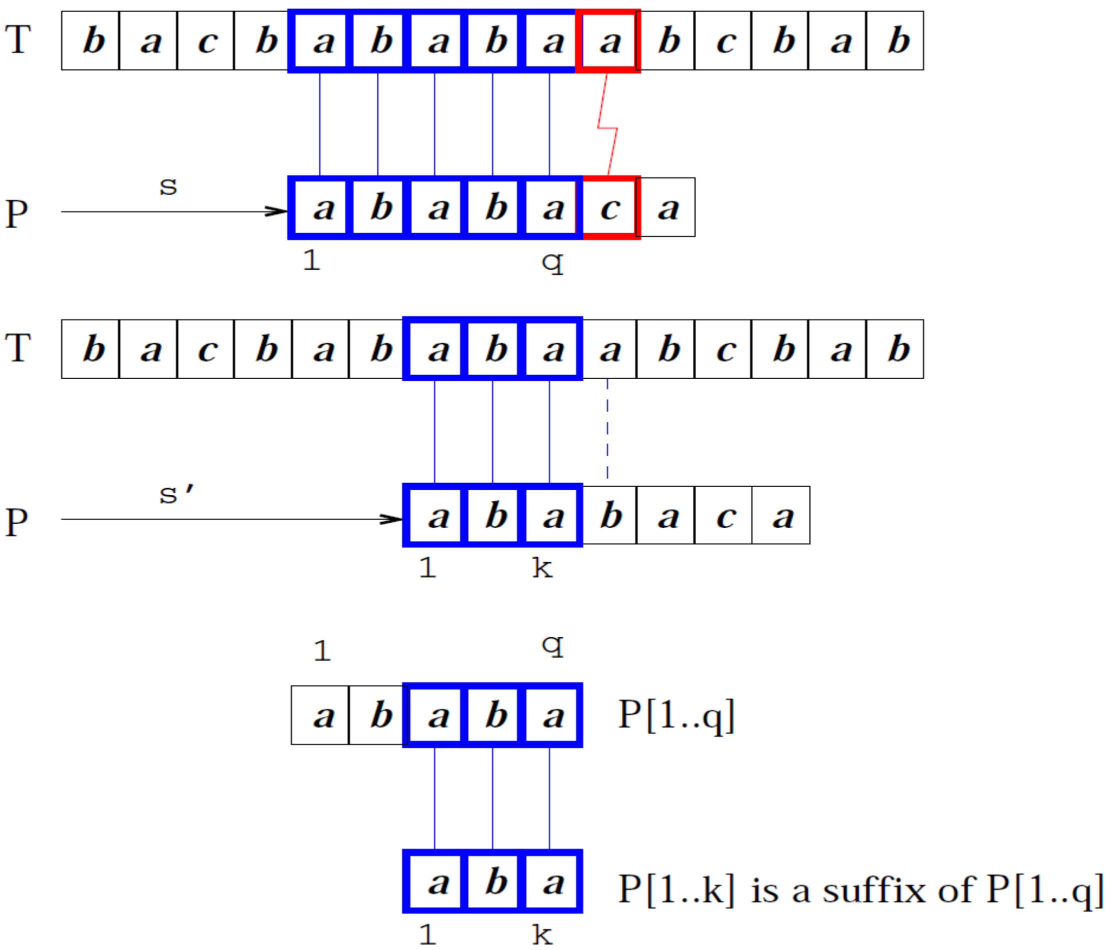
The Knuth-Morris-Pratt (KMP) Algorithm

- In the Brute-Force algorithm, if a mismatch occurs at P[j](j>1), it only slides P to right by 1 step. It throws away one piece of information that we've already known. What is that piece of information?
- Let s be the current shift value. Since it is a mismatch at P[j], we know T[s+1…s+j-1]=p[1…j-1].

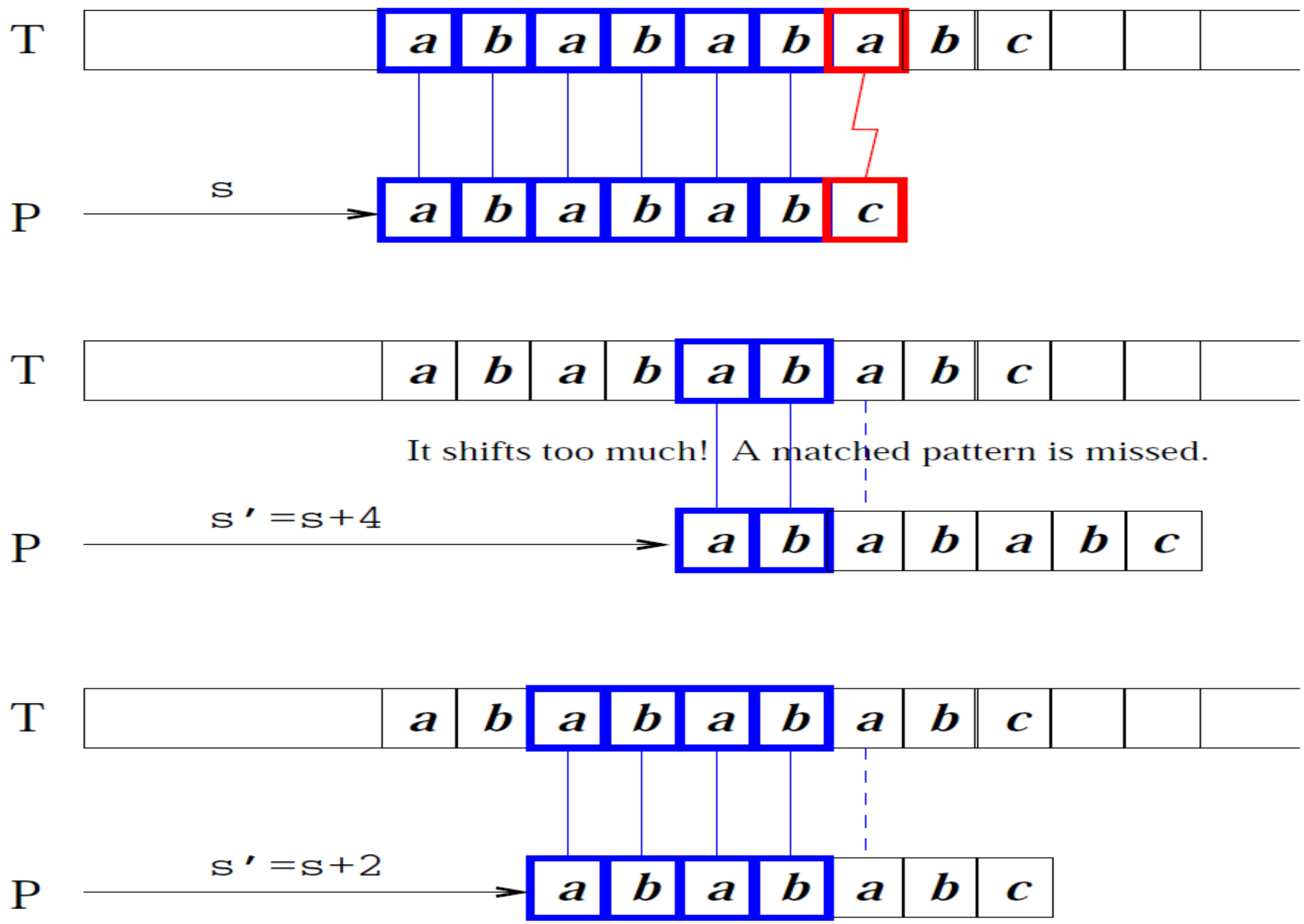


How can we make use of this information to make the next shift? In general, P should slide by $s' > s$ such that $P[1..k]$ = $T[s'+1..s'+k]$. We then compare $P[k+1]$ with $T[s'+k+1]$.

When we slide P to right, it should be a place where P could possibly occur in T .



Do not shift too much, as it may miss some matched patterns!



We need to answer the following question: Given $P[1..q]$ match text characters $T[s+1..s+q]$, what is the *least* shift $s^{'} > s$ such that

$P[1..k]$ = $T[s^{'}+1..s^{'}+k]$,

where $s^{'}+k = s+q$?

In practice, the shift $s^{'}$ can be precomputed by comparing P against itself. Observe that $T[s^{'}+1..s^{'}+k]$ is a known text, and it is a **suffix** of $P[1..q]$. To find the *least shift* $s^{'} > s$, it is the same as finding the *largest* $k < q$, s.t.,

$P[1..k]$ is a suffix of $P[1..q]$.

The *next* function

Given $P[1..m]$, let *next* be a function $\{1, 2, \ldots, m\} \rightarrow \{0, 1, \ldots, m-1\}$ such that

next(q)$= max\{k : k < q$ and $P[1..k]$ is a suffix of $P[1..q]\}$.

q	1	2	3	4	5	6	7	8	9	10
P[q]	***a***	***b***	***a***	***b***	***a***	***b***	***a***	***b***	***c***	***a***
next(q)	0	0	1	2	3	4	5	6	0	1

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
```