String Matching Algorithms

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

String Matching Problem

Let Σ denotes the set of alphabets.

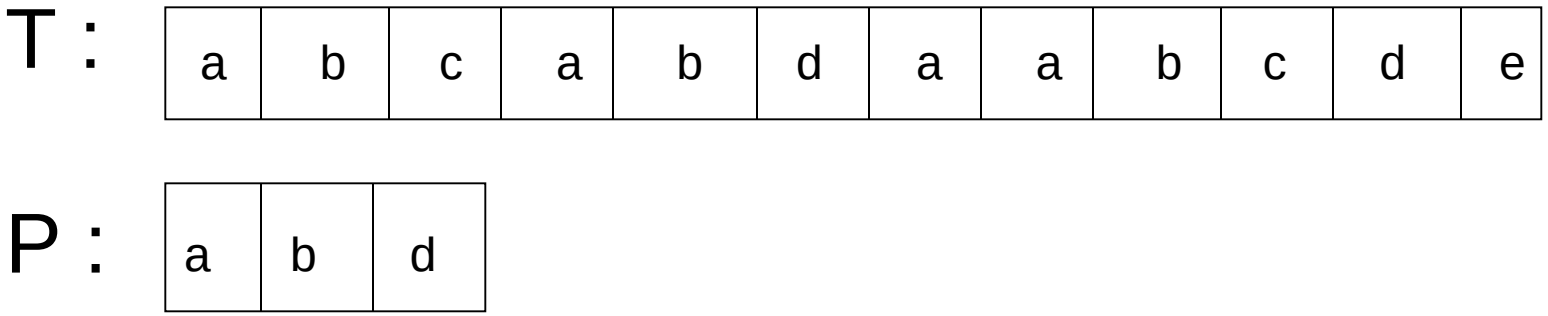
- **Given**:
  - A string of alphabets $T[1..n]$ of size n and
  - a pattern $P[1..m]$ of size m
    where $m << n$
- **To Find**:
  - Whether the pattern P occurs in text T or not
  - If it does, then give all occurrences of P in T

Pattern P occurs with shift s in T if $P[1...m] = T[s+1...s+m]$

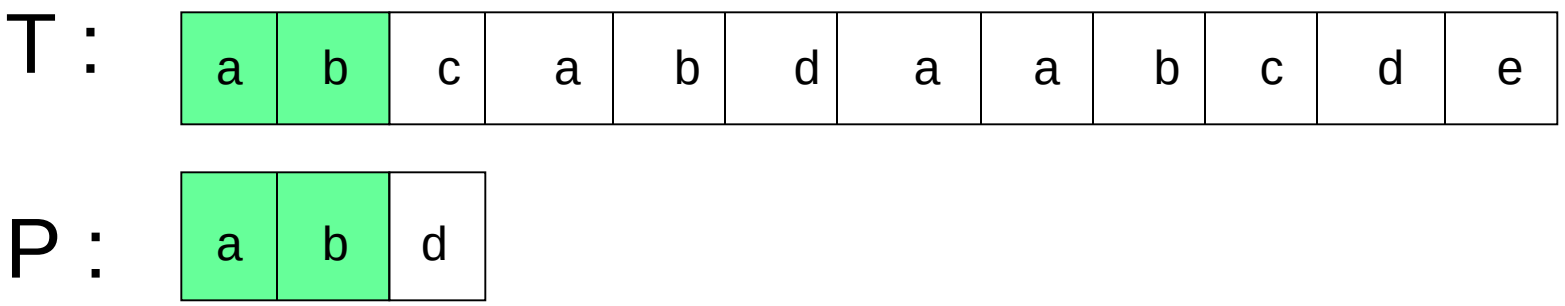
$$0 \leq s \leq n - m$$

Motivations: text-editing, pattern matching in DNA sequences

Naive String Matching Algorithm

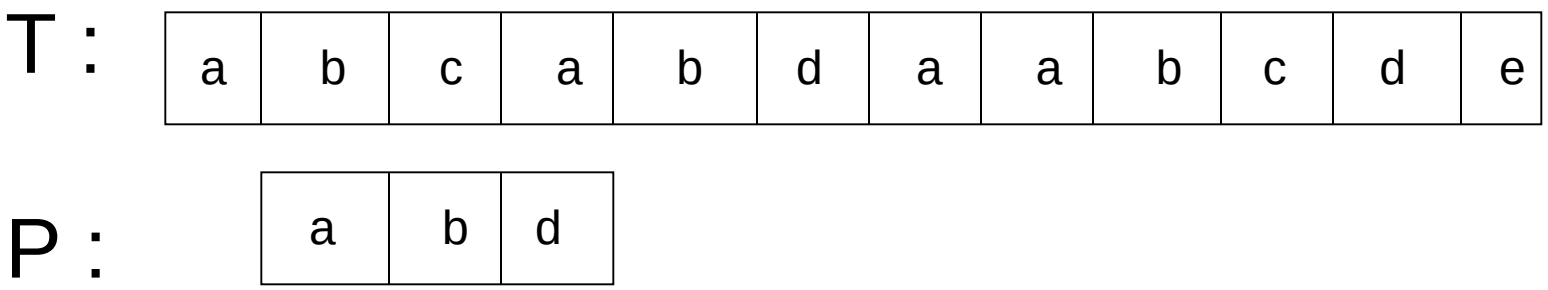


Naive String Matching Algorithm



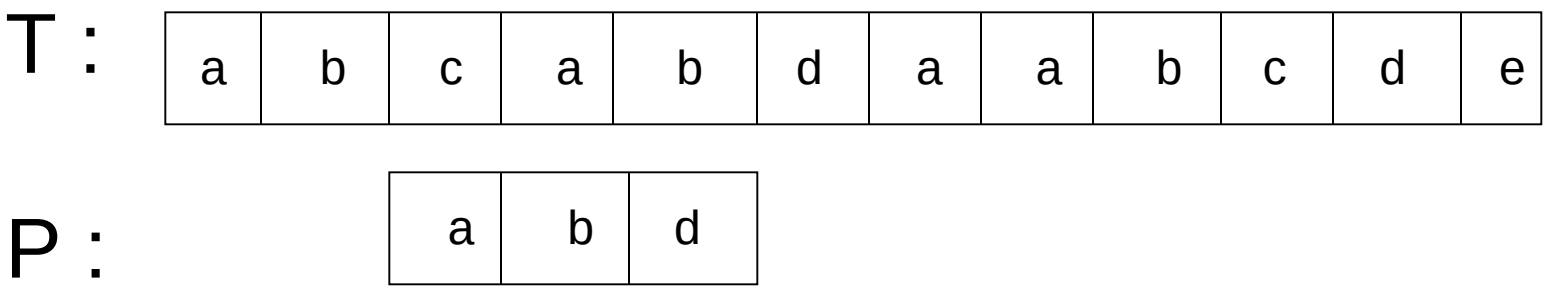
Mismatch after 3 Comparisons

Naive String Matching Algorithm



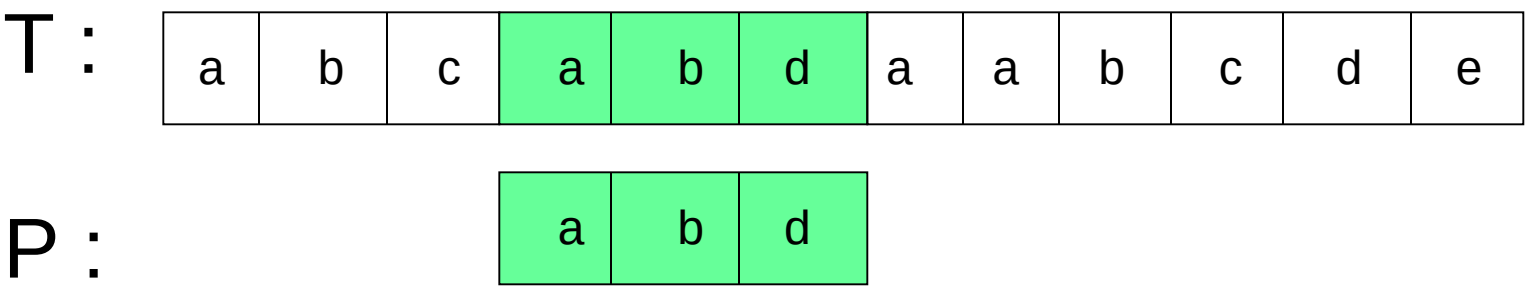
Mismatch after 1 Comparison

Naive String Matching Algorithm



Mismatch after 1 Comparison

Naive String Matching Algorithm



Match found after 3 Comparisons

Thus, after 8 comparisons the substring P is found in T.

Naive String Matching Algorithm

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

worst-case running time is $\Theta((n-m+1)m)$

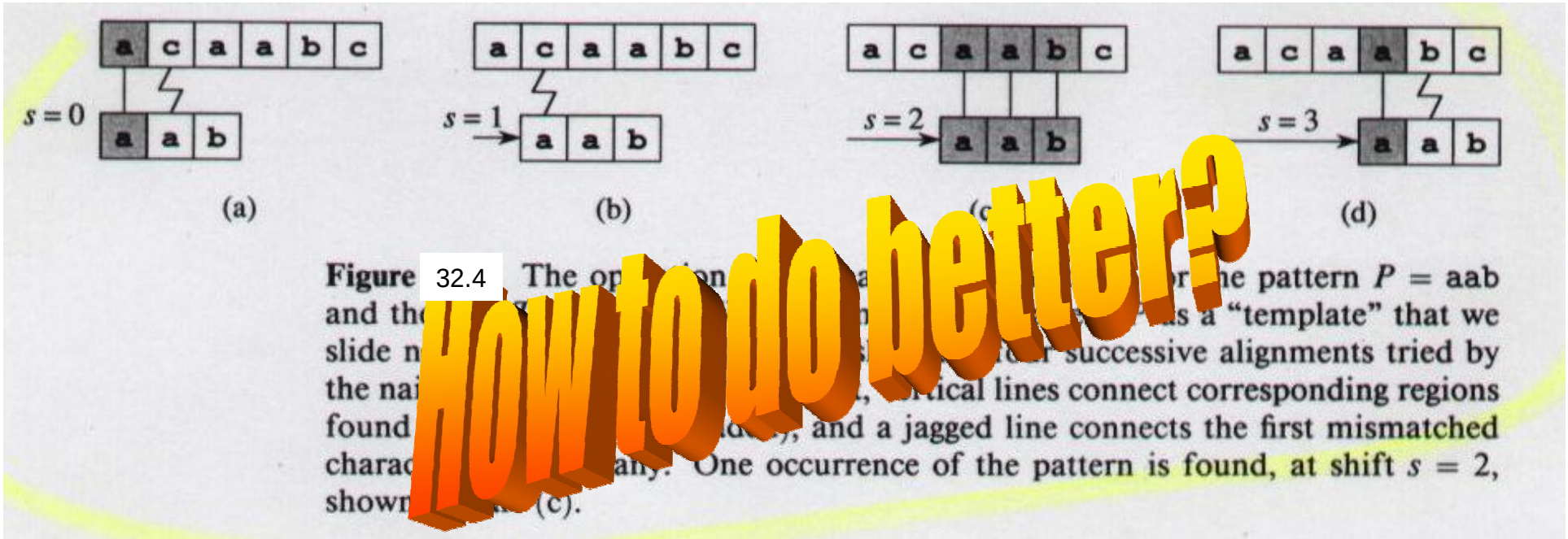


Figure 32.4 The op[illegible] pattern P = aab and th[illegible] as a "template" that we slide n[illegible] successive alignments tried by the nai[illegible]tical lines connect corresponding regions found [illegible], and a jagged line connects the first mismatched charac[illegible]. One occurrence of the pattern is found, at shift $s = 2$, shown [illegible](c).

Rabin-Karp Algorithm

- Assume each character is digit in radix-d notation (e.g. d=10)
- p = decimal value of pattern
- t_s = decimal value of substring T[s+1..s+m] for s = 0,1,..., n-m
- Strategy:
  - compute p in O(m) time
  - compute all t_i values in total of O(n) time
  - ⇒ find all valid shifts s in O(n) time by comparing p with each t_s
- Compute p in O(m) time using Horner's rule:
  - p = P[m] + d(P[m-1] + d(P[m-2] + ... + d(P[2] + dP[1]))…))
- Compute t_0 similarly from T[1..m] in O(m) time
- Compute remaining t_i's in O(n-m+1) time
  - $t_{s+1} = d(t_s - d^{m-1}T[s+1]) + T[s+m+1]$

Rabin-Karp Algorithm

But... p, t_s may be large, so use mod

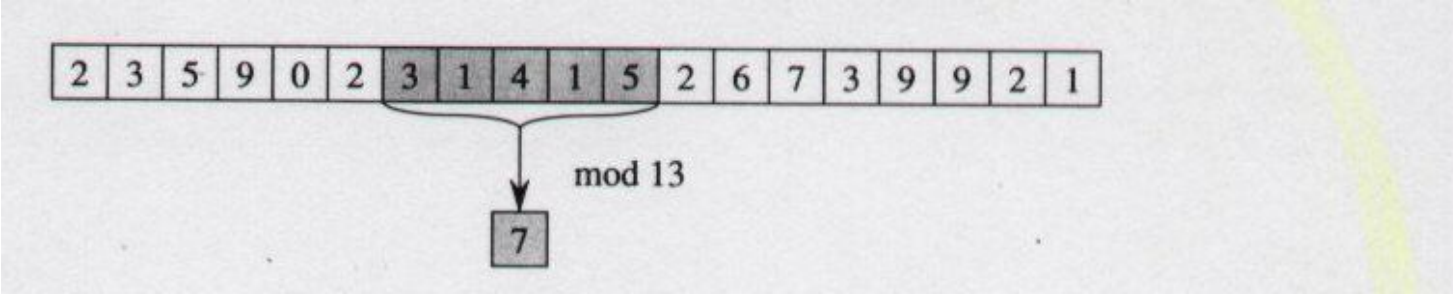


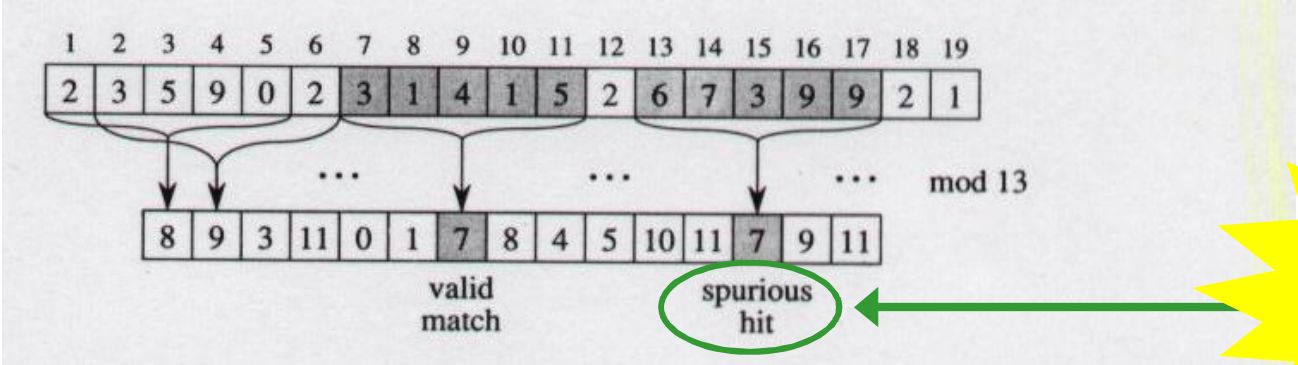
Figure 32.5 The Rabin-Karp algorithm. Each character is a decimal digit, and we compute values modulo 13. **(a)** A text string. A window of length 5 is shown shaded. The numerical value of the shaded number is computed modulo 13, yielding the value 7.

Rabin-Karp Algorithm

But...

$t_{s+1} = d(t_s - d^{m-1}T[s+1]) + T[s+m+1]$

$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q\ , \qquad (34.2)$$

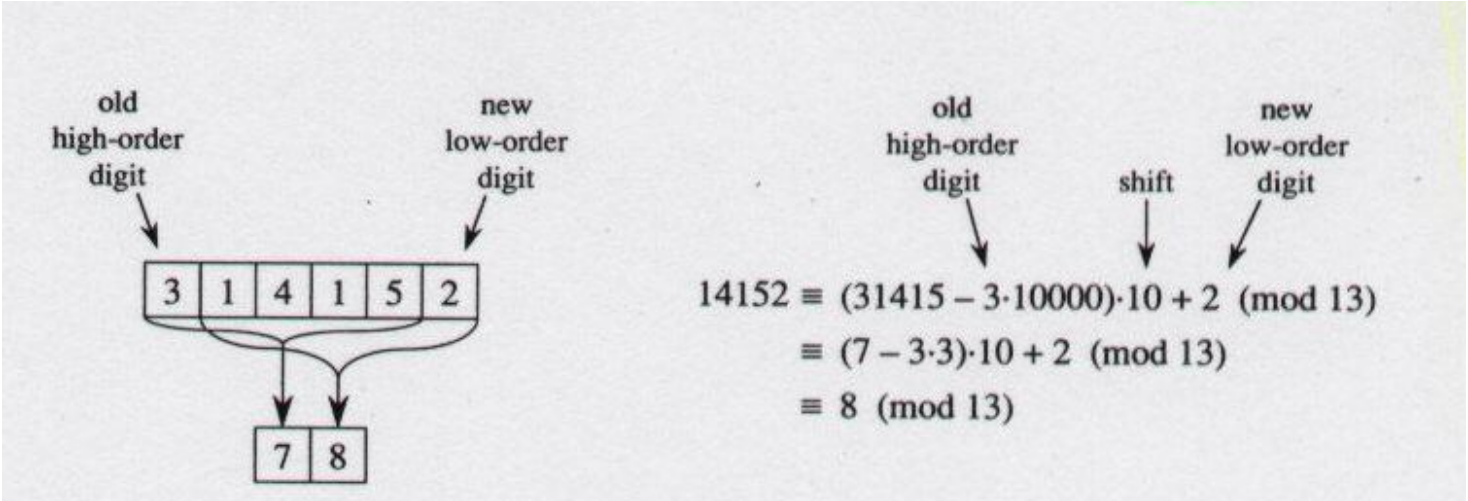


p = 31415

spurious hit

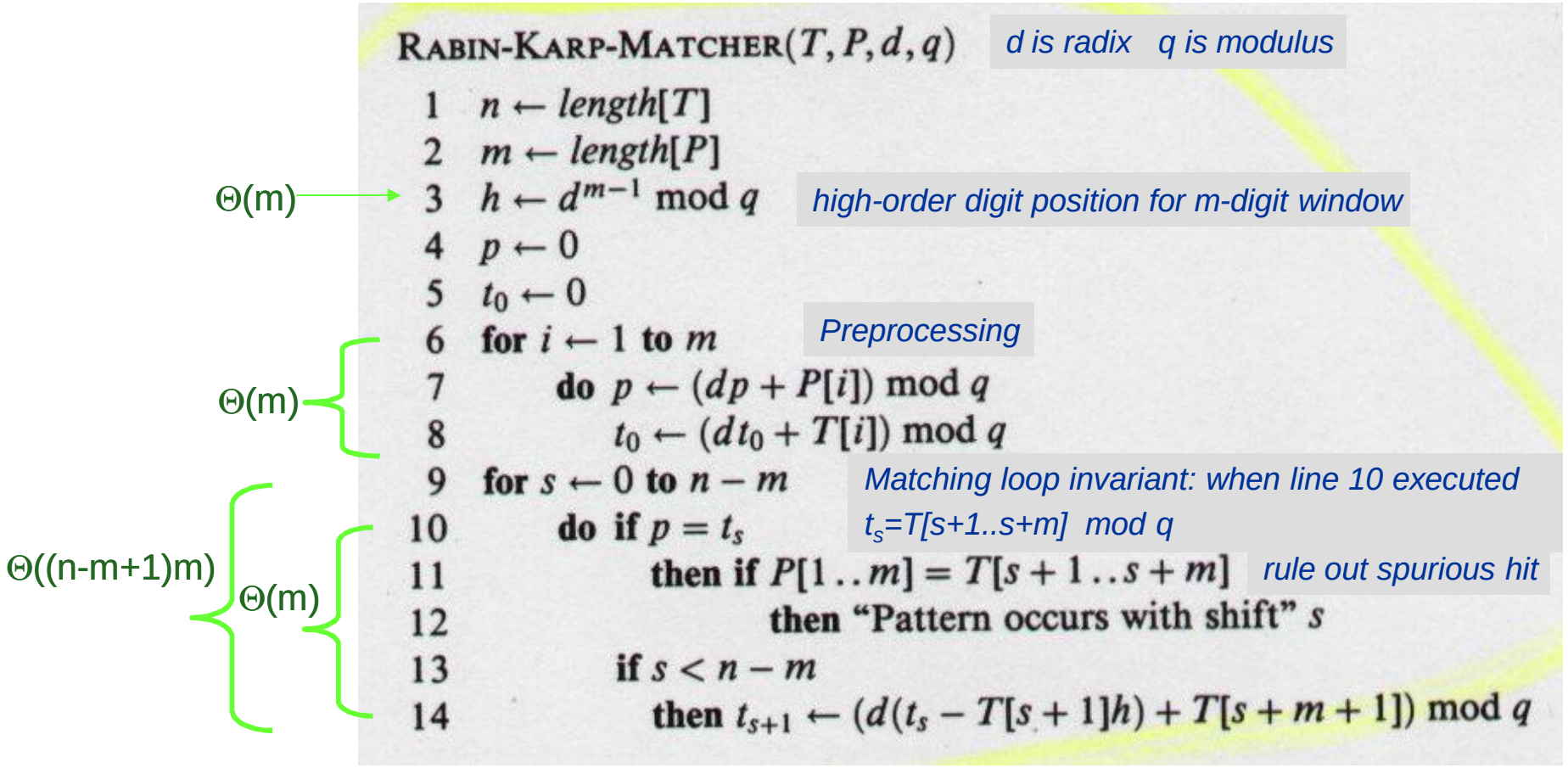
(b) The same text string with values computed modulo 13 for each possible position of a length-5 window. Assuming the pattern $P = 31415$, we look for windows whose value modulo 13 is 7, since $31415 \equiv 7 \pmod{13}$. Two such windows are found, shown shaded in the figure. The first, beginning at text position 7, is indeed an occurrence of the pattern, while the second, beginning at text position 13, is a spurious hit.

Rabin-Karp Algorithm



(c) Computing the value for a window in constant time, given the value for the previous window. The first window has value 31415. Dropping the high-order digit 3, shifting left (multiplying by 10), and then adding in the low-order digit 2 gives us the new value 14152. All computations are performed modulo 13, however, so the value for the first window is 7, and the value computed for the new window is 8.

Rabin-Karp Algorithm



worst-case running time is in Θ((n-m+1)m)