

# STRING MATCHING

Naïve String Matching ,  
Rabin Karp Algorithm

Design and Analysis of Algorithm  
Fall 2022

# REFERENCES

- Intro. to Algorithms by Cormen et al
- Algorithms on Strings, Trees, and Sequences. Comp. Science & Computational Biology by Dan Gusfield. Section 1.1 & 1.2
- Exact String Matching Algorithms.  
<http://www-igm.univ-mlv.fr/~lecroq/string/index.html>



# STRING MATCHING: INTRODUCTION

- Finding all occurrences of a pattern in some text
  - This problem arises frequently in text editing programs.
  - Efficient algorithms improve responsiveness
- In molecular biology, biological molecules can often be approximated as sequences of nucleotides or amino acids.
- Search for particular patterns in DNA sequences



# STRING MATCHING: INTRODUCTION

- Two Types:
  - **Exact String Matching**
  - **Approximate String Matching**



# EXACT STRING MATCHING PROBLEM

- Assume text is any array  $T[1..n]$  of length  $n$
- pattern is an array  $P[1..m]$  of length  $m \leq n$
- Elements of  $T$  &  $P$  are characters drawn from a finite alphabet

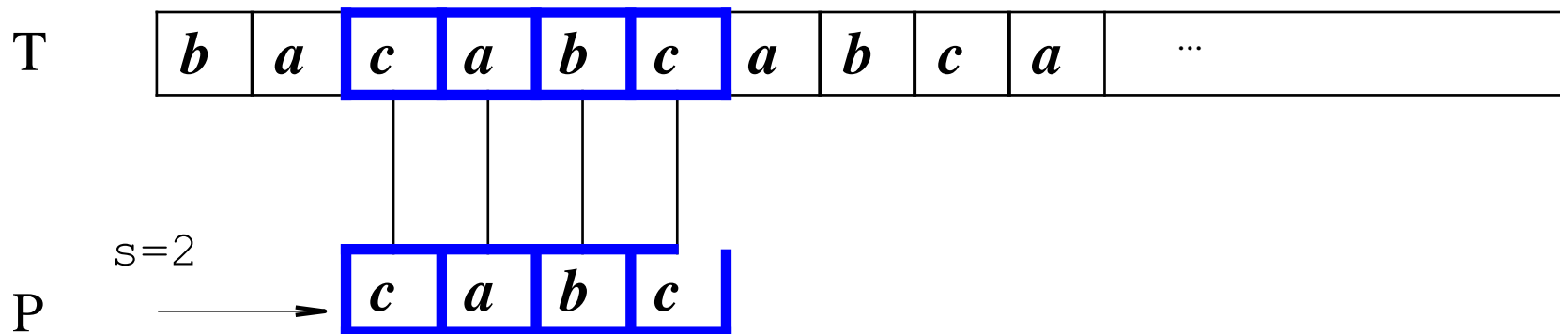
$\Sigma$ . e.g.,  $\Sigma = \{0,1\}$   
or  $\Sigma = \{a,b,\dots,z\}$ .

- $P$  &  $T$  are *strings* of characters.



- <http://whocouldthat.be/visualizing-string-matching/>

# EXACT STRING MATCHING PROBLEM



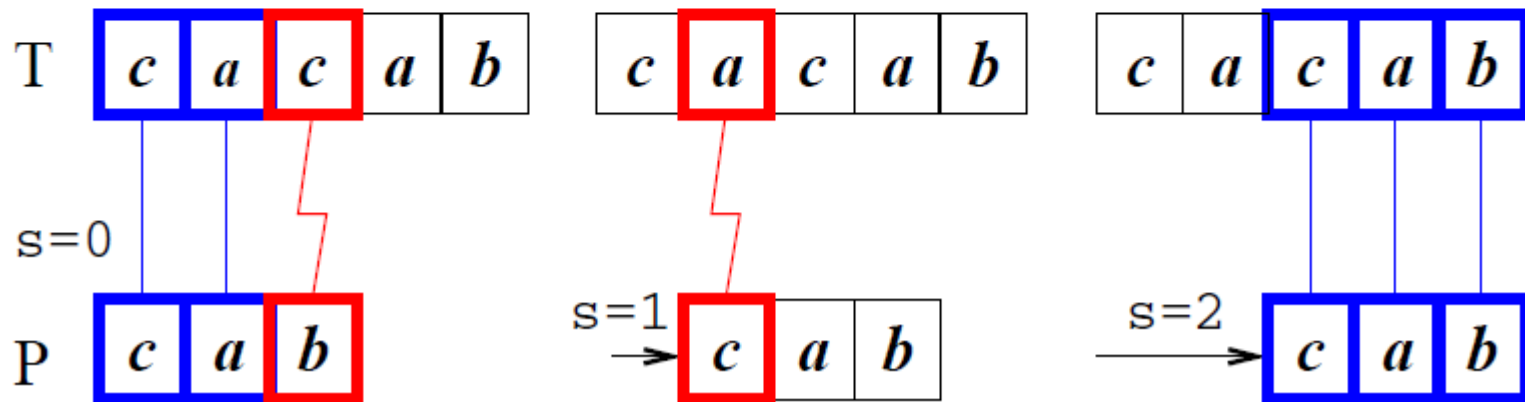
# EXACT STRING MATCHING PROBLEM

- $P$  occurs with **shift  $s$**  in  $T$  or  $P$  occurs beginning at **position  $s+1$**  in text  $T$  if
$$0 \leq s \leq n-m \text{ and } T[s+1..s+m] = P[1..m]$$
- i.e.,  $T[s+j] = P[j]$ , for  $1 \leq j \leq m$
- If  $P$  occurs with shift  $s$ ,  $s$  is a **valid shift**; otherwise we call  $s$  an **invalid shift**.
- **String matching problem**: finding all valid shifts with which a given pattern  $P$  occurs in a given text  $T$

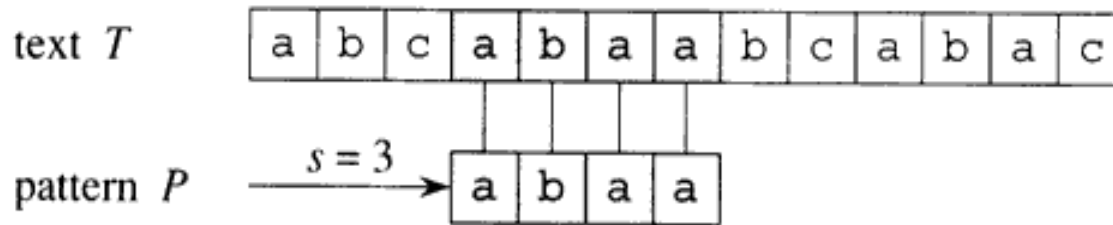


# EXACT STRING MATCHING PROBLEM

Initially,  $P$  is aligned with  $T$  at the first index position.  $P$  is then compared with  $T$  from **left-to-right**. If a mismatch occurs, "slide"  $P$  to *right* by 1 position, and start the comparison again.



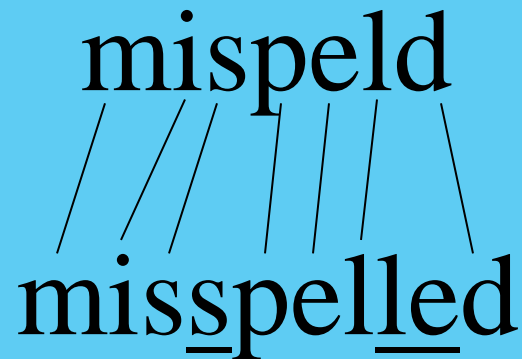
# Exact String Matching Problem



**Figure 32.1** The string-matching problem. The goal is to find all occurrences of the pattern  $P = \text{abaa}$  in the text  $T = \text{abcabaabcabac}$ . The pattern occurs only once in the text, at shift  $s = 3$ . The shift  $s = 3$  is said to be a valid shift. Each character of the pattern is connected by a vertical line to the matching character in the text, and all matched characters are shown shaded.

# APPROXIMATE STRING MATCHING

- Given a text to search (T) and a pattern to look for (P).
- Find all of the occurrences of P that exist in T, allowing a defined number of errors to be present in the matches



A diagram illustrating approximate string matching. It shows the word "mispeld" at the top and "misspelled" at the bottom, both within a light blue rectangular box. Six diagonal lines connect the letters of "mispeld" to the corresponding letters in "misspelled": from 'm' to 'm', 'i' to 'i', 's' to 's', 'p' to 'p', 'e' to 'e', and 'd' to 'd'. The second 's' in "misspelled" is underlined, and the second 'e' is underlined, indicating the positions of errors (substitutions) relative to the original pattern.



# APPROXIMATE STRING MATCHING

- The closeness of a match is measured in terms of the number of primitive operations necessary to convert the string into an exact match.
  - insertion: *cot*  $\rightarrow$  *coat*
  - deletion: *coat*  $\rightarrow$  *cot*
  - substitution: *coat*  $\rightarrow$  *cost*



# IMPORTANCE

- Not a problem for small input
  - Word Processors, utilities, in library catalog searching programs
- Large Input
  - internet crawlers, digital libraries, e-journals.
  - Several hundred specialized databases holding raw DNA, RNA and amino acid strings (e.g., US GenBank)
  - When applied on DNA databases, search may take hours or days.




# Running Time of String Matching Algorithms

Algorithm	Preprocessing time	Matching time
Naive	0	$O((n - m + 1)m)$
Rabin-Karp	$\Theta(m)$	$O((n - m + 1)m)$
Finite automaton	$O(m  \Sigma )$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$



## NOTATION & TERMINOLOGY

- $\Sigma^*$  = set of all finite-length strings from alphabet  $\Sigma$
  - $\varepsilon$  = *empty string* (zero length string)
  - $|x|$  = length of string  $x$
  - $xy$  = concatenation of strings  $x$  and  $y$ , length is  $|x| + |y|$
  - $w$  is a *prefix* of  $x$ ,  $w \preceq x$ , if  $x = wy$ , where  $y \in \Sigma^*$
  - $w$  is a *suffix* of  $x$ ,  $w \preceq^r x$ , if  $x = yw$ , where  $y \in \Sigma^*$
- 

## NOTATION & TERMINOLOGY (CONT'D)

- $\varepsilon$  is both a suffix and a prefix of every string
  - $ab \models abcca$
  - $cca \models abcca$
- We denote k-character prefix  $P[1..k]$  of the pattern  $P[1..m]$  by  $P_k$ .
- So  $P_0 = \varepsilon$  and  $P_m = P = P[1..m]$
- Similarly, k-character prefix of text  $T$  as  $T_k$
- **String matching prob.:** find all shifts in the range  $0 \leq s \leq n-m$  such that  $P \models T_{s+m}$





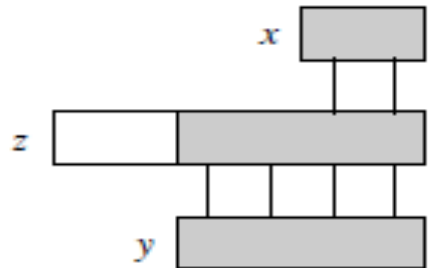
## OVERLAPPING SUFFICES

○  $x$ ,  $y$  and  $z$  are strings and  $x \sqsupseteq z$  and  $y \sqsupseteq z$   
if:

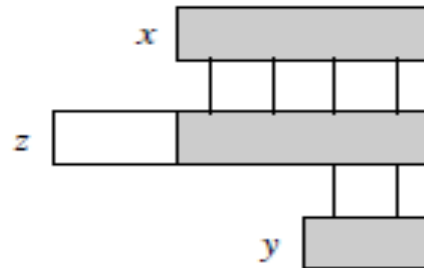
- $|x| \leq |y|$  then  $x \sqsupseteq y$
- $|x| \geq |y|$  then  $y \sqsupseteq x$
- $|x| = |y|$  then  $x = y$



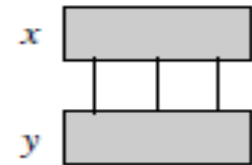
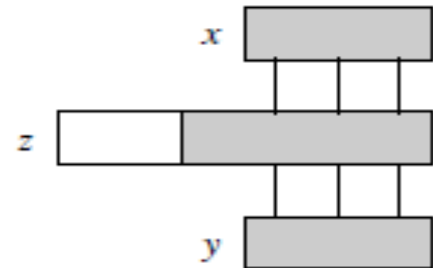
# OVERLAPPING SUFFICES



(a)



(b)



(c)

EF    **x**  
 ABCDEF    **z**  
 CDEF    **y**



# STRING COMPARISON

- The test “ $x=y$ ” is assumed to take time  $\Theta(t+1)$  where  $t$  is the length of the longest string  $z$  such that  $z \preceq x$  and  $z \preceq y$ .
- $t=0$ , when  $z = \varepsilon$

$w$  is a *prefix* of  $x$ ,  $w \preceq x$ ,  
 $w$  is a *suffix* of  $x$ ,  $w \preceq^r x$ ,



# NAÏVE STRING MATCHING ALGORITHM

The naive algorithm finds all valid shifts using a loop that checks the condition  $P[1..m] = T[s + 1..s + m]$  for each of the  $n - m + 1$  possible values of  $s$ .

NAIVE-STRING-MATCHER( $T, P$ )

```
1   $n = T.length$ 
2   $m = P.length$ 
3  for  $s = 0$  to  $n - m$ 
4      if  $P[1..m] == T[s + 1..s + m]$ 
5          print "Pattern occurs with shift"  $s$ 
```

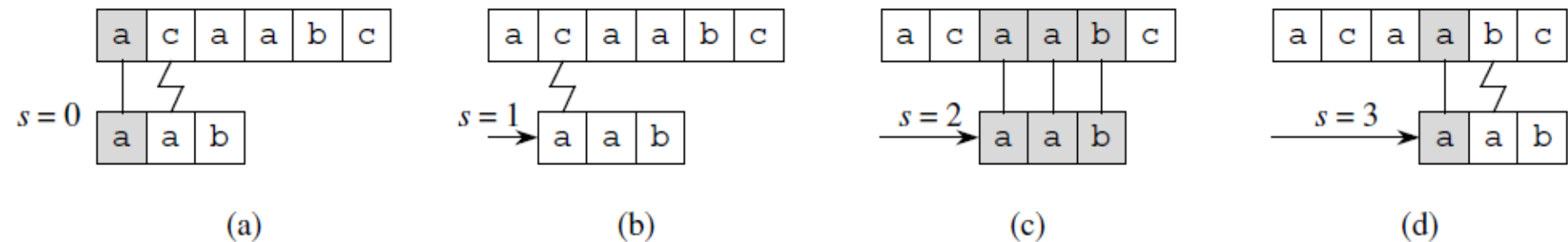
$O((n-m+1) m)$

The test in line 4 determines whether the current shift is valid; this test implicitly loops to check corresponding character positions until all positions match successfully or a mismatch is found.

# NAÏVE STRING MATCHING ALGORITHM

- Finds all valid shifts using a loop that tests  $P[1..m] = T[s+1..s+m]$  for each of  $n-m+1$  values of  $s$
- Takes  $\Theta((n-m+1)m)$  time.
- If  $m=n/2$ , it becomes  $\Theta(n^2)$





**Figure 32.4** The operation of the naive string matcher for the pattern  $P = aab$  and the text  $T = acaabc$ . We can imagine the pattern  $P$  as a “template” that we slide next to the text. (a)–(d) The four successive alignments tried by the naive string matcher. In each part, vertical lines connect corresponding regions found to match (shown shaded), and a jagged line connects the first mismatched character found, if any. One occurrence of the pattern is found, at shift  $s = 2$ , shown in part (c).

## WHY IS NAÏVE ALGO. INEFFICIENT?

- It tests for each of the  $n-m+1$  possible values of  $s$
- The info. gained about text for one value of  $s$  is ignored in considering other values of  $s$
- e.g., if  $P=aaab$  and we find that  $s=0$  is valid, then  $s=1, 2$  or  $3$  are invalid, since  $T[4]=b$





# RABIN KARP ALGORITHM

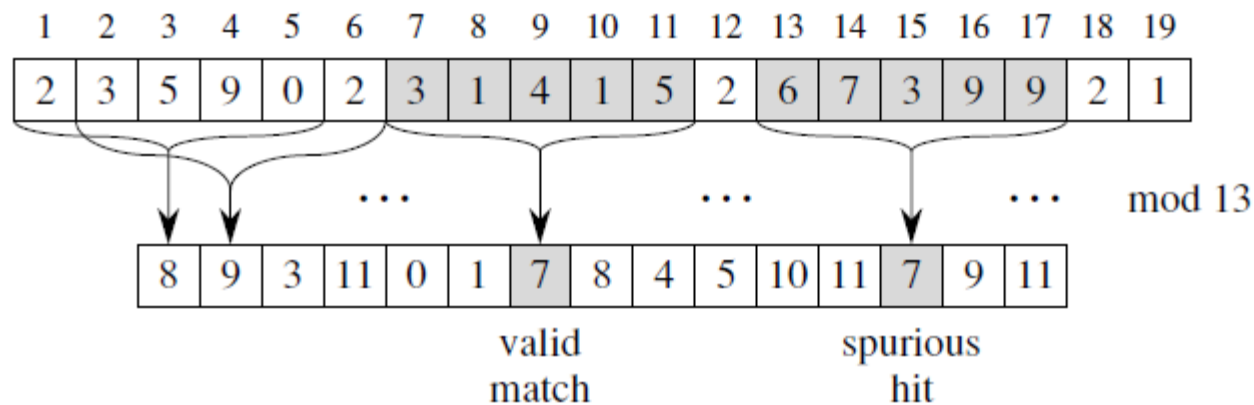
24



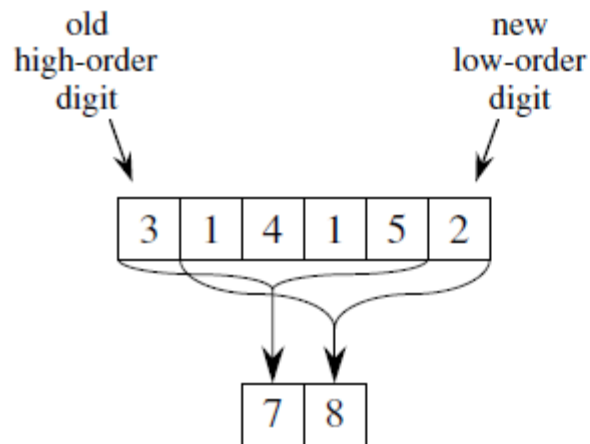
# THE RABIN-KARP ALGORITHM

- Performs well in practice
- generalizes to other algorithms for related problems such as 2D pattern matching
- Two steps:
  - Preprocessing  $\Theta(m)$
  - Matching  $\Theta((n-m+1)m)$
- Based on certain assumptions, average case running time is better
- Makes use of number theoretic notions, equivalence of two numbers modulo a third number





(b)



old high-order digit      shift      new low-order digit

$$\begin{aligned}
 14152 &\equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13} \\
 &\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13} \\
 &\equiv 8 \pmod{13}
 \end{aligned}$$

Fig 32.5

(c)

# THE RABIN-KARP ALGORITHM

- The character string 31415 thus corresponds to the decimal number 31,415.

Given pattern  $P[1..m]$ , let  $p$  denote its decimal value

Let  $t_s$  denote decimal value of length  $m$  substring  $T[s+1..s+m]$  for  $s=0,1,\dots,n-m$

$t_s=p$  if and only if  $T[s+1..s+m] = P[1..m]$

# THE RABIN-KARP ALGORITHM (CONT'D)

- Assume that  $\Sigma = \{0,1,2,\dots,9\}$ , each character is a decimal digit.
- In general, each char. is a digit in radix- $d$  notation, where  $d = |\Sigma|$
- A string of  $k$  digits = a length- $k$  decimal number
- Given pattern  $P[1..m]$ , let  $p$  denote its decimal value
- Let  $t_s$  denote decimal value of length  $m$  substring  $T[s+1..s+m]$  for  $s=0,1,\dots,n-m$
- $t_s = p$  if and only if  $T[s+1..s+m] = P[1..m]$



# THE RABIN-KARP ALGORITHM (CONT'D)

1. Compute  $p$  in  $\Theta(m)$  time
  2. Compute all  $t_s$  values in  $\Theta((n-m)+1)$  time
  3. Then, we can determine all valid shifts in time  $\Theta(m) + \Theta((n-m)+1) = \Theta(n)$
- Lets not worry that  $p$  and  $t_s$ 's might be very large
  - It takes  $\lg(a)$  bits to encode number  $a$
  - very large=do not fit in a computer word



# THE RABIN-KARP ALGORITHM (CONT'D)

- We can compute  $p$  in  $\Theta(m)$  time, using Horner's rule.

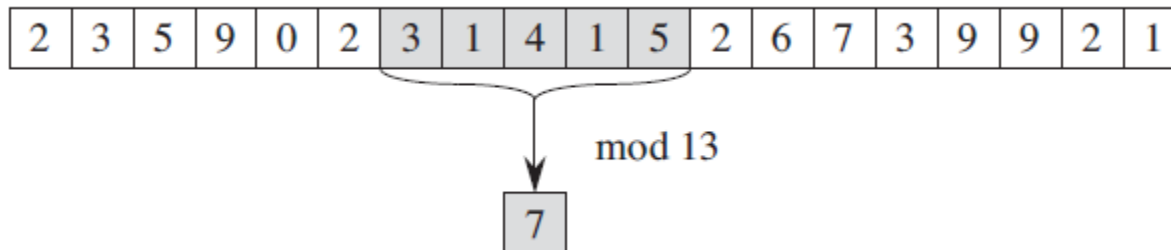
$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \dots + 10(P[2] + 10P[1]) \dots))$$

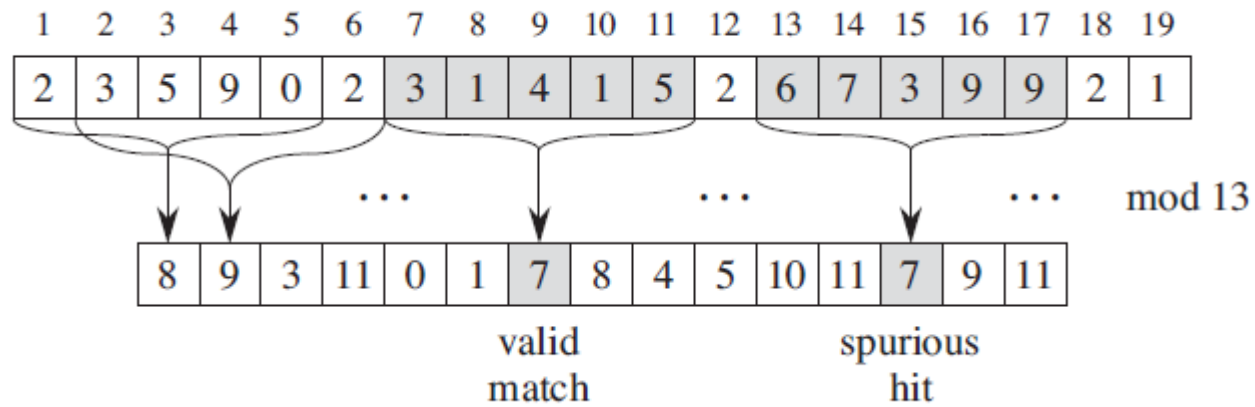
- e.g.,  $P[1..m]$ ="3457", here  $m=4$ 
  - $p = 7 + 10(5 + 10(4 + 10 \times 3))$
  - $p = 7 + 10(5 + 10(4 + 30))$
  - $p = 7 + 10(5 + 340)$
  - $p = 7 + 3450$
  - $p = 3457$



# THE RABIN-KARP ALGORITHM (CONT'D)

- Similarly compute  $t_o$  in  $\Theta(m)$  time from  $T[1..m]$
- Use  $t_o$  to compute each of  $t_1, t_2, \dots, t_{n-m}$  in constant time, which totals to  $\Theta(n-m)$  time.
- $t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$
- e.g.,  $m=5$  and  $t_s=31415$  and  $T[s+5+1]=2$ ,  $s=0$ 
  - $t_{s+1} = 10(31415 - 10000.3) + 2$
  - $\quad = 14152$



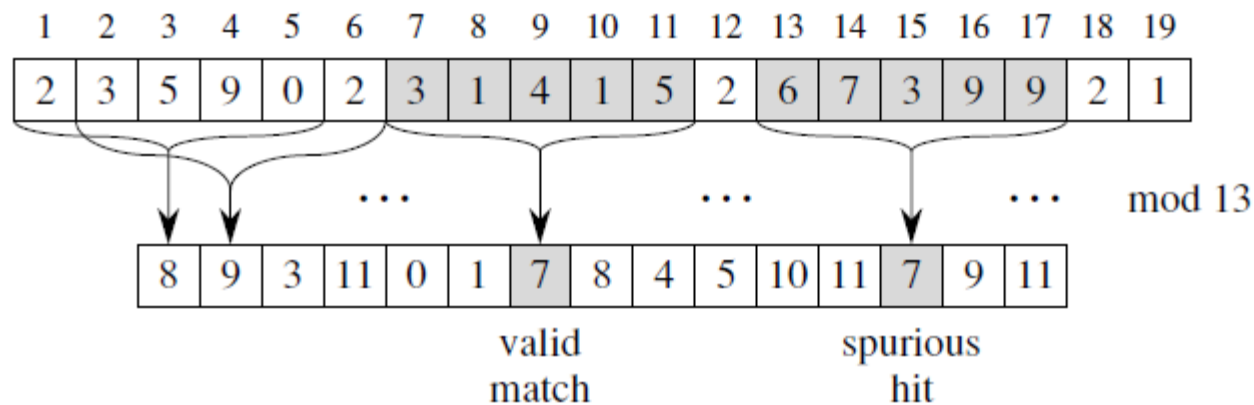




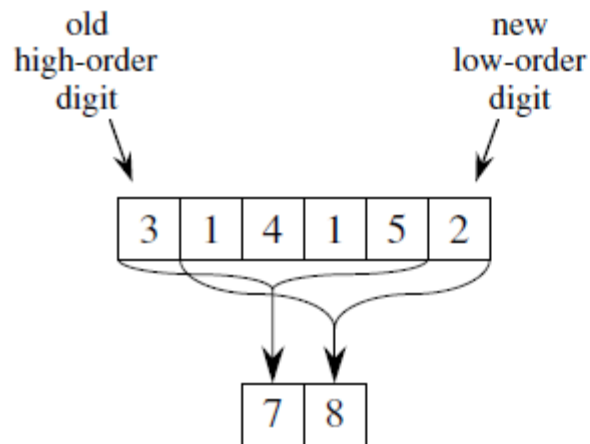
# THE RABIN-KARP ALGORITHM (CONT'D)

- If  $p$  and  $t_s$  are very large to work with, mathematical operations on  $P$  do not take “constant time”
- Simple cure, perform all operations modulo a suitable modulus  $q$
- $q$  is chosen, such that  $dq$  fits in computer word, where  $d = |\Sigma|$  and  $\Sigma = \{0, 1, \dots, d-1\}$
- We might get spurious hits, since  $t_s \equiv p \pmod{q}$  does not imply that  $t_s = p$





(b)



old high-order digit      shift      new low-order digit

$$\begin{aligned}
 14152 &\equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13} \\
 &\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13} \\
 &\equiv 8 \pmod{13}
 \end{aligned}$$

Fig 32.5

(c)

# THE RABIN-KARP ALGORITHM

- Fig 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. (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. (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-MATCHER( $T, P, d, q$ )

```
1   $n \leftarrow \text{length}[T]$ 
2   $m \leftarrow \text{length}[P]$ 
3   $h \leftarrow d^{m-1} \bmod q$    Line 3 initializes h to the value of the high-order digit position of an m-digit window.
4   $p \leftarrow 0$ 
5   $t_0 \leftarrow 0$ 
6  for  $i \leftarrow 1$  to  $m$            ▷ Preprocessing.
7      do  $p \leftarrow (dp + P[i]) \bmod q$ 
8           $t_0 \leftarrow (dt_0 + T[i]) \bmod q$ 
9  for  $s \leftarrow 0$  to  $n - m$        ▷ Matching.
10     do if  $p = t_s$ 
11         then if  $P[1..m] = T[s + 1..s + m]$ 
12             then print “Pattern occurs with shift”  $s$ 
13         if  $s < n - m$ 
14             then  $t_{s+1} \leftarrow (d(t_s - T[s + 1]h) + T[s + m + 1]) \bmod q$ 
```



# THE RABIN-KARP ALGORITHM (CONT'D)

- It takes  $\Theta(m)$  preprocessing time and  $\Theta((n-m+1)m)$  matching time in the worst case.
- In many applications, we expect few valid shifts (some constant  $c$ ), then the running time is  $O((n-m+1) + cm)$  + time for spurious hits
- and if  $q$  is large enough ( $q \geq m$ ), we can reduce spurious hits, which gives us  $O(n+m)$  running time.



# REFERENCE

## ○ Introduction to Algorithms

- Thomas H. Cormen
- Chapter # 32