String Matching

Brute-Force String Matching

- ***pattern***: a string of *m* characters to search for
- ***text***: a (longer) string of *n* characters to search in
- problem: find a substring in the text that matches the pattern

Brute-force algorithm

Step 1 Align pattern at beginning of text

Step 2 Moving from left to right, compare each character of pattern to the corresponding character in text until
  - all characters are found to match (successful search); or
  - a mismatch is detected

Step 3 While pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

Examples of Brute-Force String Matching

1. **Pattern:** 001011
   Text: 10010101101001100101111010

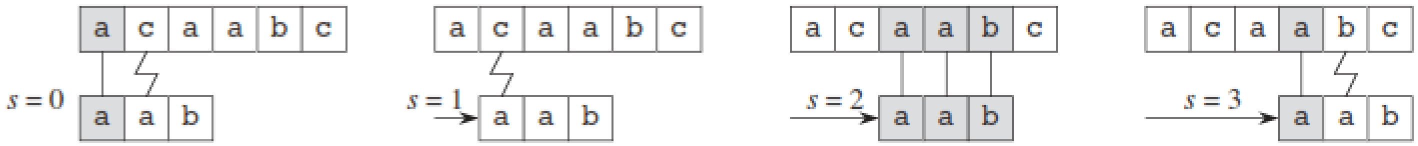
2. **Pattern:** happy
   Text: It is never too late to have a happy childhood.

Pseudocode and Efficiency

ALGORITHM *BruteForceStringMatch*$(T[0..n-1], P[0..m-1])$

```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//        an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//         matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

Efficiency:



(a) (b) (c) (d)

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
```

FIGURE 3.3 Example of brute-force string matching. (The pattern's characters that are compared with their text counterparts are in bold type.)

Shift amount	Text and pattern
0	`GTAACAGTAAACG` `AAC`
1	`GTAACAGTAAACG` ` AAC`
2	`GTAACAGTAAACG` `  AAC`
3	`GTAACAGTAAACG` `   AAC`
4	`GTAACAGTAAACG` `    AAC`
5	`GTAACAGTAAACG` `     AAC`
6	`GTAACAGTAAACG` `      AAC`
7	`GTAACAGTAAACG` `       AAC`
8	`GTAACAGTAAACG` `        AAC`
9	`GTAACAGTAAACG` `         AAC`
10	`GTAACAGTAAACG` `          AAC`

Determine the number of character comparisons made by the brute-force algorithm in searching for the pattern GANDHI in the text

THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED

Assume that the length of the text—it is 47 characters long—is known before the search starts.

Show the comparisons the naive string matcher makes for the pattern $P = 0001$ in the text $T = 000010001010001$.

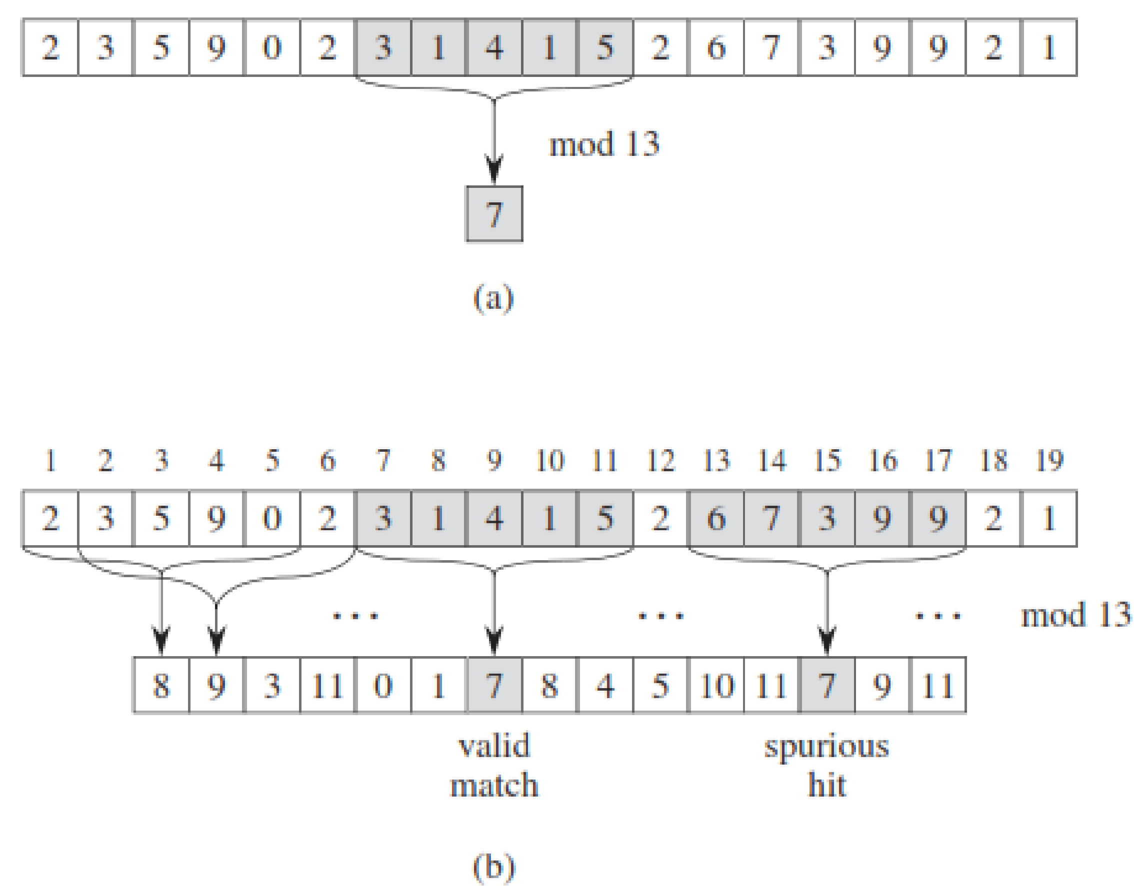
How many comparisons (both successful and unsuccessful) will be made by the brute-force algorithm in searching for each of the following patterns in the binary text of one thousand zeros?

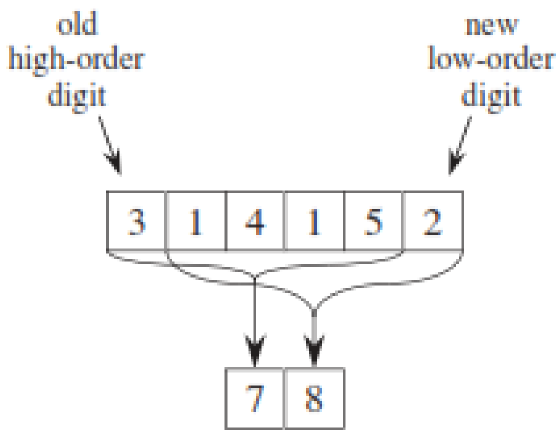
a. 00001 **b.** 10000 **c.** 01010

Give an example of a text of length n and a pattern of length m that constitutes a worst-case input for the brute-force string-matching algorithm. Exactly how many character comparisons will be made for such input?

Rabin Karp string matching

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 $s = 0, 1, \ldots, n-m$. 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 $\Theta(m)$ and all the t_s values in a total of $\Theta(n-m+1)$ time,[1]


2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1
mod 13
7
(a)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1
mod 13
8 9 3 11 0 1 7 8 4 5 10 11 7 9 11
valid
match
spurious
hit
(b)



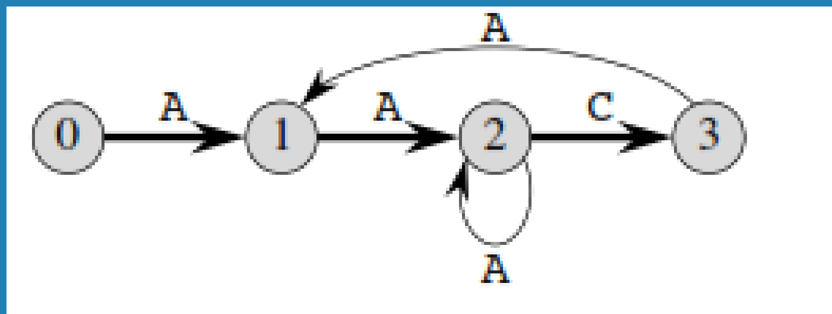
old high-order digit, shift, new low-order digit

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

(c)

Working modulo $q = 11$, how many spurious hits does the Rabin-Karp matcher encounter in the text $T = 3141592653589793$ when looking for the pattern $P = 26$?

Finite Automata



the pattern AAC does on the input text
GTAACAGTAAACG

state	0	0	0	1	2	3	1	0	0	1	2	2	3	0
character	G	T	A	A	C	A	G	T	A	A	A	C	G	

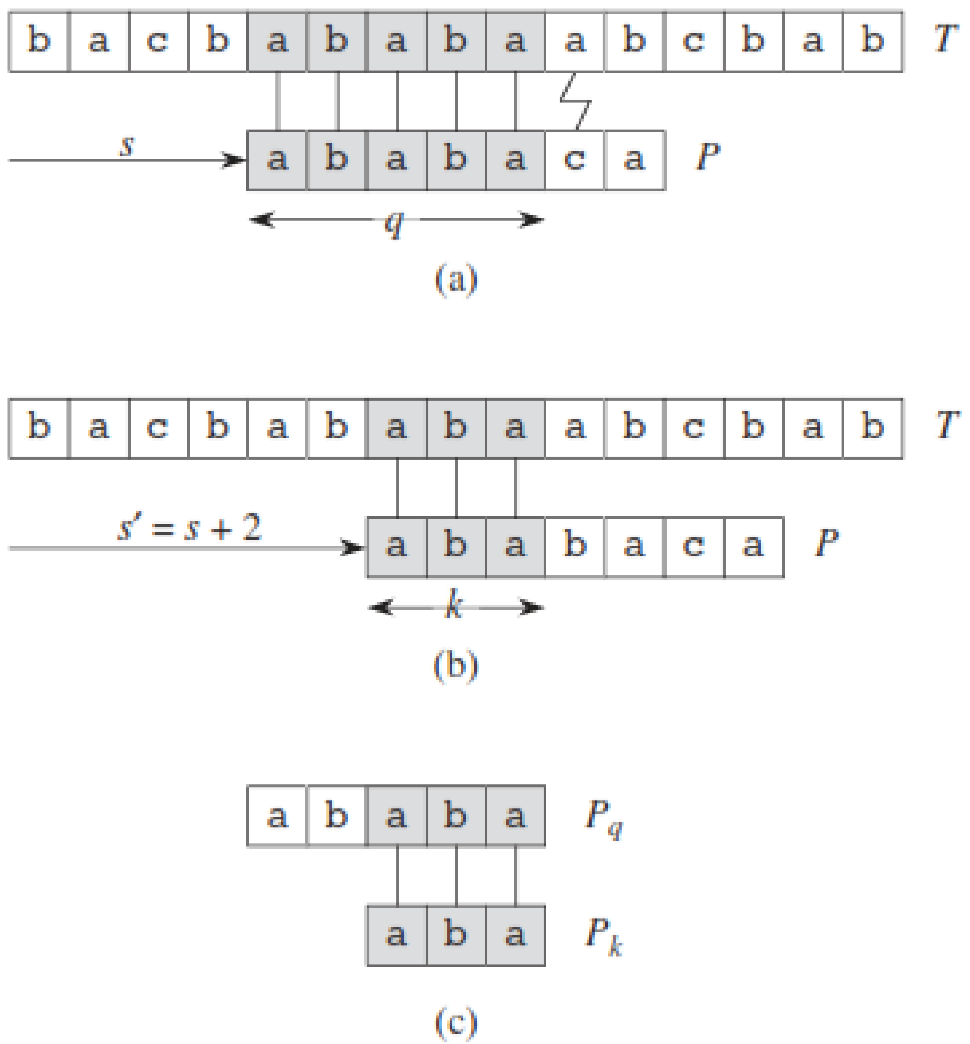
txt = "AAAAABAAABA" pat = "AAAA" We compare first window of txt with pat txt = "AAAAABAAABA" pat = "AAAA" [Initial posit

Knuth-Morris-Pratt algorithm

- Linear Time Algorithm
- txt = "AAAAABAAABA"
- pat = "AAAA"

- We compare first window of txt with pat
- txt = "AAAAABAAABA"
- pat = "AAAA" [Initial position]
- We find a match. This is same as Naive String Matching.

- In the next step, we compare next window of txt with pat.
- txt = "AAAAABAAABA"
- pat = "AAAA" [Pattern shifted one position]



b a c b a b a b a a b c b a b T
s
a b a b a c a P
q
(a)
b a c b a b a b a a b c b a b T
s′ = s + 2
a b a b a c a P
k
(b)
a b a b a P_q
a b a P_k
(c)

```
QUICKSORT(A, p, r)
    if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```