DTS203TC
Design and Analysis of Algorithms

Lecture 17: String Matching

Dr. Qi Chen
School of AI and Advanced Computing

Announcements

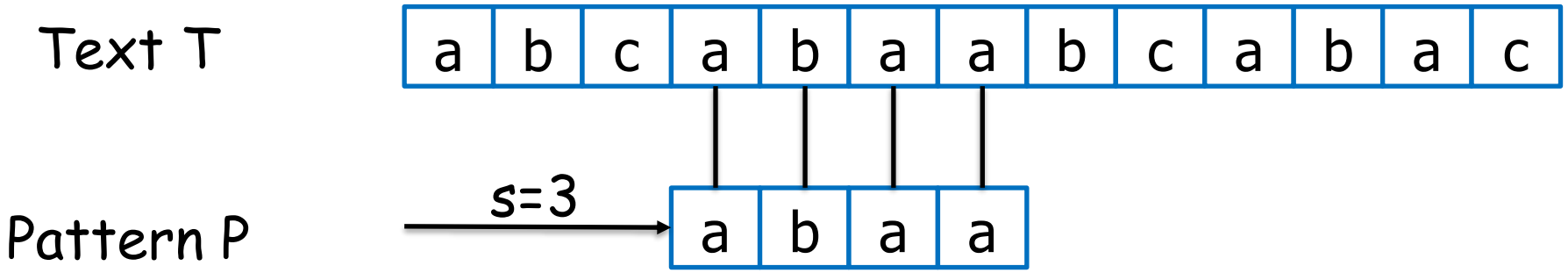
- Seminar: next Tuesday (April 2nd) 15:00 – 17:00 TC-AB-2003
- Exam: April 12th 14:00 – 16:00 M1018

Learning outcome

- String Matching
- Rabin-Karp Algorithm
- Knuth-Morris-Pratt Algorithm

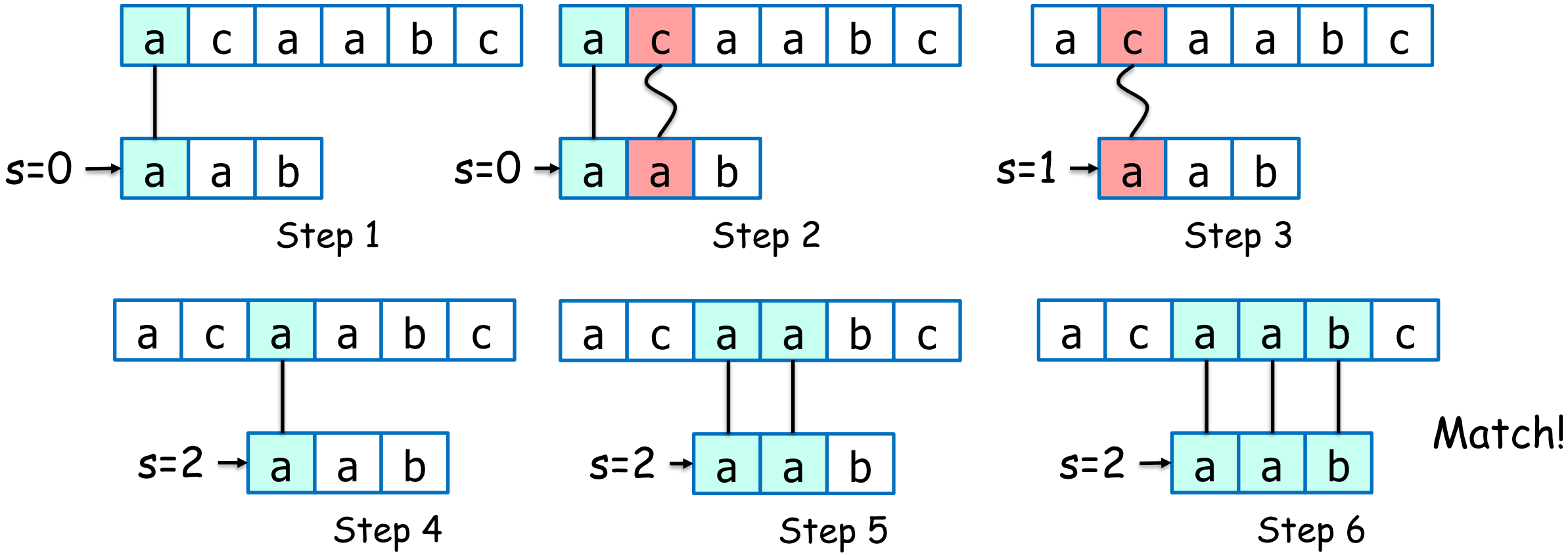
String matching problem

- Text: a longer string T
- Pattern: a shorter string P
- Find all valid shifts (s) with which a pattern P occurs in a text T



Naïve Algorithm (Brute Force)

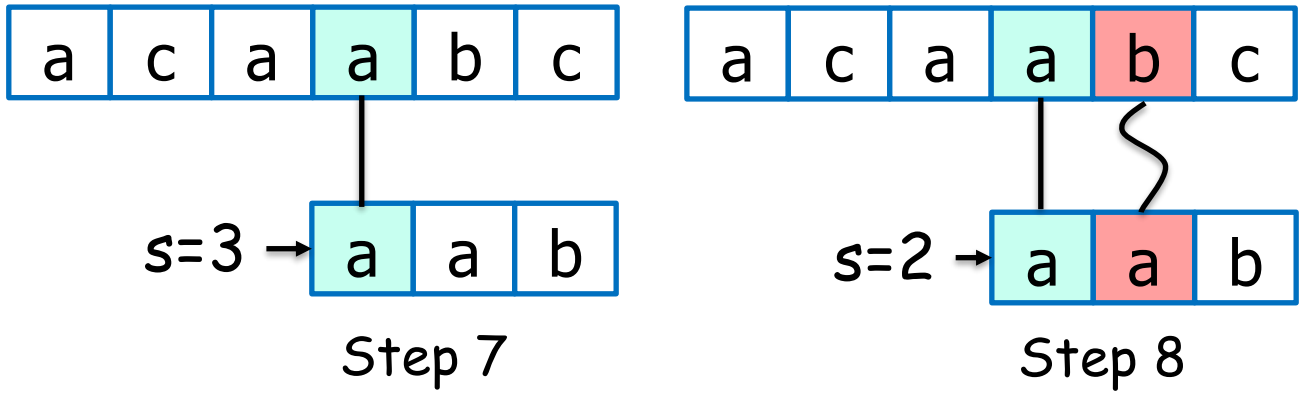
- The Naïve algorithm compares the pattern to the text, one character at a time, until unmatching characters are found:



Naïve Algorithm (Brute Force)

- The Naïve algorithm compares the pattern to the text, one character at a time, until unmatching characters are found:

- The algorithm can be designed to stop on either the first occurrence of the pattern, or upon reaching the end of the text.

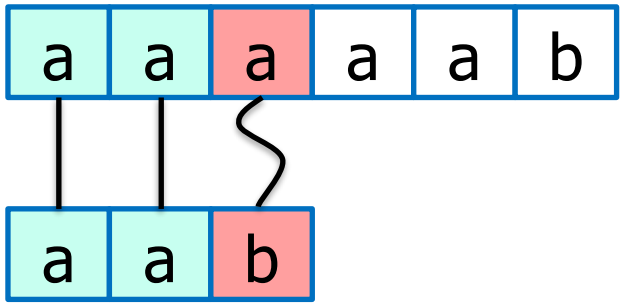


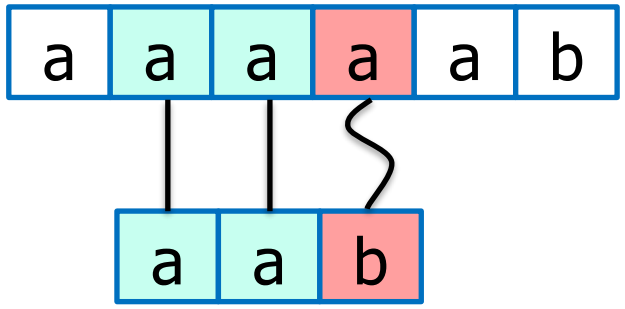
Naïve Algorithm (Brute Force)

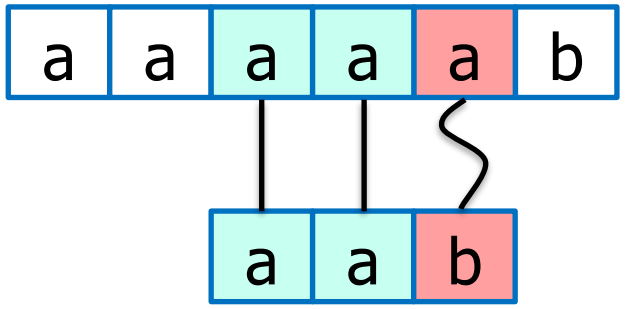
```
NAIVE-STRING-MATCHER(T, P)
n=T.length
m=P.length
for s = 0 to n - m
        if P[1..m] == T[s+1..s+m]
                print "Pattern occurs with shift" s
```

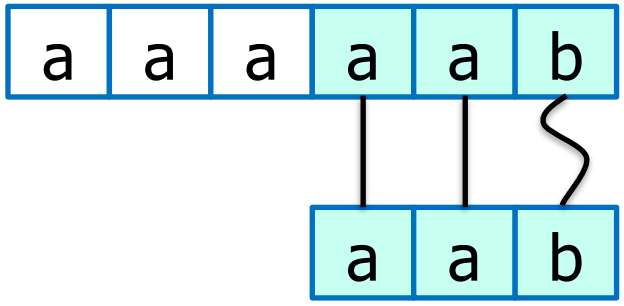
Naïve Algorithm - Complexity

- Given a length-m pattern, and a length-n text.
- Worst case: compares pattern to each substring of text.







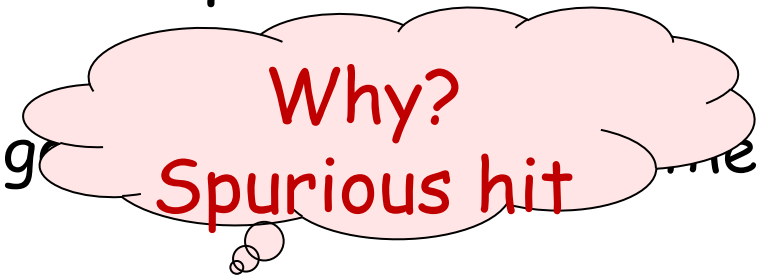


Worst Case: O(nm)

Rabin-Karp Algorithm

Rabin-Karp Algorithm

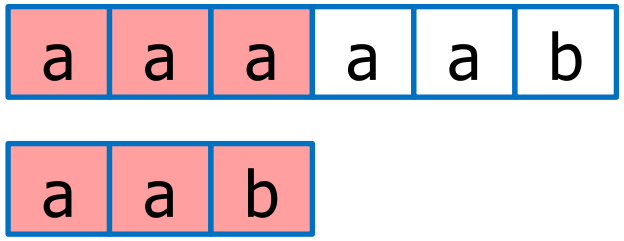
- The Rabin-Karp algorithm calculates a **hash value** for the pattern, and for each m-character subsequence of text to be compared.
  - If the hash values are not equal, the alg… hash value for next m-character sequence.
  - If equal, the algorithm will do a brute force comparison between the pattern and the m-character sequence.



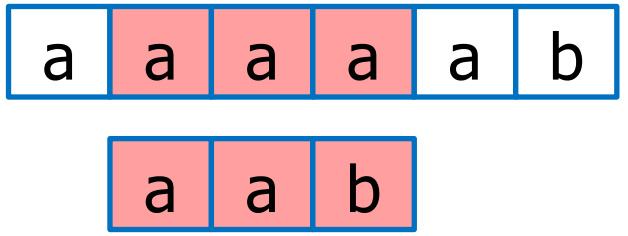
- There is only one comparison per text subsequence, and Brute Force is only needed when hash values match.

Rabin-Karp Example

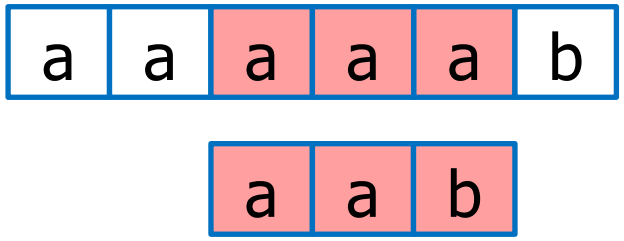
- Hash value of "aaa" is 111.
- Hash value of "aab" is 112.



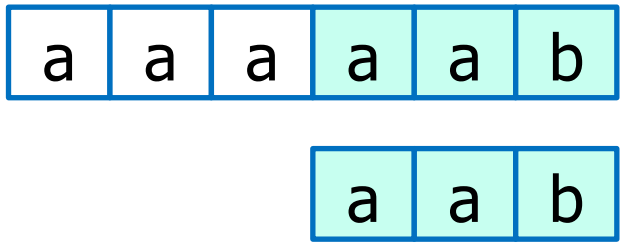
111 != 112 One Comparison



111 != 112 One Comparison



111 != 112 One Comparison



112 = 112 One Comparison,
Then, check if they match

Rabin-Karp

- What is the hash function used to calculate values for character sequences?

- Isn't it time consuming to hash every one of the m-character sequence in the text body?

Rabin-Karp Math

- Consider a m-character sequence as an m-digit number in base d, where d is the number of letters in the alphabet. The text subsequence T[s+1…s+m] is mapped to the number

$$x_s = T[s+1]\cdot d^{m-1} + T[s+2]\cdot d^{m-2} + \ldots + T[s+m]$$

- Given x_s we can compute x_{s+1} for the next subsequence T[s+2..s+m+1] in constant time as follow:

$$x_{s+1} = T[s+2]\cdot d^{m-1} + T[s+3]\cdot d^{m-2} + \ldots + T[s+m+1]$$

$$x_{s+1} = x_s \cdot d \quad \text{Shift one digit}$$

$$-T[s+1]\cdot d^m \quad \text{Subtract leftmost digit}$$

$$+T[s+m+1] \quad \text{Add new rightmost digit}$$

Rabin-Karp Mods

- If m is large, the resulting value (d^m) will be enormous. For this reason, we hash the value by taking it mod a prime number q.

$$[(x \bmod q) + (y \bmod q)] \bmod q = (x+y) \bmod q$$
$$(x \bmod q) \bmod q = x \bmod q$$

- Then:

$$t_s = (T[s+1] \cdot d^{m-1} \bmod q + T[s+2] \cdot d^{m-2} \bmod q + \ldots$$
$$+ T[s+m] \bmod$$

Rolling Hash

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

Shift one digit

Subtract leftmost digit

Add new rightmost digit

Rabin-Karp Example

- Let's assign a numerical value for each character. Here, we have taken first ten alphabets only (i.e. a to j).

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Pattern P

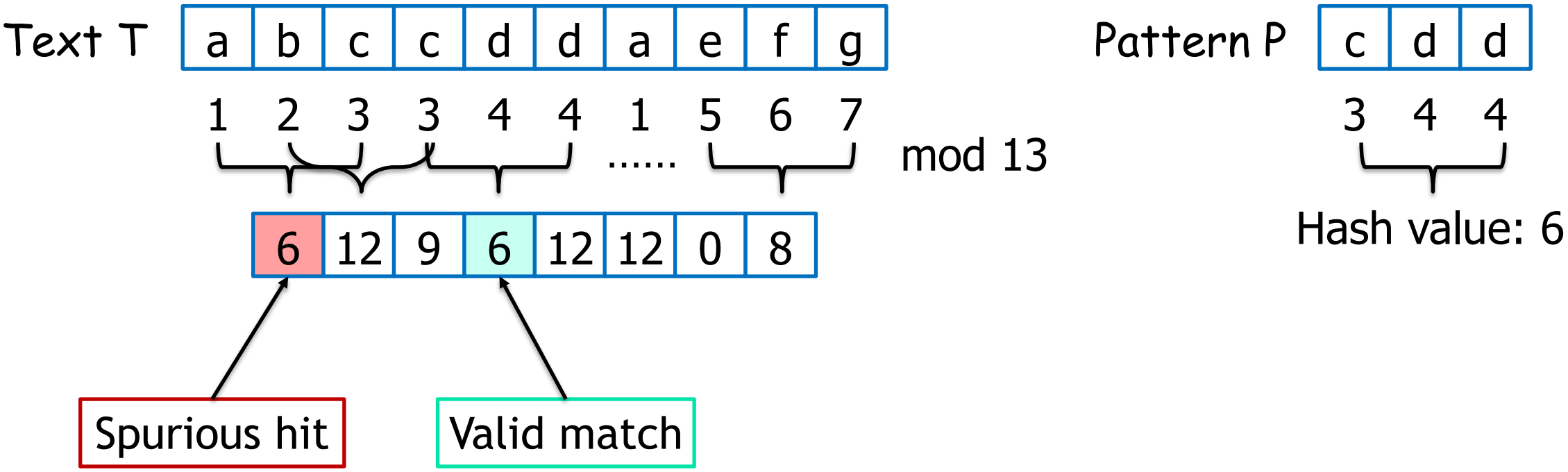
c	d	d
3	4	4

- n = 10, m=3.
- Let d be the number of characters in the input set. d = 10
- Hash value for patter p = $((3*10^2) + (4*10^1) + (4*10^0)) \bmod 13$

$= 344 \bmod 13$

$= 6$

Rabin-Karp Example



Rabin-Karp Pseudo-Code

```
RABIN-KARP(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
    P = (dp + P[i]) mod q
    t_0 = (dt_0 + T[i]) mod q
for s = 0 to n - m
    if p == t_s
        if P[1.....m] = T[s + 1..... s + m]
            print "pattern found at position" s
    If s < n-m
        t_(s+1) = (d (t_s - t[s + 1]h) + t[s + m + 1]) mod q
```

Rabin-Karp Complexity

- If a sufficiently large prime number is used for the hash function, the hashed values of two different patterns will usually be distinct.
- If this is the case, matching takes $O(n)$ time, where n is the number of characters in the larger body of text.
- If the prime number is small, and the algorithm need to verify every valid shift, then worst case time complexity is $O(mn)$.

Exercise

- Show all the spurious hits of the Rabin-Karp string matching algorithm encounter in the text T= "3141592653589" when looking for all occurrences of the pattern P = "15", working modulo q = 11 and over the alphabet $\Sigma = \{0, 1, 2, \ldots, 9\}$

Exercise

- Given a length-n string consisting of letters 'A', 'C', 'G', and 'T' representing DNA sequence. Design a O(n)-time algorithm to find all the 4-letter-long substring that occur more than once in the given string. For example, "ACGA" appears twice in "ACGATGACGA". Overlapping occurrences are allowed, e.g., "ACAC" appears twice in "ACACAC". Explain your algorithm design and analyze the time complexity.

Knuth-Morris-Pratt Algorithm

Knuth-Morris-Pratt (KMP) Algorithm

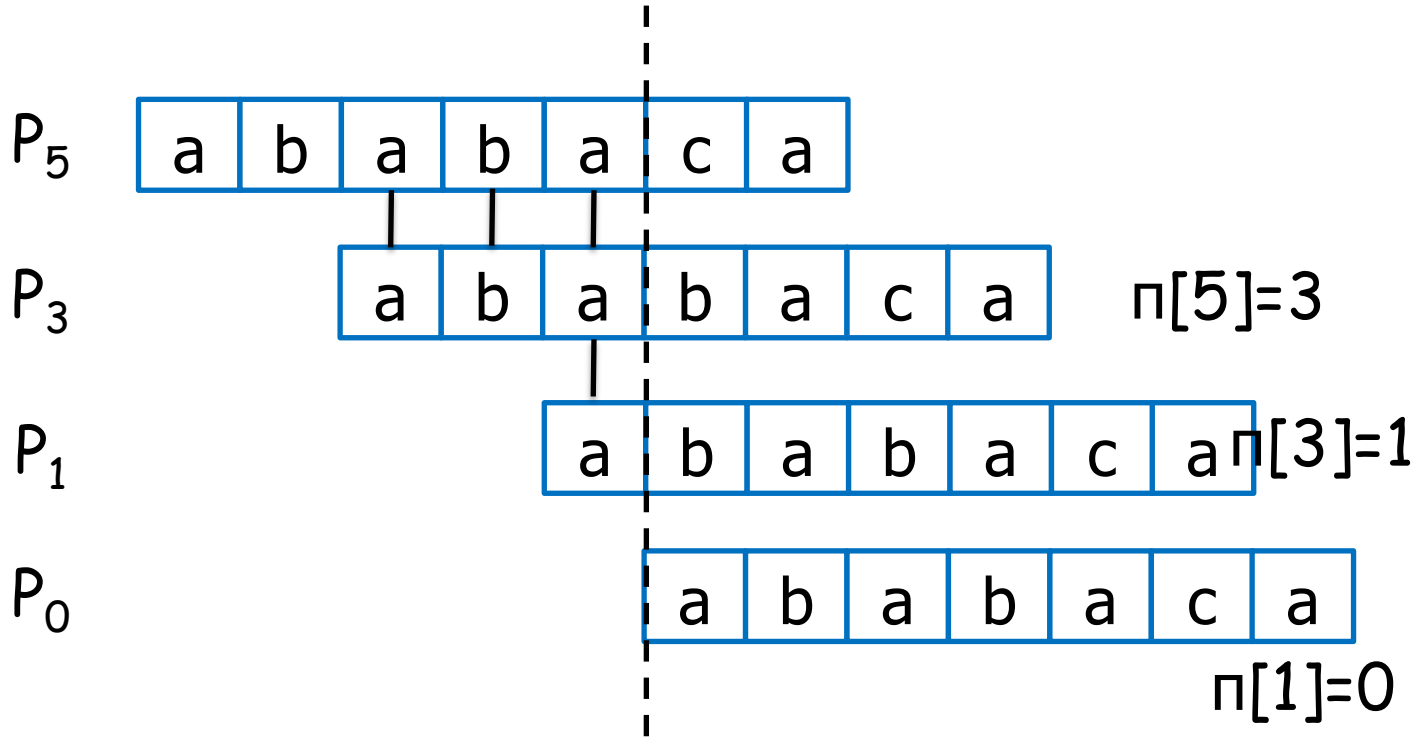
- The Knuth-Morris-Pratt (KMP) Algorithm differs from the naïve algorithm by keeping track of information gained from previous comparisons.

- A prefix function (π) is computed that indicates how much of the last comparison can be reused if it fails.

Prefix function

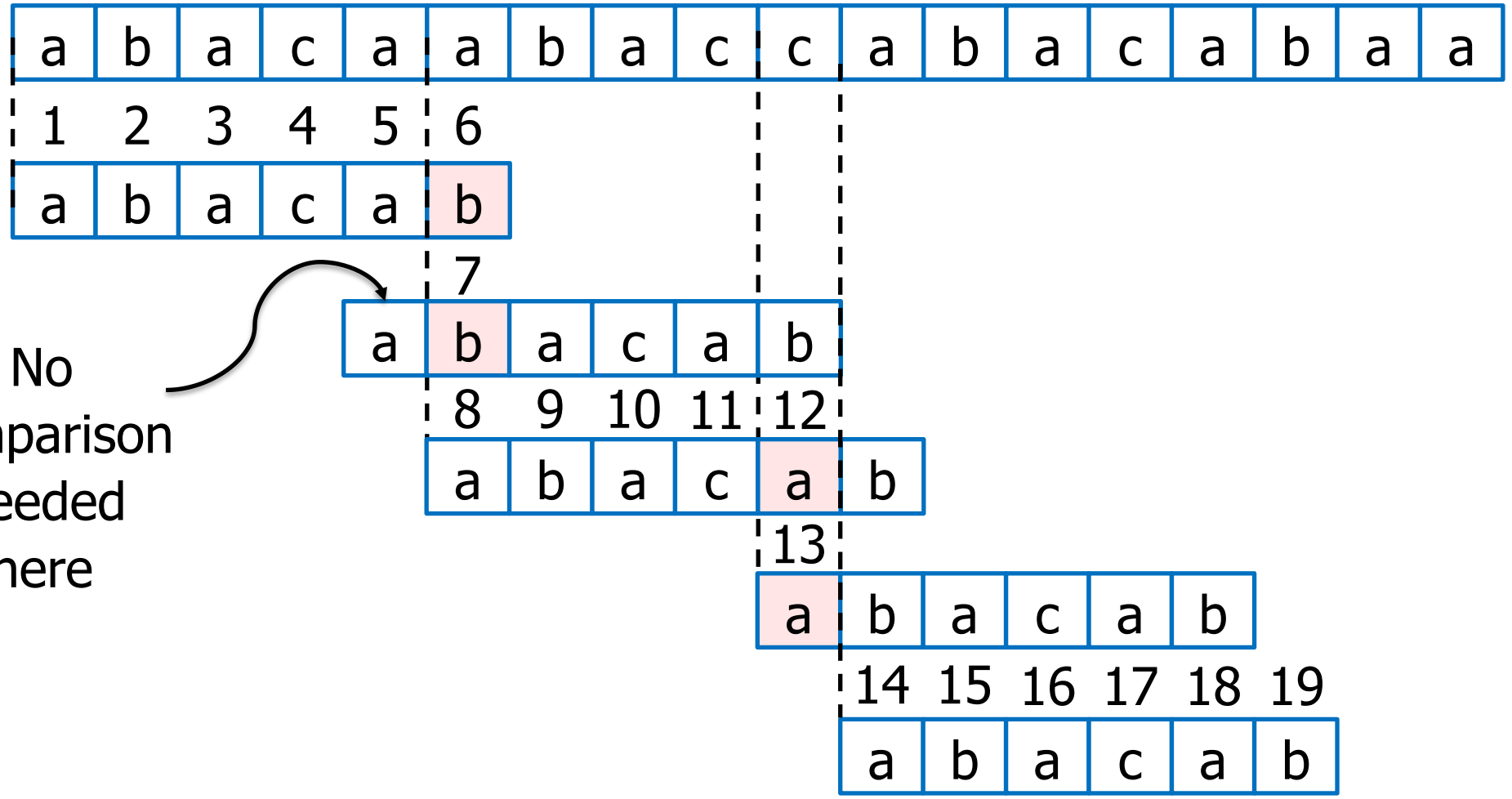
- prefix function (π) is defined to be the longest prefix of the patter P[1,…,i] that is also a suffix of P[2,…,i]
- Example:

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
π[i]	0	0	1	2	3	0	1



KMP Example

i	1	2	3	4	5	6
P[i]	a	b	a	c	a	b
π[i]	0	0	1	0	1	2



KMP pseudo code

O(m)

```
COMPUTE-PREFIX-FUNCTION(P)
 1  m = P.length
 2  let π[1..m] be a new array
 3  π[1] = 0
 4  k = 0
 5  for q = 2 to m
 6      while k > 0 and P[k + 1] ≠ P[q]
 7          k = π[k]
 8      if P[k + 1] == P[q]
 9          k = k + 1
10      π[q] = k
11  return π
```

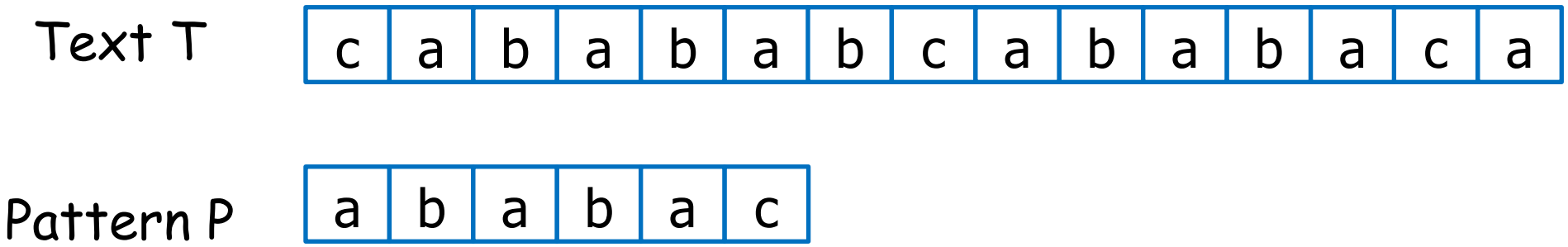
Executes O(m) times altogether

O(n)

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

Time complexity: O(n)

KMP Exercise



- Prefix function?
- Show KMP algorithm process.

Exercise

- A string is a palindrome when it reads the same backward as forward (e.g., "aba" is a palindrome while "abc" is not). Design an efficient algorithm to find the Longest Palindrome Prefix (LPP) of a string (e.g., "ana" is the LPP of the word "analysis") and analyse the complexity of your algorithm.
  - A palindrome reads the same backward as forward, so we can reverse the string and compare it to check if the string is a palindrome
  - As we want to find the longest palindrome prefix in this task(e.g., AABBB), the longest palindrome prefix becomes suffix when the string is reversed (e.g., BBBAA).
  - We can create a new string that concatenate these two strings and with "&" to separate them (e.g., AABBB&BBBAA). Now, the task becomes finding the longest prefix of the new string that is also a suffix of the same string.
  - This task can be solved by constructing a prefix table using the KMP algorithm in O(n).

Exercises

- Implement String Matching using Python
- Implement Rabin-Karp Algorithm using Python
- Implement Knuth-Morris-Pratt Algorithm using Python