IT2070 – Data Structures and Algorithms

Lecture 09

String Matching Algorithms

Contents

- String Matching
- The naïve string matching algorithm
- The Rabin-Karp Algorithm
- Finite automata

String Matching

- The problem of finding occurrence(s) of a pattern string within another string or body of text.
- There are many different algorithms for efficient searching.
- String matching is a very important subject in the wider domain of text processing.

Applications

- In string matching problems, it is required to find the occurrences of a pattern in a text.

- <u>**Applications**</u>

    - Text processing
    - Text-editing e.g.*Find and Change* in word
    - Computer security (virus detection, password checking)
    - DNA sequence analysis.
    - Data communications (header analysis)

SLIIT Discover Your Future

Example

Text *T*

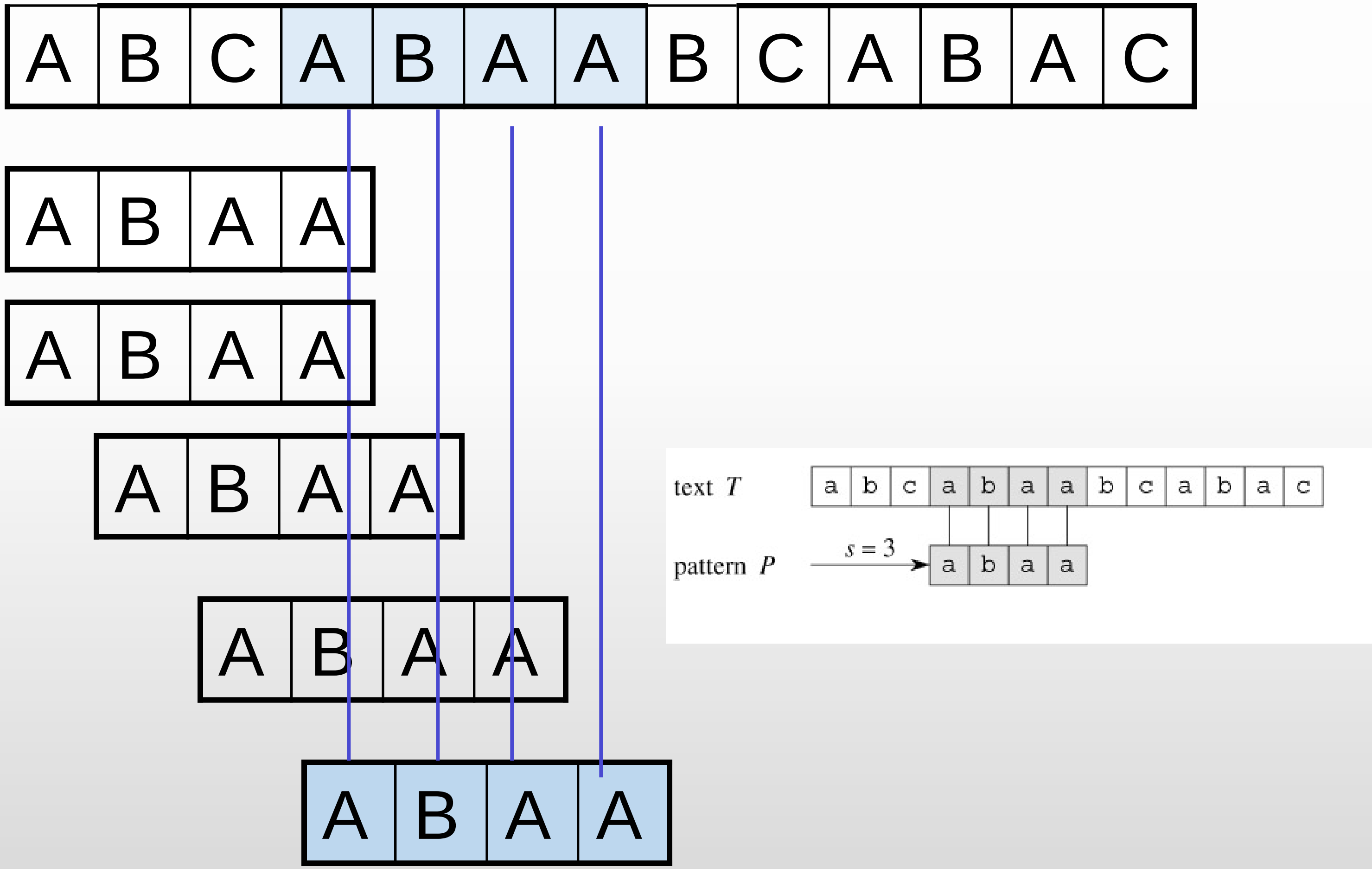
Pattern *P*

Shift 0

Shift 1

Shift 2

Shift 3



String Matching problem

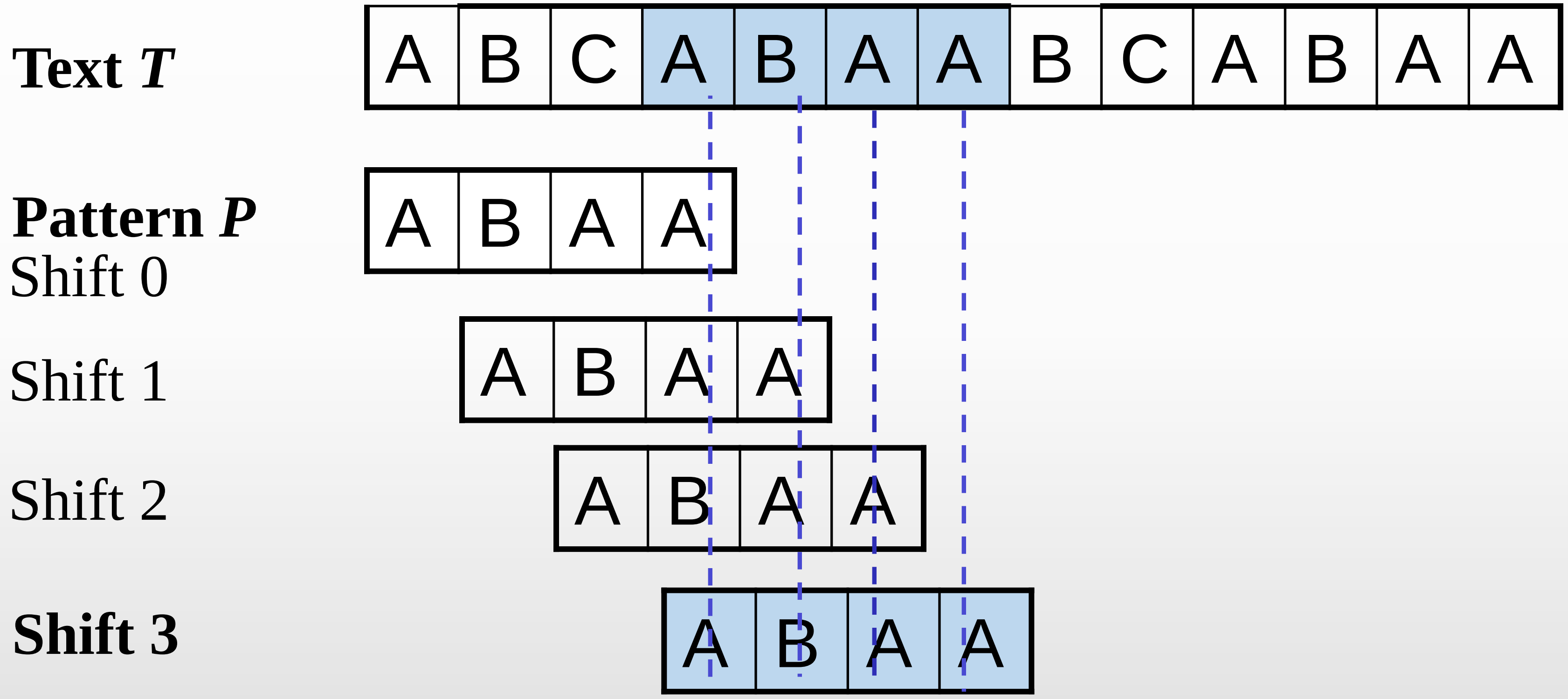
- We formulize the ***String Matching problem*** as follows.
- We assume that the text is an array **T[1..*n*]** of length ***n*** and the pattern is an array **P[1..*m*]** of length ***m***.
- We further assume that the elements of P and T are characters drawn from a finite alphabet $\sum$.
- For example we may have $\sum$={0,1} or $\sum$={a,b,..z}.
- The character arrays P and T are often called ***Strings*** of characters.

String Matching problem(contd.)

- 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***)

$$0 \leq \boldsymbol{s} \leq n\text{-}m \text{ and } T[s+1..\, s+m] = P[1..m].$$

If P occurs with shift ***s*** in ***T***, then we call s a ***valid shift***; otherwise we call ***s*** an ***invalid shift***.



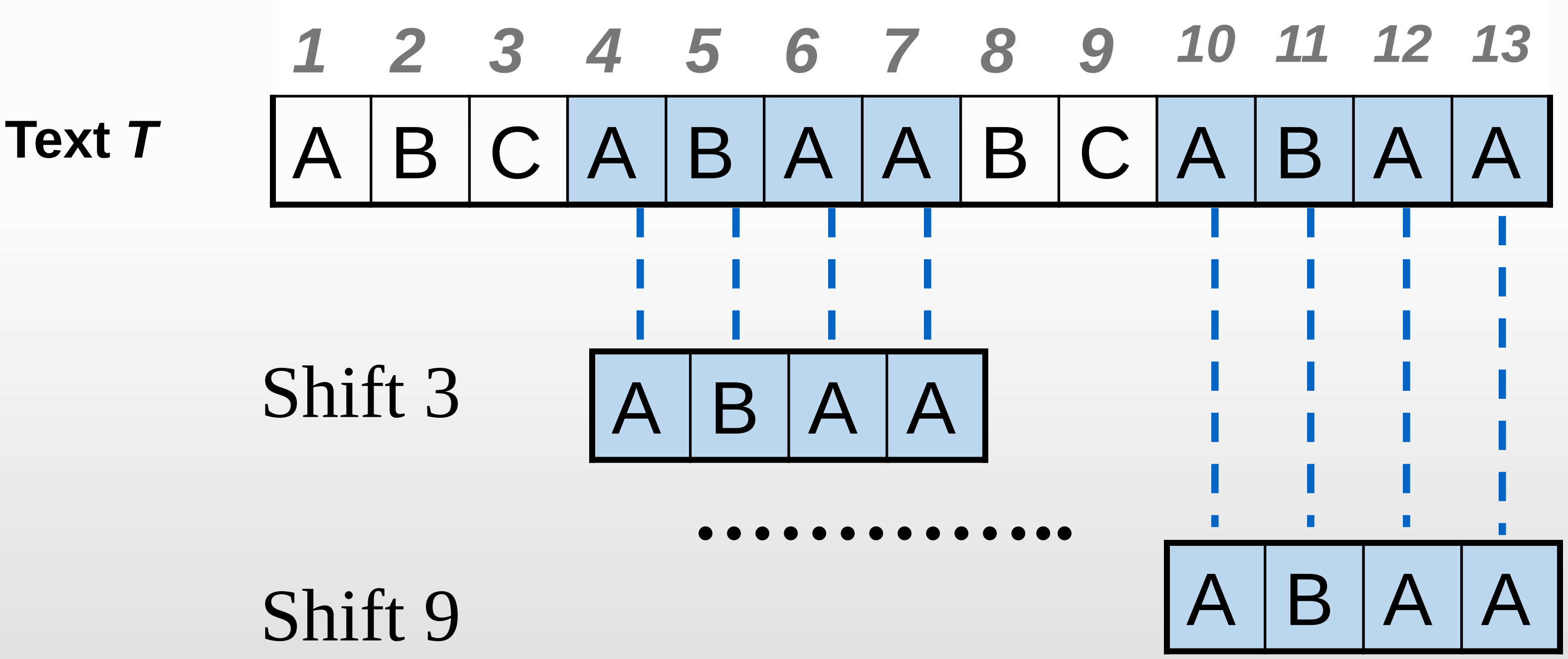
Here, n = 13 m = 4 s = 3

S = 3 is a valid shift because

$0 \leq \mathbf{3} \leq 13-4$ and $T[3+1..\,3+4] = P[1..4]$.

String Matching problem(contd.)

The string-matching problem is the problem of finding all valid shifts with which a given pattern *P* occurs in a given text *T*.

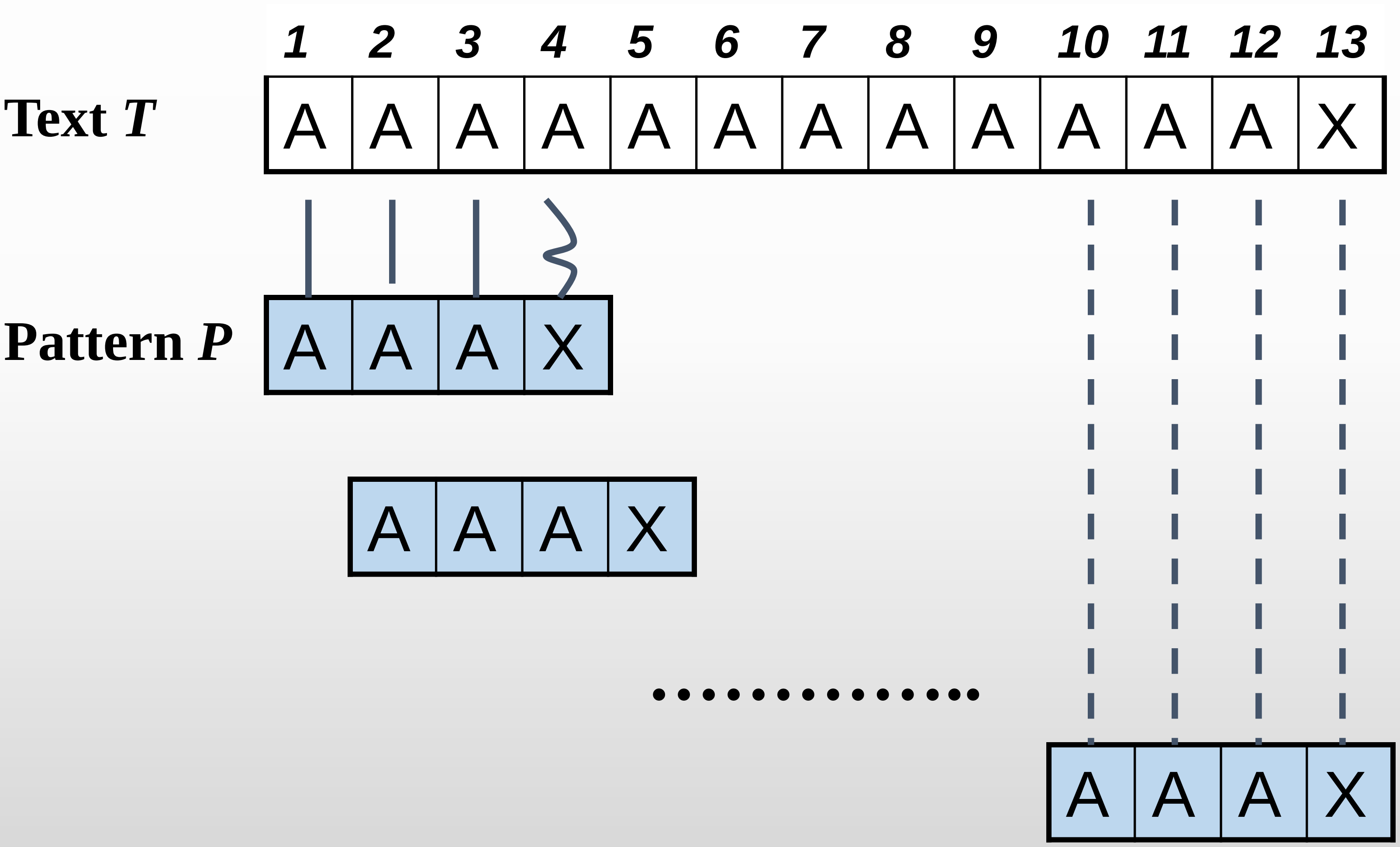


The naive string-matching algorithm

- The naïve 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.

```
Naïve-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
```

When does worst case happen?



Worst case Analysis

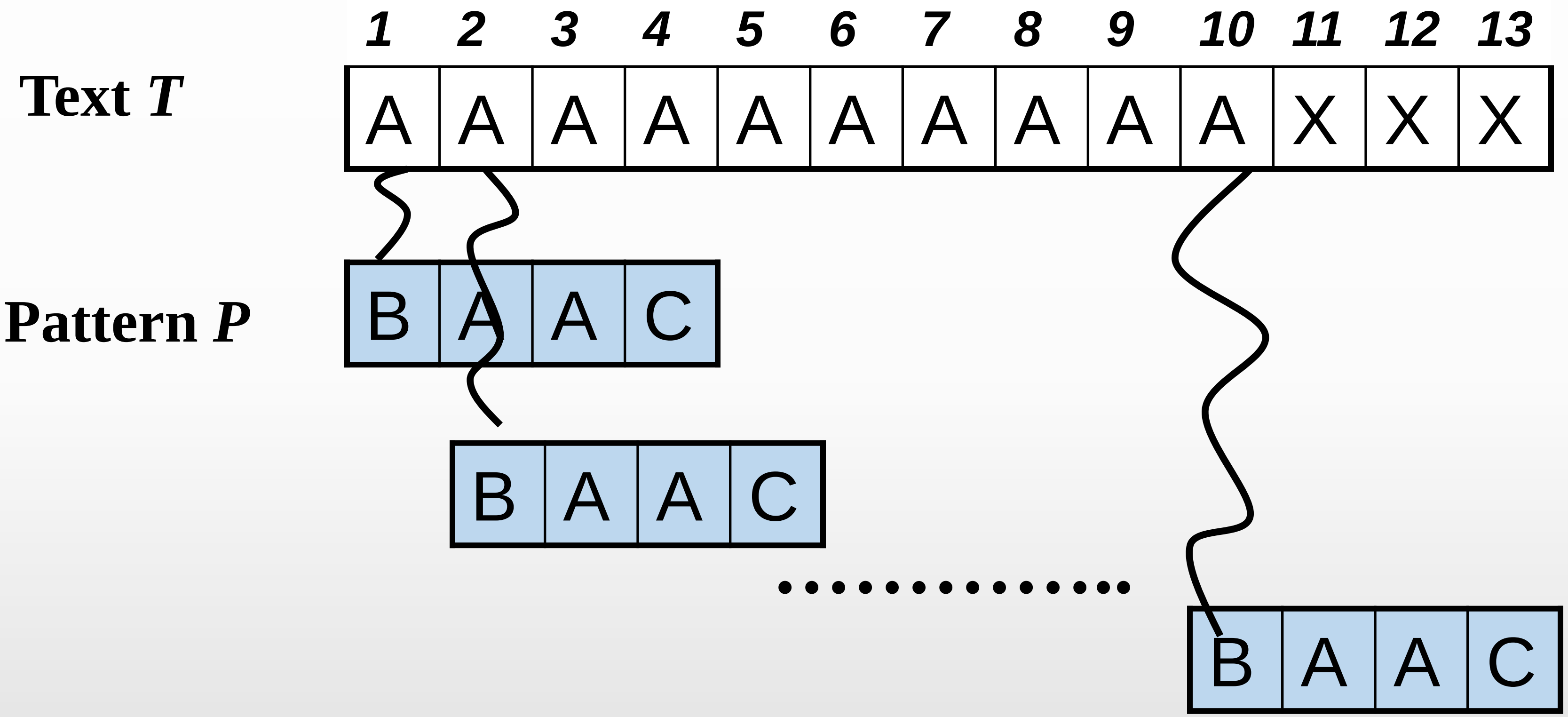
- This algorithm takes O((n-m+1)m) in the worst case
- There are at most n-m +1 shifts
- m is to compare each string after shifting
- Therefore worst case takes O((n-m+1)m)
- E.g. In above example

  O((13-4+1) 4) comparisons

Shiftings

Comparing for each shift

Best case analysis



No of shifts (13-4+1)

No of comparisons O((n-m+1) 1)

The Rabin-Karp Algorithm

Professor.Richard Karp **Harvard University**

Professor Michel Rabin Harvard University

Invented by Professor Richard Karp an Professor Michel Rabin in 1984.

The Rabin-Karp Algorithm

Given Text

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

Find the occurrence of pattern

3	1	4	1	5

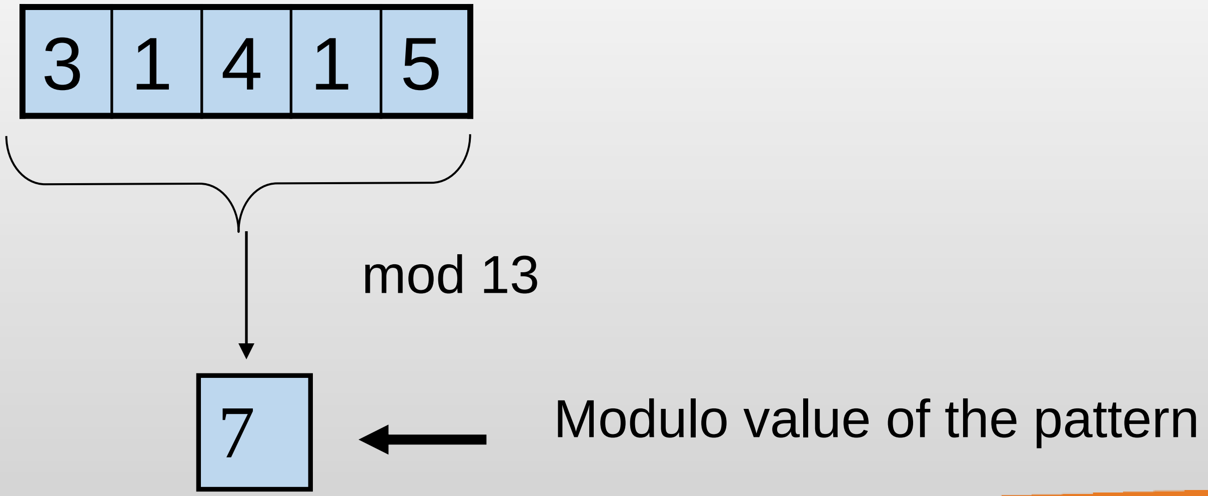
Using modulo 13

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

3	1	4	1	5

Method

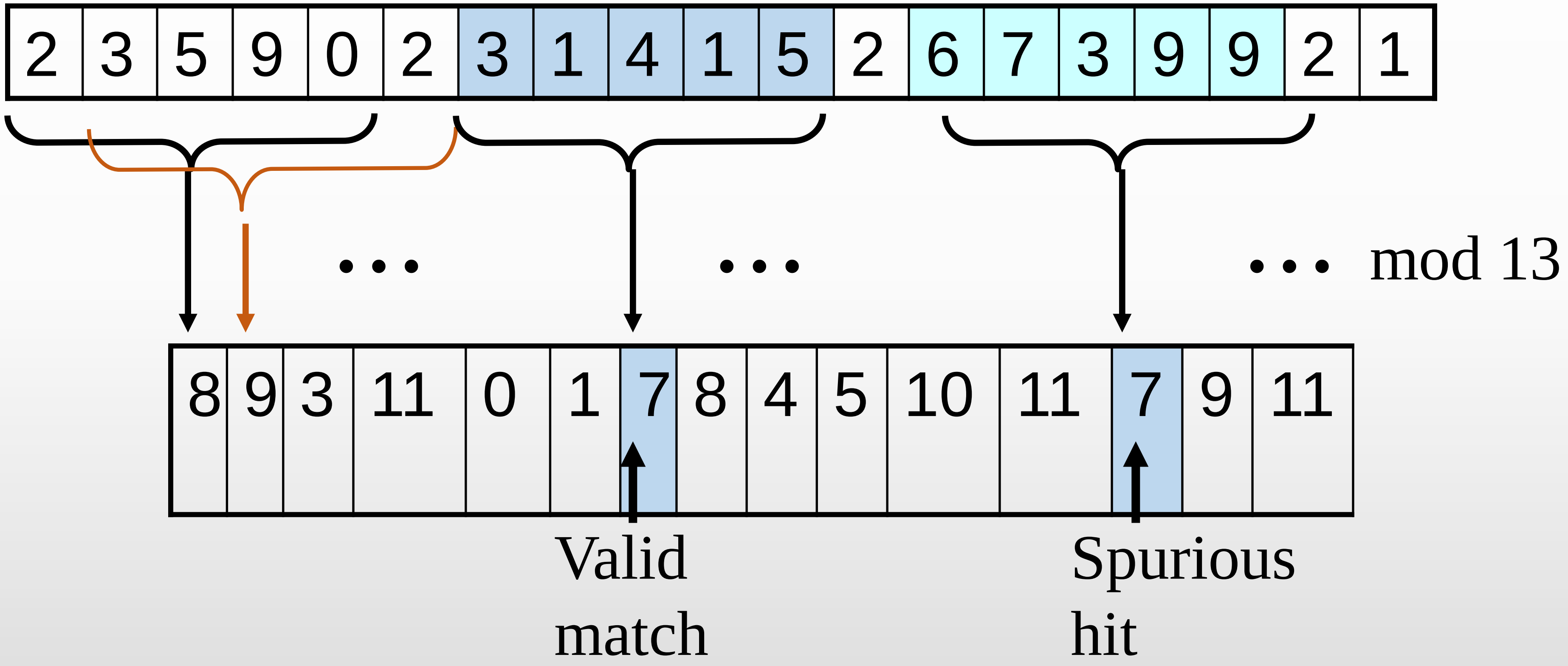
- Take the window size of the pattern
- Start from the beginning of the text taking windows
- Calculate the modulo value of that window
- Check it with the modulo value of the pattern



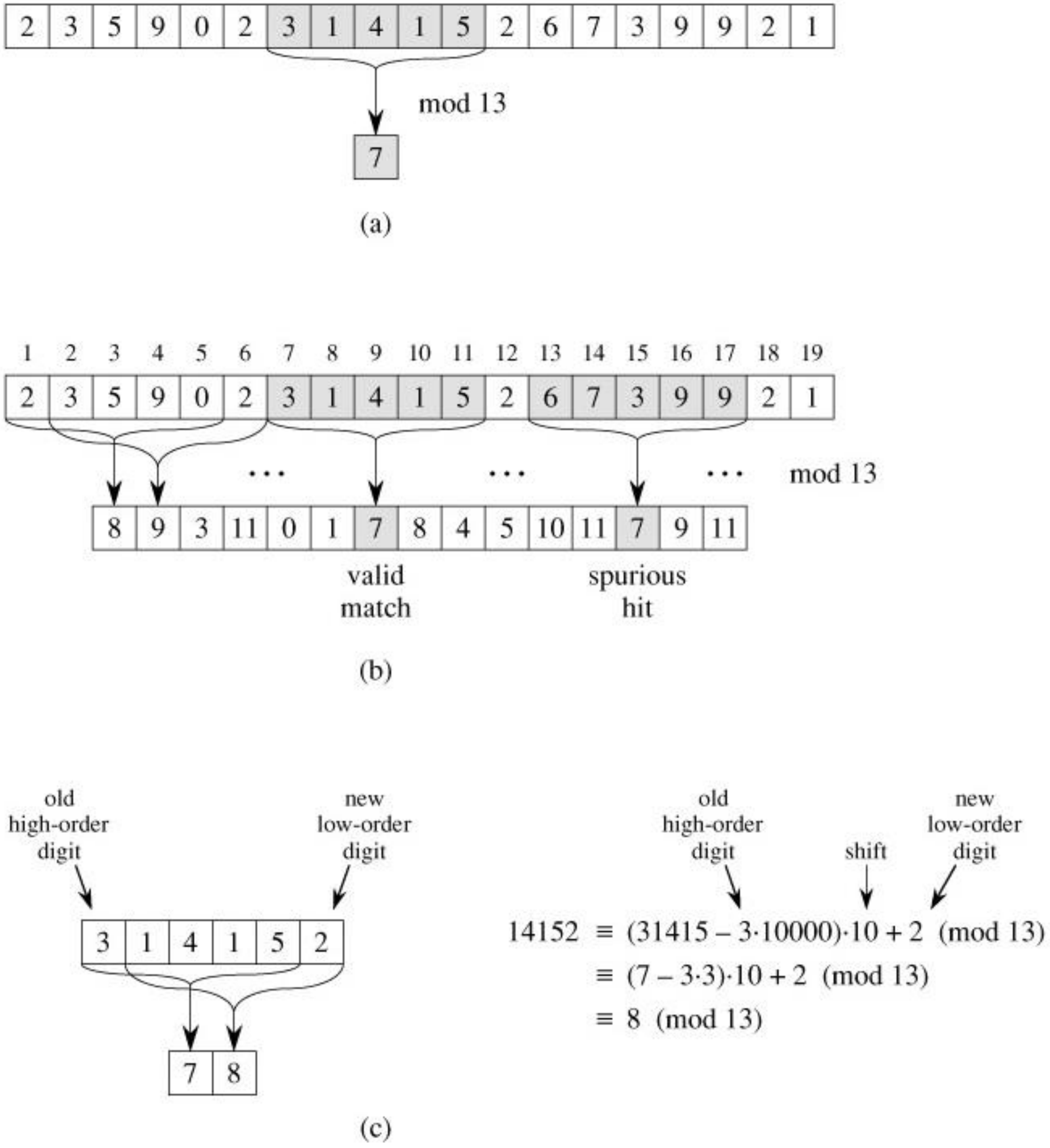
The Rabin-Karp Algorithm(contd.)

- Let us assume that the input alphabet is {0,1,2,…,9},so that each character is a decimal digit.
- We can then view a string of k consecutive characters as representing a length-k decimal number.
- The character string 31425 thus corresponds to the decimal number 31,425.

The Rabin-Karp Algorithm



All the hits should be checked further.


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
new low-order digit
3 1 4 1 5 2
7 8
old high-order digit
shift
new low-order digit
14152 ≡ (31415 – 3·10000)·10 + 2 (mod 13)
≡ (7 – 3·3)·10 + 2 (mod 13)
≡ 8 (mod 13)
(c)

Analysis of Rabin-Karp

- Comparing only the modulo value does not guarantee that the exact pattern is found.
- On the other hand, if modulo values are not matched, then we definitely know that it is not the pattern.
- All the hits must be tested further to see if the hit is a valid shift or just a spurious hit.
- This testing can be done by explicitly checking the condition $P[1..m] = T[s+1..s+m]$
- If q is large enough, then we can hope that spurious hits occur infrequently enough that the cost of the extra checking is low.

Worst case & Best case

- If all the hits are spurious hits then we have to check each of those.
- Therefore the worst case occurs when all the hits are spurious hits

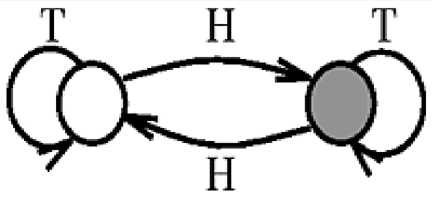
- If all the hits are neither spurious hits nor valid hits we don’t have to check each of those.
- Therefore the best case occurs when no hits occurred

String matching with finite automata

- Many string-matching algorithms build a finite automaton that scans the text string *T* for all occurrences of the pattern *P.*

- This section presents a method for building such an automaton.

- These string-matching automata are very efficient: they examine each text character *exactly once,* taking constant time per text character.

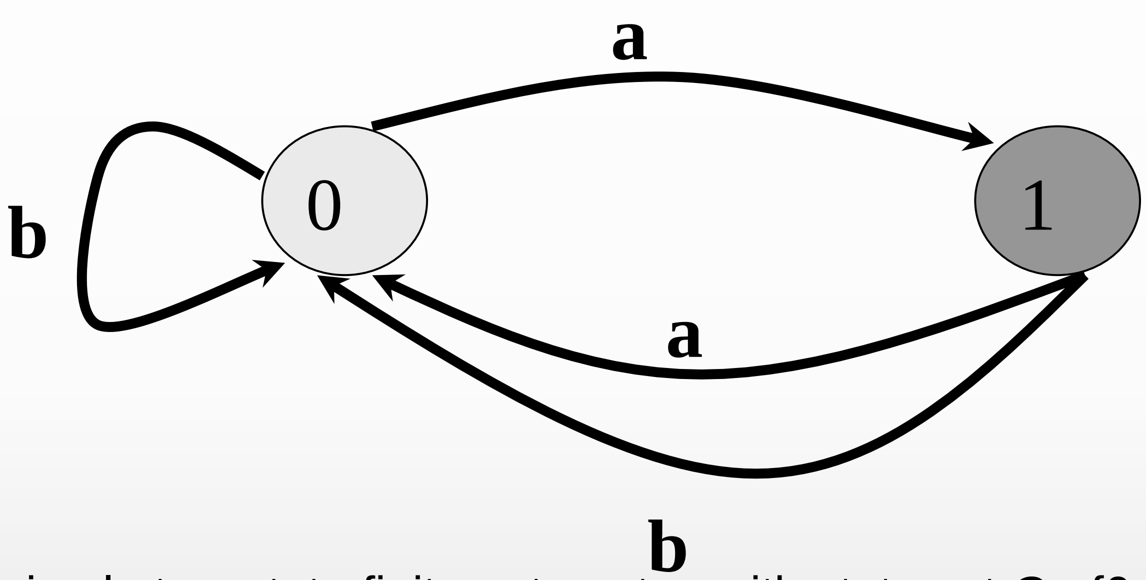


Definition of a finite automaton

A ***finite automaton M*** is a 5-tuple $(Q, q_o, A, \Sigma, \delta)$, where

- Q is a finite set of ***states***,
- $q \in Q$ is the ***start state***,
- $A \subseteq Q$ is a distinguished set of ***accepting states***,
- Σ is a finite ***input alphabet,***
- δ is a function from $Q \times \Sigma$ into Q,, called the ***transition function*** *of M.*

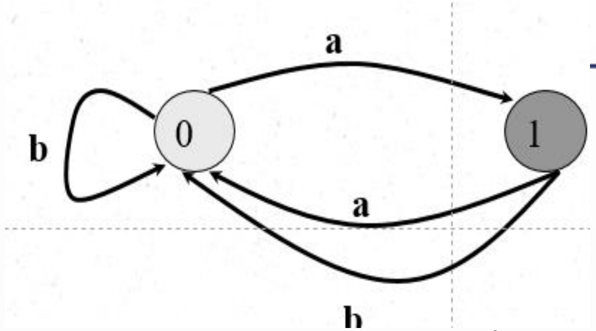
Example



state	input a	input b
0	1	0
1	0	0

A simple two-state finite automaton with state set ***Q***= {0,1},
start state q_0 = 0,
Accepting state ***A*** = 1
input alphabet ∑ = {a,b}
A tabular representation of the transition function δ

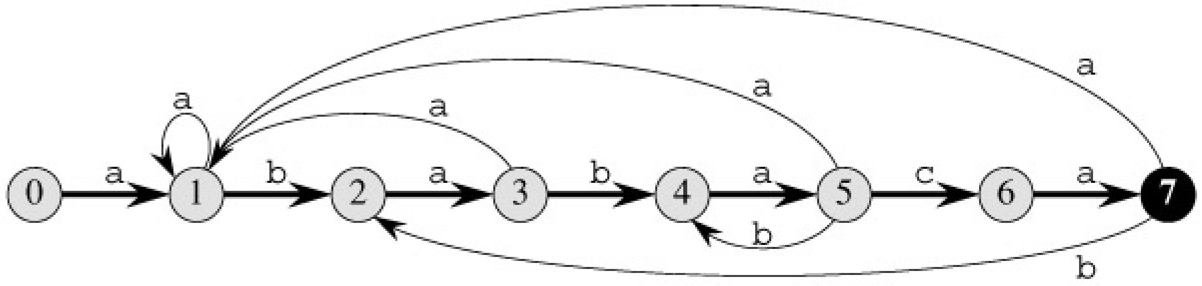
Finite automata(contd.)



- Directed edges represent transitions.
- For example, the edge from state 1 to state 0 labeled **b** indicates δ (1, b) = 0.
- This automaton accepts those strings that end in an odd number of ***a***'s.
- For example, the sequence of states this automaton enters for input **abaaa** (including the start state) is (0, 1, 0, 1, 0, 1) , and so it accepts this input.
- For input **abbaa**, the sequence of states is (0, 1,0,0, 1,0), and so it rejects this input.

Example. Accepts all strings ending in the string ababaca.

- *(a)* A state-transition diagram for the string-matching automaton that accepts all strings ending in the string ababaca. State 0 is the start state, and state 7 (shown blackened) is the only accepting state. A directed edge from state i to state j labeled a represents $\delta(i, a) = j$. The right-going edges forming the "spine" of the automaton, shown heavy in the figure, correspond to successful matches between pattern and input characters. The left-going edges correspond to failing matches. Some edges corresponding to failing matches are not shown; by convention, if a state i has no outgoing edge labeled a for some $a \in \Sigma$, then $\delta(i, a) = 0$.
- *(b)* The corresponding transition function δ, and the pattern string P = ababaca. The entries corresponding to successful matches between pattern and input characters are shown shaded.
- *(c)* The operation of the automaton on the text T = abababacaba. Under each text character $T[i]$ is given the state $\varphi(T_i)$ the automaton is in after processing the prefix T_i. One occurrence of the pattern is found, ending in position 9.



(a)

	input			
state	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

(b)

i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

(c)

Summary

- String Matching
- The naïve string matching algorithm
- The Rabin-Karp Algorithm
- Finite automata