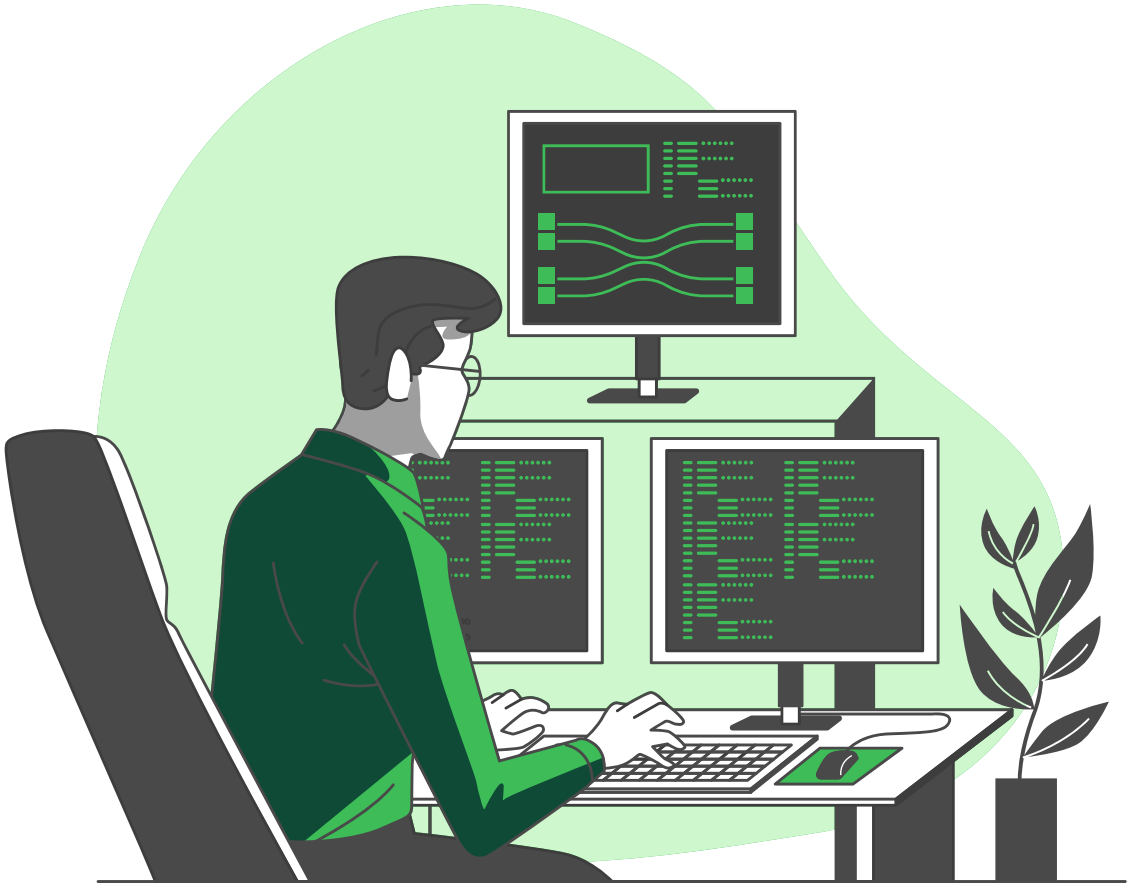


String Pattern Matching

CS 251 - Data Structures and Algorithms

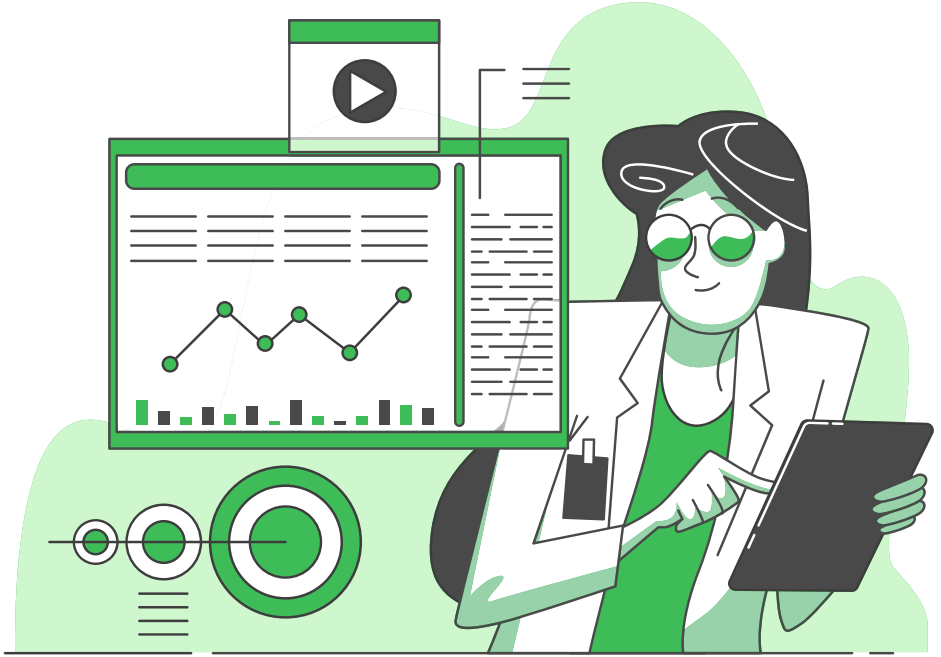
Note:
Slides complement the discussion in class

Table of Contents

01 **Definitions**
Strings and substrings

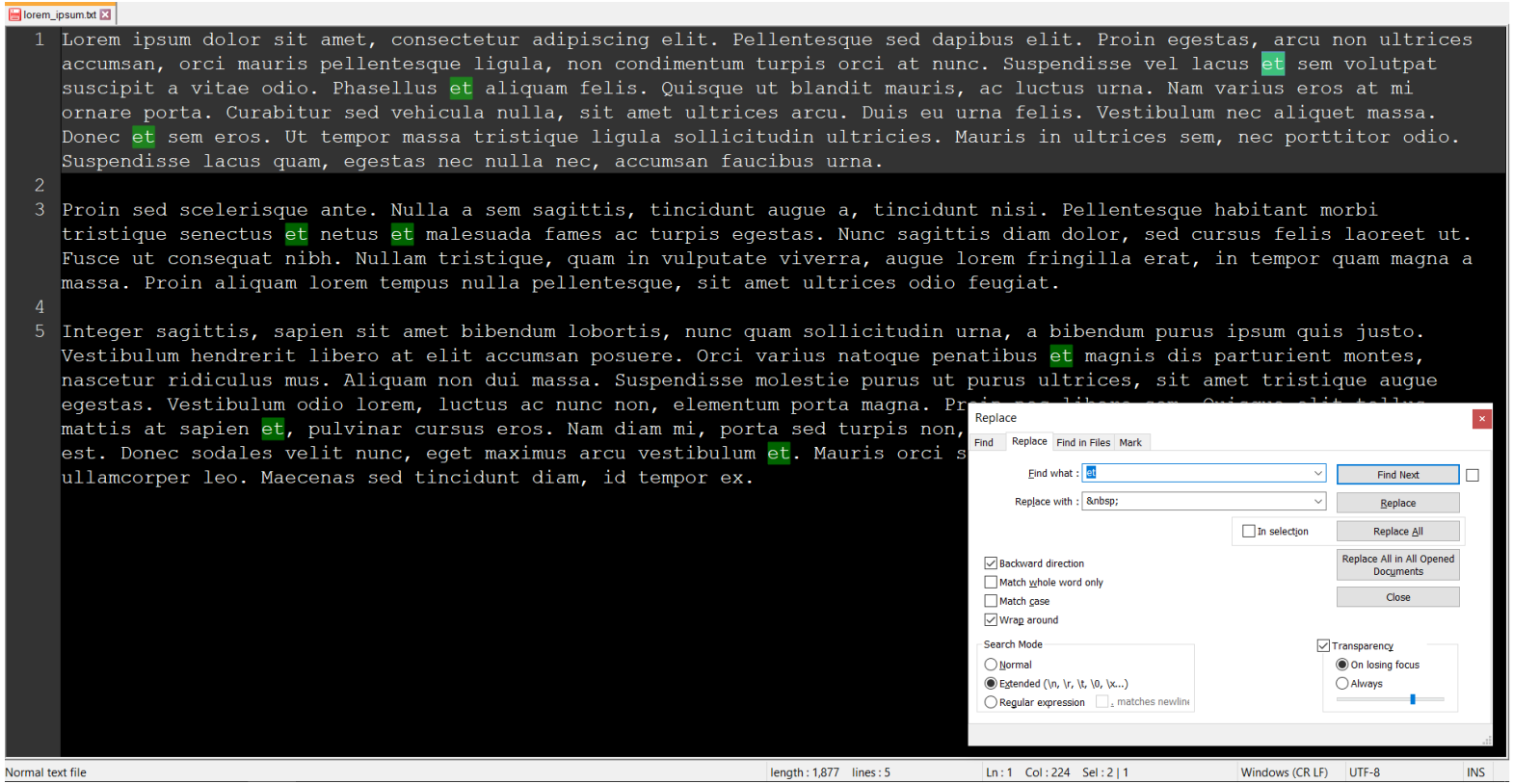
02 **Brute-Force Algorithm**
Let's be brute!

03 **Boyer-Moore Algorithm**
Heuristics to the rescue

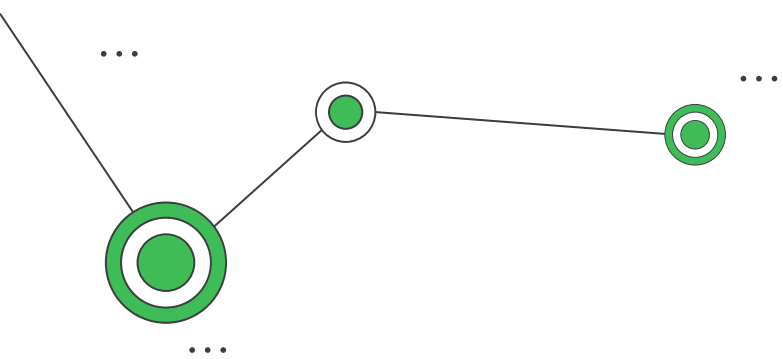


01 Definitions

Strings and Substrings



Given strings T (text) and P (pattern), find a substring of T that is equal to P.



Strings and Alphabets

A **string** is a sequence of characters.

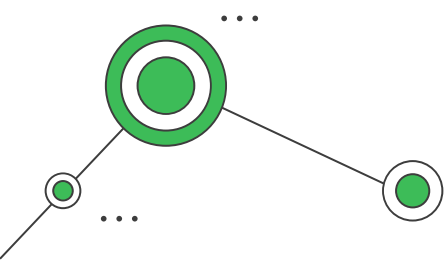
Examples:

- HTML code
- A Java program
- DNA sequence

An **alphabet** Σ is the set of possible characters for a family of strings.

Examples:

- English alphabet (26 characters)
- ASCII (7 bits per character)
- Unicode (16 bits per character)
- {A, C, G, T}



Substring, Prefix, Suffix

Let S be a string of size m. Let $i, j \in [0, m-1], i \leq j$:

Substring

$S.\text{substr}(i, j) \rightarrow S[i, j]$

Subsequence of S consisting of characters with ranks between i and j.

Prefix

$S.\text{prefix}(i) \rightarrow S[0, i]$

Substring of S between the character at index 0 and the character at index i.

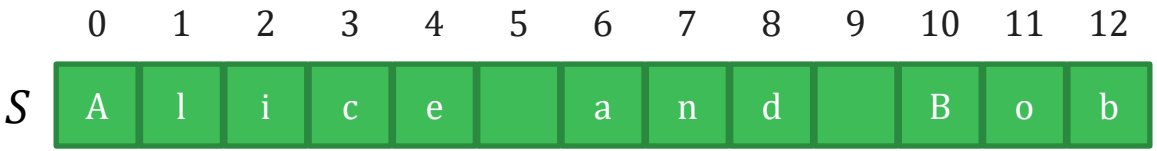
Suffix

$S.\text{suffix}(i) \rightarrow S[i, m-1]$

Substring of S between the character at index i and the last character at index $m-1$.

Substring, Prefix, Suffix

Example: $S =$ "Alice and Bob", $m = 13$:



Substring: $S[1, 7]$

"lice an"

Prefix: $S[0, 2]$

"Ali"

Suffix: $S[7, 12]$

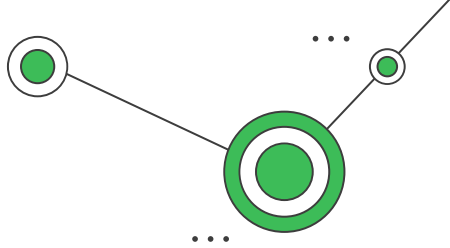
"nd Bob"

02

Brute-Force Algorithm

Let's be brute!

Fundamental Problem

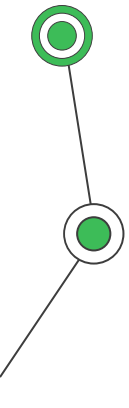


Given strings T (text) and P (pattern), find a substring of T that is equal to P.

Example:

$T = 10110110101001010100101010100101010010$

$P = 01101$



Fundamental Problem

Given strings T (text) and P (pattern), find a substring of T that is equal to P.

Example:

$T = 101101101010010101010010101010010101001010$

$P = 01101$

Fundamental Problem

Given strings T (text) and P (pattern), find a substring of T that is equal to P.

Example:

$T = 1011\boxed{01101}0100101010010101010010101001010$

$P = 01101$

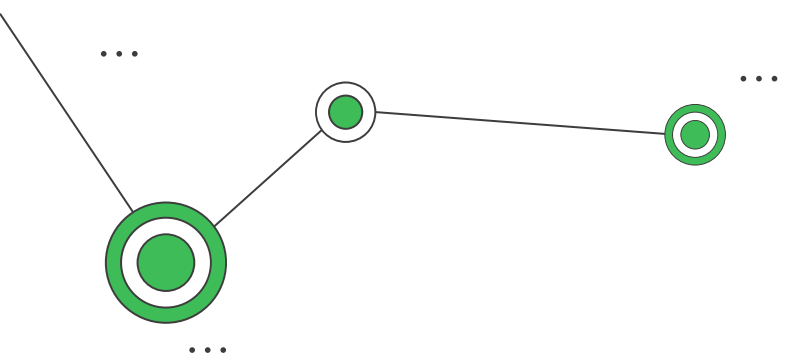
Brute-Force Algorithm

Input: Strings P of length m, and T of length n.

Output: The index in T of the first character in the match, or -1 if no match is found.

Idea: Compare the pattern P with the text T for each possible shift of P relative to T, until one of the following occur:

- A match is found.
- All placements of the pattern have been tried.



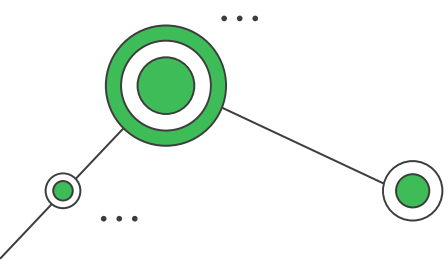
Brute-Force Algorithm

```
algorithm BruteForceMatch(T:string, P:string)

   let n be the length of T
   let m be the length of P

   for i from 0 to n-m do

      j ← 0

      while j < m and T[i+j] = P[j] do
         j ← j + 1
      end while

      if j = m then
         return i
      end if

   end for

   return -1
end algorithm
```

For loop runs $n - m + 1$ times: $O(n)$

While loop runs at most m times per for loop iteration.

Runtime: $O(nm)$



T = “a pattern matching algorithm”
P = “rithm”

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
a	**_**	**p**	**a**	**t**	**t**	**e**	**r**	**n**	**_**	**m**	**a**	**t**	**c**	**h**	**i**	**n**	**g**	**_**	**a**	**l**	**g**	**o**	**r**	**i**	**t**	**h**	**m**

T = “a pattern matching algorithm”
P = “rithm”

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
a	**_**	**p**	**a**	**t**	**t**	**e**	**r**	**n**	**_**	**m**	**a**	**t**	**c**	**h**	**i**	**n**	**g**	**_**	**a**	**l**	**g**	**o**	**r**	**i**	**t**	**h**	**m**
r	i	t	h	m																							
	r	i	t	h	m																						
		r	r	r	r	r	i	t	h	m																	
							r	i	t	h	m																
								r	l	t	h	m															
									r	r	r	r	r	r	r	r	r	r	r	r	r	r	i	t	h	m	
																							r	i	t	h	m

#compares between characters = 29

Brute-Force Algorithm

Can you imagine a worst-case scenario?

Situation:
P = ABBC
T is some string of As, Bs, and Cs

Here's a bad T:
T = ABBABBABBABBABBABBABBABBABBABBABBABBC

Brute-Force Algorithm

An even simpler but worse worst-case scenario?

Situation:
P = AAAAAH
T is some string of As and Hs

Here's a bad T:
T = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAH

03

Boyer-Moore Algorithm

Heuristics to the rescue

Heuristic (Math & CS)

*"A heuristic is a **technique** designed for **solving a problem** more **quickly** when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.*

*This is achieved by **trading** optimality, completeness, accuracy, or precision for **speed**.*

*In a way, it can be considered a **shortcut**."*

https://en.wikipedia.org/wiki/Heuristic_(computer_science)

Boyer–Moore Algorithm

Idea: Compare P with a substring of T moving backwards in P.

Character-jump heuristic: When a mismatch occurs at $T[i]$:

- If P contains $T[i]$, shift P to align the last occurrence of $T[i]$ in P with $T[i]$.
- Else, shift P to align $P[0]$ with $T[i+1]$.

Last-Occurrence Function

Boyer-Moore's algorithm preprocess the pattern P and the alphabet Σ to build the last-occurrence function L mapping Σ to integers, where $L(c)$ is defined as:

- The largest index i such that $P[i] = c$, or
- -1 otherwise (i.e., c is not in P)

The last-occurrence function can be represented by an array indexed by the numeric codes of the characters.

Example:
$\Sigma = \{a,b,c,d\}$
$P =$ abacab

c	a	b	c	d
$L(c)$	4	5	3	-1

Last-Occurrence Function

We can calculate the last-occurrence function in $O(m + s)$ where m is the length of P and s is the length of Σ.

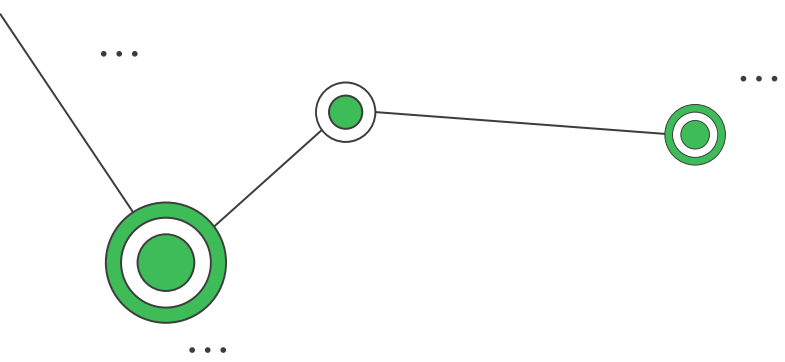
- Set all the values in the array to -1: $O(s)$
- Scan P in reverse and update values in the array for each new character: $O(m)$

$\Sigma = \{a,b,c,d\}$
$P = bbaccd$

$\boldsymbol{c}$	a	b	c	d
$\boldsymbol{L(c)}$	2	1	4	5

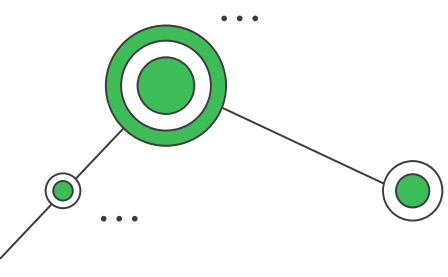
$\Sigma = \{a,b,c,d\}$
$P = dcba$

$\boldsymbol{c}$	a	b	c	d
$\boldsymbol{L(c)}$	3	2	1	0



Boyer–Moore Algorithm

```
algorithm BoyerMooreMatch(T:string, P:string, Σ:alphabet)

   let n be the length of T
   let m be the length of P

   L ← lastOccurence(P, Σ)

   i ← m - 1
   j ← m - 1

   repeat
      if T[i] = P[j] then
         if j = 0 then
            return i
         else
            i ← i - 1
            j ← j - 1
         end if
      else
         l ← L[T[i]]
         i ← i + m – min(j, 1 + l)
         j ← m – 1
      end if
   until i > n – 1

   return -1
end algorithm
```



T = “a pattern matching algorithm”
P = “rithm”

c	h	i	m	r	t	*
$L(c)$						

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
a	**_**	**p**	**a**	**t**	**t**	**e**	**r**	**n**	**_**	**m**	**a**	**t**	**c**	**h**	**i**	**n**	**g**	**_**	**a**	**l**	**g**	**o**	**r**	**i**	**t**	**h**	**m**

T = “a pattern matching algorithm”
P = “rithm”

c	h	i	m	r	t	*
$L(c)$	3	1	4	0	2	-1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
a	**_**	**p**	**a**	**t**	**t**	**e**	**r**	**n**	**_**	**m**	**a**	**t**	**c**	**h**	**i**	**n**	**g**	**_**	**a**	**l**	**g**	**o**	**r**	**i**	**t**	**h**	**m**
r	i	t	h	m																							
		r	i	t	h	m																					
							r	i	t	h	m																
												r	i	t	h	m											
																	r	i	t	h	m						
																						r	i	t	h	m	
																							r	**i**	**t**	**h**	**m**

#compares between characters = 11

Boyer-Moore Algorithm

Can you imagine a worst-case scenario?

Situation:

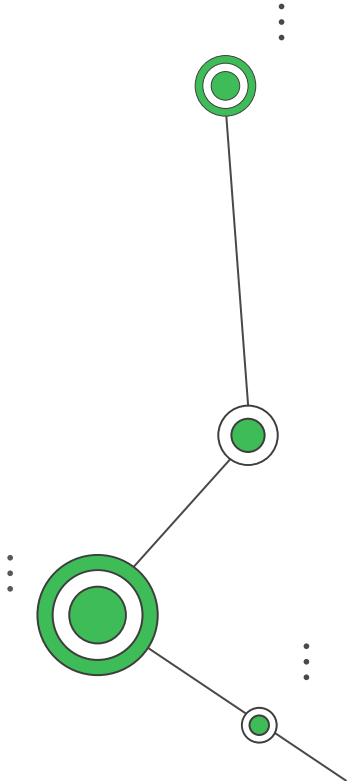
P = baaa

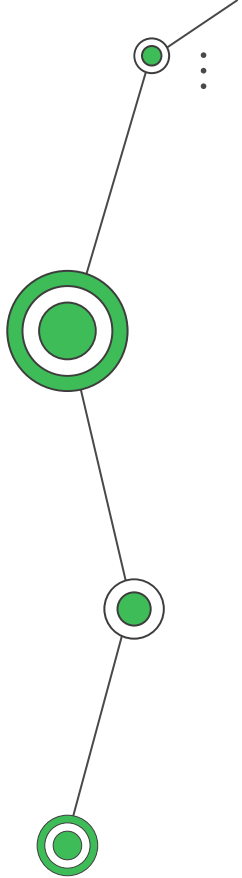
T = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Why is this bad?

“\0”

Do you have any questions?





CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik and illustrations by Stories