

# Note: Slides complement the discussion in class



#### **Definitions** Strings and substrings

## **Table of Contents**



#### Brute-Force Algorithm Let's be brute!



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







Strings and Substrings



4

#### 🔚 lorem\_ipsum.txt 🔀

Normal text file

1 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque sed dapibus elit. Proin egestas, arcu non ultrices accumsan, orci mauris pellentesque ligula, non condimentum turpis orci at nunc. Suspendisse vel lacus et sem volutpat suscipit a vitae odio. Phasellus et aliquam felis. Quisque ut blandit mauris, ac luctus urna. Nam varius eros at mi ornare porta. Curabitur sed vehicula nulla, sit amet ultrices arcu. Duis eu urna felis. Vestibulum nec aliquet massa. Donec et sem eros. Ut tempor massa tristique ligula sollicitudin ultricies. Mauris in ultrices sem, nec porttitor odio. Suspendisse lacus quam, egestas nec nulla nec, accumsan faucibus urna.

- 3 Proin sed scelerisque ante. Nulla a sem sagittis, tincidunt augue a, tincidunt nisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nunc sagittis diam dolor, sed cursus felis laoreet ut. Fusce ut consequat nibh. Nullam tristique, quam in vulputate viverra, augue lorem fringilla erat, in tempor quam magna a massa. Proin aliquam lorem tempus nulla pellentesque, sit amet ultrices odio feugiat.
- 5 Integer sagittis, sapien sit amet bibendum lobortis, nunc quam sollicitudin urna, a bibendum purus ipsum quis justo. Vestibulum hendrerit libero at elit accumsan posuere. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam non dui massa. Suspendisse molestie purus ut purus ultrices, sit amet tristique augue

egestas. Vestibulum odio lorem, luctus ac nunc non, elementum porta magna. Pr mattis at sapien et, pulvinar cursus eros. Nam diam mi, porta sed turpis non, est. Donec sodales velit nunc, eget maximus arcu vestibulum et. Mauris orci s ullamcorper leo. Maecenas sed tincidunt diam, id tempor ex.

m porta magna. Pr a sed turpis non	Replace				×
ot Mauris orci s	Find Replace Find in	n Files Mark			_
et. Mauris ordi s	Eind what :	et	~	Find Next	
	Replace with :		~	<u>R</u> eplace	
			In selection	Replace <u>A</u> ll	
	Backward direction	only		Replace All in All Opened Documents	
	Match gase			Close	
	Wrap around				
	Search Mode			[ransparency	
	○ <u>N</u> ormal			On losing focus	
	Extended (\n, \r, \/	t, \0, \x)		◯ Always	
	O Regular expression	matches newline			
length : 1,877 lines : 5	Ln:1 Col:224	Sel : 2   1	Windows (CR LF)	UTF-8	INS

#### 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**  $\Sigma$  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

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

Prefix

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

Suffix S. substr $(i, j) \rightarrow S[i, j]$  S. prefix $(i) \rightarrow S[0, i]$  S. suffix $(i) \rightarrow S[i, m-1]$ 

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



## UZ Brute-Force Algorithm

. . .

Let's be brute!

9

. . .

### **Fundamental Problem**



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

Example:

- P = 01101

### **Fundamental Problem**



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



### **Fundamental Problem**



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



- 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
                 For loop runs n - m + 1 times: O(n)
   end for
                 While loop runs at most m times per
   return -1
                 for loop iteration.
end algorithm
                 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
а	_	р	а	t	t	е	r	n	_	m	а	t	С	h	i	n	g	-	а	I	g	0	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
а	-	р	а	t	t	е	r	n	_	m	а	t	C	h	i	n	g	_	а	I	g	0	r	i	t	h	m
r	i	t	h	m																							
	r	i	t	h	m																						
		r	r	r	r	r	Î	t	h	m																	
							r	i	t	h	m																
								r		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
- ${\it T}$  is some string of As, Bs, and Cs

#### 

## **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:

## U3 Boyer-Moore Algorithm

. . .

Heuristics to the rescue

19

. . .

## 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**."



## **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  $\Sigma$  to build the last-occurrence function L mapping  $\Sigma$  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

С	а	b	с	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 sis the length of  $\Sigma$ .

- 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

С	а	b	с	d
L(c)	2	1	4	5

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

С	а	b	с	d
L(c)	3	2	1	0





## Boyer-Moore Algorithm

. . .



```
algorithm BoyerMooreMatch(T:string, P:string, \Sigma:alphabet)
   let n be the length of T
   let m be the length of P
   L \leftarrow lastOccurence(P, \Sigma)
   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
          1 \leftarrow L[T[i]]
         i \leftarrow i + m - min(j, 1 + 1)
          j ← m - 1
      end if
   until i > n - 1
   return -1
end algorithm
```

T = "a pattern matching algorithm"	
P = "rithm"	

С	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
а	_	р	а	t	t	е	r	n	_	m	а	t	С	h	i	n	g	-	а	I	g	0	r	i	t	h	m

T	= "a pattern i	matching	algorithm"
Р	="rithm"		

С	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
а	-	р	а	t	t	е	r	n	_	m	а	t	C	h	i	n	g	-	а	I	g	0	r	i	t	h	m
r	i	t	h	m																							
		r	Î	t	h	m																					
							r	i	t	h	m																
												r	i	t	h	m											
																	r	i	t	h	m						
																						r	Î	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

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