

4

String Matching – What's behind Ctrl+F?

3 March 2021

Sebastian Wild

4 String Matching

- 4.1 Introduction
- 4.2 Brute Force
- 4.3 String Matching with Finite Automata
- 4.4 The Knuth-Morris-Pratt algorithm
- 4.5 Beyond Optimal? The Boyer-Moore Algorithm
- 4.6 The Rabin-Karp Algorithm

4.1 Introduction

Ubiquitous strings

string = sequence of characters

- ▶ universal data type for ... everything!
 - ▶ natural language texts
 - ▶ programs (source code)
 - ▶ websites
 - ▶ XML documents
 - ▶ DNA sequences
 - ▶ bitstrings
 - ▶ ... a computer's memory ~→ ultimately any data is a string

~→ many different tasks and algorithms

Ubiquitous strings

string = sequence of characters

- ▶ universal data type for ... everything!
 - ▶ natural language texts
 - ▶ programs (source code)
 - ▶ websites
 - ▶ XML documents
 - ▶ DNA sequences
 - ▶ bitstrings
 - ▶ ... a computer's memory ~→ ultimately any data is a string

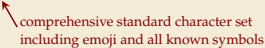
~→ many different tasks and algorithms

- ▶ This unit: finding (exact) **occurrences of a pattern** text.
 - ▶ Ctrl+F
 - ▶ grep
 - ▶ computer forensics (e. g. find signature of file on disk)
 - ▶ virus scanner
- ▶ basis for many advanced applications

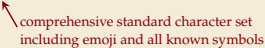
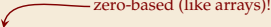
Notations

- ▶ *alphabet* Σ : finite set of allowed **characters**; $\sigma = |\Sigma|$ “a string over alphabet Σ ”
 - ▶ letters (Latin, Greek, Arabic, Cyrillic, Asian scripts, ...)
 - ▶ “what you can type on a keyboard”, Unicode characters
 - ▶ $\{0, 1\}$; nucleotides $\{A, C, G, T\}$; ...
- comprehensive standard character set
including emoji and all known symbols

Notations

- ▶ *alphabet* Σ : finite set of allowed **characters**; $|\Sigma|$ “a string over alphabet Σ ”
 - ▶ letters (Latin, Greek, Arabic, Cyrillic, Asian scripts, ...)
 - ▶ “what you can type on a keyboard”, Unicode characters
 - ▶ $\{0, 1\}$; nucleotides $\{A, C, G, T\}$; ...

- ▶ $\Sigma^n = \Sigma \times \dots \times \Sigma$: strings of **length** $n \in \mathbb{N}_0$ (n -tuples)
- ▶ $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$: set of **all** (finite) strings over Σ
- ▶ $\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$: set of **all** (finite) **nonempty** strings over Σ
- ▶ $\varepsilon \in \Sigma^0$: the *empty* string (same for all alphabets)

Notations

- ▶ *alphabet* Σ : finite set of allowed **characters**; $\sigma = |\Sigma|$ “a string over alphabet Σ ”
 - ▶ letters (Latin, Greek, Arabic, Cyrillic, Asian scripts, ...)
 - ▶ “what you can type on a keyboard”, Unicode characters
 - ▶ $\{0, 1\}$; nucleotides $\{A, C, G, T\}$; ...

- ▶ $\Sigma^n = \Sigma \times \dots \times \Sigma$: strings of **length** $n \in \mathbb{N}_0$ (n -tuples)
- ▶ $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$: set of **all** (finite) strings over Σ
- ▶ $\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$: set of **all** (finite) **nonempty** strings over Σ
- ▶ $\varepsilon \in \Sigma^0$: the *empty* string (same for all alphabets)
- ▶ for $S \in \Sigma^n$, write $S[i]$ (other sources: S_i) for ***i**th* character ($0 \leq i < n$)

- ▶ for $S, T \in \Sigma^*$, write $ST = S \cdot T$ for **concatenation** of S and T
- ▶ for $S \in \Sigma^n$, write $S[i..j]$ or $S_{i,j}$ for the **substring** $S[i] \cdot S[i+1] \dots S[j]$ ($0 \leq i \leq j < n$)
 - ▶ $S[0..j]$ is a **prefix** of S ; $S[i..n-1]$ is a **suffix** of S
 - ▶ $S[i..j) = S[i..j-1]$ (endpoint exclusive) \rightsquigarrow $S = S[0..n)$

Clicker Question



True or false: $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

A True

B False

sli.do/comp526

Click on “Polls” tab

Clicker Question



True or false: $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

A True ✓

B ~~False~~

sli.do/comp526

Click on "Polls" tab

String matching – Definition

Search for a string (pattern) in a large body of text

► **Input:**

► $T \in \Sigma^n$: The text (haystack) being searched within

► $P \in \Sigma^m$: The pattern (needle) being searched for; typically $n \gg m$

► **Output:**

► the first occurrence (match) of P in T : $\min\{i \in [0..n-m) : T[i..i+m) = P\}$

► or NO_MATCH if there is no such i (" P does not occur in T ")

variants:
find all occurrences

► Variant: Find **all** occurrences of P in T .

↪ Can do that iteratively (update T to $T[i+1..n)$ after match at i)

► **Example:**

► $T = \text{"Where is he?"}$

► $P_1 = \text{"he"} \rightsquigarrow i = 1$

► $P_2 = \text{"who"} \rightsquigarrow \text{NO_MATCH}$

► string matching is implemented in Java in `String.indexOf`