Theory: String basics

6 minutes 3 / 3 problems solved

Start practicing

§1. What is a string?

A string is one of these words that shift their meaning depending on the context: it can refer to the charming sounds of a harp or describe the structure of the universe. In computer programming, a **string** is simply an ordered sequence of symbols. The **length** $|s|$ of the string s is the total number of characters it contains. Examples:

- 100101 — a string of length 6;
- GATTACA — a string of length 7.

A string can have zero length. In this case, it is called an **empty string**.

Sometimes we need to consider a **reverse string**. For a string s, we will denote a reverse string as $reverse(s)$. For example,

$$reverse(GATTACA) = ACATTAG.$$

§2. Substrings

A **substring** is a contiguous subsequence of symbols of an original string. For instance, ATTA is a substring of the string GATTACA because the string GATTACA contains the sequence ATTA starting from the first and ending with the fourth symbol (assuming zero-based indexing).

> **Note**: Any string is a substring of itself. An empty string is a substring of any string.

A substring of the string s starting from the i-th and ending with the $(j-1)$-th symbol is denoted by $s[i:j]$. Example: for s = GATTACA, $s[1:5] =$ ATTA.

A substring starting from the index 0 is a **prefix**, and a substring ending with the last index is a **suffix**. For example, $s[0:3] =$ GAT is a prefix of s, and $s[4:7] =$ ACA is a suffix of s.

> **Note**: An empty string is a substring of any string, starting with the index 0.

§3. Strings applications

Strings are one of the most used data structures in programming. In almost any field, there are problems requiring knowledge of string processing algorithms. Consider a few typical string processing algorithms and problems.

- String-searching algorithms. Let's say we are developing a text editor and want to add a function allowing users to search for a pattern in a text: if you've ever tried finding a specific word in a wall of text, you know the ordeal. To solve this problem, we need to implement an algorithm that finds all occurrences of a given pattern in a string.
- Similarity measure. Suppose that we are developing a search engine. We want to correct users' spelling mistakes to process their requests better (recall googling something in a rush: it feels gorgeous when a search engine understands you anyway). After getting a word with a typo, we may find the most similar word in our dictionary and use it instead of the initial one. To do that, we need to implement a measure of similarity between two strings.

§4. Why study string algorithms?

Current topic:

✓ String basics ···

Topic depends on:

✓ The big O notation ···

Topic is required for:

✓ Searching a substring ···

✓ Hamming distance ···

✓ Prefix function ···

✓ String hashing ···

Edit distance ···

Table of contents:

↑ String basics

§1. What is a string?

§2. Substrings

§3. Strings applications

§4. Why study string algorithms?

Feedback & Comments

Usually, programming languages provide some standard string methods. But some of the more complex problems require you to develop and implement the solution manually.

It's often easy to come up with a simple solution to a particular problem, but a naive algorithm won't probably be efficient in terms of running time and memory consumption. More efficient algorithms are often less obvious.

For these reasons, it's essential to be familiar with some standard and well-known string processing algorithms and their applications. We will examine some of them in the next topics.

Report a typo

Thanks for your feedback!

Write here how we could improve this theory

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion