Theory: Searching a substring

10 minutes 5 / 5 problems solved Start practicing

1004 users solved this topic. Latest completion was about 5 hours ago.

§1. The problem

Strings create the programming universe, being one of the most used data structures, so it is essential to know how to handle them. **Searching for a substring** is one of the skills you're likely to use quite often. For example, while working in a text editor, you might want to find all occurrences of a particular word in a text, or you might need to find a particular phrase on a web page without having to read the entire thing. For convenience and clarity, let's differentiate: a substring that we are looking for is called a **pattern**, and a string in which we make a search is a **text**.

Say we have a pattern "ACA" and a text "ACBACAD". We can see that the pattern is a substring of the text because it's contained there starting from the third and ending with the fifth symbol (assuming zero-based indexing). With this short string, our watchful eye is enough, but in programming and with longer sequences we need to come up with an algorithm that can solve the problem for an arbitrary pattern and text.

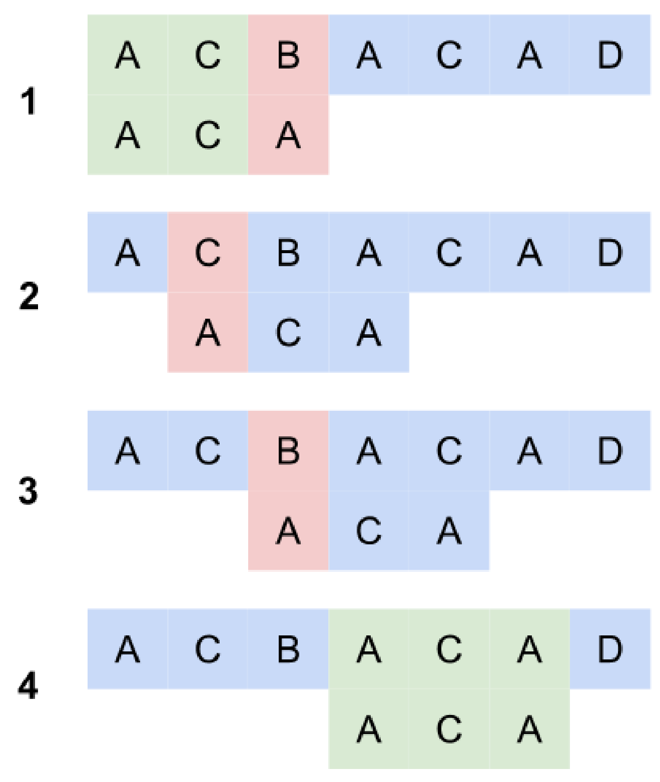
§2. The simplest algorithm

One of the simplest algorithms for finding a pattern in a text is the following:

1. Compare the pattern with the beginning of the text.
2. If every symbol of the pattern matches the corresponding symbol of the text, we're lucky: an occurrence is found.
3. If at least one symbol of the pattern doesn't match with the corresponding symbol of the text, the current attempt is a failure. In this case, we move the pattern by one symbol to the right and make a comparison again.
4. Repeat steps 2 and 3 until the pattern is found or the end of the text is reached. If none of the attempts were successful, we should indicate that there is no such pattern in the text.

§3. An example

The picture below illustrates how this algorithm works for the pattern "ACA" and the text "ACBACAD". To please your eye and aid your comprehension, matching symbols are shown in green, non-matching ones are red, and those not used in the current step are colored with a gentle blue hue:



Current topic:

✓ Searching a substring

Topic depends on:

✓ String basics

Topic is required for:

✓ Knuth-Morris-Pratt algorithm

Rabin-Karp algorithm

Searching a substring in Java

✓ Suffix tree

Searching a substring in Python

Table of contents:

Here's what's happening: in the first step, we compare the pattern with the very beginning of the text. The first two symbols match but, alas, the third doesn't. The attempt isn't successful, so we shift the pattern to the right. In the second and the third steps, we try to compare the corresponding symbols again but have an obvious mismatch in the first symbol. In the fourth step, all the corresponding symbols match, so an occurrence is successfully found.

You can see that sometimes there is no need to process all the symbols of the pattern. If we have a mismatch, say, in the first symbol, we don't need to compare the rest. In case of failure, we can immediately shift the pattern and start a new step: a nice life lesson to learn from.

§4. Complexity analysis

Let's denote $|p|$ as the length of a pattern and $|t|$ as the length of a text. In the worst case, we need to process all $|p|$ symbols of the pattern at every step. The maximal number of steps is $|t| - |p| + 1$ (assuming $|t| \geq |p|$). So, the overall running time is

$$(|t| - |p| + 1) \cdot |p| = |t| \cdot |p| - |p|^2 + |p| = O(|t| \cdot |p|)$$

Note that the algorithm requires $O(1)$ of additional memory.

§5. Additional notes

For now, we've learned about the algorithm that can answer "yes" if a pattern is present in the text and "no" if it's not. However, the problem can also be formulated in multiple other ways. We may need to:

- find the index of the first occurrence of a pattern in a text;
- find all occurrences of a pattern in a text;
- find all non-overlapping occurrences of a pattern in a text.

These variations will be suggested to you as a programming exercise.

Report a typo

91 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (8) Hints (0) Useful links (0) Show discussion