Naive Pattern Search: Conceptual

How to find a pattern within a larger body of text.

Have you ever scoured a page in a dictionary or website looking for a specific word or phrase? Checking word after word until it matches exactly what you are looking for?

This is a naive form of pattern searching!

It’s called naive because it is the simplest way to tackle the problem of finding a specific pattern (such as a word) in a text.

Naive Pattern Searching

Pattern searching requires two base components:

A text to scan

A pattern to search for

In our naive search, we can imagine the text being scanned as one long string of characters, one after another. The pattern is a separate, shorter string that we slide along the original text one character at a time, like a finger following letters in a book.

For each character of the original text, we count the number of following characters that match the pattern. If a disparity is found, then we move to the next letter of the text, but if the number of matching characters equals the length of the pattern, well then we found the pattern in the text!

Performance

The Naive Pattern Search is nice in its simple and intuitive approach to the common problem of finding a pattern or word in a larger body of text. However, for each character in the original text, we have to compare it to every character in the pattern one by one before moving further along in the text.

If we imagine a worst-case scenario where every character of the pattern k consistently matches all of the letters of the original text n, then it would take O(nk) comparisons. This means that the performance of the Naive Pattern Search approaches the slow O(n^2)!

The constant backtracking to the next character of the input text is the main cause of this slow performance, causing the algorithm to check the same characters many times. Better integrating the iteration of the pattern within the larger iteration of the text is the key to more optimized search algorithms, such as the Knuth–Morris–Pratt algorithm. It tracks collections of characters in the pattern called prefixes to intelligently skip through the original text after having checked if a pattern matches, thereby preventing backtracking, and getting a runtime of O(n+k).