CSE515 Advanced Algorithms
Notes on Lecture 26: FFT II

Antoine Vigneron

May 27, 2021

We present a solution to the pattern matching problem given at the end of the lecture.

Alphabet of size two. We begin with a simpler version of the problem where the alphabet is $\Sigma = \{a, b\}$.

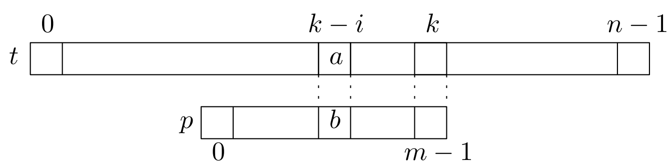

Figure 1: Mismatch of type I at position k

We will say that there is a *mismatch of type I* at position $k \geqslant m-1$ if, when we match the last character of p with the kth character of t, (at least) one of the letters of p is b and the corresponding letter of t is a. (See Figure 1.) In other words, there is a mismatch of type I at position k if there exists i such that $t[k-i] = a$ and $p[m-1-i] = b$.

In order to detect mismatches of type I, we encode the sequences differently. So let t' be the sequence of n bits defined by

$$t'[i] = \begin{cases} 1 & \text{if } s[i] = a \\ 0 & \text{otherwise} \end{cases}$$

and let p' be the sequence of m bits defined by

$$p'[i] = \begin{cases} 1 & \text{if } p[m-i-1] = b \\ 0 & \text{otherwise.} \end{cases}$$

So p' has been reversed compared with p, and has been made binary.

Using this encoding, we have a mismatch of type I at position k iff there exists $i \in \{0, \dots, m-1\}$ such that $t'[k-i] \cdot p'[i] = 1$. This condition can be rewritten

$$\sum_{i=0}^{m-1} p'[i] \cdot t'[k-i] \geqslant 1.$$

In other words, the kth coefficient of the convolution $p' \otimes t'$ is nonzero. So we can find all mismatches of type I by computing in $O(n \log n)$ time the convolution $p' \otimes t'$.

Similarly, we can define mismatches of type II when $s[i] = b$ and the corresponding letter on p is a. Then the sequence $s''[i]$ records 1 when $s[i] = b$, and $t''[p]$ records 1 when $p[m-i-1] = a$. Applying the same approach, we get all mismatches of type II in $O(n \log n)$ time by computing the convolution $p'' \otimes t''$. Then the positions where p matches t are those that do not present a mismatch of type I or II, that is, the positions $m-1 \leqslant k \leqslant n-1$ such that $(p' \otimes t')[k] = (p'' \otimes t'')[k] = 0$.

General case. First notice that the "don't care" symbols can be easily handled: just write 0 at the corresponding position of t' or p'. Then the algorithm will never consider it a mismatch, which is what we want.

To handle an alphabet Σ of arbitrary size s, we proceed as follows. Encode each letter of Σ as a $\lceil \log s \rceil$-bits integer. Then we will consider mismatches bit by bit. So if a letter a on t and the corresponding letter b on p form a mismatch, then one bit of the binary representation of a must differ from the same bit of the representation of b.

Therefore, instead of computing two convolutions, we compute $2\lceil \log s \rceil$ convolutions, because we have type I and type II mismatches for each bit of the representations of the letters. When one mismatch is found among these $2\lceil \log s \rceil$ types of mismatches, we have a mismatch. Then the positions where p match with t are the positions that remain, i.e. without mismatch.

So in total, we are computing $O(\log s)$ convolutions, and thus the running time is $O(n \log n \log s)$.

Improving the running time. In the lecture slides, we announced a running time $O(n \log(m) \log s)$ instead of $O(n \log(n) \log s)$. We now explain how to achieve it.

Suppose $n \geqslant 2m$, and we only want to find matchings within the first $2m$ characters of t, i.e. we want to match p with $t_0 t_1 \ldots t_{2m-1}$. Then we can use the algorithm above which runs in time $O(m \log(m) \log s)$. Similarly, if we want to find matchings between p and $t_m t_{m+1} \ldots t_{3m-1}$, then we can also do it in time $O(m \log(m) \log s)$.

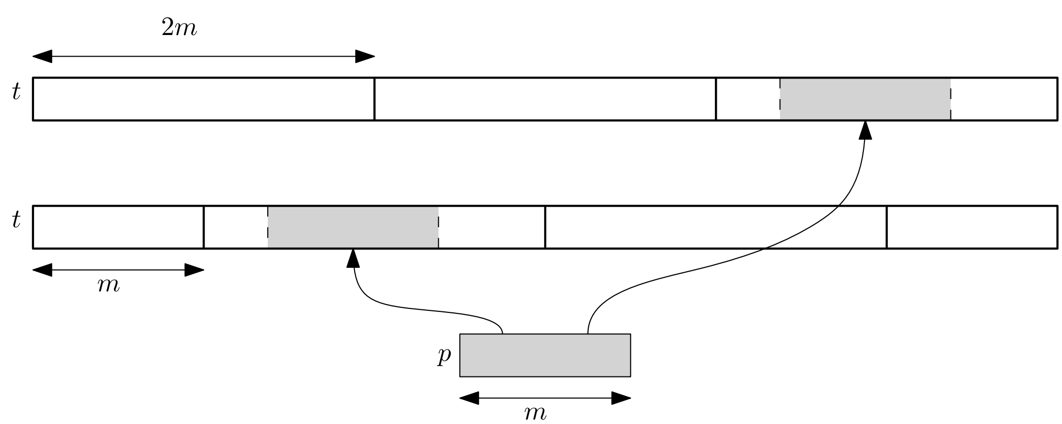

Figure 2: Splitting t into windows of size $2m$.

So in order to find matches within the whole sequence t, we can just shift a "window" of size $2m$ by successive distances m, and solve the problem within each window. (See Figure 2.) As there are less than $2n/m$ windows and each is handled in time $O(m \log(m) \log s)$, the overall running time is $O(n \log(m) \log s)$, as announced.