THE KMP ALGORITHM

Yanheng Wang

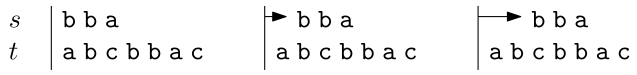
The string matching problem asks if a "snippet" $s[0\dots m]$ appears as a substring in the "text" $t[0\dots n]$. In other words, we are searching for a shift i such that $s[0\dots m]=t[i\dots i+m]$.

Here is a skeleton algorithm, with details to be filled:

Algorithm Skeleton

```
for $i=0,\dots,n$ do
    find the maximum $\ell$ such that $s[0\dots\ell]=t[i\dots i+\ell]$
    if $\ell=m$ then
        return true
return false
```

We can interpret it as sliding the snippet over the text. Initially s and t are aligned to the left. In each iteration we shift s to the right and verify if it agrees with t vertically.



Naïvely we could implement the "find" by comparing $s[j]$ with $t[i+j]$ for every j, which costs $O(m)$ time. A worst-case example is $s=$ `aaab` and $t=$ `aaaaaaaaa`: for each i we need $\Theta(m)$ comparisons, only to find that s and t disagree on a single letter. The total running time is thus $\Theta(mn)$.

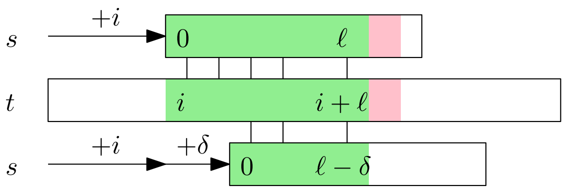
The Knuth-Morris-Pratt algorithm swiftly reuses previous comparisons to save time. Assume that we have computed ℓ, thus

$$s[0\dots\ell]=t[i\dots i+\ell]. \qquad (1)$$

Now we want to increase the shift i to $i+\delta$. However, not all δ's are worth considering. If $i+\delta$ is correct, i.e. $s[0\dots m]=t[i+\delta\dots i+\delta+m]$, then in particular

$$s[0\dots\ell-\delta]=t[i+\delta\dots i+\ell]. \qquad (2)$$

The picture below illustrates the two equations.



Now from (1) and (2) we deduce a necessary condition

$$s[\delta\ldots\ell] = t[i+\delta\ldots i+\ell] = s[0\ldots\ell-\delta],$$

so it is safe to increase i by

$$\delta_\ell := \min\{\delta \geqslant 1 : s[\delta\ldots\ell] = s[0\ldots\ell-\delta]\}.$$

In case that δ does not exist we set $\delta_\ell := \ell+1$. The correctness of the algorithm below follows.

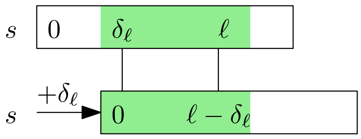
Algorithm KMP (part 1)

```
compute $\delta_0, \ldots, \delta_m$
$i := 0$
$\ell := -1$
**while** $i \leqslant n$ **do**
    **while** $s[\ell+1] = t[i+\ell+1]$ **do**
        $\ell := \ell+1$
    *{now $\ell$ is maximal such that $s[0\ldots\ell] = t[i\ldots i+\ell]$}*
    **if** $\ell = m$ **then**
        **return** true
    $i := i+\delta_\ell$
    $\ell := \ell-\delta_\ell$
    *{now we still have $s[0\ldots\ell] = t[i\ldots i+\ell]$}*
```

Assuming that $\delta_0, \ldots, \delta_m$ can be computed quickly, let us analyse the running time of the rest. Visually, each iteration of the outer loop first pushes the red cell to the right by some (possibly 0) distance, and then shifts the snippet to the right by some positive distance. The cost is proportional to their combined displacements. Since neither the red cell or the snippet can travel more than n units over the course of the algorithm, the total cost is bounded by $O(n)$.

Formally, $2i+\ell$ strictly increases at each iteration of the inner loop, and also after every update of (i,ℓ) in the outer loop. Since $2i+\ell \leqslant 2n+m = O(n)$ throughout, the number of steps is upper bounded by $O(n)$.

Next we explain how to compute $\delta_0,\ldots,\delta_m$ in linear time. A picture would help clarify the definition of δ_ℓ:



Exercise. Show that $1=\delta_0 \leqslant \cdots \leqslant \delta_m \leqslant m+1$.

To hunt for $\delta_{\ell+1}$, we start from $\delta := \delta_\ell$ and increase the value gradually. Visually this corresponds to shifting the second s to the right. In the beginning we have $s[\delta\ldots\ell] = s[0\ldots\ell-\delta]$. Our goal is keeping this equation while making $s[\ell+1] = s[\ell+1-\delta]$ as well.

A familiar situation, isn't it? Using the argument as in (1) and (2), we can safely increase δ by $\delta_{\ell-\delta}$ instead of one. Moreover, as $s[\delta\ldots\ell] = s[0\ldots\ell-\delta]$ is preserved, we only have to check if $s[\ell+1] = s[\ell-\delta+1]$. If so then we are done; otherwise we iterate.

Algorithm KMP (part 2)

```
$\delta = \delta_0 := 1$
**for** $\ell = 0,\ldots,m-1$ **do**
    **while** $s[\ell+1] \neq s[\ell+1-\delta]$ **do**
        $\delta := \delta + \delta_{\ell-\delta}$
        $\{$*invariant:* $s[\delta\ldots\ell] = s[0\ldots\ell-\delta]\}$
    $\delta_{\ell+1} := \delta$
```

The running time is captured by the number of updates to δ. Since every update strictly increases δ, and since $\delta \leqslant m+1$ always, the time is linear as desired.

To conclude the note, we mention that δ_m is exactly the *period* of string s. That is, $s[i] = s[i+\delta_m]$ for all i, and no natural number below δ_m satisfies the property.

Exercise. Prove it.