String Matching

Last time we presented the last of our approximation algorithms, Subset-Sum. We now focus our attention to a different issue, string matching. We will present the Knuth-Morris-Pratt (KMP) algorithm for string matching, which is an exact algorithm that runs in linear time.

Problem Definition: Given text T and pattern P, find all occurrences of P in T, where $|T| = n$, $|P| = m$, and $m << n$ (the length of the pattern is much less than the length of the full text). The algorithm should return the starting positions within T for each occurrence of P.

Consider the following example:

$$T = abbabbaaab \Longrightarrow n = 10$$
$$P = abba \Longrightarrow m = 4$$
$$return : \{1, 4\}$$

In practice, often we could be required to solve the following problem: given a lengthy text, such as a journal article or even a textbook of a thousand pages (assuming we have no prior knowledge of the structure of the text), we need to find all the places where a certain term is found. We had better have an algorithm that runs as close to linear in the input size $(n + m)$ as possible.

The naive way of solving the problem would be to check for each position in T if it can be a starting position for an occurrence of P. That is, start at position i, $1 \leq i \leq n$, and move forward until at position $i + j$ there is a mismatch, or until an occurrence of P is found. At that point, move to the next i. This algorithm is given below. The naive algorithm returns a list of

Naive Algorithm

$L = \emptyset$
For $i = 1$ to $n - m + 1$
  If $P = T[i...i + m]$ then
    add i to L
Return L

all the positions within T that are the beginning of an occurrence of P.

A word of notation is in order. For ease of analysis, we will allow the character index j within the pattern P to be in the range $1...m$, so an index of 0 will not be permitted. Moreover, $P[i...j]$ will denote the *substring starting at i and ending at j*, with $|P[1...j]| = j$. Thus, $P[1...i] = P[1 + j - i...j]$ will mean that $P[1...i]$ is a suffix of $P[1...j]$.

The running time of the naive algorithm is $\Theta((n - m)m) = \Theta(mn)$. Our goal is to achieve a running time of $O(m + n)$, that is, to develop an algorithm that runs in time linear in the input size. KMP is one such algorithm, and we will describe it next.

KMP Approach

The idea is to maintain two pointers, l and r, into the text T, such that the following KMT invariants are satisfied:

- $0 \leq r - l \leq m$
- $T[l...r] = P[1...1 + r - l]$

- All matches starting before l have been identified and added to L (list of matches).

The first point tells us that the length of the text between the two pointers cannot be more than the length of the pattern P; the second point requires that the substring in T in positions $l, ..., r$ be identical to the prefix in P ending at position $1 + r - l$ (see Fig. 1); the third point is clear.

The algorithm outline is presented below. Observe that by the KMP-invariants, the algorithm is correct.

KMP outline

$l = r = 1$; $L = \emptyset$
Preprocess(P)
loop: (*while* $r \leq n$)
  at each iteration
    increase either l or r or both
    modify L
output L

Now, note that in the loop, l can be increased at most n times; the same is true for r. Thus, there are at most $2n$ iterations of the loop. We will show that each iteration takes $O(1)$ time, with a preprocessing mechanism that takes $O(m)$ time. Thus, the total time in the loop is $O(n)$. Combined with the preprocessing time, this gives an $O(n + m)$ algorithm. We will consider several cases that can occur in a given iteration.

Case 1: $r - l = m$

Found a match; add l to L
$l = l + m - \pi(m) = r - \pi(m)$; r unchanged

Case 2: $r - l < m$

2.1: $T[r] = P[1 + r - l]$ (see Fig. 2)

We have just matched one more character in the pattern, so increment r:
l unchanged; $r \leftarrow r + 1$

2.2: $T[r] \neq P[1 + r - l]$ and $r = l$

In other words, $T[r] \neq P[1]$: so far we have matched nothing
$l \leftarrow l + 1$; $r \leftarrow r + 1$

2.3: $T[r] \neq P[1 + r - l]$ and $l < r$

$l = r - \pi(r - l)$; r unchanged

We now make the following definition:

$$\pi(j) \quad = \quad \max\{i : \ i < j \ \textit{and} \ P[1...i] = P[1 + j - i...j]\}$$

Thus, $\pi(j)$ is the length of the longest prefix of P that is also a nontrivial suffix of $P[1...j]$. Thus,

$\pi(i)$ is the length of the longest prefix of P that is also a nontrivial suffix of $P[0...i]$. Let us consider two examples.

Example:

$$T = bbababababaabc$$
$$P = ababa$$
$$l = 3,\ r = 8$$

Here, $r - l = 5 = m$, so we are in case 1. So,

$$\begin{aligned} \pi(5) &= \max\{i < 5 :\ P[1...i] = P[1+5-i...5]\} \\ &= \max\{1, 3\} = 3 \end{aligned}$$

Thus, $l = r - \pi(m) = 8 - \pi(5) = 8 - 3 = 5$; r is unchanged.

Example:

$$T = bbababababaabc$$
$$P = babaa$$
$$l = 2,\ r = 6$$

Here, $r - l = 4 < m$, $T[r] \neq P[1 + r - l]$, and $l < r$, so we are in case 2.3. So,

$$\begin{aligned} \pi(4) &= \max\{i < 4 :\ P[1...i] = P[1+4-i...4]\} \\ &= \max\{2\} = 2 \end{aligned}$$

Thus, $l = r - \pi(r - l) = 6 - \pi(4) = 6 - 2 = 4$; r is unchanged.

Let us now consider an efficient algorithm for computing the prefix function.

Alg1

```
π[1] = 0; i = 0
For j = 2 to m
    While i > 0 and P[i+1] ≠ P[j]
        i = π[i]
    If P[i+1] = P[j]
        i = i + 1
    π[j] = i
Return π
```

Example: Let us apply this algorithm to the simple example pattern $P = ababcb$. The resulting π is $\{0, 0, 1, 2, 0, 0\}$.

Running time

We will use amortized analysis to show that the running time of the above algorithm is $\Theta(m)$. Let us associate a potential Φ with the current state i of the algorithm. First, note that the only two places in the algorithm that can modify i (and thus change the potential) are lines 4 and 6. So,

- $\Phi_0 = 0$ (by line 1)
- i decreases every time line 4 is executed, since $\pi[i] < i$ by definition
- i is increased by *at most* 1 on line 6 each time through the *For* loop, since the *If* condition on line 5 may not always evaluate to true
- $i \geq 0$ is an invariant, since $\pi[i] \geq 0, \ \forall i$
- $i < j$ is an invariant, since $i = 0 < 2 = j$ initially in the *For* loop and since j is incremented by 1 *exactly* once per iteration, while i is incremented by *at most* 1

Now,

- Since $\pi[i] < i$, we can take the cost of each iteration k of the *While* loop (line 4) to be $\Delta_k\Phi$, the decrease in Φ for the current step
- On line 6, Φ is increased by at most 1. In total, this gives $m-2$ increments of Φ by 1, so the total decrease in potential cannot be $> (m-2)$, as $i \geq 0$ is an invariant
- Thus, a single iteration of lines 3-7 gives a constant amortized time: $O(1)$

The number of iterations of the *For* loop on lines 2-7 is $m-2$. Note that the *total potential drop* $= \Delta\Phi = \Phi_0 - \Phi_f \leq 0$ (remember that $i \geq 0$ is an invariant). Thus, the total actual worst-case running time of the prefix algorithm is $\Theta(m)$:

$$\begin{aligned} \textit{total actual time} &= \textit{total amortized time} + \textit{potential drop} \\ &= \Theta(m) + \Delta\Phi = \Theta(m) \end{aligned}$$

Correctness

Lemma 0: Let $P[1...k_1]$, $P[1...k_2]$, and $P[1...k]$ be strings such that $P[1...k_1] = P[1+k-k_1...k]$ and $P[1...k_2] = P[1+k-k_2...k]$. If $k_1 \leq k_2$, then $P[1...k_1] = P[1+k_2-k_1...k_2]$.

Proof: Draw the three strings aligned together. $\diamondsuit$

Now, let

$$\pi^*[j] = \{\pi^{(1)}[j], \pi^{(2)}[j], ..., \pi^{(t)}[j]\},$$

where $\pi^{(0)}[j] = j$ and $\pi^{(k+1)}[j] = \pi[\pi^{(k)}[j]]$ for $k \geq 1$. The terminating condition for the sequence is $\pi^{(t)}[j] = 0$.

Example: Consider again the pattern $P = ababcb$. For $j = 4$, $\pi^*[j] = \{2, 0\}$, while for $j = 3$, $\pi^*[j] = \{1, 0\}$, and for $j = 6$, $\pi^*[j] = \{0\}$.

Lemma 1: $\pi^*[j] = \{i : \ i < j \textit{ and } P[1...i] = P[1+j-i...j]\}$ for $j \in [1, m]$.

Compare *Lemma 1* to the definition of $\pi[j]$. The lemma tells us that $\pi^*[j]$ includes all numbers i satisfying the given conditions, while $\pi[j]$ gives the maximum of these numbers. Let us now prove the claim in the lemma.

Proof:

1. First, note that

$$k \in \pi^*[j] \Rightarrow P[1...k] = P[1+j-k...j], \tag{1}$$

   since $k \in \pi^*[j] \Rightarrow \exists x > 0 : \ k = \pi^{(x)}[j]$, by the definition of π^*. By induction on x, Eq.(1) holds. Thus, generalizing Eq.(1),

$$\pi^*[j] \subseteq \{i : i < j \ \mathit{and} \ P[1...i] = P[1+j-i...j]\} \tag{2}$$

2. Now, assume that the set $S = \{i : i < j \ \mathit{and} \ P[1...i] = P[1+j-i...j]\} - \pi^*[j] \ \neq \ \emptyset$ and let $z = \max_x\{x \in S\}$. Note that the elements in S are not in $\pi^*[j]$, so $z \notin \pi^*[j]$.

   - As $\pi[j] = \max_i\{i < j \ \mathit{and} \ P[1...i] = P[1+j-i...j]\}$, $z < \pi[j]$.
   - Now, $\pi[j] \in \pi^*[j]$ by definition, so let $y = \min_x\{x \in \pi^*[q] : \ x > z\}$
   - $P[1...z] = P[1+j-z...j]$ by the definition of z
   - $P[1...y] = P[1+j-y...j]$ by the definition of y
   - So, by *Lemma 0*, $P[1...z] = P[1+y-z...y]$
   - Thus, $\pi[y] = z$, by the definitions of π, y, and z
   - So, $z = \pi[y] \in \pi^*[j]$, which is a contradiction. Thus, our assumption does not hold, and

$$\pi^*[q] \supseteq \{k : k < q \ \mathit{and} \ P[1...k] = P[1+q-k...q]\} \tag{3}$$

Combining Eqs.(2) and (3), we prove *Lemma 1*. $\diamondsuit$

The prefix algorithm correctly gives $\pi[1] = 0$ on line 1, directly by the definition of π. Let us consider the remaining cases.

Lemma 2: $\pi[j] > 0 \Rightarrow \pi[j] - 1 \in \pi^*[j-1]$ for $j \in [1, m]$

Proof: Let $t = \pi[j]$, so $0 < t < j$ by the condition of the lemma

- $P[1...t] = P[1+j-t...j]$ by the definition of π
- So, $t-1 < j-1$ and $P[1...t-1] = P[1+(j-1)-(t-1)...j-1]$, since only the last identical character is deleted
- Thus, by Lemma 1, $t-1 \in \pi^*[j-1]$
- As $\pi[j] - 1 = t - 1$ by the definition of t, $\pi[j] - 1 \in \pi^*[j-1]$. $\diamondsuit$

We now define the set $E_{j-1} \subseteq \pi^*[j-1]$ for $j \in [2, m]$:

$$\begin{aligned}
E_{j-1} &= \{i \in \pi^*[j-1] : \ P[i+1] = P[j]\} \\
&= \{i : \ i < j-1 \ \mathit{and} \ P[1...i] = P[1+(j-1)-i...j-1] \ \mathit{and} \ P[i+1] = P[j]\} \\
&= \{i : \ i < j-1 \ \mathit{and} \ P[1...i+1] = P[1+j-(i+1)...j]\} \\
&= \{i : \ i < j-1 \ \mathit{and} \ P[1...i+1] = P[j-i...j]\}
\end{aligned}$$

The second line follows by *Lemma 1.* The third line is acquired by just adding an identical character at the end of both strings.

Example: For the pattern $P = ababcb$: for $j = 4$, $E_{j-1} = \{1\}$; for $j = 5$, $E_{j-1} = \emptyset$.

All the proofs so far lead us to our final theorem, which at last tells us what the values of $\pi[j]$ have to be.

Theorem 1: $\pi[j] = 0$ if $E_{j-1} = \emptyset$; otherwise, $\pi[j] = 1 + max\{i \in E_{j-1}\}$

Proof:

1. $E_{j-1} = \emptyset \Rightarrow \neg\exists\, i$ such that $P[1...i+1] = P[j-i...j]$. So, $\pi[j] = 0$ by the definition of π.
2. $E_{j-1} \neq \emptyset \Rightarrow i + 1 < j$ and $P[1...i+1] = P[j-i...j]$, $\forall i \in E_{j-1}$, by definition. So,
   - $\pi[j] \geq 1 + \max\{i \in E_{j-1}\}$ by the definitions of π and E_{j-1}
   - Now, let $t + 1 = \pi[j] > 0$, so $P[1...t+1] = P[j-t...j]$ (by the definition of π) $\Rightarrow P[t+1] = P[j]$, since the last characters of both strings are identical
   - $t \in \pi^*[j-1]$ by the definition of t and *Lemma 2*
   - So, from the previous two points, $t \in E_{j-1}$ directly by the definition of E_{j-1}
   - Thus, $t \leq \max\{i \in E_{j-1}\}$, and so $\pi[j] \leq 1 + \max\{i \in E_{j-1}\}$ by the definition of t

   Thus, since $\pi[j] \geq 1 + \max\{i \in E_{j-1}\}$ and $\pi[j] \leq 1 + \max\{i \in E_{j-1}\}$, we finally have $\pi[j] = 1 + \max\{i \in E_{j-1}\}$. $\diamondsuit$

Example: For the pattern $P = ababcb$: for $j = 4$, $E_{j-1} = \{1\}$, so $\pi[4] = 1 + \max\{i \in E_{j-1}\} = 1 + 1 = 2$; for $j = 5$, $E_{j-1} = \emptyset$, so $\pi[5] = 0$.

To finish the proof of correctness, first note that $i = \pi[j-1]$, $\forall j$ in the *For* loop, by lines 1 and 7.

1. If the *While* loop terminates because $P[i+1] \neq P[j]$ becomes true, then we have found $i = \max\{k \in E_{j-1}\}$, by the definition of E_{j-1} and by the fact that no other number greater than the current i is in E_{j-1}, since we decrease the value of i each time through the *While* loop. By *Theorem 1*, $\pi[j] = 1 + i$. This is what we get on line 7, after first synchronizing the value of i on line 6, so that $i = \pi[j-1]$ will be true when the next iteration starts. Thus, *Alg1* gives a correct $\pi[j]$ in this case.
2. If the *While* loop terminates because $i = 0$ becomes true, then the only number that can be in E_{j-1} is 0. So, we check if $0 \in E_{j-1}$ on line 5:
   - If $0 \in E_{j-1}$, then $\max\{k \in E_{j-1}\} = 0$. By *Theorem 1*, $\pi[j] = 1 + 0 = 1$, which is what we get on line 7, again after first synchronizing i on line 6 for the next iteration.
   - If $0 \notin E_{j-1}$, then $E_{j-1} = \emptyset$, and, by *Theorem 1*, $\pi[j] = 0$. Since line 7 gives us exactly $\pi[j] = i = 0$, the algorithm is correct in this case as well.

With this, we complete the proof of correctness.

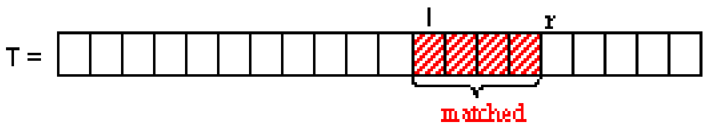

Figure 1

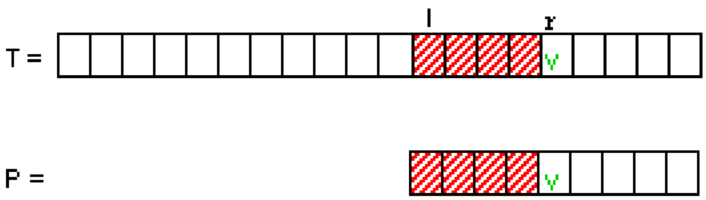

Figure 2