String Matching: Knuth-Morris-Pratt Algorithm

Greg Plaxton
Theory in Programming Practice, Fall 2005
Department of Computer Science
University of Texas at Austin
Some Notation

- We index the symbols in a string starting at 0.

- For any string $s$, let $\bar{s}$ denote the length of $s$.

- For any string $s$ and integer $i$ such that $0 \leq i < \bar{s}$, let $s[i]$ denote the symbol of $s$ with index $i$.

- For any string $s$ and integers $i$ and $j$ such that $0 \leq i < \bar{s}$ and $i \leq j \leq \bar{s}$, $s[i..j]$ denotes the (possibly empty) substring of $s$ starting at index $i$ and ending just before $j$.
  
  
  - $s[2..2]$ is the empty string.
  
  - $s[0..\bar{s}] = s$. 

---

Theory in Programming Practice, Plaxton, Fall 2005
The (Exact) String Matching Problem

• Given a text string $t$ and a pattern string $p$, find all occurrences of $p$ in $t$
Three Efficient String Matching Algorithms

• Rabin-Karp
  – This is a simple randomized algorithm that tends to run in linear time in most scenarios of practical interest
  – The worst case running time is as bad as that of the naive algorithm, i.e., $\Theta(p \cdot t)$

• Knuth-Morris-Pratt (this lecture and the next)
  – The worst case running time of this algorithm is linear, i.e., $O(p + t)$

• Boyer-Moore
  – This algorithm tends to have the best performance in practice, as it often runs in sublinear time
  – The worst case running time is as bad as that of the naive algorithm
The KMP String Matching Algorithm: Plan

• We maintain two indices, $\ell$ and $r$, into the text string

• We iteratively update these indices and detect matches such that the following loop invariant is maintained
  
  – KMP Invariant: $\ell \leq r$, $t[\ell..r] = p[0..r - \ell]$, and all occurrences of the pattern $p$ starting prior to $\ell$ in the text $t$ have been detected

• We ensure that the invariant holds initially by setting $\ell$ and $r$ to zero

• Remark: We will see later that the algorithm also requires a preprocessing phase involving only the pattern string $p$
Achieving Linear Time Complexity: The Plan

• The algorithm performs only a constant amount of computation in each iteration

• The algorithm never decreases $\ell$ or $r$

• In each iteration, either $\ell$ or $r$ is increased

• Note that the indices $\ell$ and $r$ are at most $\bar{t}$

• By the KMP invariant, all matches have been detected once $\ell$ reaches $\bar{t}$, so we can terminate at that point

• The preprocessing phase, which involves only $p$, runs in $O(\bar{p})$ time
KMP Iteration

• Let’s see how to define an iteration of the KMP loop
• Assume the KMP invariant holds at the beginning of the iteration
• Since the loop has not terminated, \( \ell < \bar{t} \)
• We’d like to increase \( \ell \) or \( r \), while maintaining the invariant
• There are two cases to consider
  – Case 1: \( 0 \leq r - \ell < \bar{p} \), i.e., we do not yet know whether there is a match starting at index \( \ell \)
  – Case 2: \( r - \ell = \bar{p} \), i.e., we have found a match starting at index \( \ell \)
**Case 1:** \(0 \leq r - \ell < p\)

- **Case 1.1:** \(t[r] = p[r - \ell]\)
  - We've matched another symbol; increment \(r\)

- **Case 1.2:** \(r = \ell\) and \(t[r] \neq p[r - \ell]\)
  - Our current match is the empty string and the next symbol does not match \(p[0]\); increment \(\ell\) and \(r\)

- **Case 1.3:** \(r > \ell\) and \(t[r] \neq p[r - \ell]\)
  - Our current match is a nonempty proper prefix of \(p\) and the next symbol does not extend this match
  - How should we update \(\ell\) and \(r\) in this remaining subcase?
Case 1.3: $0 \leq r - \ell < \bar{p}$, $r > \ell$, and $t[r] \neq p[r - \ell]$

- Our current match $u$ is a nonempty proper prefix of $p$ and the next symbol does not extend this match

- We cannot set $\ell$ to $r$ because we might skip over one or more matches
  - Example: Suppose $p$ is $axbcyaxbts$ and we’ve already matched $axbcyaxb$, but the next symbol is not $t$
  - In this example, we advance $\ell$ by 5

- In general, we advance $\ell$ by the smallest $k > 0$ such that the suffix $v = u[k..\bar{u}]$ of $u$ is a prefix of $p$

- Note that $v$ is simply the longest string that is both a proper prefix and a proper suffix of $u$
  - This string is called the core of $u$, denoted $c(u)$
  - Later we will discuss how the KMP algorithm computes such cores
Case 2: \( r - \ell = \bar{p} \)

- We output that a match exists starting at index \( \ell \)
- How do we update \( \ell \) and \( r \)?
- Note that this case is very similar to Case 1.3 treated earlier
- We increase \( \ell \) by \( \bar{p} - c(p) \)
Core Computation

- It remains only to describe how the KMP algorithm computes the cores required in Cases 1.3 and 2
- Recall that each iteration of KMP is supposed to run in a constant number of operations
- How can we hope to compute the core of a string in constant time?
KMP Core Computation: A Key Observation

• Note that in Case 1.3 we need to compute the core of some proper prefix of $p$, while in Case 2 we need to compute the core of $p$

• Thus, if we precompute the core of every prefix of $p$, we will be able to execute each iteration of the KMP loop in constant time

• It remains to prove that we can compute the core of every prefix of $p$ in $O(p)$ time
Some Properties of Core

- Let $u \preceq v$ mean that $u$ is both a prefix and a suffix of $v$
  - For any string $u$, $\epsilon \preceq u$
  - The $\preceq$ relation is a partial order
- Let $u \prec v$ denote $u \preceq v$ and $u \neq v$
- The core $c(v)$ of a string $v$ is the unique string such that for all strings $u$
  \[ u \preceq c(v) \equiv u \prec v \]
  - It follows, by replacing $u$ with $c(v)$, that $c(v) \prec v$ and hence $\overline{c(v)} < \overline{v}$
- Let $c^0(u)$ denote $u$ and for any $i \geq 0$ such that $c^i(v)$ is a nonempty string, let $c^{i+1}(u)$ denote $c(c^i(u))$
A Key Property

- Claim: For any \( u \) and \( v \), \( u \preceq v \equiv \langle \exists i : 0 \leq i : u = c^i(v) \rangle \)

- The proof is by induction on the length of \( v \)

- Base case (\( \bar{v} = 0 \)):

\[
\begin{align*}
 u \preceq v \\
\equiv \{ \bar{v} = 0, \text{i.e., } v = \epsilon \} \\
\{ u = \epsilon \land v = \epsilon \} \\
\equiv \{ \text{definition of } c^0: v = \epsilon \Rightarrow c^i(v) \text{ is defined for } i = 0 \text{ only} \} \\
\langle \exists i : 0 \leq i : u = c^i(v) \rangle
\end{align*}
\]
Induction Step: $\overline{v} = n + 1$, $n \geq 0$

\[
\begin{align*}
  u & \preceq v \\
  \equiv & \{ \text{definition of } \preceq \} \\
  & u = v \land u < v \\
  \equiv & \{ \text{definition of core} \} \\
  & u = v \lor u \preceq c(v) \\
  \equiv & \{ c(v) < \overline{v}; \text{ induction hypothesis on second term} \} \\
  & u = v \lor \langle \exists i : 0 \leq i : u = c^i(c(v)) \rangle \\
  \equiv & \{ \text{rewrite} \} \\
  & u = c^0(v) \lor \langle \exists i : 0 < i : u = c^i(v) \rangle \\
  \equiv & \{ \text{rewrite} \} \\
  & \langle \exists i : 0 \leq i : u = c^i(v) \rangle
\end{align*}
\]