The Knuth-Morris-Pratt Algorithm

The original KMP algorithm does not compute the failure function through Z-values.

- Start with $s_{p1} = 0$.
- Assume we have $s_{pi}$ for $i = 1, 2, \ldots, k$.
- **Goal:** Compute $s_{pk+1}$.

\[ P \begin{array}{c} \alpha \cr s_{pk} \end{array} \begin{array}{c} x \cr k \end{array} +1 \]

Let $\beta' = \beta x$ be the prefix of length $s_{pk+1}$ of $P$.

\[ P \begin{array}{c} \beta' \cr s_{pk+1} \end{array} \begin{array}{c} x \cr k \end{array} +1 \]

\( (*) \) $\beta$ is the longest proper prefix of $P[1 \ldots k]$ that matches a suffix of $P[1 \ldots k]$ and where $P[|\beta| + 1] = x$.

**Lemma.** For all $k$, $s_{pk+1} \leq s_{pk} + 1$. Further, $s_{pk+1} = s_{pk} + 1$ if and only if $P[s_{pk}+1] = P[k+1]$.

\[ P \begin{array}{c} \beta' \cr s_{pk+1} \end{array} \begin{array}{c} \beta' \cr k \end{array} x \]

What if $P[s_{pk}+1] \neq P[k+1]$?

\[ P \begin{array}{c} \alpha \cr s_{pk} \end{array} \begin{array}{c} \beta \cr k \end{array} \begin{array}{c} x \cr k+1 \end{array} \]

\( (**) \) $\beta$ is the longest proper prefix of $P[1 \ldots s_{pk}]$ that matches a suffix of $P[1 \ldots s_{pk}]$ and where $P[|\beta| + 1] = x$. 

\[ P \begin{array}{c} \alpha \cr s_{pk} \end{array} \begin{array}{c} \beta \cr k \end{array} \begin{array}{c} x \cr k+1 \end{array} \]

\( (**) \) $\beta$ is the longest proper prefix of $P[1 \ldots s_{pk}]$ that matches a suffix of $P[1 \ldots s_{pk}]$ and where $P[|\beta| + 1] = x$. 

\[ P \begin{array}{c} \alpha \cr s_{pk} \end{array} \begin{array}{c} \beta \cr k \end{array} \begin{array}{c} x \cr k+1 \end{array} \]
The reduction

- Find longest proper prefix of $P[1 \ldots sp_k]$ that matches a suffix of $P[1 \ldots sp_k]$ — its length must be $sp[sp_k]$.
- If $P[sp[sp_k] + 1] = x$, then we’re done: $sp_{k+1} = sp[sp_k] + 1$. Otherwise, recurse again.
- Eventually, either valid prefix is found or beginning of $P$ is reached. In latter case, if $P[1] = x$, set $sp_{k+1} = 1$; else set $sp_{k+1} = 0$.

$SP(P)$:

\[
sp_1 \leftarrow 0
\]
\[
\text{for } k \leftarrow 1 \text{ to } n - 1 \\
\quad \text{do } x \leftarrow P[k+1]; \ v \leftarrow sp_k \\
\quad \text{while } P[v + 1] \neq x \text{ and } v \neq 0 \\
\quad \quad \text{do } v \leftarrow sp_v \\
\quad \text{if } P[v + 1] = x \\
\quad \quad sp_{k+1} \leftarrow v + 1 \\
\quad \text{else } \ sp_{k+1} \leftarrow 0 \\
\text{return } sp
\]

$SP'(P)$:

\[
sp'_1 \leftarrow 0
\]
\[
\text{for } i \leftarrow 2 \text{ to } n - 1 \\
\quad \text{do } v \leftarrow sp_i \\
\quad \text{if } P[v + 1] \neq P[i + 1] \\
\quad \quad \text{then } sp'_i \leftarrow v \\
\quad \text{else } sp'_i \leftarrow sp'_v \\
\quad \text{return } sp'
\]

Exact Matching for Sets of Patterns

**Problem:** Find all occurrences in $T$ of any pattern in the set of patterns $P = \{P_1, \ldots, P_z\}$.

**Naive solution:** Run KMP or BM $z$ times. If $n$ is the total length of all the patterns, then total time is $O(n + zm)$.

**Aho-Corasick Algorithm:** An extension of KMP that solves set matching in $O(n + m + k)$ time, where $k =$ number of occurrences in $T$ of patterns in $P$. 
A **keyword tree** for set of patterns $P = \{P_1, \ldots, P_z\}$ is a rooted directed tree $K$ such that:
1. every edge is labeled with one character,
2. any two edges out of the same node have different labels,
3. each $P_i$ maps to some node $v$ in $K$ such that the characters on the path from the root spell out $P_i$, and every leaf of $K$ is mapped to by some $P_i$.

A keyword tree for $P$ can be constructed in $O(n)$ time.

---

**Naive application of keyword trees:**
- To find matches that begin at $T[l]$, $l = 1, \ldots, m$, follow unique path in $K$ that matches a substring of $T$ starting at $l$. Numbered nodes along the path indicate all patterns in $P$ that start at position $l$. Time $= O(\min\{n, m\})$
- Repeat this for each $l$. Time $= O(nm)$

**Dictionary problem:** The patterns are the words in the dictionary. To see if a word is in the dictionary, follow a path in keyword tree.

---

**Assumption.** No pattern in $P$ is a proper substring of any other pattern in $P$.

**Definition.** Let $v$ be a node in $K$. The **label** $L(v)$ of $v$ is the concatenation of the characters on the path from the root to $v$ in $K$.

**Definition.** For each node $v$ in $K$, $lp(v)$ is the length of the longest proper suffix of $L(v)$ that is also a prefix of some pattern in $P$.

---

**Lemma.** Let $\alpha$ be the $lp(v)$-length suffix of $L(v)$. Then there is a unique node in $K$ whose label is $\alpha$.

**Definition.** For each node $v$ in $K$, $n_v$ is the unique node in $K$ labeled with a suffix of $L(v)$ of length $lp(v)$. When $lp(v) = 0$, $n_v$ is the root of $K$. The pair $(v, n_v)$ is the **failure link** at $v$.
Algorithm ACSearch:
\[
l \leftarrow 1; \quad c \leftarrow 1; \quad w \leftarrow root(K)
\]
repeat
\[
\text{while there exists an edge } (w, w') \text{ labeled } T[c] \\
\quad \text{do } \quad \text{if } w' \text{ is numbered by } P_i \\
\quad \quad \text{then report occurrence of } P_i \\
\quad \quad \text{starting at position } l \text{ of } T \\
\quad w \leftarrow w'; \quad c \leftarrow c + 1 \\
\quad w \leftarrow n_w; \quad l \leftarrow c - lp(w)
\]
until \( c > n \)

Algorithm \( n_v \):
\[
\text{let } v' \text{ be the parent of } v \text{ in } K \\
\text{let } x \text{ be the character on edge } (v', v) \\
w \leftarrow n_v' \\
\text{while there is no } (w, w') \text{ labeled } x \text{ and } w \neq \text{root} \\
\quad \text{do } \quad w \leftarrow n_w \\
\text{if there exists a } (w, w') \text{ labeled } x \\
\quad \text{then } n_v \leftarrow w' \\
\quad \text{else } n_v \leftarrow root
\]

To find all the failure links in the keyword tree \( K \):

order nodes in \( K \) by non-decreasing distance from the root (breadth-first search order)
for each node \( v \) in \( K \), in order
\[
\text{do } \quad \text{apply Algorithm } n_v \text{ to } v
\]

Theorem. The total time needed to find all the failure links is \( O(n) \).

Proof sketch. Let \( t = |P_i| \).

If nodes \( v' \) and \( v \) are nodes on the path from the root to the node labeled \( i \) in \( K \), and \( v' \) is the parent of \( v \), then \( lp(v) \leq lp(v') + 1 \).
\[
\Rightarrow \text{total increase in } lp() \text{ over all nodes on path is } t.
\]

On the other hand, each time \( w \) is set inside the while loop, the potential value for \( lp(v) \) decreases by at least 1. But \( lp() \) is never negative.
\[
\Rightarrow \text{total decrease in } lp() \text{ is also at most } t\]
Lemma. Suppose there is a (possibly empty) path of failure links from a node $v$ to a node numbered by $P_i$. Then, $P_i$ must occur in $T$ ending at position $c$ (the current character) whenever $v$ is reached during $ACSearch$.

Lemma. Suppose a node $v$ is reached by $ACSearch$. Then, $P_i$ occurs in $T$ ending at position $c$ only if $v$ is numbered $i$ or there is a directed path of failure links from $v$ to the node numbered $i$.

**FullACSearch**:

$l \leftarrow 1; \; c \leftarrow 1; \; w \leftarrow root(K)$

repeat

while there exists an edge $(w,w')$ labeled $T[c]$ do

if $w'$ is numbered by $P_i$ or there is a path of failure links from $w'$ to a node numbered $P_i$ then report occurrence of $P_i$ starting at position $l$ of $T$

$w \leftarrow w'; \; c \leftarrow c + 1$

$w \leftarrow n_w; \; l \leftarrow c - lp(w)$

until $c > n$

A full implementation of Aho-Corasick requires *output links*.

For each $v$ in $K$, the *output link* for $v$ points to the numbered node that is reachable from $v$ through the fewest failure links. All output links can be computed in $O(n)$ time.

**Theorem.** After $O(n)$-time preprocessing, all occurrences in $T$ of patterns in $P$ can be found in $O(m+k)$ time, where $k$ is the number of occurrences.

**Application: Exact matching with wild cards**

A *wild card* is a character $\varnothing$ that matches any single character.

**Problem.** Given a pattern string $W$ containing wild card characters and a text $T$, find all occurrences of $W$ in $T$.

**Example.** $b\varnothing b\varnothing a\varnothing$ appears twice in $baabcabcabb$
Exact matching with wild cards:
1. Let $C$ be a vector of length $|T|$, set to all 0’s
2. Let $P = \{P_1, P_2, \ldots, P_k\}$ be the multi-set of all maximal substrings of $W$ that do not contain any wild cards. Let $l_1, l_2, \ldots, l_k$ be the starting positions in $P$ of these substrings.
3. Use Aho-Corasick to find all starting positions in $T$ of each $P_i$ in $P$. For each staring position $j$ of $P_i$, increment $C[j - l_i + 1]$ by one.
4. Scan $C$: There is an occurrence of $W$ in $T$ starting at $T[p]$ if and only if $C[p] = k$.

Theorem. All matches can be found in $O(n + km)$ time. If $k$ is bounded, all matches are found in $O(n+m)$ time.

Comments

- No solution with run time $O(n + m)$, independent of $k$, is known.
- There are cases where wild cards also appear in $T$. 