The KNUTH MORRIS PRATT (KMP) Algorithm

The Pattern Matching Problem

Given a pattern \( p = p_1 \ldots p_k \), \( p_i \in \Sigma \)
and a text \( t = t_1 \ldots t_n \), \( t_i \in \Sigma \)

patterns short
text = long, very long

Compute the first exact occurrence of \( p \) in \( t \).

\[ \begin{align*}
n = 1000,000 \quad l = 1000 & \quad \text{naive} \\
\end{align*} \]
KMP $O(\ell + m)$ tremendous speedup

optimal, practical

We will construct a finite automata $M_p$ based only on $p$.

Find first exact occurrence of $p$ in $t$.

$p = \cdots \quad (\text{no $p$ exact match found})$

$\Sigma = \{a, b\}$

We first build a skeletal machine DFA (det FA) with $\ell + 1$ states

$p = p_1 \cdots p_e, \quad p_i \in \Sigma$

we stay in state $\not\circ$ till the first $p_1$ symbol
• State 0 has a transition to itself on all symbols except $p_i \neq p_1$

• We can think of state $i$ as a pointer to the $i^{th}$ position in the pattern $p$

• The $M_p$ operates like a deterministic finite automata except that it can make several state transitions while reading the state input symbol

\[ s \rightarrow s' \rightarrow s'' \]

\[ \begin{array}{c} \text{Input} \\
\text{Symbols} \end{array} \]

(At the end we will "fix" the algorithm to operate exactly as a deterministic finite automata)

• $M_p$ has the same states as the skeletal machine
Thus the state $j$ of $M_p$ corresponds to prefix $p_1p_2...p_j$ of pattern string $p$.

- The text $t = t_1t_2...t_n$, $t_i \in \Sigma$

- $M_p$ starts in state $0$ with its tape head reading the first symbol of $t$

  $t = t_1...t_n$
IF \( p_1 = t_1 \) THEN state \( 1 \) and advance its tape head to position 2 in \( t_j \), i.e., \( t_2 \).

IF \( p_1 \neq t_1 \) then \( M_p \) remains in state \( 0 \) and advances its tape head to \( t_2 \).

Suppose that after we "read" \( t_1 t_2 t_3 \ldots t_k \) and \( M_p \) then is in state \( j \). This implies that the last \( j \) symbols from \( t_1 \) to \( t_k \).
are $P_1, P_2, \ldots, P_j$.

Prefix of $P$.

"suffix-prefix" the key to the algorithm insight.

---

IF $t = P_{k+1}$, the next symbol of text $t$, agrees with $P_{k+1}$, then $M_p$ advances to state $j+1$ and its tape head reads now $t_{k+2}$.

IF $t_{k+1} \neq P_{j+1}$, THEN $M_p$ enters the highest number.
State $i$ such that $\forall j \leq i \text{ max such } \rho_1 \rho_2 \ldots \rho_i$ is a suffix of $t_1 t_2 \ldots t_{k+1}$

"Skip as much as you can without missing any match of $p$" is the largest $i$

Going back is failure function

How to find $i$?

To help finding $i$ the machine $M$ has associated with it an integer valued function $f$ called the failure function defined as follows:
f(j) is the largest s < j for which
\[ p_1 p_2 \ldots p_s \text{ is a suffix of } p_1 p_2 \ldots p_j \]
\[ p_1 \ldots p_s \text{ is a prefix of } p \]
\[ p_1 \ldots p_s \text{ is a suffix of } p_1 p_2 \ldots p_j \]

That is, \( f(j) \) is the largest \( s \), \( 1 < j \) such that:
\[ p_1 p_2 \ldots p_s = p_{j-s+1} p_{j-s+2} \ldots p_j \]
Example

\[ p = aabbaaab \]

Failure function \( f \) of \( p \):

<table>
<thead>
<tr>
<th>( j )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(j) )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ j = 1 \quad f(1) = 0 \]

\[ j = 2 \quad p_1 p_2 = aa \quad i = 1 \]

\[ a \text{ is a prefix} \]
\[ a \text{ is a suffix of } \sum p_i p_k \]

\[ |a| = 1 \quad f(j) = 1 = f(2) \]

\[ j = 3 \quad p_1 p_2 p_3 = aab \quad \text{no suffix is prefix} \]

\[ f(j) = 0 = f(4) \]

\[ j = 4 \quad p_1 p_2 p_3 p_4 = aa \ 6b \]

\[ \text{no suffix} \]

\[ f(j) = 0 = f(4) \]
\[ j = 5 \quad p_1 p_2 p_3 p_4 p_5 = a a b b a \]

largest prefix that is suffix is a
\[ f(5) = 1 = f(5) \]

\[ j = 6 \quad p_1 p_2 p_3 p_4 p_5 p_6 = a a b b a a \]
a is prefix that is also suffix
aa is prefix that is also suffix
max: aa is the max such
\[ f(j) = 2 \]

\[ j = 7 \quad p_1 p_2 p_3 p_4 p_5 p_6 p_7 = a a b b a a b \]

yes aab is prefix and also suffix
\[ f(j) = 3 \quad f(7) \]
What is:

<table>
<thead>
<tr>
<th>$f(7)$</th>
<th>$f(3)$</th>
<th>$f(6)$</th>
<th>$f(2)$</th>
<th>$f(1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

will use $f$ repeatedly till we find the right match to extend with $t$ as a match $k+1$.

The Failure Function Algorithm

To see how the failure function is used by $M_p$, let us define the function

\[ f(m)(j) \]

as follows:

i) $f(j) = f(j)$, and
ii) \( f^m(j) = f(f^{m-1}(j)) \)
for \( m > 1 \)

ex. \( f^{(3)}(j) = f(f(f(j))) \)

Ex. \( f^{(2)}(6) = 1 \)

\( f(6) = 2, f(2) = 1 \)

That is, \( f^{(m)}(j) \) is just \( f \) applied \( m \) times to \( j \)

Suppose once again that \( M_p \) is in state \( j \) having read \( t_1, t_2, \ldots, t_k \) and \( t_k \neq P_{j+1} \)
At trip point $M_p$ applies its failure function repeatedly to $j$ until it finds the smallest value of $m$ for which:

**Case 1.** $f^{(m)} (j) = 0$ and $t_{k+1} = P_{i+1}$

**Case 2.** $f^{(m)} (j) = 0$ and $t_{k+1} \neq P_i$

That is, $M_p$ backs up through states $f^{(1)} (j), f^{(2)} (j), ...$ until we get to either **Case 1** or
Case 2 holds for \( f(\text{su}) \) but not for \( f(\text{m}^{-1}) \) for \( f(\text{T}_j) \).

In Case 1: \( M_p \) enters state \( u+1 \).

In Case 2: \( M_p \) enters state \( 0 \).

In either case, the tape head advances to \( t_{k+2} \).

In Case 1: it is easy to see that if \( p_1 p_2 \ldots p_j \) was the longest prefix of \( p \) that is also a suffix of \( t_1 t_2 \ldots t_k \) then \( p_1 p_2 \ldots P_f(\text{m})(j)+1 \) is the longest.
Prefix of $p$ that is also a suffix of $t_1 t_2 \ldots t_k t_{k+1}$

In Case 2: no prefix of $p$ is a suffix of $t_1 t_2 \ldots t_k t_{k+1}$

. $M_p$ then proceeds processing the text symbol $t_{k+2}$.

. $M_p$ continues operating in this fashion either until it enters the final state (in which case we know that the last text symbol
constitute an instance of the pattern
\[ p = P_1 P_2 \cdots P_r, \]
or until \( M_p \) has processed the last symbol of text \( t \) without entering the final state 1, in which case we know that \( p \) is not a substring of \( t \).

\[ \text{EXAMPLE} \]

INPUT: \( p = a a b b a a b \)
\( t = a b a a b a a b b a b a b a a b \)
failure (see table) \( \Sigma = \{a, b\} \)

\[ p = a a b b a a a b \]

\[ t = a b a a b a a b a a b a a b a a b a a b \]

State:

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 1 & 2 & 3 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}
\]

At pos 6-12

is the pattern \( P \)

**FAILURE FUNCTION ALGORITHM**

**INPUT:** \( p = p_1 p_2 \cdots p_e, e \geq 1 \)

**OUTPUT:** \( f \), The failure function for \( p \)
\begin{aligned}
\text{BEGIN} \\
L_1 & \quad f(i) = 0, \quad i = 0 \\
L_2 & \quad \text{FOR } j = 2 \text{ to } n \text{ DO} \\
L_3 & \quad \text{BEGIN} \\
L_4 & \quad \quad i = f(j-1) \\
L_5 & \quad \quad \text{WHILE } p_j \neq p_{i+1} \text{ AND } i > 0 \\
L_6 & \quad \quad \quad \text{DO } i = f(i) \\
L_7 & \quad \quad \quad \text{IF } p_j \neq p_{i+1} \text{ AND } i = 0 \\
L_8 & \quad \quad \quad \quad \text{THEN } f(j) = 0 \\
L_9 & \quad \quad \quad \quad \text{ELSE } f(j) = i + 1 \\
L_10 & \quad \text{END} \\
\text{END}
\end{aligned}
\[ p = a a b b a a b \]

\[ j = 1 \quad f(i) = 0, j \cdot i = 0 \]

\[ j = 2 \quad i = f(j - 1) = f(1) = 0 \]

\[ \text{while } p_j \neq p_{i+1} \text{ and } i > 0 \]

\[ p_2 \neq p_1 \text{ and } i > 0? \]

\[ \text{if } p_j \neq p_{i+1} \text{ and } i = 0 \]

\[ \text{NOT } \]

\[ \text{else } f(j) = i + 1 = 1 \]

\[ j = 3 \quad i = f(j - 1) = f(2) = 1 \]

\[ \text{while } p_3 \neq p_2 \text{ and } i > 0 \]

\[ b \neq a \text{ YES } \]

\[ \text{YES} \]

\[ i = f(i) = 0 \]
If $p_3 + p_2 = \varphi \wedge i = 0$

$j = 4 \quad j = 4$

$i = \mathcal{F}(j-1) = \mathcal{F}(3) = 0$

which

$p_4 \neq p_1$ and $i > 0$

$s \neq a$ and $i > 0$

If $p_4 \neq p_1$ and $i = 0$

Then $\mathcal{F}(4) = 0$

$j = 5$

$i = \mathcal{F}(j-1) = \mathcal{F}(4) = 0$

while $p_5 \neq p_1$ and $i > 0$

no
\[ i = 5 \neq p_1 \land i = 0 \quad \text{YES} \]
\[ \text{ELSE } f(5) = 1 \]

\[ j = 6 \]
\[ i = f(j - 1) = f(5) = 1 \]

\[ \text{while } p_j \neq p_{i+1} \land i = 0 \]
\[ a \neq a \quad \text{NO} \]
\[ \text{ELSE } f(6) = 2 \]

\[ j = 7 \]
\[ i = f(j - 1) = f(6) = 2 \]
while \( p_1 \neq p_3 \) and \( i < 5 \)

\[ \text{not} \]

\[ (\text{if } p_1 \neq p_3 \text{ and } i = 0 \text{ then true}) \]

\[ \text{else } \ell(?) = i+i = 3 \]

**THEOREM**

The Failure Function Algorithm computes \( \ell \) in \( O(|e|) \) steps where \( |e| = |p_1| \cdot (\text{length of } p) \)

**PROOF**

L.3 and L.5 have constant computation time/cost, i.e., the number of time units
The cost of the while statement is proportional to the number of times \( i \) is decreased by the value \( f(i) \) on line 24 following the do. Remember: by definition \( f(i) < i \). So decrease indeed.

The only way \( i \) increases is by the assignment
\[ f(j) = i \times 1 \] by \textit{L6} then incrementing \( j \) by 1 at \textit{L2} and then setting \( i = f(j-1) \) at \textit{L3}

Since \( i = 0 \) initially, and \textit{L6} is executed \( n-1 \) times, we conclude that the \underline{while} statement on \textit{L4} cannot be executed more than \( n \) times.
Thus the total cost of executing L4 is $O(t)$. The remaining instructions in the algorithm take $O(p)$ time, and thus the entire algorithm is $O(t)$. 

Q E D
General time complexity of the KMP Algorithm

We can show by induction (like in proof of correctness of the algorithm) that the pattern matching machine \( M_p \) will be in state \( q_i \) after reading the
if and only if

$p_1p_2\ldots p_i$ is the longest prefix of $p$

that is a suffix of $t_1t_2\ldots t_k$.

Thus $M_p$ correctly finds the leftmost occurrence of $p$ in
the text

$t = t_1\ldots t_n$
By the correctness of the Failure Function Algorithm, showing linear time for computing it, we can show that $M_p$ will execute at most $2 \cdot |I| \cdot |S|$ state transitions when processing input $t$.

Thus we can determine whether $p$ is a substring.
of $t$ by tracing out the state transitions of $M_p$ on input $t$.

To do this all we need is the failure function $\text{fail}$ for $p$.

This can be constructed in time $O(|p|) = O(e)$.

Thus we can determine whether $p$ is a substring of $t$ in
$O(1/p1 + 1/p2)$ time independent of the alphabet size.