Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

December 18, 2011
- Definition
- History
- Components of KMP
- Algorithm
- Example
- Run-Time Analysis
- Advantages and Disadvantages
- References
Definition:

- Best known for linear time for exact matching.
- Compares from left to right.
- Shifts more than one position.
- Preprocessing approach of Pattern to avoid trivial comparisons.
- Avoids recomputing matches.
This algorithm was conceived by Donald Knuth and Vaughan Pratt and independently by James H.Morris in 1977.
Knuth, Morris and Pratt discovered first linear time string-matching algorithm by analysis of the naive algorithm.

It keeps the information that naive approach wasted gathered during the scan of the text. By avoiding this waste of information, it achieves a running time of $O(m + n)$.

The implementation of Knuth-Morris-Pratt algorithm is efficient because it minimizes the total number of comparisons of the pattern against the input string.
Components of KMP:

- The prefix-function $\pi$:
  - It preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself.
  - It is defined as the size of the largest prefix of $P[0..j-1]$ that is also a suffix of $P[1..j]$.
  - It also indicates how much of the last comparison can be reused if it fails.
  - It enables avoiding backtracking on the string ‘$S$’.
Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

\( m \leftarrow \text{length}[p] \)
\( a[1] \leftarrow 0 \)
\( k \leftarrow 0 \)

\textbf{for} \( q \leftarrow 2 \) \textbf{to} \( m \) \textbf{do}

\hspace{1em} \textbf{while} \( k > 0 \) \textbf{and} \( p[k + 1] \neq p[q] \) \textbf{do}

\hspace{2em} \( k \leftarrow a[k] \)

\hspace{1em} \textbf{end while}

\hspace{1em} \textbf{if} \( p[k + 1] = p[q] \) \textbf{then}

\hspace{2em} \( k \leftarrow k + 1 \)

\hspace{1em} \textbf{end if}

\hspace{1em} \( a[q] \leftarrow k \)

\textbf{end for}

\textbf{return} \( \square \)

Here \( a = \square \)
Let us consider an example of how to compute $\sqcap$ for the pattern ‘$p$’.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>c</th>
<th>a</th>
</tr>
</thead>
</table>

Initially: $m = \text{length}[p] = 7$

$\sqcap[1] = 0$

$k = 0$

where $m$, $\sqcap[1]$, and $k$ are the length of the pattern, prefix function and initial potential value respectively.
Step 1: $q = 2, \ k = 0$
$\sqcup[2] = 0$

<table>
<thead>
<tr>
<th>q</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>p</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 2: $q = 3, \ k = 0$
$\sqcup[3] = 1$

<table>
<thead>
<tr>
<th>q</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>p</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 3: $q = 4$, $k = 1$
$\sqcup[4] = 2$

<table>
<thead>
<tr>
<th>q</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>$p$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 4: $q = 5$, $k = 2$
$\sqcup[5] = 3$

<table>
<thead>
<tr>
<th>q</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>$p$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 5: \( q = 6, \ k = 3 \)
\[ \sqcap[6] = 1 \]

\[
\begin{array}{ccccccc}
q & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
p & a & b & a & b & a & b & a \\
\sqcap & 0 & 0 & 1 & 2 & 3 & 1 & \\
\end{array}
\]

Step 6: \( q = 7, \ k = 1 \)
\[ \sqcap[7] = 1 \]

\[
\begin{array}{ccccccc}
q & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
p & a & b & a & b & a & c & a \\
\sqcap & 0 & 0 & 1 & 2 & 3 & 1 & 1 \\
\end{array}
\]
After iterating 6 times, the prefix function computations is complete:

<table>
<thead>
<tr>
<th>q</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>p</td>
<td>a</td>
<td>b</td>
<td>A</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>□</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The running time of the prefix function is $O(m)$. 
Step 1: Initialize the input variables:
  n = Length of the Text.
  m = Length of the Pattern.
  \( \mathfrak{n} \) = Prefix–function of pattern (p).
  q = Number of characters matched.

Step 2: Define the variable:
  q = 0, the beginning of the match.

Step 3: Compare the first character of the pattern with first character of text.
  If match is not found, substitute the value of \( \mathfrak{n}[q] \) to q.
  If match is found, then increment the value of q by 1.

Step 4: Check whether all the pattern elements are matched with the text elements.
  If not, repeat the search process.
  If yes, print the number of shifts taken by the pattern.

Step 5: look for the next match.
n ← length[S]
m ← length[p]
a ← Compute Prefix function
q ← 0
for i ← 1 to n do
    while q > 0 and p[q + 1] ≠ S[i] do
        q ← a[q]
        if p[q + 1] = S[i] then
            q ← q + 1
        end if
    end while
end for

Here a = □
Now let us consider an example so that the algorithm can be clearly understood.

String | b | a | c | b | a | b | a | b | a | b | a | c | a | a | b

Pattern | a | b | a | b | a | c | a

Let us execute the KMP algorithm to find whether ‘p’ occurs in ‘S’.
Initially: \( n = \text{size of } S = 15; \)
\( m = \text{size of } p = 7 \)

Step 1: \( i = 1, q = 0 \)
comparing \( p[1] \) with \( S[1] \)

String \( b \ a \ c \ b \ a \ b \ a \ b \ a \ b \ a \ c \ a \ a \ b \)

Pattern \( a \ b \ a \ b \ a \ c \ a \)

\( P[1] \) does not match with \( S[1] \). ‘p’ will be shifted one position to the right.

Step 2: \( i = 2, q = 0 \)

String \( b \ a \ c \ b \ a \ b \ a \ b \ a \ b \ a \ b \ a \ c \ a \ a \ b \)

Pattern \( a \ b \ a \ b \ a \ c \ a \)
Step 3: $i = 3, q = 1$

<table>
<thead>
<tr>
<th>String</th>
<th>b a c b a b a b a b a c a a b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>a b a b a c a</td>
</tr>
</tbody>
</table>

Backtracking on $p$, comparing $p[1]$ and $S[3]$
Step 4: $i = 4, q = 0$

<table>
<thead>
<tr>
<th>String</th>
<th>b a c b a b a b a b a c a a b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>a b a b a c a</td>
</tr>
</tbody>
</table>
Step 5: $i = 5, q = 0$

String $b\ a\ c\ b\ a\ b\ a\ b\ a\ b\ c\ a\ a\ b$

Pattern $a\ b\ a\ b\ a\ c\ a\ a$

Step 6: $i = 6, q = 1$

String $b\ a\ c\ b\ a\ b\ a\ b\ a\ b\ a\ b\ a\ c\ a\ a\ b$

Pattern $a\ b\ a\ b\ a\ c\ a\ a$
Step 7: $i = 7$, $q = 2$


String $bacbacbaabababacaabb$

Pattern $ababaca$

Step 8: $i = 8$, $q = 3$


String $bacbacbaabababacaabb$

Pattern $ababaca$
Step 9: \( i = 9,\ q = 4 \)


<table>
<thead>
<tr>
<th>String</th>
<th>b</th>
<th>a</th>
<th>c</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>c</th>
<th>a</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Step 10: \( i = 10,\ q = 5 \)


\( p[6] \) doesn’t matches with \( S[10] \)

| String  | b | a | c | b | a | b | a | b | a | b | a | b | a | b | a | c | a | a | b |
| Pattern | a | b | a | b | a | c | a | a | c | a | a | b |

Backtracking on \( p \), comparing \( p[4] \) with \( S[10] \) because after mismatch \( q = \sqcap[5] = 3 \)
Step 11: $i = 11$, $q = 4$

String $babcabababaacaab$

Pattern $ababaca$

Step 12: $i = 12$, $q = 5$

String $babcabababaacaab$

Pattern $ababaca$
Step 13: $i = 13$, $q = 6$


String $bacbababababacaab$

Pattern $ababaca$

pattern ‘$p$’ has been found to completely occur in string ‘$S$’. The total number of shifts that took place for the match to be found are: $i - m = 13 - 7 = 6$ shifts.
Run-Time analysis:

- $O(m)$ - It is to compute the prefix function values.
- $O(n)$ - It is to compare the pattern to the text.
- Total of $O(n + m)$ run time.
Advantages and Disadvantages:

- Advantages:
  - The running time of the KMP algorithm is optimal \( O(m + n) \), which is very fast.
  - The algorithm never needs to move backwards in the input text \( T \). It makes the algorithm good for processing very large files.
Advantages and Disadvantages:

- Disadvantages:
  
  ★ Doesn’t work so well as the size of the alphabets increases. By which more chances of mismatch occurs.
