The Knuth-Morris-Pratt algorithm

Kshitij Bansal

Chennai Mathematical Institute
Undergraduate Student

August 25, 2008
Introduction

The Problem

Naive algorithm

Knuth-Morris-Pratt algorithm

Can we do better?

Improved algorithm

Computing $f$

Final remarks

Conclusion

Think/read about

References

Thankyou
The Problem

Given a piece of text, find if a smaller string occurs in it.
The Problem

Given a piece of text, find if a smaller string occurs in it.

Let $T[1..n]$ be an array which holds the text. Call the smaller string to be searched pattern: $p[1..m]$. 
Naive Algorithm

Naive-String-Matcher( $T[1..n], P[1..m]$ )

- Time complexity: $O(mn)$.
- Space complexity: $O(1)$. 
Naive Algorithm

Naive-String-Matcher( $T[1..n], P[1..m]$ )

1. for $i$ from 0 to $n - m$
Naive Algorithm

Naive-String-Matcher( $T[1..n], P[1..m]$)

1  for $i$ from $0$ to $n - m$
2    if $P[1..m] = T[i + 1..i + m]$
Naive Algorithm

Naive-String-Matcher( $T[1..n], P[1..m]$ )

1. for $i$ from 0 to $n - m$
2. if $P[1...m] = T[i + 1...i + m]$
3. print “Pattern found starting at position”, $i$
Naive Algorithm

Naive-String-Matcher( $T[1..n], P[1..m]$ )

1. for $i$ from 0 to $n - m$
2. if $P[1...m] = T[i + 1...i + m]$
3. print “Pattern found starting at position”, $i$

- Time complexity: $O(mn)$.
- Space complexity: $O(1)$. 
Introduction
   The Problem
   Naive algorithm

Knuth-Morris-Pratt algorithm
   Can we do better?
   Improved algorithm
   Computing $f$

Final remarks
   Conclusion
   Think/read about
   References
   Thankyou
Can we do better?

Text: \textit{abaabaabac}
Pattern: \textit{abaabac}
Can we do better?

Text: *abaabaabac*
Pattern: *abaabac*

The naive algorithm when trying to match the seventh character of the text with the pattern fails. It discards all information about text read from first seven characters and starts afresh. Can we somehow use this information and improve?
Improved algorithm

Improved-String-Matcher( $T[1..n], P[1..m], f[1..m]$ )

1. $q = 0$ \hspace{1cm} (number of characters matched)
Improved algorithm

Improved-String-Matcher( T[1..n], P[1..m], f[1..m] )

1. \( q = 0 \) \hspace{1cm} (number of characters matched)
2. \textbf{for} \( i \) \textbf{from} 1 \textbf{to} \( n \):
Improved algorithm

Improved-String-Matcher( T[1..n], P[1..m], f[1..m] )

1  \( q = 0 \)  \hspace{1cm} (number of characters matched)
2  for \( i \) from 1 to \( n \):
3     if \( P[q + 1] = T[i] \):
4         \( q = q + 1 \)
Improved algorithm

Improved-String-Matcher(\(T[1..n], P[1..m], f[1..m]\))

1. \(q = 0\) \(\text{(number of characters matched)}\)
2. \textbf{for} \(i\) \textbf{from} 1 \textbf{to} \(n\):
3. \hspace{1em} \textbf{if} \(P[q+1] = T[i]\):
4. \hspace{2em} \(q = q + 1\)
5. \hspace{1em} \textbf{else if} \(q > 0\)
6. \hspace{2em} \(q = f[q]\)

Does this really help? What happens when text is \(abaabadaba\)?

Note that \(q\) can increase by at most 1 at each step.
Improved algorithm

Improved-String-Matcher\((T[1..n], P[1..m], f[1..m])\)

1. \(q = 0\) \((\text{number of characters matched})\)
2. for \(i\) from 1 to \(n\):
3. \hspace{1em} if \(P[q + 1] = T[i]\):
4. \hspace{2em} \(q = q + 1\)
5. \hspace{1em} else if \(q > 0\)
6. \hspace{2em} \(q = f[q]\)
7. \hspace{2em} goto line 3

Does this really help? What happens when text is `abaabadaba`...? Note that \(q\) can increase by at most 1 at each step.
Improved algorithm

Improved-String-Matcher( T[1..n], P[1..m], f[1..m] )

1  \( q = 0 \) \hspace{1cm} \textit{(number of characters matched)}
2  \textbf{for} i \textbf{from} 1 \textbf{to} n:
3    \hspace{1cm} \textbf{if} P[q + 1] = T[i]:
4        \hspace{1cm} q = q + 1
5    \hspace{1cm} \textbf{else if} q > 0
6        \hspace{1cm} q = f[q]
7    \hspace{1cm} \textbf{goto} \hspace{0.5cm} \text{line 3}

Does this really help? What happens when text is \textit{abaabadaaba}...?
Improved algorithm

Improved-String-Matcher( $T[1..n], P[1..m], f[1..m]$ )

1. $q = 0$  \hspace{1cm} (number of characters matched)
2. for $i$ from 1 to $n$:
3. \hspace{1cm} if $P[q + 1] = T[i]$:
4. \hspace{2cm} $q = q + 1$
5. \hspace{1cm} else if $q > 0$
6. \hspace{2cm} $q = f[q]$
7. \hspace{1cm} goto line 3

Does this really help? What happens when text is $abaabadaba$...? Note that $q$ can increase by atmost 1 at each step.
Computing $f$

- $f[i]$ denotes the longest *proper* suffix of $P[1 \ldots i]$ which is a prefix of $P[1 \ldots n]$
Computing $f$

- $f[i]$ denotes the longest *proper* suffix of $P[1 \ldots i]$ which is a prefix of $P[1 \ldots n]$
- Just match the pattern with itself.
Introduction
  The Problem
  Naive algorithm

Knuth-Morris-Pratt algorithm
  Can we do better?
  Improved algorithm
  Computing $f$

Final remarks
  Conclusion
  Think/read about
  References
  Thankyou
Conclusion

KMP algorithm

- Time complexity: $O(n + m)$
Conclusion

KMP algorithm

- Time complexity: $O(n + m)$
- Space complexity: $O(m)$
Think/read about

- Number of distinct substrings in a string
Think/read about

- Number of distinct substrings in a string
- Multiple patterns and single text
Think/read about

- Number of distinct substrings in a string
- Multiple patterns and single text
- Matching regular expressions
References


Thank you

Questions?
The End