The Knuth-Morris-Pratt algorithm

Kshitij Bansal

Chennai Mathematical Institute
Undergraduate Student

August 25, 2008

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($T[1..n]$,$P[1..m]$)

Naive Algorithm

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

1 **for** i **from** 0 **to** $n - m$

Naive Algorithm

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

for i **from** 0 **to** $n - m$

  if $P[1 \ldots m] = T[i+1 \ldots i+m]$

Naive Algorithm

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

for i **from** 0 **to** $n - m$
    if $P[1 \ldots m] = T[i + 1 \ldots i + m]$
        print "Pattern found starting at position ", i

Naive Algorithm

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

```
for i from 0 to n − m
    if P[1 ... m] = T[i + 1 ... i + m]
        print "Pattern found starting at position ", i
```

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

Can we do better?

Text: *abaabaabac*
Pattern: *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 $q = 0$ *(number of characters matched)*

Improved algorithm

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

$q = 0$ *(number of characters matched)*

for i **from** 1 **to** n:

Improved algorithm

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

$q = 0$ *(number of characters matched)*

for i **from** 1 **to** n:

    if $P[q+1] = T[i]$:

        $q = q + 1$

Improved algorithm

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

```
q = 0          (number of characters matched)
for i from 1 to n:
    if P[q + 1] = T[i]:
        q = q + 1
    else if q > 0
        q = f[q]
```

Improved algorithm

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

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

Improved algorithm

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

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

Does this really help? What happens when text is *abaababadaba*...?

Improved algorithm

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

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

Does this really help? What happens when text is *abaababadaba*...?
Note that q can increase by atmost 1 at each step.

Computing f

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

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.

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 expresssions

References

- Thomas H. Cormen; Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2001). “Section 32.4: The Knuth-Morris-Pratt algorithm”, *Introduction to Algorithms*, Second edition, MIT Press and McGraw-Hill, 923931. ISBN 978-0-262-03293-3.
- The original paper: Donald Knuth; James H. Morris, Jr, Vaughan Pratt (1977). “Fast pattern matching in strings”. *SIAM Journal on Computing* **6** (2): 323350. doi:10.1137/0206024.
- An explanation of the algorithm and sample C++ code by David Eppstein: http://www.ics.uci.edu/ eppstein/161/960227.html

Thank you

Questions?

The End