Knuth-Morris-Pratt

String searching algorithm

By: Brandon Stafford

Date: April 12, 2005

Overview

- What is string searching
- Why do we need string searching
- What is the Knuth-Morris-Pratt Algorithm
- How does it Work
- Time Complexity
- Examples

What is string Searching and why do we need it

- Finding a given pattern in a string
  - Given the string "abogogwithgshls"
  - Find pattern "gog"
- Used in text searching programs
  - Search and replace
  - Find word
  - We can search any type of string
  - Binary strings, text, numerical data

What is the Knuth-Morris-Pratt algorithm

- Based on Naïve algorithm
- Reduces the number of comparisons
  - Uses information learned in inner loop to determine how far to skip in the outer loop

How does it work

- Uses pattern to pre-compute the number of skips
- Then searches like the Naïve algorithm
- When a mismatch is found, uses the pre-computed number of skips, to determine how far to skip

pseudo code

```
n=length(T)
M=Length(P)
Pre=Compute-Prefix-Function(P)
q=0
For i=1 to n
    do while q>0 and P[q+1]!=T[i]
        do q=pre[q]
    if P[q+1] = T[i]
        then q=q+1
    if q=m
        then print "pattern occurs
                with shift" i-m
                q=pre[q]
```

```
Compute-Prefix-Function(P)
M=length[P]
Pre[1] =0
K=0
For q=2 to m
        do while k>0 and
        P[k+1] != P[q]
            do k=P[q]
        if P[k+1]=P[q]
            then k=k+1
        pre[q] =k
Return pre
```

Time Complexity

- KMP Matcher Θ(n)
  - N is length of text
- Compute prefix function Θ(m)
  - M is length of pattern

Examples

Code from
http://www-igm.univ-mlv.fr/~lecroq/string/node8.html

- Example C code

PreComputeFunction

```
void preKmp(char *x, int m, int kmpNext[])
{
int i, j; i = 0; j = kmpNext[0] = -1;
 while (i < m) {
     while (j > -1 && x[i] != x[j])
         j = kmpNext[j];
     i++; j++;
     if (x[i] == x[j])
         kmpNext[i] = kmpNext[j];
     else kmpNext[i] = j; } }
```

```
void KMP(char *x, int m, char *y, int n) {
 int i, j, kmpNext[XSIZE];
/* Preprocessing */ preKmp(x, m, kmpNext);
/* Searching */
i = j = 0;
 while (j < n) {
     while (i > -1 && x[i] != y[j])
         i = kmpNext[i];
     i++; j++;
     if (i >= m)
         { OUTPUT(j - i);
           i = kmpNext[i]; }
 } }
```

Computing the Precompute Function

Example pattern
"abcd"

i	0	1	2	3	4
P[i]	A	B	C	d	
Pre[i]	-1	0	0	0	0

Precompute Function Continued

- Example Pattern
  "aaababa"

i	0	1	2	3	4	5	6	7
P[i]	a	a	a	b	a	b	a	
Pre[i]	-1	-1	-1	2	-1	1	-1	1

String matching example

- Text = "ddabcdeddc"
- Pattern = "abcd"

PreComputeValues
Pre[0]=-1
[1]=0
[2]=0
[3]=0
[4]=0

Step1:

ddabcdeddc

abcd

Mismatch, so move 0 - -1 to the right

Cont

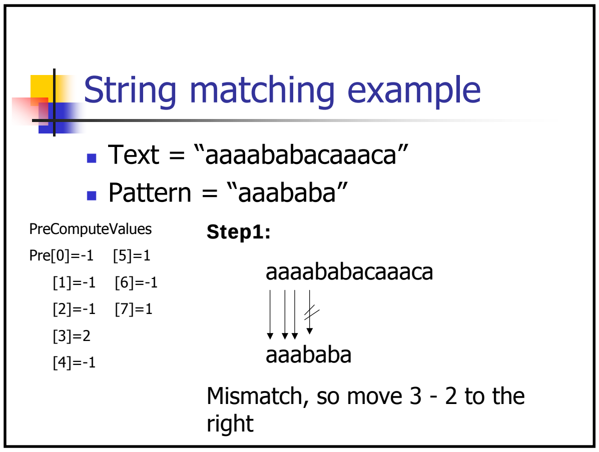
Step 2:
dabcdeddc

abcd

Match, so move 4 – 0 places to the right

Step 3:
Ddabcdeddc

abcd   Mismatch, so you are done.



String matching example

- Text = “aaaababacaaaca”
- Pattern = “aaababa”

PreComputeValues

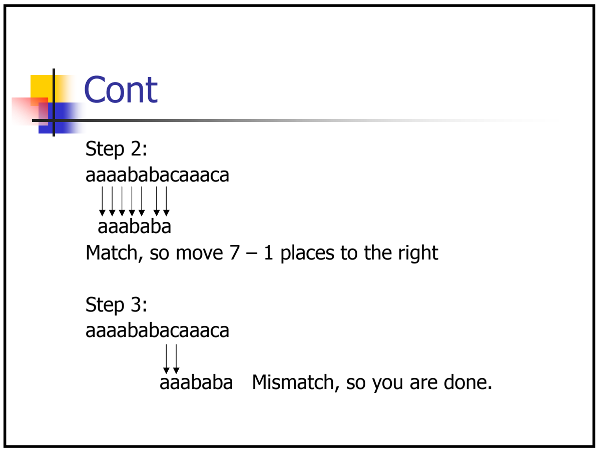
Pre[0]=-1 [5]=1
[1]=-1 [6]=-1
[2]=-1 [7]=1
[3]=2
[4]=-1

Step1:

aaaababacaaaca

aaababa

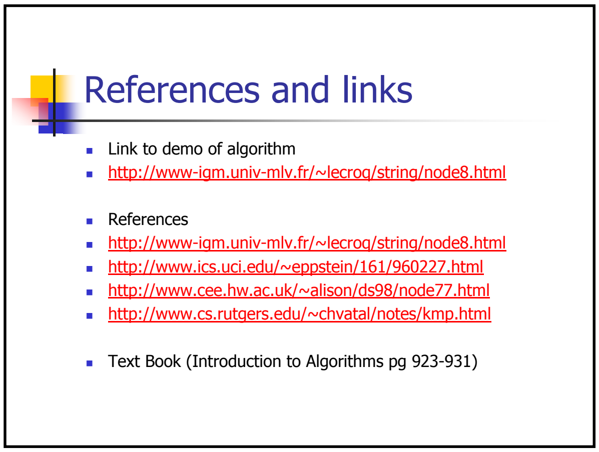
Mismatch, so move 3 - 2 to the right



Cont

Step 2:
aaaababacaaaca
aaababa
Match, so move 7 – 1 places to the right

Step 3:
aaaababacaaaca
aaababa Mismatch, so you are done.



References and links

- Link to demo of algorithm
- http://www-igm.univ-mlv.fr/~lecroq/string/node8.html

- References
- http://www-igm.univ-mlv.fr/~lecroq/string/node8.html
- http://www.ics.uci.edu/~eppstein/161/960227.html
- http://www.cee.hw.ac.uk/~alison/ds98/node77.html
- http://www.cs.rutgers.edu/~chvatal/notes/kmp.html

- Text Book (Introduction to Algorithms pg 923-931)