Text Algorithms (6EAP)
Lecture 3: Exact pattern matching II

Jaak Vilo
2010 fall

Algorithms
• Brute force \( O(nm) \)
• Knuth-Morris-Pratt \( O(n) \)
• Karp-Rabin
• Shift-OR, Shift-AND
• Boyer-Moore
• Factor searches

Find occurrences in text

• R. Boyer, S.Moore: A fast string searching algorithm. *CACM* 20 (1977), 762-772 [PDF]

Find occurrences in text

• Have we missed anything?

• What have we learned if we test for a potential match from the end?
Our search algorithm may be specified as follows:

\[ \text{strings} \leftarrow \text{length of string} \]
\[ i \leftarrow \text{pattern length} \]
\[ \text{top: if } i > \text{strings then return false.} \]
\[ j \leftarrow \text{pattern length} \]
\[ \text{loop: if } j = 0 \text{ then return } j + 1. \]
\[ \text{if } \text{string}(i) = \text{pat}(j) \]
\[ \text{then} \]
\[ j = j - 1. \]
\[ i = i - 1. \]
\[ \text{goto loop.} \]
\[ \text{else:} \]
\[ i = i + \max(\text{delta}, (\text{string}(i)), \text{delta}, j)). \]
\[ \text{goto top.} \]

If the above algorithm returns false, then \text{pat} does not occur in \text{string}. If the algorithm returns a number, then it is the position of the left end of the first occurrence of \text{pat} in \text{string}.

**Bad character heuristics**

maximal shift on \( S[i] \)

\[
\begin{align*}
\text{void bmInitocc() } & \{ \\
\text{char a; int j; } & \\
\text{for( a=0; a < alphabetsize; a++) } & \text{occ[a]=-1;} \\
\text{for (j=0; j<} & \text{m; j++) } \\
\text{a=p[j]; } & \text{occ[a]=j; } \\
\text{\}} \\
\end{align*}
\]

**Good suffix heuristics**

\[
\begin{align*}
\text{void preprocess_BM() } & \{ \text{// delta1 and delta2 } \\
\text{i=} & \text{m;} \\
\text{while i <= n } & \{ \\
\text{for( j=m; j>0 } & \text{and P[j]=S[i-m+j]; j-- ) } \\
\text{if j=0 } & \text{report match at position i-m+1 } \\
\text{i = i + max(\text{delta}, S[i)], delta2[j]) } \\
\text{\}} \\
\end{align*}
\]

**Boyer-Moore algorithm**

Input: Text \( S[1..n] \) and pattern \( P[1..m] \)
Output: Occurrences of \( P \) in \( S \)

\[
\begin{align*}
\text{while i <= n } & \{ \\
\text{for( j=m; j>0 } & \text{and P[j]=S[i-m+j]; j-- ) } \\
\text{if j=0 } & \text{report match at position i-m+1 } \\
\text{i = i + max(\text{delta1}[ S[i] ], \text{delta2}[ j ]) } \\
\end{align*}
\]
• http://www.itc.fh-flensburg.de/lang/algorithmen/pattern/bmen.htm
• http://www.iwm.univ-mlv.fr/~lecroq/string/
• Animation: http://www-igm.univ-mlv.fr/~lecroq/string/

Simplifications of BM

• There are many variants of Boyer-Moore, and many scientific papers.
• On average the time complexity is sublinear
• Algorithm speed can be improved and yet simplify the code.
• It is useful to use the last character heuristics (Horspool (1980), Baeza-Yates(1989), Hume and Sunday(1991)).

Algorithm BMH (Boyer-Moore-Horspool)

• RN Horspool - Practical Fast Searching in Strings
  Software - Practice and Experience, 10(6):501-506 1980

Input: Text S[1..n] and pattern P[1..m]
Output: occurrences of P in S
1. for a in Σ do δ[a] = m
2. for j=1..m-1 do δ[P[j]] = m-j
3. i=m
4. while i <= n do
5.   if S[i] == P[m] then
6.     j = m-1
7.     while (j>0 and P[j]==S[i-m+j]) j = j-1;
8.     if j==0 report match at i-m+1
9.     i = i + δ[S[i]];

Algorithm Boyer-Moore-Horspool-Hume-Sunday (BMHHS)

• Use delta in a tight loop
• If match (delta=0) then check and apply original delta d

Input: Text S[1..n] and pattern P[1..m]
Output: occurrences of P in S
1. for a in Σ do δ[a] = m
2. for j=1..m-1 do δ[P[j]] = m-j
3. d = δ[P[m]]; // memorize d on P[m]
4. δ[P[m]] = 0; // ensure delta on match of last char is 0
5. for ( i=m ; i<= n ; i = i+d )
6.   repeat
7.     t = δ[S[i]] ; i = i + t
8.   until t==0
9.   for ( j=m-1 ; j>0 and P[j]==S[i-m+j] ; j = j-1 )
10.  if j==0 report match at i-m+1

BMHHS requires that the text is padded by P: S[n+1..n+m] = P
(in order for the algorithm to finish correctly – at least one occurrence!)

String Matching: Horspool algorithm

• How the comparison is made?
• Which is the next position of the window?

Then it is necessary a preprocess that determines the length of the shift.

• Daniel M. Sunday: A very fast substring search algorithm [PDF]
  Communications of the ACM August 1990, Volume 33 Issue 8

• Loop unrolling:
  • Avoid too many loops (each loop requires tests) by just repeating code within the loop.
  • Line 7 in previous algorithm can be replaced by:

7. i += δ[S[i]] ;
   i += δ[S[i]] ;
   i += δ[S[i]] ;

9/19/10
Forward-Fast-Search: Another Fast Variant of the Boyer-Moore String Matching Algorithm

- The Prague Stringology Conference ’03
- Domenico Cantone and Simone Faro

Abstract: We present a variation of the Fast-Search string matching algorithm, a recent member of the large family of Boyer-Moore-like algorithms, and we compare it with some of the most effective string matching algorithms, such as Horspool, Quick Search, Tuned Boyer-Moore, Reverse Factor, Berry-Ravindran, and Fast-Search itself. All algorithms are compared in terms of run-time efficiency, number of text character inspections, and number of character comparisons. It turns out that our new proposed variant, though not linear, achieves very good results especially in the case of very short patterns or small alphabets.

PS.gz (local copy)

Factor searches

Do not compare characters, but find the longest match to any subregion of the pattern.

Several hits: use bitparallelism

NDA on the suffixes of ‘announce’

Deterministic version of the same Backward Factor Oracle
String Matching of one pattern
The cost of Brute Force algorithm is $O(nm)$.

Can the search be made with lower cost?

CTACTACGTCTATACGTACTACTGC
TACTACGTATGACTACATGC
TACTACGGTATGACTAA

Prefix search

Suffix search

Factor search

26