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

Jaak Vilo
2010 fall
Find occurrences in text
Algorithms

- Brute force $O(nm)$
- Knuth-Morris-Pratt $O(n)$
- Karp-Rabin
- Shift-OR, Shift-AND
- Boyer-Moore
- Factor searches
• R. Boyer, S. Moore: A fast string searching algorithm. *CACM* 20 (1977), 762-772 [PDF]
Find occurrences in text

• Have we missed anything?
Find occurrences in text

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

\[
\text{stringlen} \leftarrow \text{length of } \text{string}.
\]
\[
i \leftarrow \text{patlen}.
\]
\[
\text{top: if } i > \text{stringlen} \text{ then return false.}
\]
\[
j \leftarrow \text{patlen}.
\]
\[
\text{loop: if } j = 0 \text{ then return } j + 1.
\]
\[
\text{if } \text{string}(i) = \text{pat}(j)
\]
\[
\text{then}
\]
\[
j \leftarrow j - 1.
\]
\[
i \leftarrow i - 1.
\]
\[
goto \text{loop}.
\]
\[
\text{close;}
\]
\[
i \leftarrow i + \max(\text{delta}_1 (\text{string}(i)), \text{delta}_2 (j)).
\]
\[
goto \text{top}.
\]

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}.
Find occurrences in text

P

S

A

B
Bad character heuristics
maximal shift on $S[i]$

\[ \delta_1( S[i] ) = |m| - \text{patlen-j} \]

if pattern does not contain $S[i]$

\[ \text{max } j \text{ so that } P[j] = S[i] \]

First x in pattern (from end)
void bmInitocc() {
    char a; int j;
    for(a=0; a<alphabetsize; a++)
        occ[a]=-1;
    for (j=0; j<m; j++) {
        a=p[j];
        occ[a]=j;  }  
}
Good suffix heuristics

\[ \delta_2(S[i]) - \text{ minimal shift so that matched region is fully covered or that the suffix of match is also a prefix of } P \]
Boyer-Moore algorithm

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

preprocess_BM() // delta1 and delta2  
i=m  
while $i \leq n$
  for( $j=m$; $j>0$ and $P[j]==S[i-m+j]$; $j--$ ) ;  
  if $j==0$ report match at position $i-m+1$
  $i = i + \max( \text{delta1}[ S[i] ], \text{delta2}[ j ] )$
• http://www.iti.fh-flensburg.de/lang/algorithmen/pattern/bmen.htm


• 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 delta[a] = m
2. for j=1..m-1 do delta[P[j]] = m-j
3. i=m
4. while i <= n
5.       if S[i] == P[m]
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 + delta[ S[i] ]
String Matching: Horspool algorithm

• How the comparison is made?

Text: ____________________________
Pattern: ________

From right to left: suffix search

• Which is the next position of the window?

Text: ____________________________ a
Pattern: ________

It depends of where appears the last letter of the text, say it ‘a’ in the pattern:

Then it is necessary a preprocess that determines the length of the shift.
**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** delta[a] = m
2. **for** j=1..m-1 **do** delta[P[j]] = m-j
3. d = delta[P[m]]; // memorize d on P[m]
4. delta[P[m]] = 0; // ensure delta on match of last char is 0
5. for ( i=m ; i<= n ; i = i+d )
   6. **repeat** // skip loop
      7. t=delta[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]..S[n+m] = P
(in order for the algorithm to finish correctly – at least one occurrence!).
• **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:

```c
7.   i += delta[ S[i] ];
     i += delta[ S[i] ];
     i += (t = delta[ S[i] ]); 
```
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.

• http://cs.felk.cvut.cz/psc/event/2003/p2.html
• PS.gz (local copy)
Factor searches

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

Fig. 2.14. Basic search of the BDM algorithm with the suffix automaton. The variable last stores the beginning position of the longest suffix of the part read that is also a prefix of the pattern.

Do not compare characters, but find the longest match to any subregion of the pattern.
Several hits: use bitparallelism

with 0 and 1 as with **Shift-And**. The number 1, representing an active state at position \( j \) of \( p \), means that the factor \( p_j \ldots p_{j+|u|-1} \) is equal to \( u \). Figure 2.15 shows this relationship. If the pattern is of size less than \( w \), then the set fits in a computer word \( D = d_m \ldots d_1 \).

\[\text{Text} \quad \text{Factor search} \quad \quad u\]

\[\text{Pattern} \quad u \quad u\]

\[\text{D table} \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0\]

Fig. 2.15. Bit-parallel factor search. The table \( D \) keeps a list of the positions in \( p \) where the factor \( u \) begins.
NDA on the suffixes of ‘announce’

Fig. 2.17. Nondeterministic automaton recognizing all factors of the reverse string of “announce”.
Deterministic version of the same Backward Factor Oracle

Fig. 2.20. Factor oracle for the reverse string of "announce".
String Matching of one pattern
The cost of Brute Force algorithm is $O(nm)$.

Can the search be made with lower cost?

CTACTACGTCTATACTGATCGTACTACATGC
TACTACCGGTATGACTAA

Prefix search

Suffix search

Factor search
Fig. 2.22. Map of experimental efficiency for different string matching algorithms.

2.6 Other algorithms and references

BNDM - Backward Non-Deterministic DAWG Matching
BOM - Backward Oracle matching
BIG DATA

SCIENCE IN THE PETABYTE ERA

THE BITER BIT
Viral infections for viruses

TROPICAL CYCLONES
The strong get stronger

BLACK HOLE PHYSICS
A new window on the Galactic Centre