Time-Space Optimal String Matching

Zvi Galil and Joel Seiferas, 1982

David Robillard

School of Computer Science
Carleton University

March 21, 2013

The String Matching Problem

Find all full instances of a given “pattern” word x in a “text” string y.

Pattern:	a	b	d						
Text:	a	b	a	b	d	a	a	b	d
Matches:			$\uparrow$				$\uparrow$		

Naïve Algorithm

Search for pattern starting at each index of the text.

Pattern:	a	b	d						
Text:	a	b	a	b	d	a	a	b	d
Search 0:	?	?	X						
Search 1:		X							
Search 2:			?	?	✓				
Search 3:				X					
Search 4:					X				
Search 5:						?	X		
Search 6:							?	?	✓
Search 7:								X	
Search 8:									X

Time Complexity

Searching at every index takes time $O(|x| \cdot |y|)$ in the worst case.

Pattern:	a	a	a	a	a	a	a	b
Text:	a	a	a	a	a	a	a	z
Search 0:	?	?	?	?	?	?	?	X
Search 1:		?	?	?	?	?	?	X
Search 2:			?	?	?	?	?	X
Search 3:				?	?	?	?	X
Search 4:					?	?	?	X
Search 5:						?	?	X
Search 6:							?	X

Improving the Naïve Algorithm

- Can we do better?
- Yes, because when checking the *m*th pattern character:
  - We know the previous *m* text characters match
  - Therefore, there is no need to check them again on failure

The Knuth-Morris-Pratt Algorithm

- Several methods of eliminating these redundant checks have been proposed
- Most well-known is the Knuth-Morris-Pratt algorithm (KMP)
  - Uses a precomputed $O(|x|)$ table
  - Table is used to decide how much to backtrack on a failed match
- Other approaches:
  - $O(\lg(|x|))$ storage
  - Reusing pattern for storage
  - Constant space, but with catches (random numbers, restrictions, possible error, etc.)

Time-Space-Optimal String Matching

The result of this paper[1] is an algorithm with:

- Linear time
- Small constant storage requirements
- Minimal requirements: can be implemented on a six-head two-way finite automaton

Periods

Definition (Period)

String z is a period of string w if w is a prefix of $z^m = zzz\ldots$. Equivalently, z is a period of w if and only if w is a prefix of zw.

Definition (Basic string)

String z is basic if it is not of the form z'^i for any integer $i > 1$.

Definition (Prefix period)

String z is a prefix period of w if it is basic and z^k is a prefix of w.

Another View of Periods

For example, $s =$"abracadabra" has two periods, of length 7 and 10, because for i, $s[i] = s[i+7] = s[i+10]$.

Reach

Definition (Reach)

$$reach_w(p) = \max\{q \leq |w| : [0,p]_w \textit{ is a period of } [0,q]_w\}$$

Periodicity

Periodicity Lemma

If a string of length $p_1 + p_2$ *has periods of lengths* p_1 *and* p_2*, then it has a period of length* $\gcd(p_1, p_2)$*. [3]*

Searching

Several earlier algorithms follow the same general scheme, to scan the text while maintaining:

p Position in text (increasing, ≥ 0)
q Length of pattern prefix known to match starting at p (≥ 0)

- If q reaches $|x|$, then a match has been found.
- Update (p, q) to (p', q') and continue the search.
- The problem of an efficient algorithm is to compute the ideal (p', q') efficiently.

Shift

Earlier work by the authors[2] computed (p', q') as

$$(p', q') = \begin{cases} (p + \text{shift}_x(q), q - \text{shift}_x(q)) & \text{if } \text{shift}_x(q) \leq \frac{q}{k} \\ (p + max(1, \lceil \frac{q}{k} \rceil), 0) & \text{otherwise} \end{cases}$$

for some fixed integer k. Note that:

- The first case is unlikely if k is large
- Only the first case uses the shift function
- It would be nice if we could eliminate that case entirely...

Occurrence of $\mathrm{shift}_x(q) \leq \frac{q}{k}$

Lemma (1)

If $\mathrm{shift}_x(q) \leq \frac{q}{k}$, *then* $[0, \mathit{shift}_x(q)]_x$ *is a prefix period of* x.

Lemma (2)

If $[0, \mathit{shift}]_x$ *is a prefix period of* x, *then*
$\mathrm{shift} = \mathrm{shift}_x(q) \leq \frac{q}{k} \Leftrightarrow k \cdot \mathrm{shift} \leq q \leq \mathrm{reach}_x(\mathrm{shift})$.

Theorem (Decomposition)

Each pattern x *has a parse* $x = uv$ *such that* v *has at most one prefix period and* $|u| = O(\mathit{shift}_v(|v|))$.

Efficient Searching

Given a decomposition $x = uv$, the algorithm searches the text for the suffix v. There are two cases:

1. v has no prefix period, and Lem. 1 guarantees the first case ($shift_x(q) \leq \frac{q}{k}$) never occurs.
2. v has one prefix period of length p_1, and Lem. 1 and Lem. 2 guarantee that the first case occurs only for $kp_q \leq q \leq \text{reach}_v(p_1)$.

Suffix Search

So, when searching for the suffix v, we have

$$(p', q') = \begin{cases} (p + p_1, q - p_1) & \text{if } kp_q \leq q \leq \text{reach}_v(p_1) \\ (p + max(1, \lceil \frac{q}{k} \rceil), 0) & \text{otherwise} \end{cases}$$

with the desired property that a general shift function is not required.
This takes time $O(|v| + |y|)$, since $(k+1)p + q$ increases in O steps.

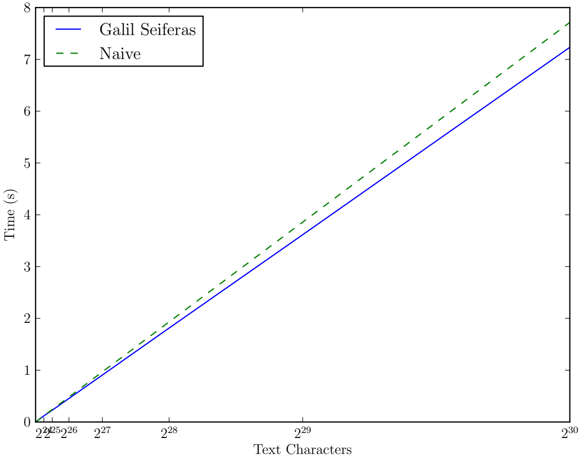
Pattern Search

The algorithm naïvely checks if u precedes every found suffix v. Since v can occur at most $\frac{|y|}{\mathrm{shift}_v(v)}$ times, the total time will be $O(|u|)\frac{|y|}{\mathrm{shift}_v(v)}$.

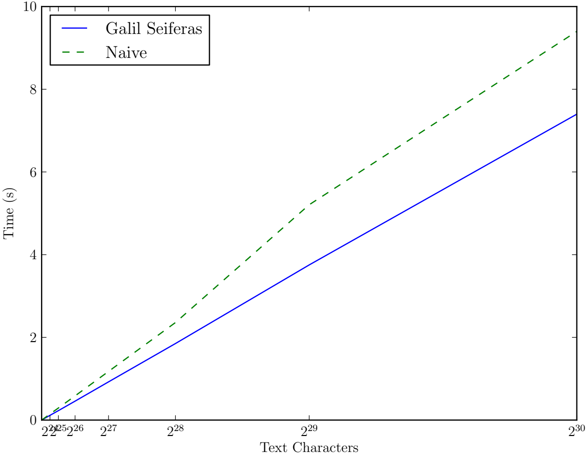
Finding a Decomposition

- Given a decomposition $x = uv$,we can search quickly
- Such a decomposition can be found in linear time
- Details ommitted here, but basic idea is to match x with itself

Performance: $|x| = 16$



Performance: $|x| = \Theta(\lg |y|)$



Limitations and Questions

- Main limitation is that this is not a "real-time" algorithm (must go backwards in text)
- How much time/space is required for a forward-only algorithm?
- How few heads are required?

References

GALIL, Z., AND SEIFERAS, J.
Time-space-optimal string matching.
Journal of Computer and System Sciences 26, 3 (1983), 280–294.

GALIL, Z., AND SEIFERAS, J.
Saving space in fast string-matching.
In *Foundations of Computer Science, 1977., 18th Annual Symposium on* (31 1977-Nov. 2), pp. 179–188.

KNUTH, D. E., MORRIS JR, J. H., AND PRATT, V. R.
Fast pattern matching in strings.
SIAM journal on computing 6, 2 (1977), 323–350.