Computing Longest Common Substrings Using Suffix Arrays

Maxim A. Babenko, Tatiana A. Starikovskaya

Moscow State University

Computer Science in Russia, 2008

Outline

1. Problem Definition

Outline

1. Problem Definition
2. Suffix Arrays

Outline

1. Problem Definition
2. Suffix Arrays
3. The Algorithm

Outline

1. Problem Definition
2. Suffix Arrays
3. The Algorithm
4. Conclusions

Part 1
Problem Definition

LCS

- **Problem** (LCS, Longest Common Substring): Given a collection of N strings $A = \{\alpha_1, \ldots, \alpha_N\}$ and an integer K ($2 \leq K \leq N$) find the longest string β that is a substring of at least K strings in A.

LCS

- **Problem** (LCS, Longest Common Substring): Given a collection of *N* strings $A = \{\alpha_1, \ldots, \alpha_N\}$ and an integer *K* ($2 \leq K \leq N$) find the longest string β that is a substring of at least *K* strings in *A*.
- **Tools:** Suffix Arrays
- **Time and Space:** Linear and alphabet-independent
- **Model of Computation:** RAM

Part 2
Suffix Arrays

Useful Definitions

- **Definition** (Suffix): Let $\omega = \omega_1\omega_2\dots\omega_n$ be an arbitrary string of length n. For each i ($1 \le i \le n$)

$$\omega[i..] = \omega_i\omega_{i+1}\dots\omega_n$$

  is a suffix of ω.
- **Definition** (Lexicographic order): Suppose we have some order on letters of the alphabet Σ. This order can be extended in a standard way to strings over Σ: $\alpha < \beta$ iff either α is proper prefix of β or $\alpha[1] = \beta[1], \dots, \alpha[i] = \beta[i], \alpha[i+1] < \beta[i+1]$.

Suffix Arrays

- **Definition** (Suffix Array): Let ω be an arbitrary string of length n. Consider its non-empty suffixes

$$\omega[1..],\ \omega[2..],\ \ldots,\ \omega[n..].$$

and order them lexicographically. Let $SA(i)$ denote the starting position of the suffix appearing on the i-th place ($1 \le i \le n$):

$$\omega[SA(1)..] < \omega[SA(2)..] < \ldots < \omega[SA(n)..].$$

An Example of Suffix Array

	suffixes	*SA*	sorted suffixes
1	`mississippi`	11	`i`
2	`ississippi`	8	`ippi`
3	`ssissippi`	5	`issippi`
4	`sissippi`	2	`ississippi`
5	`issippi`	1	`mississippi`
6	`ssippi`	10	`pi`
7	`sippi`	9	`ppi`
8	`ippi`	7	`sippi`
9	`ppi`	4	`sissippi`
10	`pi`	6	`ssippi`
11	`i`	3	`ssissippi`

Figure: String `mississippi`, its suffixes, and the corresponding suffix array.

Why Suffix Arrays?

- A **simple** data structure containing all the necessary information.

Why Suffix Arrays?

- A **simple** data structure containing all the necessary information.
- Many **nice** and **simple** efficient construction algoritms (e.g. **Kärkäinen, Sanders [2003]**) with **alphabet-independent** time and space complexity.

Part 3
The Algorithm

Our Main Result

Theorem

Let the total length of strings $\alpha_1, \ldots, \alpha_N$ be equal to L. Then the answer to the LCS problem can be computed in $O(L)$ time and in $O(L)$ space.

LCS Example

- Consider the following **example** with $N = 3$, $K = 2$:

$$\begin{aligned} \alpha_1 &= abb \\ \alpha_2 &= cb \\ \alpha_3 &= abc \end{aligned}$$

Clearly, the answer is *ab*.

Observation

- The longest common substring for K strings of our set is the longest common prefix of some suffixes of these strings.

Observation

- The longest common substring for K strings of our set is the longest common prefix of some suffixes of these strings.
- We calculate **the longest common prefix of every K suffixes** of different strings and take the longest one; the latter is the answer to the LCS problem.

Preprocessing: Step 1

- Combine the strings in A as follows:

$$\alpha = \alpha_1 \$_1 \alpha_2 \$_2 \ldots \alpha_N \$_N.$$

  Here $\$_i$ are special symbols (**sentinels**) that are different and lexicographically less than other symbols of the initial alphabet Σ

Preprocessing: Step 1

- Combine the strings in A as follows:

$$\alpha = \alpha_1 \$_1 \alpha_2 \$_2 \ldots \alpha_N \$_N.$$

  Here $\$_i$ are special symbols (**sentinels**) that are different and lexicographically less than other symbols of the initial alphabet Σ

- **Example**: $\alpha = abb\$_1 cb\$_2 abc\$_3$

Preprocessing: Step 2

- **Definition** (Longest Common Prefixes (LCP) array): The array containing lengths of the longest common prefixes for every pair of consecutive suffixes (w.r.t. lexicographical order).
- LCP array can be easily constructed in linear time and space.

Preprocessing: Step 2

- **Definition** (Longest Common Prefixes (LCP) array): The array containing lengths of the longest common prefixes for every pair of consecutive suffixes (w.r.t. lexicographical order).
- LCP array can be easily constructed in linear time and space.
- We construct **the suffix array** and **the LCP array** for α.

Step 2. Example of SA and LCP

String: abb$\$_1cb\$_2abc\$_3$

SA:	4	7	11	1	8	3	6	2	9	10	5
LCP:	0	0	0	2	0	1	1	1	0	1	

	suffixes	*SA*	sorted suffixes	*LCP*
1	abb$\$_1cb\$_2abc\$_3$	4	$\$_1cb\$_2abc\$_3$	0
2	bb$\$_1cb\$_2abc\$_3$	7	$\$_2abc\$_3$	0
3	b$\$_1cb\$_2abc\$_3$	11	$\$_3$	0
4	$\$_1cb\$_2abc\$_3$	1	abb$\$_1cb\$_2abc\$_3$	2
5	cb$\$_2abc\$_3$	8	abc$\$_3$	0
6	b$\$_2abc\$_3$	3	b$\$_1cb\$_2abc\$_3$	1
7	$\$_2abc\$_3$	6	b$\$_2abc\$_3$	1
8	abc$\$_3$	2	bb$\$_1cb\$_2abc\$_3$	1
9	bc$\$_3$	9	bc$\$_3$	0
10	c$\$_3$	10	c$\$_3$	1
11	$\$_3$	5	cb$\$_2abc\$_3$	

Further Ideas

- The longest prefix of suffixes of K different strings in A is the longest common prefix of suffixes of K different colors in α.

Further Ideas

- The longest prefix of suffixes of K different strings in A is the longest common prefix of suffixes of K different colors in α.
- Consider K suffixes at positions $i_1, \ldots, i_K$ and assume that $SA[i_1] < SA[i_2] < \ldots < SA[i_K]$. The length of the longest common prefix of these K suffixes is equal to the **minimum** of $LCP[i_1], \ldots, LCP[i_K - 1]$.

Further Ideas

- The longest prefix of suffixes of K different strings in A is the longest common prefix of suffixes of K different colors in α.
- Consider K suffixes at positions $i_1, \ldots, i_K$ and assume that $SA[i_1] < SA[i_2] < \ldots < SA[i_K]$. The length of the longest common prefix of these K suffixes is equal to the **minimum** of $LCP[i_1], \ldots, LCP[i_K - 1]$.
- **Example:**

SA:	4	7	11	1	8	3	6	2	9	10	5
LCP:	0	0	0	2	0	1	1	1	0	1	

Suffixes: $\texttt{abb}\$_1\texttt{cb}\$_2\texttt{abc}\$_3$

$\texttt{abc}\$_3$

Extensions

Theorem

- **Problem**: *Given a collection of N strings $A = \{\alpha_1, \ldots, \alpha_N\}$, for each K ($2 \le K \le N$) find the longest string β that is a substring of at least K strings in A.*

Extensions

Theorem

- **Problem**: *Given a collection of* N *strings* $A = \{\alpha_1, \ldots, \alpha_N\}$, *for each* K ($2 \le K \le N$) *find the longest string* β *that is a substring of at least* K *strings in* A.
- *Let the total length of strings* $\alpha_1, \ldots, \alpha_N$ *be equal to* L. *Then the answer to the above problem can be computed in* $O(L \cdot \log^* L)$ *time and in* $O(L)$ *space.*

Part 4
Conclusions

Open Problem

- How to compute **an inexact longest common substring**?

Acknowledgements

The authors are thankful to the students of Department of Mathematical Logic and Theory of Algorithms and to **Maxim Ushakov** and **Victor Khimenko** (Google Moscow) for many helpful discussions.

The End :-)

Thank you for your attention.
Questions are welcome!