Shortest Unique Substring Query Revisited

Atalay Mert İleri
Bilkent University, Turkey

M. Oğuzhan Külekci
Istanbul Medipol University, Turkey

Bojian Xu
Eastern Washington University, USA

June 18, 2014

Outline

1. Preliminary Information
2. Left Bounded Shortest Unique Substrings
3. Finding Shortest Unique Substring For All Positions
4. Experimental Results
5. Conclusion
6. Acknowledgments
7. Q&A

Shortest Unique Substring (SUS)

A substring $S[i \cdots j]$ is called **Shortest Unique Substring** covering a position k, if $i \leq k \leq j$ and there is no other unique substring $S'[i' \cdots j']$ exists such that $i' \leq k \leq j'$ and $j' - i' < j - i$

Shortest Unique Substring (SUS)

A substring $S[i \cdots j]$ is called **Shortest Unique Substring covering a position k**, if $i \leq k \leq j$ and there is no other unique substring $S'[i' \cdots j']$ exists such that $i' \leq k \leq j'$ and $j' - i' < j - i$

Problem 1 (find details in paper)

Given a string location k, find the leftmost SUS covering location k.

Problem 2 (find details in paper)

Given a string location k, find all the SUSes covering location k.

Problem 3 (discussed in this talk)

Find the leftmost SUS covering every string location $1, 2, \ldots, n$.

Problem 4 (find details in paper)

Find all the SUSes covering every string location $1, 2, \ldots, n$.

Why SUS queries ?

- Distinguishing results in a text search
- Finding DNA signatures of organisms
- Extracting the event context in historical events

Preliminaries: Suffix Array and Rank Array

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

Preliminaries: Suffix Array and Rank Array

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	$\Rightarrow$	9	ppi
8	ippi		7	sippi
9	ppi		4	sissippi
10	pi		6	ssippi
11	i		3	ssissippi

Preliminaries: Suffix Array and Rank Array

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

$\Rightarrow$

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

SA	11	8	5	2	1	10	9	7	4	6	3
RANK	5	4	11	9	3	10	8	2	7	6	1

Preliminaries: Longest Common Prefix Array

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

LCP	0	1	1	4	0	0	1	0	2	1	3

Left Bounded Shortest Unique Substring

Definition

For a particular string location $k \in \{1, 2, \ldots, n\}$, the **left-bounded shortest unique substring (LSUS) starting at location k**, denoted as $\mathbf{LSUS_k}$, is a unique substring $S[k \ldots j]$, such that either $k = j$ or any proper prefix of $S[k \ldots j]$ is not unique.

Examples:

mississippi $\longrightarrow LSUS_1 = S[1, 1]$
mississippi $\longrightarrow LSUS_2 = S[2, 6]$
$\vdots$
mississippi $\longrightarrow LSUS_{10} = S[10, 11]$
mississippi $\longrightarrow LSUS_{11}$ does not exist

Fast computation of LSUS'es from LCP array

For $i = 1, 2, \ldots, n$:

$$LSUS_i = \begin{cases} S[i \ldots i + L_i], & \text{if} \quad i + L_i \leq n \\ \textit{not existing}, & \text{otherwise} \end{cases}$$

where $L_i = \max\{LCP[RANK[i]], LCP[RANK[i] + 1]\}$.

Fast computation of LSUS'es from LCP array

mississippi

RANK	5	4	11	9	3	10	8	2	7	6	1
LCP	0	1	1	4	0	0	1	0	2	1	3
L_i	0										
$LSUS_i$	m										

Fast computation of LSUS'es from LCP array

mississippi

RANK	5	4	11	9	3	10	8	2	7	6	1
LCP	0	1	1	4	0	0	1	0	2	1	3
L_i	0	4									
$LSUS_i$	m	issis									

Fast computation of LSUS'es from LCP array

mississippi

RANK	5	4	11	9	3	10	8	2	7	6	1
LCP	0	1	1	4	0	0	1	0	2	1	3
L_i	0	4	3								
$LSUS_i$	m	issis	ssis								

Fast computation of LSUS'es from LCP array

mississippi

RANK	5	4	11	9	3	10	8	2	7	6	1
LCP	0	1	1	4	0	0	1	0	2	1	3
L_i	0	4	3	2	4	3	2	1	1		
$LSUS_i$	m	issis	ssis	sis	issip	ssip	sip	ip	pp		

Fast computation of LSUS'es from LCP array

mississippi

RANK	5	4	11	9	3	10	8	2	7	6	1
LCP	0	1	1	4	0	0	1	0	2	1	3
L_i	0	4	3	2	4	3	2	1	1	1	
$LSUS_i$	m	issis	ssis	sis	issip	ssip	sip	ip	pp	pi	

Fast computation of LSUS'es from LCP array

mississippi

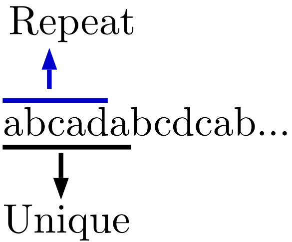
RANK	5	4	11	9	3	10	8	2	7	6	1
LCP	0	1	1	4	0	0	1	0	2	1	3
L_i	0	4	3	2	4	3	2	1	1	1	1
$LSUS_i$	m	issis	ssis	sis	issip	ssip	sip	ip	pp	pi	-

Properties of LSUS

- $LSUS_1$ always exists, because at least it can be the whole string, which is unique.

Properties of LSUS

- $LSUS_1$ always exists, because at least it can be the whole string, which is unique.
- $LSUS_i$ cannot end before $LSUS_{i-1}$



Finding leftmost SUS for all positions: Overview

$$SUS_i = \begin{cases} SUS_{i-1} + S[i], & \text{if} \quad |SUS_{i-1}| + 1 \leq |SLS_i| \\ SLS_i, & \text{if} \quad |SUS_{i-1}| + 1 > |SLS_i| \end{cases}$$

SLS_i

Leftmost shortest LSUS covering string location i.

abcabcbcab

abcabcbcab

abcabcbcab

abcabcbcab

abcabcbcab

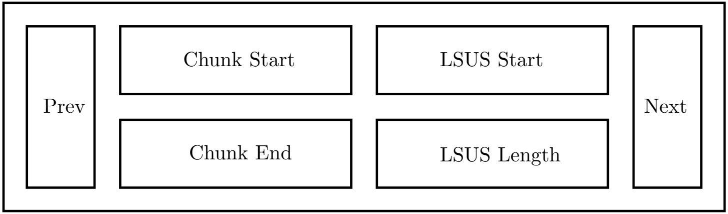
abcabcbcab

abcabcbcab

abcabcbcab

abcabcbcab

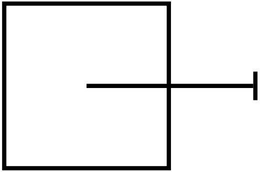
Finding the candidate: Data structure



Example Run

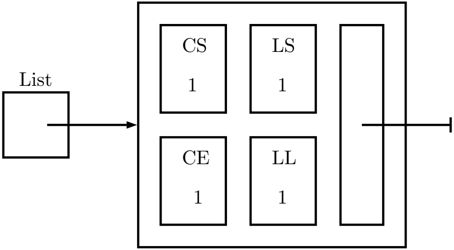
S = mississippi

List



Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

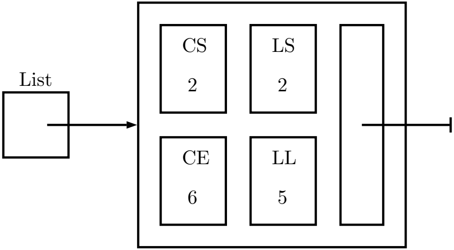


$$SUS_0 = -$$
$$SLS_1 = (1, 1)$$
$$SUS_1 = (1, 1)$$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

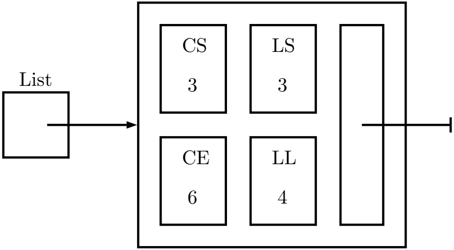


$SUS_1 = (1, 1)$
$SLS_2 = (2, 5)$
$SUS_2 = (1, 2)$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

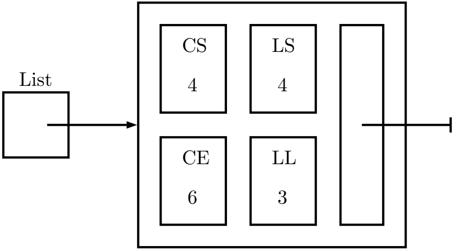


$SUS_2 = (1, 2)$
$SLS_3 = (3, 4)$
$SUS_3 = (1, 3)$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

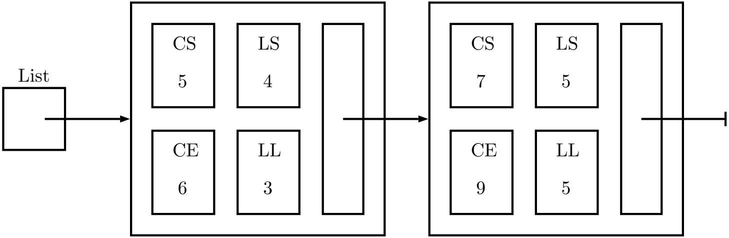


$SUS_3 = (1, 3)$
$SLS_4 = (4, 3)$
$SUS_4 = (4, 3)$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

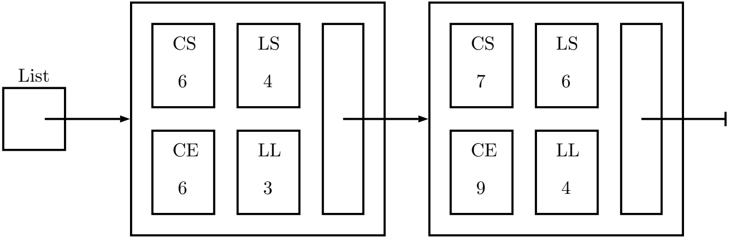


$$SUS_4 = (4, 3)$$
$$SLS_5 = (4, 3)$$
$$SUS_5 = (4, 3)$$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

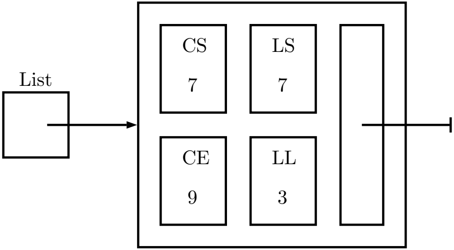


$$SUS_5 = (4, 3)$$
$$SLS_6 = (4, 3)$$
$$SUS_6 = (4, 3)$$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

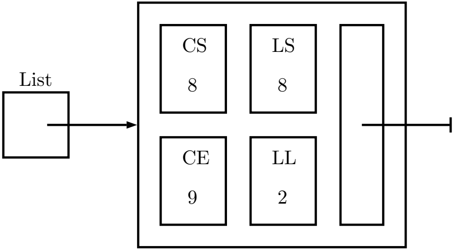


$$SUS_6 = (4, 3)$$
$$SLS_7 = (7, 3)$$
$$SUS_7 = (7, 3)$$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

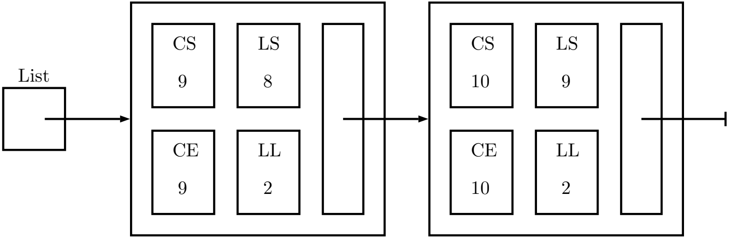


$SUS_7 = (7, 3)$
$SLS_8 = (8, 2)$
$SUS_8 = (8, 2)$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

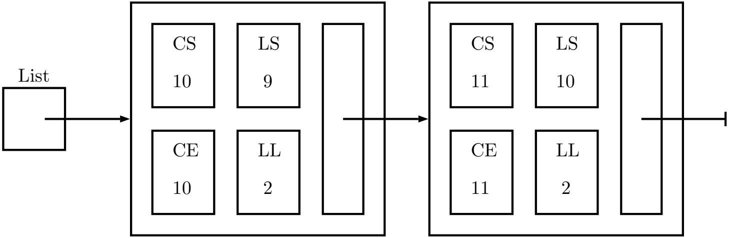


$$SUS_8 = (8,2)$$
$$SLS_9 = (8,2)$$
$$SUS_9 = (8,2)$$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i

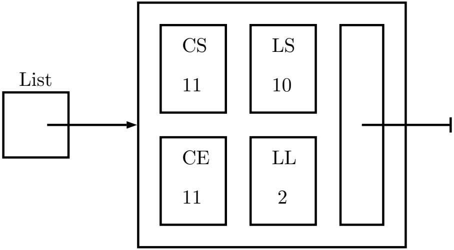


$SUS_9 = (8, 2)$
$SLS_{10} = (9, 2)$
$SUS_{10} = (9, 2)$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Example Run

1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i



$$SUS_{10} = (9, 2)$$
$$SLS_{11} = (10, 2)$$
$$SUS_{11} = (10, 2)$$

CS: Chunk Start
CE: Chunk End
LS: LSUS Start
LL: LSUS Length

Time Complexity

	Time
Derivation of LCP and Rank array	$O(n)$
Calculation of LSUS	$O(1)$
Maintenance of data structure	$O(1)$
Finding SLS	$O(1)$
Determining SUS	$O(1)$
Number of iterations:	n

Total Time: $O(n) + n \cdot (O(1) + O(1) + O(1) + O(1)) = O(n)$

Space Complexity

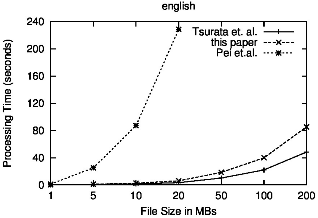
	Memory (in words)
LCP array	$O(n)$
Rank array	$O(n)$
Data structure	$O(n)$
Previous SUS	$O(1)$

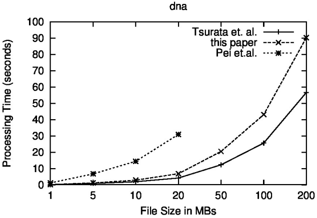
Total Memory: $O(n) + O(n) + O(n) + O(1) = O(n)$

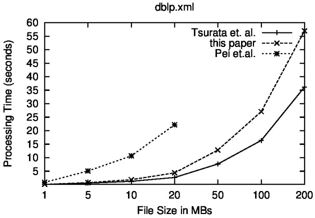
Involved work in experimental study

- J. Pei, W. C. H. Wu, M. Y. Yeh, "On shortest unique substring queries", ICDE 2013
- K. Tsuruta, S. Inenaga, H. Bannai, M. Takeda, "Shortest unique substrings queries in optimal time", SOFSEM 2014

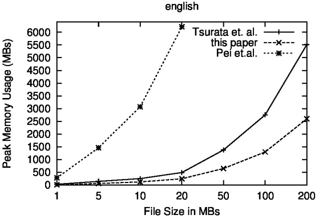
Time Measurements

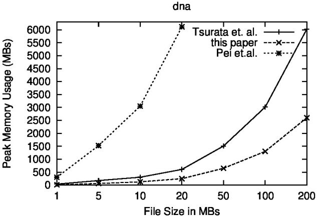


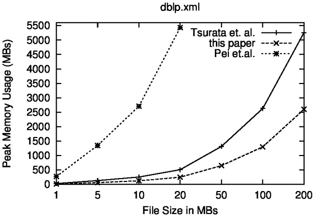




Memory Measurements







Conclusion

- We developed O(n) time and space algorithms for finding SUS covering one position and all positions.
- Our work improved the prior work of Pei et al. 8 times in terms of time and 20 times in terms of memory.

Acknowledgments

I want to thank them for their monetary support for my trip to CPM2014.

CPM Organizing Commitee

Bilim Akademisi

Thanks !

Questions ?