Suffix Arrays on Words

Paolo Ferragina[1] and Johannes Fischer[2]

[1] Dipartimento di Informatica
Università di Pisa

[2] Institut für Informatik
Ludwig-Maximilians-Universität München

July 2007, London (Ontario)

Outline

Outline

Formal Problem Definition

The Problem

given: text $T[1, n]$ over integer alphabet Σ, tokenized into k words by word-delimiters

task: preprocess T such that all occ word-aligned occurrences of $P[1, m]$ can be found "efficiently"

Example

$T =$"... he talks ... she talks ... she talks ... he talks ... the talks at CPM ... he talks"
$P =$"he talks"

applicable to natural language, URLs, structured text, ...

Formal Problem Definition

The Problem

given: text $T[1, n]$ over integer alphabet Σ, tokenized into k words by word-delimiters

task: preprocess T such that all occ word-aligned occurrences of $P[1, m]$ can be found "efficiently"

Example

$T =$"... he talks ... she talks ... she talks ... he talks ... the talks at CPM ... he talks"
$P =$"he talks"

applicable to natural language, URLs, structured text, ...

Naive Solution 1

The Problem

given: text $T[1, n]$, tokenized into k words

task: preprocess T such that all occ word-aligned occurrences of $P[1, m]$ can be found

Naive Solution 1

preprocessing: build full-text index on T

pattern matching: locate all occ' occurrences of P, filter out the occ word-aligned occurrences

Problem: index-size $O(n \log n + n \log |\Sigma|)$ bits, overhead in pattern-matching ($occ' \gg occ$, $n \gg k$)

Naive Solution 2

The Problem

given: text $T[1, n]$, tokenized into k words

task: preprocess T such that all occ word-aligned occurrences of $P[1, m]$ can be found

Naive Solution 2

preprocessing: build full-text index on T, filter out word-aligned positions

pattern matching: locate occ word-aligned occurrences

Problem: still needs $O(n \log n + n \log |\Sigma|)$ bits at construction time $\rightsquigarrow$ memory bottleneck!

Main Result 1: Construction

Definition (Word Suffix Array WSA)

$I[1,k] \subseteq [1:n]$: positions in T where words start
WSA A: permutation of I, $T_{A[i-1]..n} <_{\text{lex}} T_{A[i]..n} \forall 1 < i \leq k$

Example ($T = ab\#a\#aa\#a\#ab\#baa\#aab\#a\#$)

A=	22	4	9	6	18	1	11	14
	a	a	a	a	a	a	a	b
	#	#	#	a	a	b	b	a
		a	a	#	b	#	#	a
		a	b	a	#	a	b	#
		·	·	·	·	·	·	·

Theorem (Optimal Construction of WSAs)

WSA can be constructed in $O(n)$ time using $O(k)$ words of extra space (assuming integer alphabet).

Main Result 2: Searching

- easy to adapt search-methods for full-text suffix arrays (FTSAs)

method	space (words)	time bounds
binary	k	$O(m \log k)$
improved	$(1+C)k$, $C \leq 1$	$O((m - \log(Ck)) \log k)$
lcp	$2k$	$O(m + \log k)$
ESA	$2k + O(k/\log k)$	$O(m\|\Sigma\|)$
ESA-log	$2k + O(k/\log k)$	$O(m \log \|\Sigma\|)$

$+O(occ)$ for displaying all occ word-aligned occurrences

- FTSA: same, with k substituted by n

Previous Results

- Anderson et al.'96: word-based suffix tree in $O(n)$ expected time, $O(k)$ working space
- Inenaga and Takeda'06: word-based suffix tree and (C)DAWG in $O(n)$ deterministic time, $O(k)$ working space

Previous Results

- Anderson et al.'96: word-based suffix tree in $O(n)$ expected time, $O(k)$ working space
- Inenaga and Takeda'06: word-based suffix tree and (C)DAWG in $O(n)$ deterministic time, $O(k)$ working space
- Where is the word-based suffix array??? HERE!
- advantages
  - can be constructed using only 3 integer arrays
  - simple!
  - easy-to-implement alternative for constructing word-based suffix trees

Outline

Summary of Algorithm

1. Radix-Sort indexed suffixes up to next #
2. Build new text T' from bucket-numbers obtained in step 1
3. Build FTSA for T'
4. Derive WSA A from FTSA

Summary of Algorithm

1. Radix-Sort indexed suffixes up to next # $O(n)$
2. Build new text T' from bucket-numbers obtained in step 1 $O(k)$
3. Build FTSA for T' $O(k)$
4. Derive WSA A from FTSA $O(k)$

Step 1/4

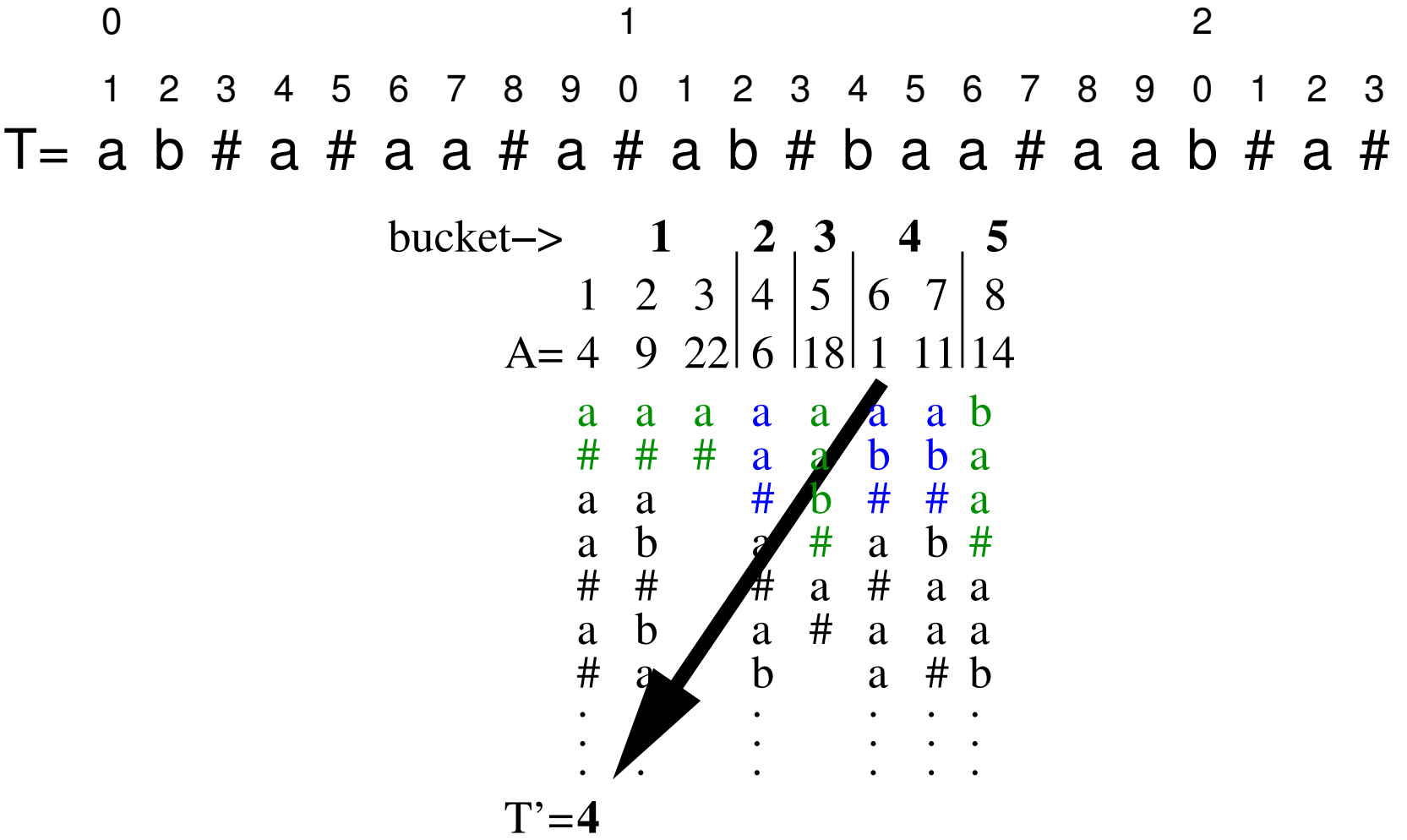
Radix-Sort indexed suffixes up to next #

		0									1										2		
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
T=	a	b	#	a	#	a	a	#	a	#	a	b	#	b	a	a	#	a	a	b	#	a	#

bucket–>	**1**			**2**	**3**	**4**		**5**
	1	2	3	4	5	6	7	8
A=	4	9	22	6	18	1	11	14
	a	a	a	a	a	a	a	b
	#	#	#	a	a	b	b	a
	a	a		#	b	#	#	a
	a	b		a	#	a	b	#
	#	#		#	a	#	a	a
	a	b		a	#	a	a	a
	#	a		b		a	#	b
	.	.		.		.	.	.
	.	.		.		.	.	.
	.	.		.		.	.	.

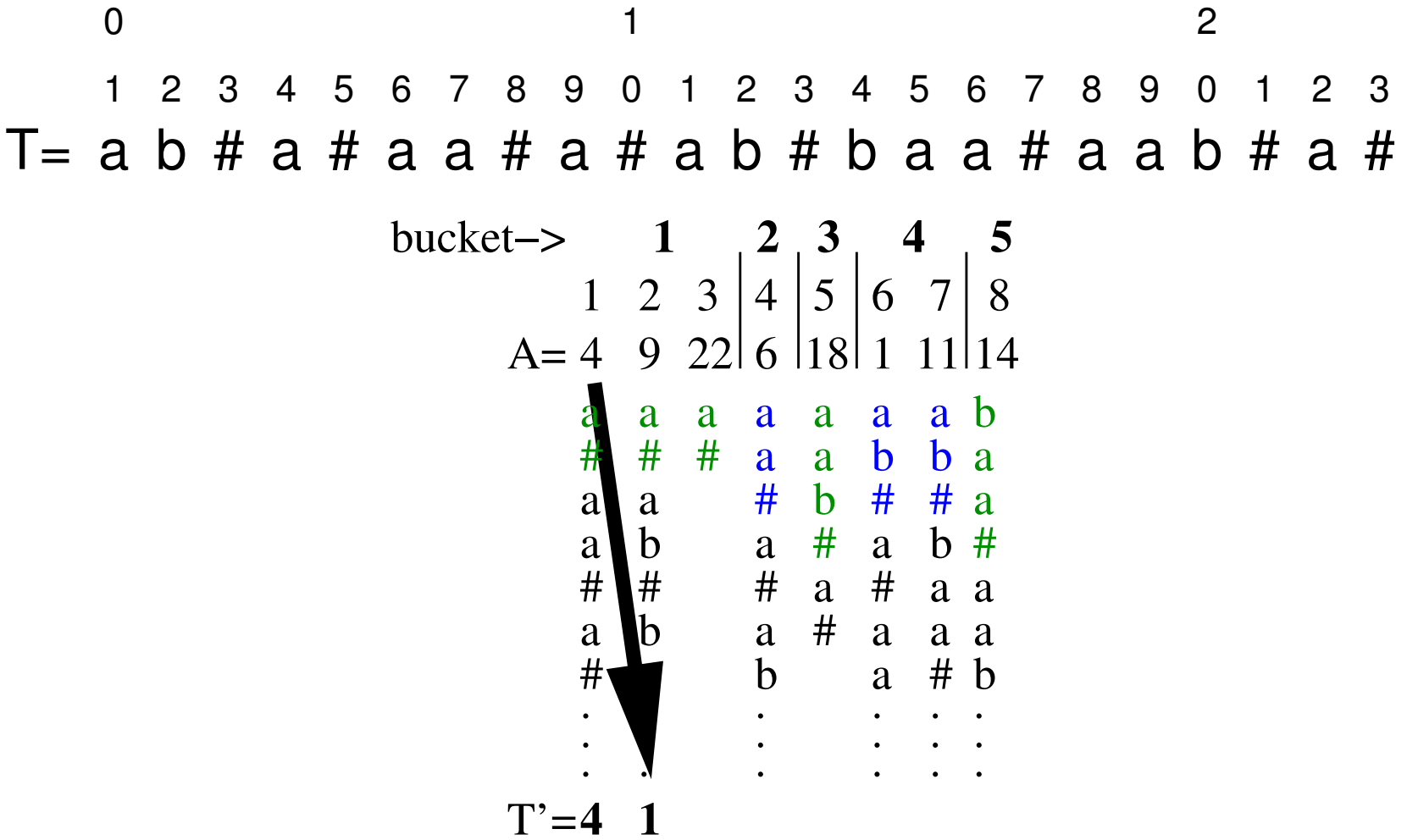
Step 2/4

Build new text T' from bucket-numbers



Step 2/4

Build new text T' from bucket-numbers



Step 2/4

Build new text T' from bucket-numbers

		0								1										2			
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
T=	a	b	#	a	#	a	a	#	a	#	a	b	#	b	a	a	#	a	a	b	#	a	#

bucket–>	**1**			**2**	**3**	**4**		**5**
	1	2	3	4	5	6	7	8
A=	4	9	22	6	18	1	11	14
	a	a	a	a	a	a	a	b
	#	#	#	a	a	b	b	a
	a	a		#	b	#	#	a
	a	b		a	#	a	b	#
	#	#		#	a	#	a	a
	a	b		a	#	a	a	a
	#	a		b		a	#	b
	.	.		.		.	.	.
	.	.		.		.	.	.
	.	.		.		.	.	.
T'=	**4**	**1**	**2**	**1**	**4**	**5**	**3**	**1**

Step 3/4

Build full-text suffix array for T'

		0								1										2			
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
T=	a	b	#	a	#	a	a	#	a	#	a	b	#	b	a	a	#	a	a	b	#	a	#

	1	2	3	4	5	6	7	8
T'=	**4**	**1**	**2**	**1**	**4**	**5**	**3**	**1**
SA=	8	2	4	3	7	1	5	6
	1	**1**	**1**	**2**	**3**	**4**	**4**	**5**
		2	**4**	**1**	**1**	**1**	**5**	**3**
		1	**5**	**4**		**2**	**3**	**1**
		4	**3**	**5**		**1**	**1**	
		5	**1**	**3**		**4**		
		3		**1**		**5**		
		1				**3**		
						1		

Step 4/4

Derive word suffix array *A* from full-text SA

		0								1									2			
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
T=	a	b	#	a	#	a	a	#	a	#	a	b	#	b	a	a	#	a	a	b	#	a	#

bucket–>	**1**			**2**	**3**	**4**		**5**
	1	2	3	4	5	6	7	8
A=	4	9	22	6	18	1	11	14
	a	a	a	a	a	a	a	b
	#	#	#	a	a	b	b	a
	a	a		#	b	#	#	a
	a	b		a	#	a	b	#
	#	#		#	a	#	a	a
	a	b		a	#	a	a	a
	#	a		b		a	#	b
	.	.		.		.	.	.
	.	.		.		.	.	.
	.	.		.		.	.	.

	1	2	3	4	5	6	7	8
T'=	**4**	**1**	**2**	**1**	**4**	**5**	**3**	**1**
SA=	8	2	4	3	7	1	5	6
	1	**1**	**1**	**2**	**3**	**4**	**4**	**5**
		2	**4**	**1**	**1**	**1**	**5**	**3**
		1	**5**	**4**		**2**	**3**	**1**
		4	**3**	**5**		**1**	**1**	
		5	**1**	**3**		**4**		
		3		**1**		**5**		
		1				**3**		
						1		

Step 4/4

Derive word suffix array *A* from full-text SA

		0								1										2			
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
T=	a	b	#	a	#	a	a	#	a	#	a	b	#	b	a	a	#	a	a	b	#	a	#

FINAL RESULT:

A=	22	4	9	6	18	1	11	14
	a	a	a	a	a	a	a	b
	#	#	#	a	a	b	b	a
		a	a	#	b	#	#	a
		a	b	a	#	a	b	#
		#	#	#	a	#	a	a
		.	.	.	#	.	.	.
		.	.	.		.	.	.
		.	.	.		.	.	.

Manber and Myers's Search Algorithms

- task: find interval of $P[1, m]$ in A
- binary search: $O(m \log n)$ (MM'90) $\rightsquigarrow$ $O(m \log k)$
- $O(m + \log n)$ (MM'90) $\rightsquigarrow$ $O(m + \log k)$
  - needs additional array of size k
- reduction of costly binary search steps!

Enhanced Suffix Array Functionality

- enhance WSA with word-based LCP-array LCP$[1, k]$
  - computable in $O(n)$ time using $O(k)$ extra space (adapted from Kasai et al.'01)
  - can be stored in $n + k + o(n)$ bits (adapted from Sadakane'02)
  - allows simulation of bottom-up traversals of word suffix tree
- build RMQ on LCP
  - computable in $O(k)$ time using $o(k)$ extra space (Fischer-Heun'07)
  - allows simulation of top-down traversals of word suffix tree
  - $\rightsquigarrow$ $O(m|\Sigma|)$ pattern matching
  - variant of RMQ $\rightsquigarrow$ $O(m \log |\Sigma|)$ pattern matching

Outline

Index Construction: Space

- "English," "XML," and "sources" from Pizza & Chili
- plus: dictionary of URLs; random words ($|\Sigma| = 2$)
- used Larsson-Sadakane SA-construction algorithm
- compared to MSufSort-3.0 (Maniscalco-Puglisi'07) and Deep-shallow (Manzini-Ferragina'02)

dataset	peak space consumption (MB)		
	MSufSort-3.0	deep-shallow	WSA
English	—	1,365.3	1,118.0
XML	1,976.9	1,129.7	890.9
sources	1,407.7	804.4	807.9
URLs	484.3	276.7	132.9
random	1,735.6	991.8	362.4

Index Construction: Time

- "English," "XML," and "sources" from Pizza & Chili
- plus: dictionary of URLs; random words ($|\Sigma| = 2$)
- used Larsson-Sadakane SA-construction algorithm
- compared to MSufSort-3.0 (Maniscalco-Puglisi'07) and Deep-shallow (Manzini-Ferragina'02)

dataset	execution time (s)		
	MSufSort-3.0	deep-shallow	WSA
English	—	755.8	533.64
XML	363.9	410.8	328.38
sources	193.4	260.6	281.12
URLs	75.2	71.0	45.75
random	452.7	332.2	224.75

Comparison of Search Methods

method	space (words)	time bounds
binary	k	$O(m \log k)$
improved	$(1+C)k$, $C \leq 1$	$O((m - \log(Ck)) \log k)$
lcp	$2k$	$O(m + \log k)$
ESA	$2k + O(k/\log k)$	$O(m\|\Sigma\|)$

Comparison of Search Methods