Reduction ratio of the IS-algorithm: worst and random cases

Vincent Jugé

Université Gustave Eiffel (LIGM)

28/06/2022

Contents

Suffix arrays[1]

Suffix array: permutation that orders lexicographically suffixes of a word

B A L A L A I K A

Suffix arrays[1]

Suffix array: permutation that orders lexicographically suffixes of a word

B	A	L	A	L	A	I	K	A

A								
K	A							
I	K	A						
A	I	K	A					
L	A	I	K	A				
A	L	A	I	K	A			
L	A	L	A	I	K	A		
A	L	A	L	A	I	K	A	
B	A	L	A	L	A	I	K	A

Suffix arrays[1]

Suffix array: permutation that orders lexicographically suffixes of a word

B	A	L	A	L	A	I	K	A

A								
A	I	K	A					
A	L	A	I	K	A			
A	L	A	L	A	I	K	A	
B	A	L	A	L	A	I	K	A
I	K	A						
K	A							
L	A	I	K	A				
L	A	L	A	I	K	A		

Suffix arrays[1]

Suffix array: permutation that orders lexicographically suffixes of a word

B	A	L	A	L	A	I	K	A
4	3	8	2	7	1	5	6	0

A								
A	I	K	A					
A	L	A	I	K	A			
A	L	A	L	A	I	K	A	
B	A	L	A	L	A	I	K	A
I	K	A						
K	A							
L	A	I	K	A				
L	A	L	A	I	K	A		

Suffix arrays[1]

Suffix array: permutation that orders lexicographically suffixes of a word

B	A	L	A	L	A	I	K	A
4	3	8	2	7	1	5	6	0

A								
A	I	K	A					
A	L	A	I	K	A			
A	L	A	L	A	I	K	A	
B	A	L	A	L	A	I	K	A
I	K	A						
K	A							
L	A	I	K	A				
L	A	L	A	I	K	A		

Useful for longest common factors, Burrows-Wheeler transform[2], ...

Induced-sorting (SA-IS) algorithm[3]

Goal: Computing the suffix array of a word w

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

B A L A L A I K A

0. If no symbol of w occurs twice, just sort them

Induced-sorting (SA-IS) algorithm[3]

Goal: Computing the suffix array of a word w

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

B	A	L	A	L	A	I	K	A	\$

0. If no symbol of w occurs twice, just sort them
1. Append a \$ symbol (minimal symbol) to w

Induced-sorting (SA-IS) algorithm[3]

Goal: Computing the suffix array of a word w

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

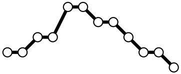
B A L A L A I K A \$

A L A

A L A

A I K A \$

0. If no symbol of w occurs twice, just sort them
1. Append a \$ symbol (minimal symbol) to w
2. Subdivide $w \cdot \$$ into **unimodal** (LMS) factors



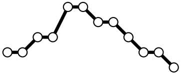
Induced-sorting (SA-IS) algorithm[3]

Goal: Computing the suffix array of a word w

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

	B	A	L	A	L	A	I	K	A	\$
1		A	L	A						
1				A	L	A				
0						A	I	K	A	\$

0. If no symbol of w occurs twice, just sort them
1. Append a \$ symbol (minimal symbol) to w
2. Subdivide $w \cdot \$$ into **unimodal** (LMS) factors
3. Sort these and relabel them in increasing order



(gives you w')

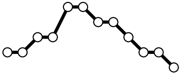
Induced-sorting (SA-IS) algorithm[3]

Goal: Computing the suffix array of a word w

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

	B	A	L	A	L	A	I	K	A	$
		2		1		0				
1		A	L	A						
1				A	L	A				
0						A	I	K	A	$

0. If no symbol of w occurs twice, just sort them
1. Append a $ symbol (minimal symbol) to w
2. Subdivide $w \cdot \$$ into **unimodal** (LMS) factors
3. Sort these and relabel them in increasing order
4. Compute the suffix array of w'



(gives you w')

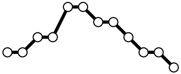
Induced-sorting (SA-IS) algorithm[3]

Goal: Computing the suffix array of a word w

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

	B	A	L	A	L	A	I	K	A	$
	4	3	8	2	7	1	5	6	0	
1		A	L	A						
1				A	L	A				
0						A	I	K	A	$

0. If no symbol of w occurs twice, just sort them
1. Append a \$ symbol (minimal symbol) to w
2. Subdivide $w \cdot \$$ into **unimodal** (LMS) factors
3. Sort these and relabel them in increasing order
4. Compute the suffix array of w'
5. Finish computing the suffix array of w



(gives you w')

Induced sorting (SA-IS) algorithm

Theorem

IS algorithm computes the suffix array of w in time linear in $|w|$.

Proof elements:

- Steps ❶ and ❷ can be performed in time $\mathcal{O}(|w|)$
- Unimodal words of total length ℓ and their suffixes can be sorted in time $\mathcal{O}(\ell)$: Steps ❸ and ❺ can be performed in time $\mathcal{O}(|w|)$
- Step ❹ is performed on a word of length $|w'| \leqslant (|w|-1)/2$

Suffix array computed in time $\mathcal{O}(|w| + |w|/2 + |w|/4 + \cdots) = \mathcal{O}(|w|)$

Induced sorting (SA-IS) algorithm

Theorem

IS algorithm computes the suffix array of w in time linear in $|w|$.

Proof elements:

- Steps ❶ and ❷ can be performed in time $\mathcal{O}(|w|)$
- Unimodal words of total length ℓ and their suffixes can be sorted in time $\mathcal{O}(\ell)$: Steps ❸ and ❺ can be performed in time $\mathcal{O}(|w|)$
- Step ❹ is performed on a word of length $|w'| \leqslant (|w| - 1)/2$

Suffix array computed in time $\mathcal{O}(|w| + |w|/2 + |w|/4 + \cdots) = \mathcal{O}(|w|)$

Further questions:

- Can we **repeatedly** have $|w'| = (|w| - 1)/2$?
- What is the **reduction ratio** $|w'|/|w|$ **in practice**?
- How many **recursive calls** shall we expect?

Reduction ratio: worst case

Worst-case scenario[5]

We can keep having $|w'| = (|w| - 1)/2$ for $\log_2(|w|)$ recursive steps

Example:

2 1 2 0 4 1 4 0 2 1 4 0 4 1 3 \$

Reduction ratio: worst case

Worst-case scenario[5]

We can keep having $|w'| = (|w| - 1)/2$ for $\log_2(|w|)$ recursive steps

Example:

$$\begin{array}{cccccccccccccccc}
2 & 1 & 2 & 0 & 4 & 1 & 4 & 0 & 2 & 1 & 4 & 0 & 4 & 1 & 3 & \$ \\
\\
 & 1 & 2 & 0 & & 1 & 4 & 0 & & 1 & 4 & 0 & & 1 & 3 & \$ \\
 & & & 0 & 4 & 1 & & 0 & 2 & 1 & & 0 & 4 & 1 & &
\end{array}$$

Reduction ratio: worst case

Worst-case scenario[5]

We can keep having $|w'| = (|w| - 1)/2$ for $\log_2(|w|)$ recursive steps

Example:

2	1	2	0	4	1	4	0	2	1	4	0	4	1	3	$
	2		1		4		0		4		1		3		
	1	2	0		1	4	0		1	4	0		1	3	$
			0	4	1		0	2	1		0	4	1		

Reduction ratio: worst case

Worst-case scenario[5]

We can keep having $|w'| = (|w| - 1)/2$ for $\log_2(|w|)$ recursive steps

Example:

2	1	2	0	4	1	4	0	2	1	4	0	4	1	3	$
	2		1		4		0		4		1		3		
	1	2	0		1	4	0		1	4	0		1	3	$
			0	4	1		0	2	1		0	4	1		

Word obtained by applying the **increasing morphism**

$$0 \mapsto 02 \qquad 1 \mapsto 04 \qquad 2 \mapsto 12 \qquad 3 \mapsto 13 \qquad 4 \mapsto 14$$

k times on the letter 3, and then deleting the first letter

Infinitely many independent letters

Sample the letters of $w \colon \mathbb{Z} \mapsto \{0,1\}$ independently uniformly at random:

Example:

$$\ldots \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad \ldots$$

Infinitely many independent letters

Sample the letters of $w \colon \mathbb{Z} \mapsto \{0, 1\}$ independently uniformly at random:

- Ends of unimodal factors are the subwords 10: $|w'| \sim |w|/4$

Example:

...	1	0	1	1	0	1	0	1	1	0	0	1	1	0	1	1	...
...	1	0			0	1	0			0	0	1	1	0			
		0	1	1	0		0	1	1	0				0	1	1	...

Infinitely many independent letters

Sample the letters of $w \colon \mathbb{Z} \mapsto \{0,1\}$ independently uniformly at random:

- Ends of unimodal factors are the subwords 10: $|w'| \sim |w|/4$
- Unimodal factors of w are independent words, with $\mathbb{P}[0^a 1^b 0] = 2^{-a-b}$

Example:

...	1	0	1	1	0	1	0	1	1	0	0	1	1	0	1	1	...
...	1	0			0	1	0			0	0	1	1	0			
		0	1	1	0		0	1	1	0				0	1	1	...

Infinitely many independent letters

Sample the letters of $w\colon \mathbb{Z} \mapsto \{0,1\}$ independently uniformly at random:

- Ends of unimodal factors are the subwords 10: $|w'| \sim |w|/4$
- Unimodal factors of w are independent words, with $\mathbb{P}[0^a 1^b 0] = 2^{-a-b}$
  - **Infinite** alphabet! (countable, not isomorphic to $\mathbb{Z}$ or $\mathbb{N}$)

Example:

...	1	0	1	1	0	1	0	1	1	0	0	1	1	0	1	1	...
...	1	0			0	1	0			0	0	1	1	0			
		0	1	1	0		0	1	1	0				0	1	1	...

$\ldots \quad 0^1 1^2 0 \quad 0^2 1^2 0 \quad 0^1 1^1 0 \quad 0^2 1^1 0 \quad 0^2 1^2 0 \quad 0^2 1^2 0 \quad 0^3 1^1 0 \quad 0^1 1^4 0 \quad \ldots$

Infinitely many independent letters

Sample the letters of $w\colon \mathbb{Z} \mapsto \{0,1\}$ independently uniformly at random:

- Ends of unimodal factors are the subwords 10: $|w'| \sim |w|/4$
- Unimodal factors of w are independent words, with $\mathbb{P}[0^a 1^b 0] = 2^{-a-b}$
  - **Infinite** alphabet! (countable, not isomorphic to $\mathbb{Z}$ or $\mathbb{N}$)
- Unimodal factors of w' are **not** independent, and $|w''| \sim 0.353\ldots|w'|$
- Things keep getting more complicated after further recursive calls

Example:

...	1	0	1	1	0	1	0	1	1	0	0	1	1	0	1	1	...
...	1	0			0	1	0			0	0	1	1	0			
		0	1	1	0		0	1	1	0				0	1	1	...

...	$0^1 1^2 0$	$0^2 1^2 0$	$0^1 1^1 0$	$0^2 1^1 0$	$0^2 1^2 0$	$0^2 1^2 0$	$0^3 1^1 0$	$0^1 1^4 0$	...
...	$0^1 1^2 0$	$0^2 1^2 0$		$0^2 1^1 0$	$0^2 1^2 0$	$0^2 1^2 0$	$0^3 1^1 0$		
		$0^2 1^2 0$	$0^1 1^1 0$	$0^2 1^1 0$			$0^3 1^1 0$	$0^1 1^4 0$	...

Main challenges

Questions:

- What about relabelling (in step 2)?
- What about letters that are **not** independent?
- What when leftmost and rightmost letters are eventually reached?

Main challenges

Questions:

- What about relabelling (in step 2)?
- What about letters that are **not** independent?
- What when leftmost and rightmost letters are eventually reached?

Answers:

- Relabelling is useful for actual computations, not here
- Assume that letters are given (from left to right or right to left) by a **nice** Markov chain
- Truncate your Markov chain when you have enough symbols!

Main challenges

Questions:

- What about relabelling (in step 2)?
- What about letters that are **not** independent?
- What when leftmost and rightmost letters are eventually reached?

Answers:

- Relabelling is useful for actual computations, not here
- Assume that letters are given (from left to right or right to left) by a **nice** Markov chain
- Truncate your Markov chain when you have enough symbols!

1	0	1	1	0	1	0	1	1	0	0	1	1	0	0	...
	0	1	1	0		0	1	1	0				0	0	...
				0	1	0			0	0	1	1	0		

Main challenges

Questions:

- What about relabelling (in step 2)?
- What about letters that are **not** independent?
- What when leftmost and rightmost letters are eventually reached?

Answers:

- Relabelling is useful for actual computations, not here
- Assume that letters are given (from left to right or right to left) by a **nice** Markov chain
- Truncate your Markov chain when you have enough symbols!

1	0	1	1	0	1	0	1	1	0	0	1	1	0	$
	0	1	1	0		0	1	1	0				$	
				0	1	0			0	0	1	1	0	

Nice Markov chains

Contraints to satisfy:

- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

Nice Markov chains

Contraints to satisfy:

- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain M is **almost surely eventually positive, recurrent and irreducible** if it has a terminal component $\mathcal{X}$ that is almost surely reached, and on which M is positive recurrent.

Nice Markov chains

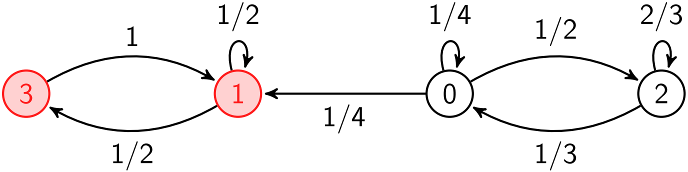
Contraints to satisfy:

- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain M is **almost surely eventually positive, recurrent and irreducible** if it has a terminal component $\mathcal{X}$ that is almost surely reached, and on which M is positive recurrent.

Example: $\mathbb{E}[1 \to 3] = 2$



Nice Markov chains

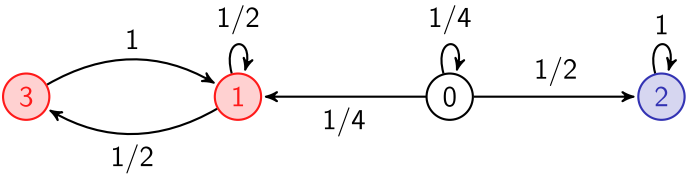
Contraints to satisfy:

- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain M is **almost surely eventually positive, recurrent and irreducible** if it has a terminal component $\mathcal{X}$ that is almost surely reached, and on which M is positive recurrent.

Counter-example: $\mathbb{E}[2 \to 1] = +\infty$



Nice Markov chains

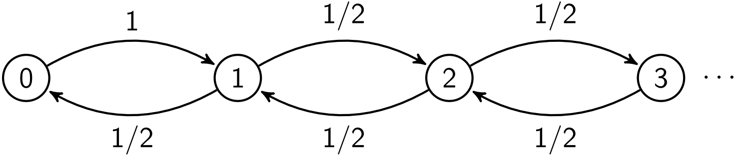
Contraints to satisfy:

- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain M is **almost surely eventually positive, recurrent and irreducible** if it has a terminal component $\mathcal{X}$ that is almost surely reached, and on which M is positive recurrent.

Counter-example: $\mathbb{E}[1 \to 0] = +\infty$



Nice Markov chains

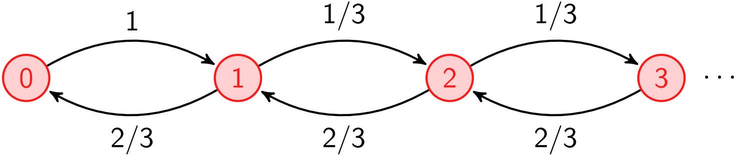
Contraints to satisfy:

- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain M is **almost surely eventually positive, recurrent and irreducible** if it has a terminal component $\mathcal{X}$ that is almost surely reached, and on which M is positive recurrent.

Example: $\mathbb{E}[1 \to 0] = 3$



Letters generated by a nice Markov chain

Theorem[5]

Let w be a word whose letters are generated by an EPRI Markov chain, and let $w^{(k)}$ be the word obtained after k recursive calls. The ratios

$$\frac{|w^{(k)}|}{|w|}$$

converge, in probability, towards a constant $\gamma^{(k)}$.

Letters generated by a nice Markov chain

Theorem[5]

Let w be a word whose letters are generated by an EPRI Markov chain, and let $w^{(k)}$ be the word obtained after k recursive calls. The ratios

$$\frac{|w^{(k)}|}{|w|}$$

converge, in probability, towards a constant $\gamma^{(k)}$.

Bonus result[4,5]

If the letters of w are i.i.d, $\gamma^{(1)} < 1/3$.

Number of recursive calls

Step 0 (direct letter sorting if possible) is very useful!

Number of recursive calls

Step ❶ (direct letter sorting if possible) is very useful!

Theorem[5]

Let w be a word whose letters are generated by a finite Markov chain. There exists a constant k such that, for all $\ell \geqslant 0$, the SA-IS algorithm has a probability

$$\mathbb{P} \leqslant k/|w|^{2^\ell}$$

of performing more than $2\log_2(\log_2(|w|)) + \ell$ recursive calls.

Number of recursive calls

Step ⓿ (direct letter sorting if possible) is very useful!

Theorem[5]

Let w be a word whose letters are generated by a finite Markov chain. There exists a constant k such that, for all $\ell \geqslant 0$, the SA-IS algorithm has a probability

$$\mathbb{P} \leqslant k/|w|^{2^\ell}$$

of performing more than $2\log_2(\log_2(|w|)) + \ell$ recursive calls.

Proof elements:

- Each letter of $w^{(i)}$ represents at least 2^i letters of w
- Letters of w reach a terminal component $\mathcal{X}$ in expected time $\mathcal{O}(1)$
- If $\mathcal{X}$ is a cycle, end up with a one-letter word in $\mathcal{O}(1)$ recursive calls
- Otherwise, factors of w of length $2^\ell(\log_2(|w|))^2$ are likely to be distinct

Some references

[1] *Suffix arrays: a new method for on-line string searches*,
U. Manber & G. Meyers (1993)

[2] *A block-sorting lossless data compression algorithm*
M. Burrows & D. Wheeler (1994)

[3] *Two efficient algorithms for linear time suffix array construction*
G. Nong, S. Zhang & W. H. Chan (2010)

[4] *A probabilistic analysis of the reduction ratio in the suffix-array IS-algorithm*
C. Nicaud (2015)

[5] *Reduction ratio of the IS-algorithm: worst and random cases*
V. Jugé (2022)


THANK YOU FOR LISTENING!
DO YOU HAVE EASY QUESTIONS?