Fast practical computation of the longest common Cartesian substrings of two strings

Simone Faro, Thierry Lecroq and Kunsoo Park

PSC 2020
August 31st – September 2nd, 2020 – Prague, Czech Republic

Outline

1. Introduction
2. Longest cartesian substrings
3. A fast practical method
4. Experiments

Outline

1. Introduction
2. Longest cartesian substrings
3. A fast practical method
4. Experiments

Cartesian Tree

Definition

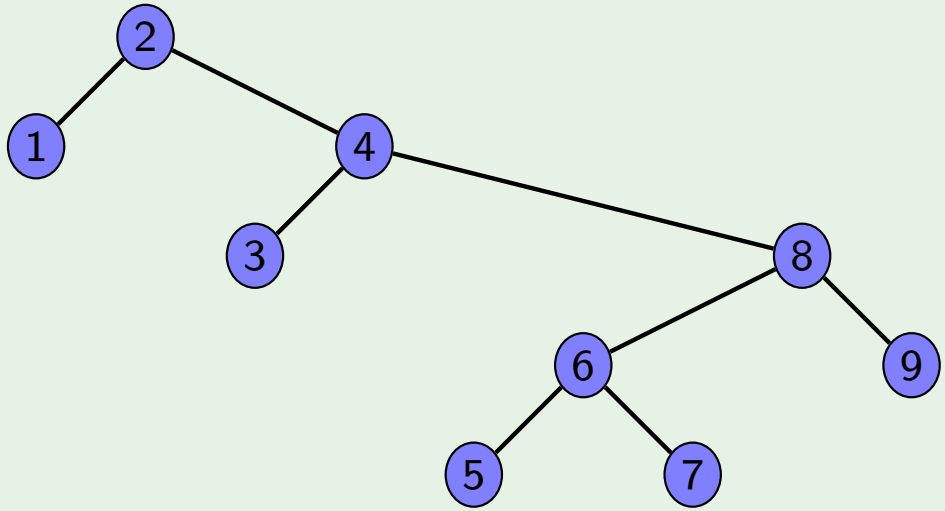
Let x be a string of numbers of length m.
The Cartesian Tree $\mathcal{CT}(x)$ of x is the binary tree where:

- the root corresponds to the index i of the minimal element of x (if there are several occurrences of the minimal element, the leftmost one is chosen)
- the left subtree of the root corresponds to the Cartesian Tree of $x[1\,..\,i-1]$
- the right subtree of the root corresponds to the Cartesian Tree of $x[i+1\,..\,m]$

Cartesian Tree

Example

i	1	2	3	4	5	6	7	8	9
$x[i]$	3	1	6	4	8	6	7	5	9



Cartesian Tree

Jean Vuillemin
A Unifying Look at Data Structures
Commun. ACM 23(4) 1980, 229–239

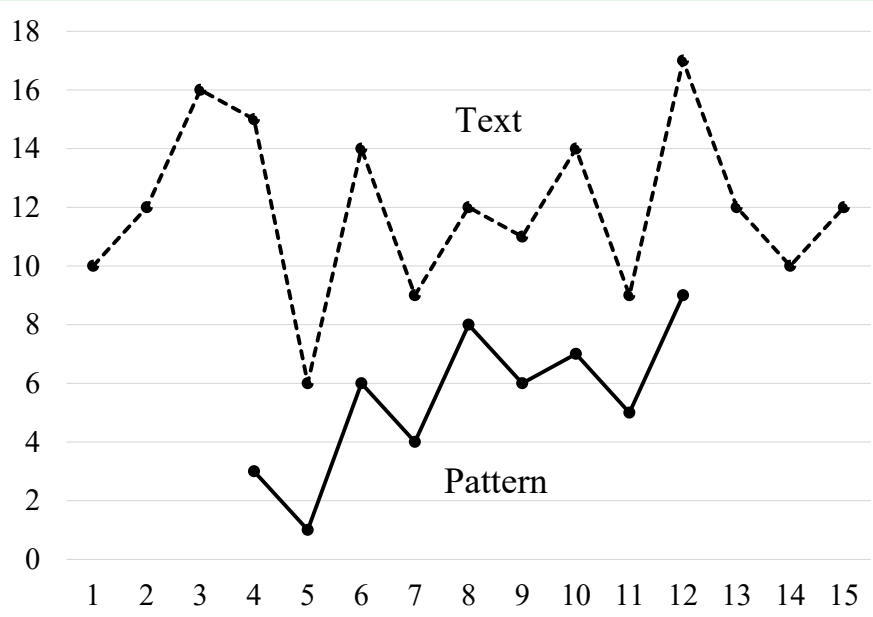
Cartesian Tree Matching

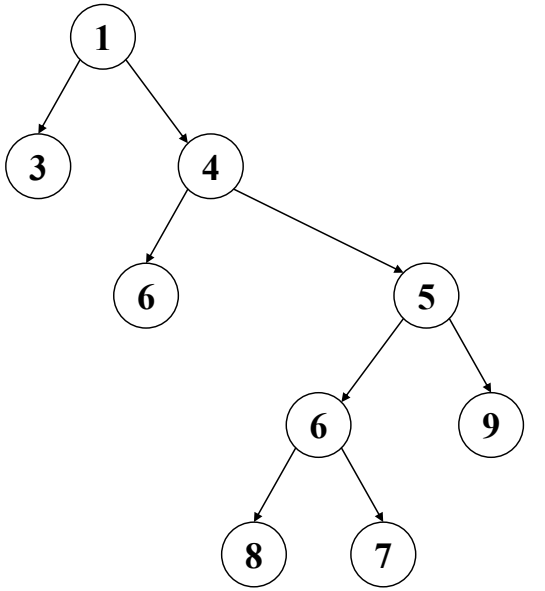
Definition

Given two strings of numbers x and y, find all the substrings of y that have the same Cartesian tree as x

Cartesian Tree Matching

Example





Cartesian Tree Matching

Motivations

Finding patterns in time series data such as :

- share prices in stock markets
- temperatures
- ...

Notations

$x[i] \prec x[j]$ if and only if $x[i] < x[j]$ or $x[i] = x[j]$ and $i < j$

Parent-Distance representation

Definition

Given a string $x[1\,..\,n]$, the Parent-Distance representation of x is a function $\mathcal{PD}_x$, which is defined as follows:

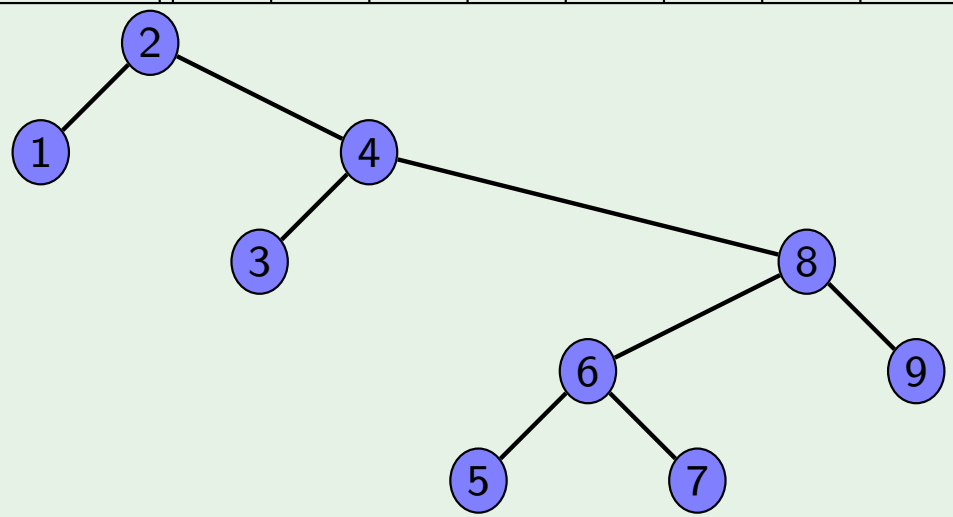
$$\mathcal{PD}_x(i) = \begin{cases} i - \max_{1 \le j < i}\{j \mid x[j] \prec x[i]\} & \textit{if such } j \textit{ exists} \\ 0 & \textit{otherwise.} \end{cases}$$

The Parent-Distance representation has a one-to-one mapping to the Cartesian tree

Parent-Distance representation

Example

i	1	2	3	4	5	6	7	8	9
$x[i]$	3	1	6	4	8	6	7	5	9
$\mathcal{PD}_x(i)$	0	0	1	2	1	2	1	4	1



Parent-Distance representation of a substring of x

$$\mathcal{PD}_{x[i..j]}[k] = \begin{cases} 0 & \text{if } \mathcal{PD}_x[i+k-1] \geq k \\ \mathcal{PD}_x[i+k-1] & \text{otherwise.} \end{cases}$$

Parent-Distance representation of a substring of x

Example

i	1	2	3	4	5	6	7	8	9
$x[i]$	3	1	6	4	8	6	7	5	9
$\mathcal{PD}_{x[1..9]}$	0	0	1	2	1	2	1	4	1
$\mathcal{PD}_{x[2..9]}$		0	1	2	1	2	1	4	1
$\mathcal{PD}_{x[3..9]}$			0	0	1	2	1	4	1
$\mathcal{PD}_{x[4..9]}$				0	1	2	1	4	1
$\mathcal{PD}_{x[5..9]}$					0	0	1	0	1
$\mathcal{PD}_{x[6..9]}$						0	1	0	1
$\mathcal{PD}_{x[7..9]}$							0	0	1
$\mathcal{PD}_{x[8..9]}$								0	1
$\mathcal{PD}_{x[9..9]}$									0

Previous works


Sung Gwan Park, Amihood Amir, Gad M. Landau and Kunsoo Park
Cartesian Tree Matching and Indexing
CPM'19


Siwoo Song, Cheol Ryu, Simone Faro, T L, Kunsoo Park
Fast Cartesian Tree Matching
SPIRE 2019: 124–137


Geonmo Gu, Siwoo Song, Simone Faro, T L, Kunsoo Park
Fast Multiple Pattern Cartesian Tree Matching
WALCOM 2020: 107–119

Outline

1. Introduction
2. **Longest cartesian substrings**
3. A fast practical method
4. Experiments

Longest common Cartesian substrings of two strings

We want to common substrings of maximal length and that have the same Cartesian tree

Usual method

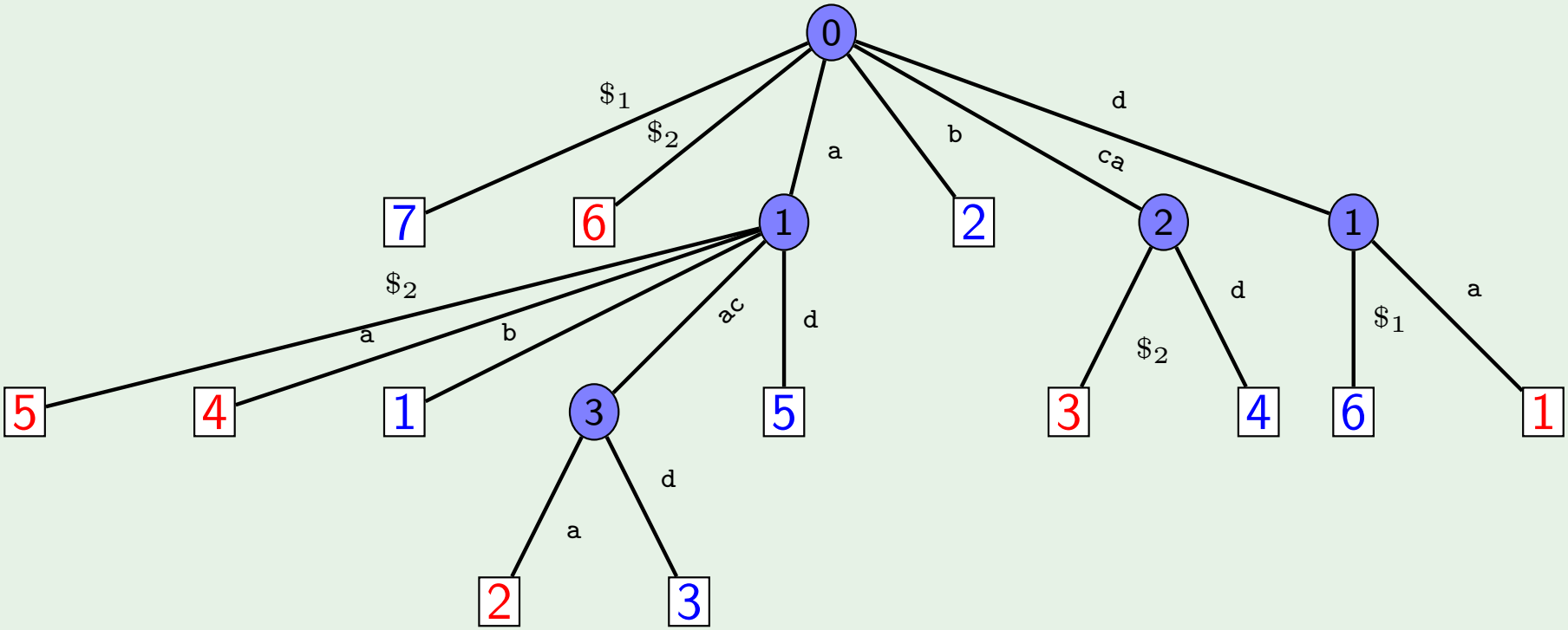
For computing the longest common substrings of x and y

- build the generalize suffix tree for x and y
- the deepest nodes (in terms of string depth) of the tree having leaves for suffixes of both x and y represents the longest common substrings of x and y

Example

	1	2	3	4	5	6	7
$x =$	a	b	a	c	a	d	$\$_1$

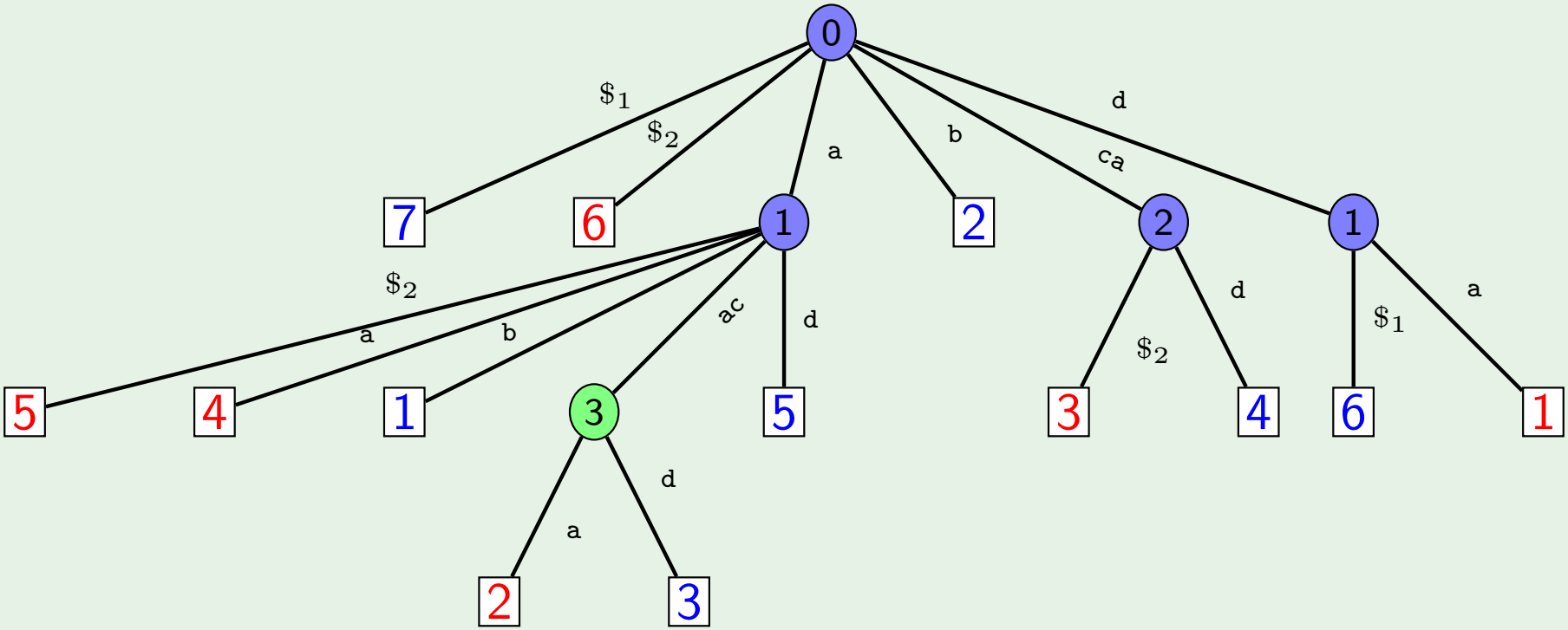
	1	2	3	4	5	6
$y =$	d	a	c	a	a	$\$_2$



Example

	1	2	3	4	5	6	7
$x =$	a	b	a	c	a	d	$\$_1$

	1	2	3	4	5	6
$y =$	d	a	c	a	a	$\$_2$



Suffix tree construction

The distinct right context property

if

$$lcp(x[i\,..\,m], x[j\,..\,m]) = \ell$$

then

$$lcp(x[i+1\,..\,m], x[j+1\,..\,m]) = \ell - 1$$

for $1 \leq i, j \leq m$ where $lcp(u, v)$ is the length of the longest prefix common to two strings u and v.

This means that the suffix link of every internal node should point to an explicit node.

Suffix tree construction

The Cartesian suffix tree does not have the distinct right context property meaning that if

$$lcp(\mathcal{PD}_x[i\,..\,m], PD_x[j\,..\,m]) = \ell$$

then

$lcp(\mathcal{PD}_x[i+1\,..\,m], PD_x[j+1\,..\,m])$ can be greater than $\ell - 1$.

With $x = \langle 3, 1, 6, 4, 8, 6, 7, 5, 9 \rangle$

$$lcp(\mathcal{PD}_{x[2..9]}, \mathcal{PD}_{x[6..9]}) = lcp(\langle 0, 1, 2, 1, 2, 1, 4, 1 \rangle, \langle 0, 1, 0, 1 \rangle) = 2$$

and

$$lcp(\mathcal{PD}_{x[3..9]}, \mathcal{PD}_{x[7..9]}) = lcp(\langle 0, 0, 1, 2, 1, 4, 1 \rangle, \langle 0, 0, 1 \rangle) = 3$$

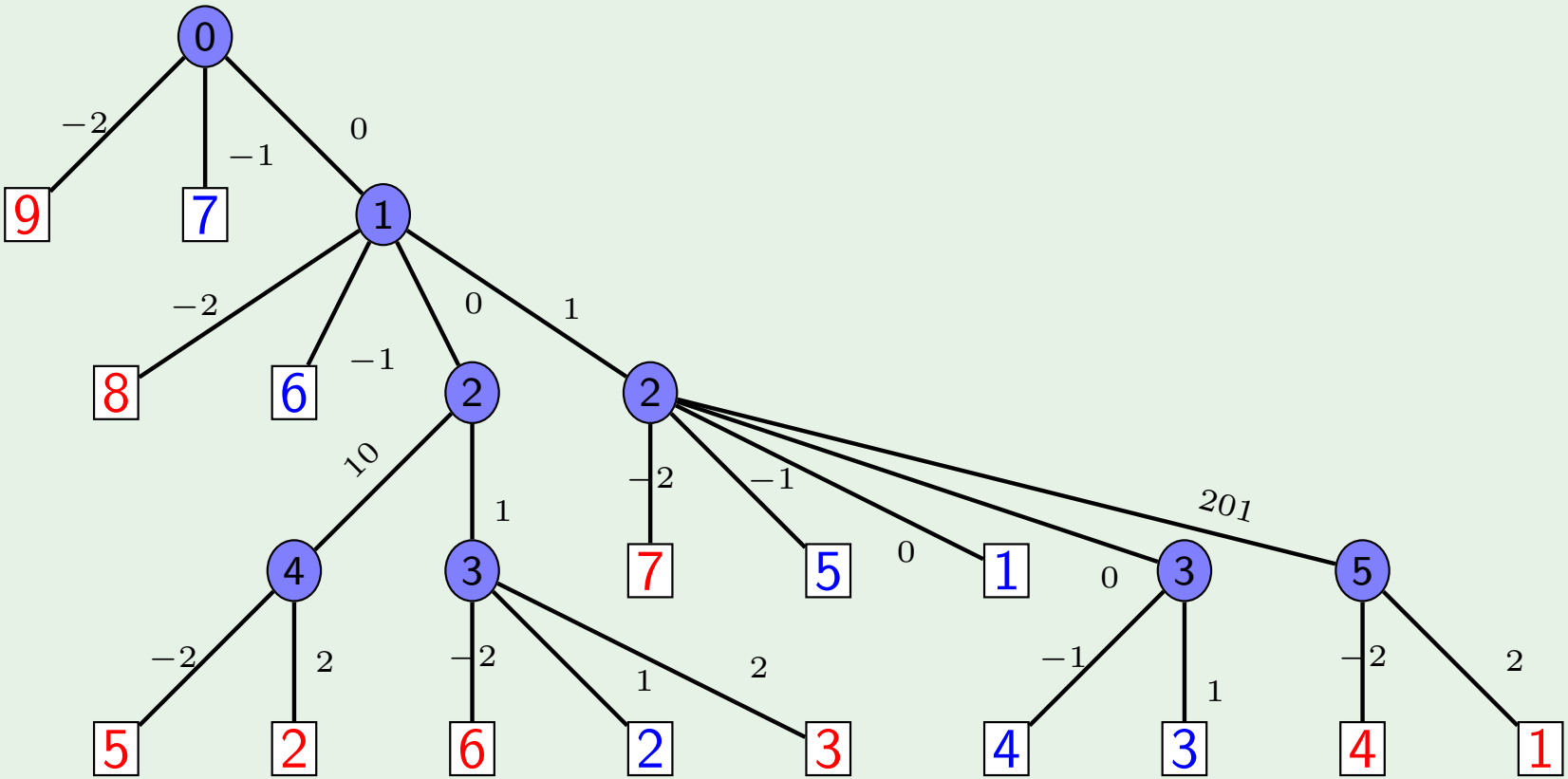
Suffix tree with missing suffix links

Richard Cole and Ramesh Hariharan
Faster Suffix Tree Construction with Missing Suffix Links
SIAM J. Comput. **33** (1) (2003) 26–42

Example

i	1	2	3	4	5	6	7
$x[i]$	70	84	63	74	86	97	
$\mathcal{PD}$	0	1	0	1	1	1	-1
		0	0	1	1	1	-1
			0	1	1	1	-1
				0	1	1	-1
					0	1	-1
						0	-1
							-1

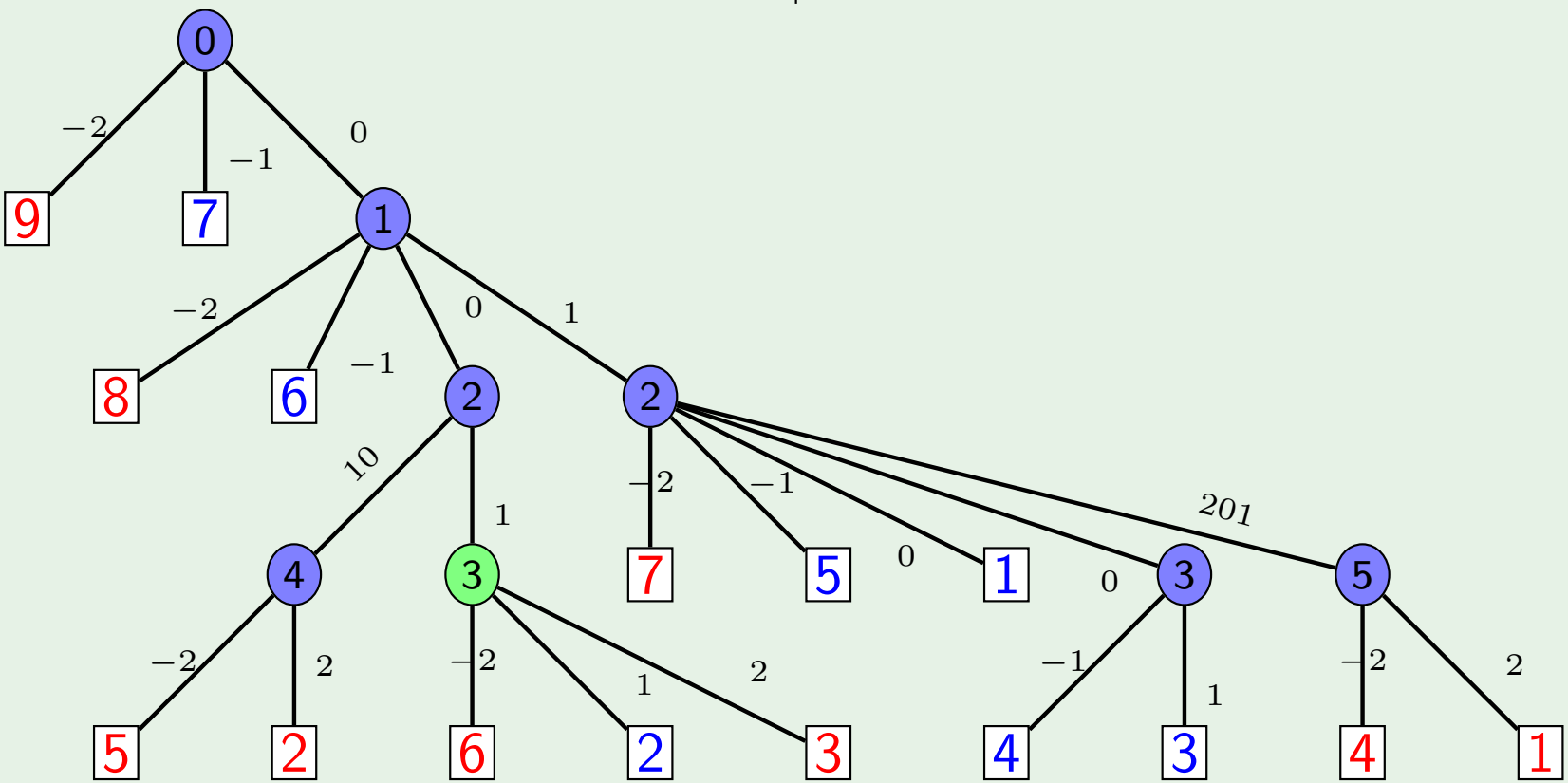
j	1	2	3	4	5	6	7	8	9
$y[j]$	50	83	76	39	90	67	1	6	
$\mathcal{PD}$	0	1	2	0	1	2	0	1	-2
		0	0	0	1	2	0	1	-2
			0	0	1	2	0	1	-2
				0	1	2	0	1	-2
					0	0	0	1	-2
						0	0	1	-2
							0	1	-2
								0	-2
									-2



Example

i	1	2	3	4	5	6	7
$x[i]$	70	84	63	74	86	97	
$\mathcal{PD}$	0	1	0	1	1	1	-1
		0	0	1	1	1	-1
			0	1	1	1	-1
				0	1	1	-1
					0	1	-1
						0	-1
							-1

j	1	2	3	4	5	6	7	8	9
$y[j]$	50	83	76	39	90	67	1	6	
$\mathcal{PD}$	0	1	2	0	1	2	0	1	-2
		0	0	0	1	2	0	1	-2
			0	0	1	2	0	1	-2
				0	1	2	0	1	-2
					0	0	0	1	-2
						0	0	1	-2
							0	1	-2
								0	-2
									-2



Suffix tree based method

Result

The longest substrings of x and y having the same Cartesian tree can be computed in
time $O(m+n)$ and in space $O(m+n)$
and in linear space in addition to $\mathcal{PD}_x$ and $\mathcal{PD}_y$.

Outline

1. Introduction
2. Longest cartesian substrings
3. **A fast practical method**
4. Experiments

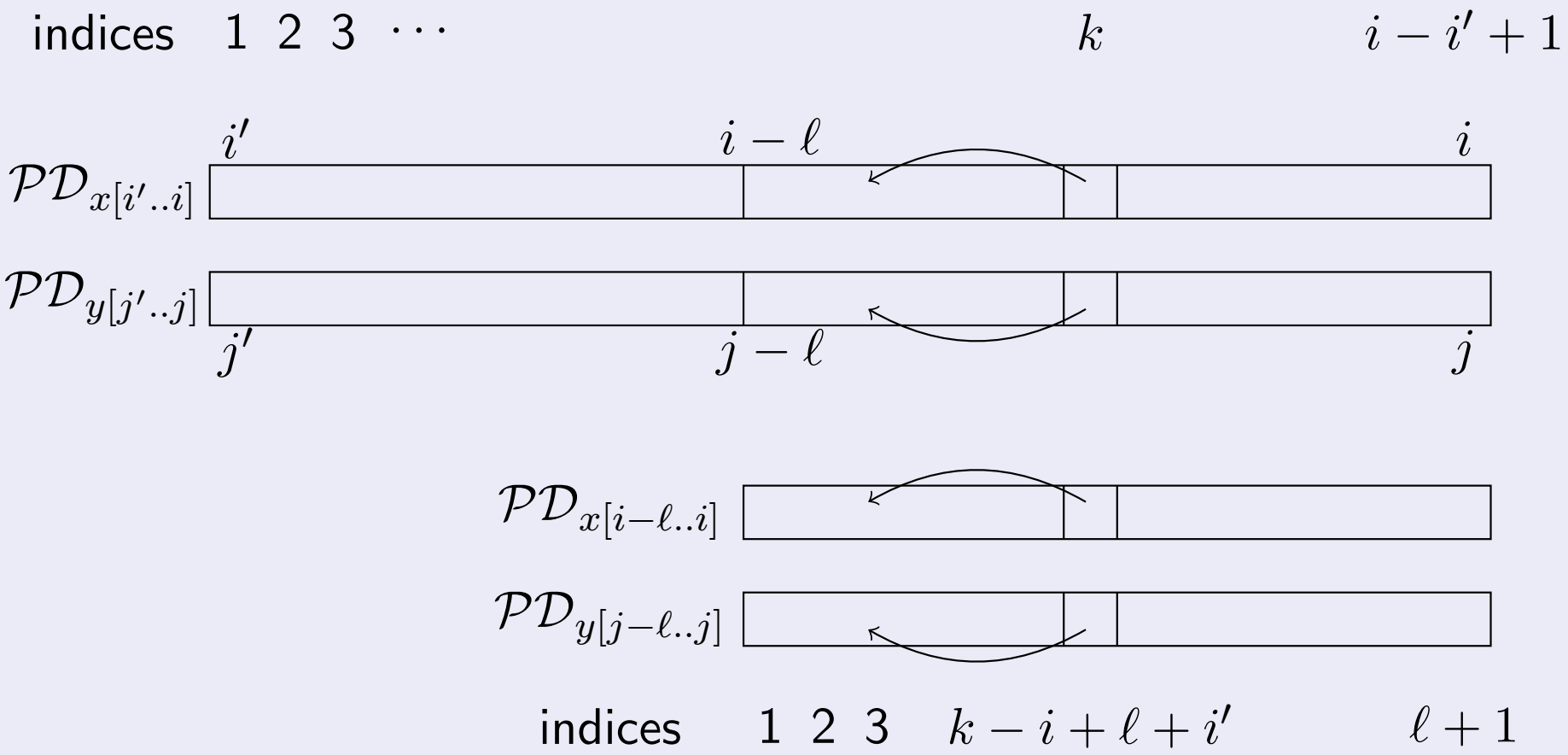
Parent-distance representation of suffixes

Let us first state that if two substrings have the same parent-distance so have their suffixes.

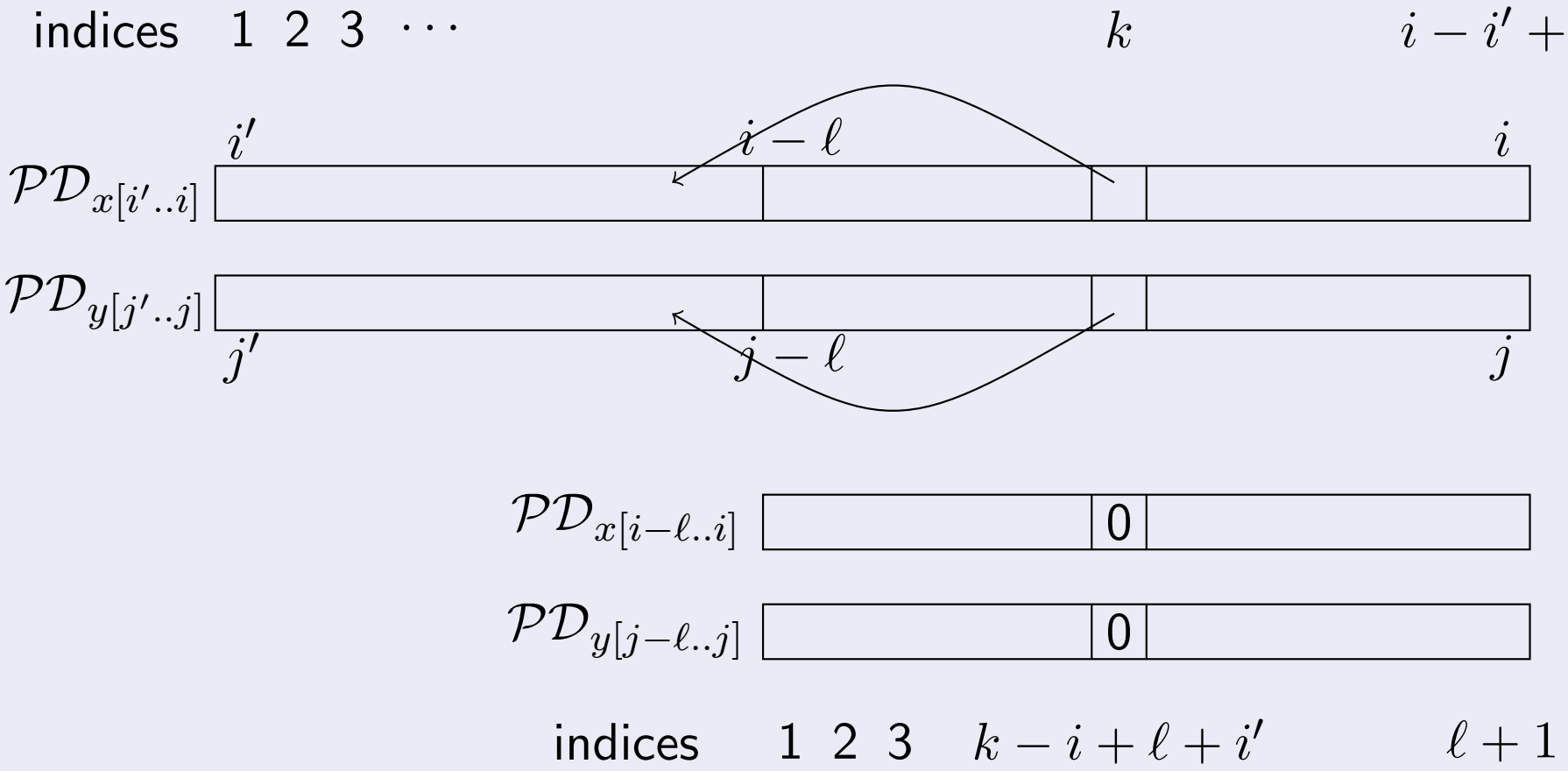
Fact

If $\mathcal{PD}_{x[i'..i]} = \mathcal{PD}_{y[j'..j]}$ *then* $\mathcal{PD}_{x[i-\ell..i]} = \mathcal{PD}_{y[j-\ell..j]}$ *for* $0 \leq \ell < i - i'$.

Case 1: $\mathcal{PD}_{x[i'..i]}[k] = \mathcal{PD}_{y[j'..j]}[k] < k - i + \ell$



Case 2: $\mathcal{PD}_{x[i'..i]}[k] = \mathcal{PD}_{y[j'..j]}[k] \geq k - i + \ell$



Extension of common Cartesian substring

Lemma

Let $x[i' \,..\, i-1]$ *and* $y[j' \,..\, j-1]$ *be the longest suffixes of* $x[1 \,..\, i-1]$ *and* $y[1 \,..\, j-1]$ *having the same Cartesian tree.*
Let $\ell_x = \mathcal{PD}_{x[i'..i]}[i-i'+1]$, $\ell_y = \mathcal{PD}_{y[j'..j]}[j-j'+1]$ *and* $\ell = \min\{\ell_x, \ell_y\}$.
The longest suffixes of $x[1 \,..\, i]$ *and* $y[1 \,..\, j]$ *having the same Cartesian tree are:*

1. $x[i' \,..\, i]$ *and* $y[j' \,..\, j]$ *if* $\ell_x = \ell_y$;
2. $x[i-\ell_y+1 \,..\, i]$ *and* $y[j-\ell_y+1 \,..\, j]$ *if* $\ell_x \neq \ell_y$ *and* $\ell_x = 0$;
3. $x[i-\ell_x+1 \,..\, i]$ *and* $y[j-\ell_x+1 \,..\, j]$ *if* $\ell_x \neq \ell_y$ *and* $\ell_y = 0$;
4. $x[i-\ell+1 \,..\, i]$ *and* $y[j-\ell+1 \,..\, j]$ *if* $\ell_x \neq \ell_y$ *and* $\ell_x \neq 0$ *and* $\ell_y \neq 0$.

Example

i	1	2	3	4	5	6
$x[i]$	70	84	63	74	86	97
$\mathcal{PD}$	0	1	0	1	1	1

j	1	2	3	4	5	6	7	8
$y[j]$	50	83	76	39	90	67	1	6
$\mathcal{PD}$	0	1	2	0	1	2	0	1

Example

i	1	2	3	4	5	6
$x[i]$	70	84	63	74	86	97
$\mathcal{PD}$	0	1	0	1	1	1

j	1	2	3	4	5	6	7	8
$y[j]$	50	83	76	39	90	67	1	6
$\mathcal{PD}$	0	1	2	0	1	2	0	1

Example

i	1	2	3	4	5	6
$x[i]$	70	84	63	74	86	97
$\mathcal{PD}$	0	1	0	1	1	1
			0			

j	1	2	3	4	5	6	7	8
$y[j]$	50	83	76	39	90	67	1	6
$\mathcal{PD}$	0	1	2	0	1	2	0	1
				0				

Example

i	1	2	3	4	5	6
$x[i]$	70	84	63	74	86	97
$\mathcal{PD}$	0	1	0	1	1	1
			0	1		

j	1	2	3	4	5	6	7	8
$y[j]$	50	83	76	39	90	67	1	6
$\mathcal{PD}$	0	1	2	0	1	2	0	1
				0	1			

Example

i	1	2	3	4	5	6
$x[i]$	70	84	63	74	86	97
$\mathcal{PD}$	0	1	0	1	1	1
			0	1		
					0	

j	1	2	3	4	5	6	7	8
$y[j]$	50	83	76	39	90	67	1	6
$\mathcal{PD}$	0	1	2	0	1	2	0	1
				0	1			
						0		

Example

i	1	2	3	4	5	6
$x[i]$	70	84	63	74	86	97
$\mathcal{PD}$	0	1	0	1	1	1
			0	1		
					0	
						0

j	1	2	3	4	5	6	7	8
$y[j]$	50	83	76	39	90	67	1	6
$\mathcal{PD}$	0	1	2	0	1	2	0	1
				0	1			
						0		
							0	

Example

	y	50	83	76	39	90	67	1	6
x	$\mathcal{PD}$	0	1	2	0	1	2	0	1
70	0	1	1	1	1	1	1	1	1
84	1	1	2	1	1	2	1	1	2
63	0	1	1	2	2	1	2	2	1
74	1	1	2	1	1	3	1	1	3
86	1	1	2	1	1	2	1	1	2
97	1	1	2	1	1	2	1	1	2

Example

	y	50	83	76	39	90	67	1	6
x	$\mathcal{PD}$	0	1	2	0	1	2	0	1
70	0	1	1	1	1	1	1	1	1
84	1	1	2	1	1	2	1	1	2
63	0	1	1	2	2	1	2	2	1
74	1	1	2	1	1	3	1	1	3
86	1	1	2	1	1	2	1	1	2
97	1	1	2	1	1	2	1	1	2

Our method

Result

Doing the computation for every pairs of starting positions i in x and j in y the longest substrings of x and y having the same Cartesian tree can be computed in
time $O(mn)$ and in space $O(m+n)$
and in constant space in addition to $\mathcal{PD}_x$ and $\mathcal{PD}_y$.

Outline

1. Introduction
2. Longest cartesian substrings
3. A fast practical method
4. **Experiments**

Implementation

- $m < n$
- simplified version of ST with missing links
- build the ST for $\mathcal{PD}_y$
- suffixes of $\mathcal{PD}_x$ are not inserted but forks are used to compute the searched value
- main diagonals are processed first
- computation for one diagonal is stopped when it cannot contribute to a larger value

Random data

Settings

- 4 different alphabets: $[0; 10[$, $[0; 100[$, $[0; 1000[$ and $[0; 10000[$
- 3 different values for n: 50, 125 and 200
- 4 values of m: $n/10$, $n/5$, $n/2.5$ and $n/1.25$

Random data

$n = 50$

σ	m	5	10	20	40
10	alt	4283	6189	13621	24711
	ST	36120	33230	46744	56796
100	alt	4963	7210	13702	25644
	ST	40437	39994	46495	58973
1000	alt	4779	7618	14708	25621
	ST	39886	42869	50787	58650
10000	alt	4280	7083	14636	25825
	ST	36074	40522	51166	58184

Random data

$n = 125$

σ	m	12	24	48	96
10	alt	16589	34500	71790	150029
	ST	98865	95663	108624	132481
100	alt	15577	37178	77217	139539
	ST	96253	106864	121404	128341
1000	alt	14420	35633	72197	141706
	ST	85163	101248	111795	126697
10000	alt	15112	34439	74415	140266
	ST	94438	95248	114660	128906

Random data

$n = 200$

σ	m	20	40	80	160
10	alt	38677	91698	203764	436527
	ST	127756	150781	181238	261578
100	alt	50687	114319	204344	444743
	ST	197489	209726	196585	280190
1000	alt	46502	104542	190889	393760
	ST	172624	183040	174690	216348
10000	alt	43269	97887	198550	414818
	ST	150034	160674	181007	236281

Real data

From COVID-19

- pairwise computations in 2 series for 15 countries
- serie $\sharp 1$: lengths range from 45 to 102
- serie $\sharp 2$: lengths range from 32 to 98
- times for serie $\sharp 1$: ST 9381, alt 5904
- times for serie $\sharp 2$: ST 8727, alt 4758

Conclusion

Method for computing the longest common Cartesian substrings of 2 strings x and y:

- fast for short strings
- constant space in addition to x, y, $\mathcal{PD}_x$, $\mathcal{PD}_y$

Thank you for your attention!