Two-Dimensional Longest Common Extension Queries in Compact Space

Arnab Ganguly
Daniel Gibney
Rahul Shah
Sharma V. Thankachan

42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025)

Text Indexing and Basic Queries

Index a text $T[1...n]$ over an alphabet $[\sigma]$ to support the pattern matching queries

Query: Pattern $P[1...m]$
Output: Set of occurrences of P in T

Text Indexing and Basic Queries

Index a text $T[1...n]$ over an alphabet $[\sigma]$ to support the pattern matching queries

Query: Pattern $P[1...m]$
Output: Set of occurrences of P in T

	1	2	3	4	5	6	7	8	9	10	11	12
$T[1...n]$	G	A	**C**	**A**	**T**	C	T	**C**	**A**	**T**	T	G

Set of occurrences of P = **CAT** in the above text is {3,8}

Text Indexing and Basic Queries

Index a text $T[1...n]$ over an alphabet $[\sigma]$ to support the pattern matching queries

Query: Pattern $P[1...m]$
Output: Set of occurrences of P in T

$T[1...n]$

1	2	3	4	5	6	7	8	9	10	11	12
G	A	**C**	**A**	**T**	C	T	**C**	**A**	**T**	T	G

Set of occurrences of P = **CAT** in the above text is {3,8}

ALGORITHMS
Knuth–Morris–Pratt (KMP)
FFT based algorithm
Rabin Karp etc

Time is at least O(n), i.e., linear in text length

Text Indexing and Basic Queries

Index a text $T[1...n]$ over an alphabet $[\sigma]$ to support the pattern matching queries

Query: Pattern $P[1...m]$
Output: Set of occurrences of P in T

$T[1...n]$

1	2	3	4	5	6	7	8	9	10	11	12
G	A	**C**	**A**	**T**	C	T	**C**	**A**	**T**	T	G

Set of occurrences of P = **CAT** in the above text is {3,8}

ALGORITHMS
Knuth–Morris–Pratt (KMP)
FFT based algorithm
Rabin Karp etc

Time is at least O(n), i.e., linear in text length

DATA STRUCTURES (Indexes)
Suffix Trees
Suffix Arrays
Their compressed versions (FM-index, CSA, r-index, etc)

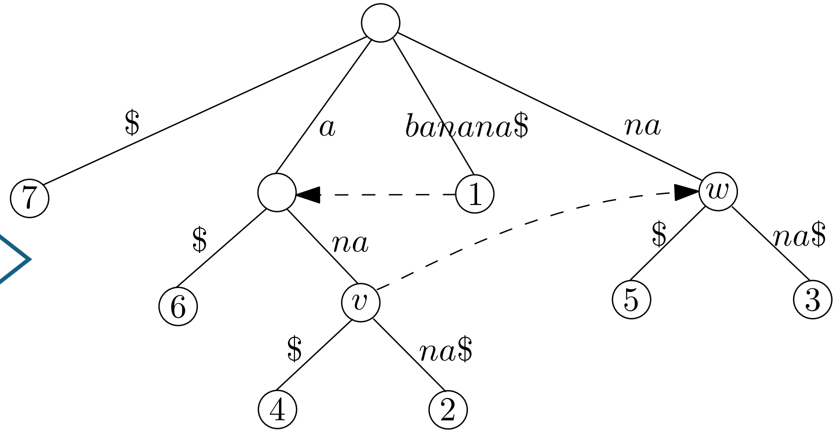
Query time is linear in "m"and output size

Suffix Trees and Suffix Arrays (text = banana$)

1 banana$
2 anana$
3 nana$
4 ana$
5 na$
6 a$
7 $

SORT

7 $
6 a$
4 ana$
2 anana$
1 banana$
5 na$
3 nana$

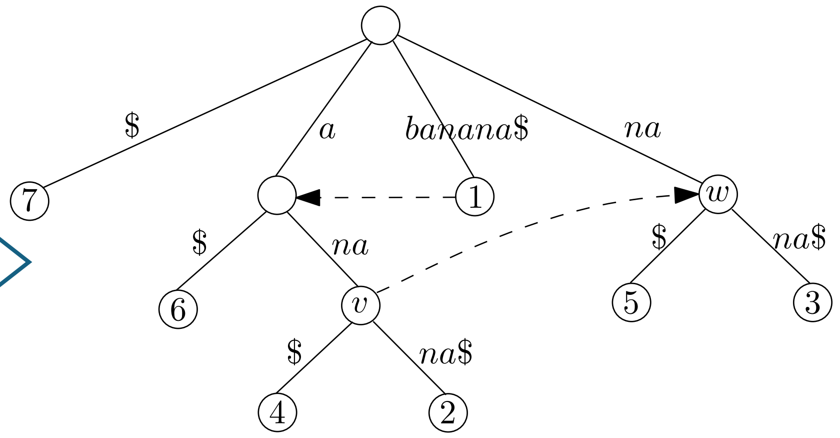


Suffix Trees and Suffix Arrays
(text = banana$)

1 banana$
2 anana$
3 nana$
4 ana$
5 na$
6 a$
7 $

SORT

7 $
6 a$
4 ana$
2 anana$
1 banana$
5 na$
3 nana$



	1	2	3	4	5	6	7
SUFFIX ARRAY	7	6	4	2	1	5	3
Inverse SUFFIX ARRAY	5	4	7	3	6	2	1

$LCE(2,4) = 3$

$LCE(3,5) = 2$

$LCE(2,5) = 0$

Longest Common Extension $LCE(i, j)$ *is length of the longest common prefix of suffixes starting at i and j*

Suffix Trees and Suffix Arrays

Good news: O(1) time for all 3 operations

- **suffix array**
- **inverse suffix array**
- **LCE operation**

Bad news: $O(n \log n)$ **bits space,**

while data needs only $n \log \sigma$ **bits**

Suffix Trees and Suffix Arrays

Good news: O(1) time for all 3 operations

- **suffix array**
- **inverse suffix array**
- **LCE operation**

Bad news: $O(n \log n)$ **bits space,**

while data needs only $n \log \sigma$ **bits**

Can we encode suffix arrays/trees in space close to text's space ~$n \log \sigma$ **bits?**

Can we encode suffix arrays/trees in space close to text's space $\sim n \log \sigma$ bits?

YES !!! ... (1D case) is well solved...

Compressed Suffix Arrays (CSA) [Grossi and Vitter, 2000]
FM-index [Ferragina and Manzini, 2000]
Compressed Suffix Trees [Sadakane, SODA 2004]
.....
....
....
r-index [Gagie, Prezza, Navarro, SODA 2018]
Suffix array in delta-compressed space [Kempa Kociumaka, FOCS 2023]

2D texts & 2D suffix Arrays & Trees

Index a 2D text for matching SQUARE patterns

T	0	1	2	3	4
0	a	a	b	b	\$
1	a	b	b	c	\$
2	b	b	a	a	\$
3	b	c	a	b	\$
4	\$	\$	\$	\$	\$

2D texts & 2D suffix Arrays & Trees

Index a 2D text for matching SQUARE patterns

T	0	1	2	3	4
0	a	a	b	b	\$
1	a	b	b	c	\$
2	b	b	a	a	\$
3	b	c	a	b	\$
4	\$	\$	\$	\$	\$

Define L-suffixes L(2,1) = b aac ab$$$...

If an mxm square pattern occurs at a positions,
the its L-suffix is a prefix of the (corresponding) L-suffix of the text

2D texts & 2D suffix Arrays & Trees

Index a 2D text for matching SQUARE patterns

T	0	1	2	3	4
0	a	a	b	b	$\$$
1	a	b	b	c	$\$$
2	b	b	a	a	$\$$
3	b	c	a	b	$\$$
4	$\$$	$\$$	$\$$	$\$$	$\$$

Define L-suffixes   L(2,1) = b aac ab$$$...

If an mxm square pattern occurs at a positions,
the its L-suffix is a prefix of the (corresponding) L-suffix of the text

Suffix Tree: Compacted Trie over L-suffixes

$O(N \log N)$ bits space ($N = n \times n$; matrix size)

O(1) for all 3 operations (SA/ISA/LCE)

Efficient (square) pattern matching

LCE((i,j), (i',j')) = m if mxm is the largest square matrix matching ...

2D texts & 2D suffix Arrays & Trees

Space Efficient Encoding for 2D strings?

$N \log \sigma$ **or even any** $o(N \log N)$ **bits?**

Space Efficient Index for 2D strings

T	0	1	2	3	4
0	a	a	b	b	$\$$
1	a	b	b	c	$\$$
2	b	b	a	a	$\$$
3	b	c	a	b	$\$$
4	$\$$	$\$$	$\$$	$\$$	$\$$

Theorem: $O(N \log \sigma)$-bit index for LCE in ~ $O(\log^{2/3} n)$ time

LCE((0,2)&(2,0)) = 2

LCE((0,0)&(2,2)) = 2

LCE((1,2)&(2,0)) =1

Space Efficient Index for 2D strings

T	0	1	2	3	4
0	a	a	b	b	$\$$
1	a	b	b	c	$\$$
2	b	b	a	a	$\$$
3	b	c	a	b	$\$$
4	$\$$	$\$$	$\$$	$\$$	$\$$

Theorem: $O(N \log \sigma)$-bit index for LCE in ~ $O(\log^{2/3} n)$ time

LCE((0,2)&(2,0)) = 2

LCE((0,0)&(2,2)) = 2

LCE((1,2)&(2,0)) =1

Corollaries

$O(N \log \sigma + N \log \log N)$

bits index for

ISA queries via O(log n)* LCE queries
Sub-linear time SA queries $O(N/poly(\sigma \log n))$
Pattern Matching $O(M + occ + N/poly(\sigma \log n))$

2D LCE

Give two positions in the matrix,
find the largest common square sub-matrix LxL
whose top-left-corner align to those positions

- Linearize the 2D text into 1D text(s)
- Reduce the 2D-LCE query into logarithmic number of 1D-LCE's

- 1D LCE can be answered in O(1) time using an $O(n \log \sigma)$ bit structure
- 1D LCE can be answered in O(t) time using O(n/t) words
  [& text in read only], t is any parameter.
- **Difference Cover ......**

2D LCE - A simple O(L) time solution

T	0	1	2	3	4
0	a	a	b	b	\$
1	a	b	b	c	\$
2	b	b	a	a	\$
3	b	c	a	b	\$
4	\$	\$	\$	\$	\$

Concatenate all ROWS into a single 1D text,
make a compact space (1D) LCE structure over it

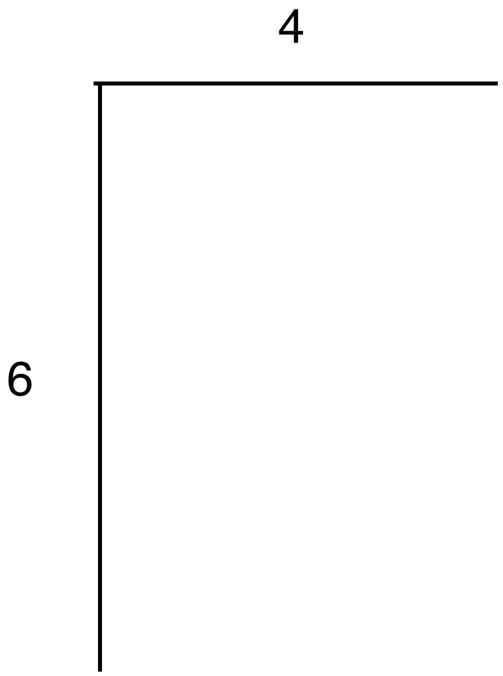
Similarly, concatenate all COLUMNS into a single 1D text,
make a compact space (1D) LCE structure over it

Maintain 1D LCE structure with space $O(n \log \sigma)$ bits and time O(1)

2D LCE - A simple O(L) time solution

Given two positions (a,b) and (c,d), initialize i = a, j = b, k = c, l = d

- *Compute LCE of R_i[j...] & R_k[l...] and LCE of C_j[i..]&C_l[k..]*
- *increment all 4 values are repeat (keep guessing L until we get it)*

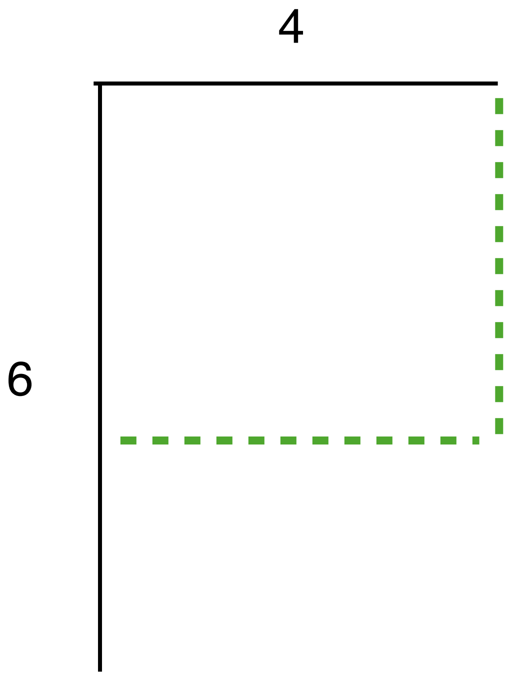


2D LCE - A simple O(L) time solution

Given two positions (a,b) and (c,d), initialize i = a, j = b, k = c, l = d

- *Compute LCE of R_i[j...] & R_k[l...] and LCE of C_j[i..]&C_l[k..]*
- *increment all 4 values are repeat (keep guessing L until we get it)*

$L \in [1,4]$

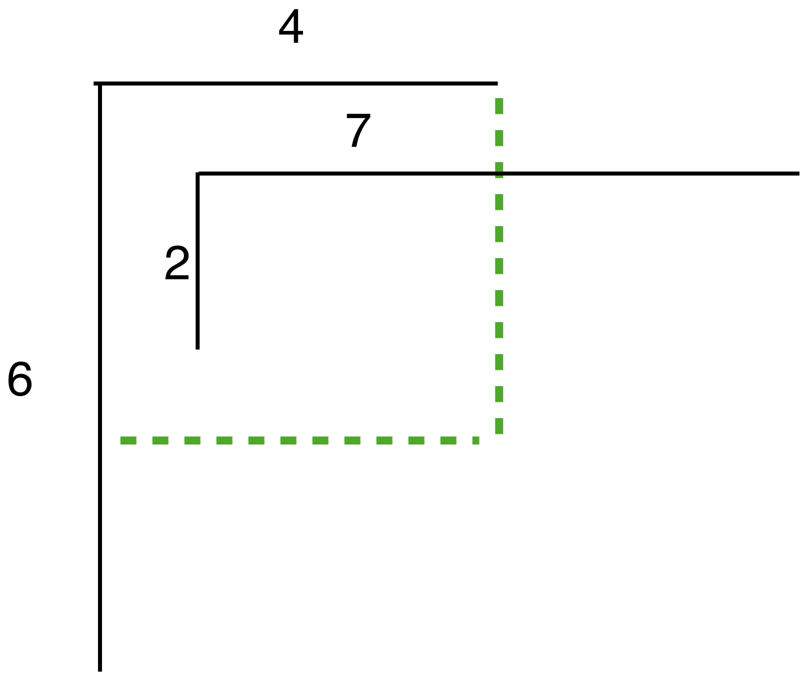


2D LCE - A simple O(L) time solution

Given two positions (a,b) and (c,d), initialize i = a, j = b, k = c, l = d

- *Compute LCE of R_i[j...] & R_k[l...] and LCE of C_j[i..]&C_l[k..]*
- *increment all 4 values are repeat (keep guessing L until we get it)*

$$L \in [1,4]$$

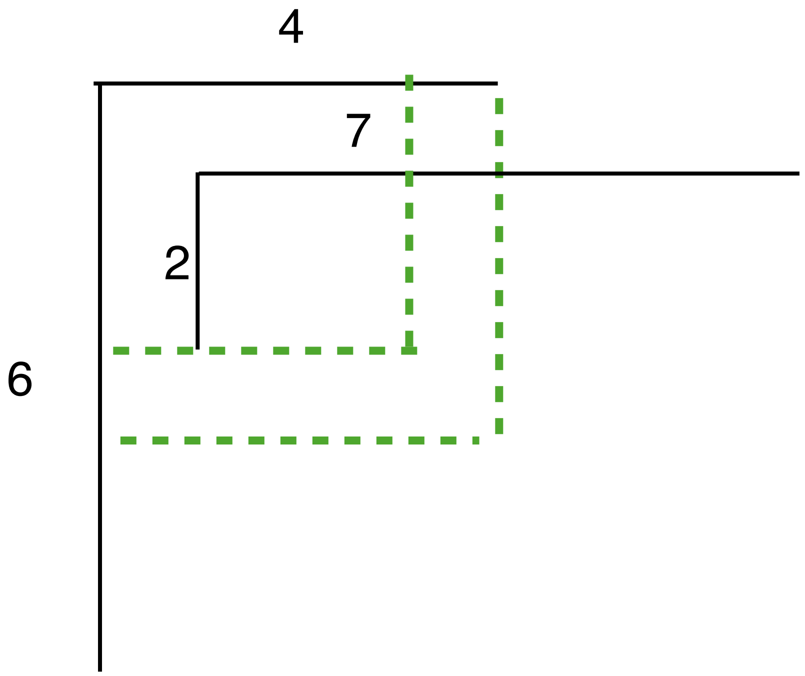


2D LCE - A simple O(L) time solution

Given two positions (a,b) and (c,d), initialize i = a, j = b, k = c, l = d

- *Compute LCE of R_i[j...] & R_k[l...] and LCE of C_j[i..]&C_l[k..]*
- *increment all 4 values are repeat (keep guessing L until we get it)*

$L \in [2,3]$

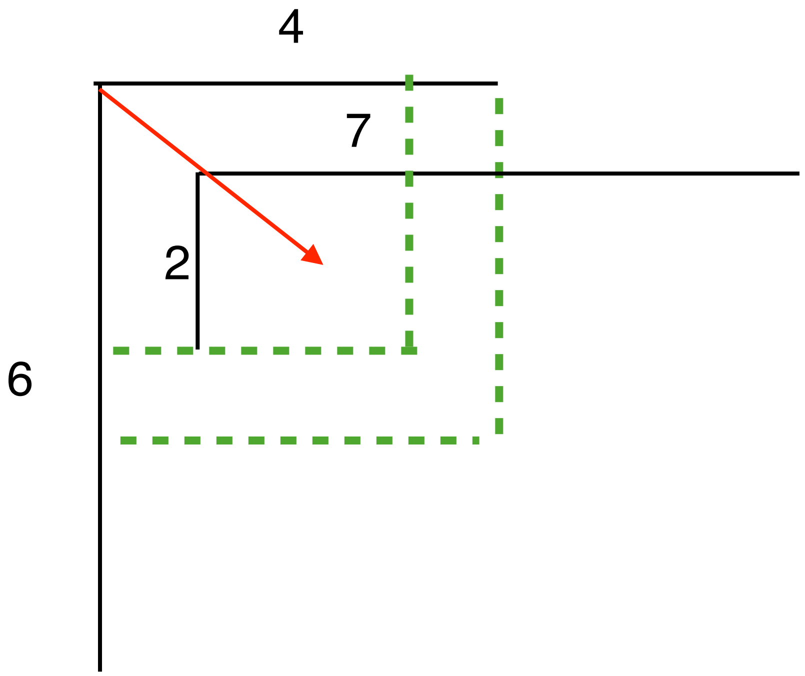


2D LCE - A simple O(L) time solution

Given two positions (a,b) and (c,d), initialize i = a, j = b, k = c, l = d

- *Compute LCE of R_i[j...] & R_k[l...] and LCE of C_j[i..]&C_l[k..]*
- *increment all 4 values are repeat (keep guessing L until we get it)*

$L \in [2,3]$



Continue .... O(L) steps and O(L) time

2D LCE - $O(\log^2 n)$ time solution

2D LCE - $O(\log_\sigma^2 n)$ time solution

Difference Cover (DC)

There exists an $S \subseteq \{1,2,3,...,n\}$ of size $O(n/\sqrt{d})$ such that $\forall(i,j), \exists h \in [0,d)$ such that $i+h, j+h \in s$

Solution: Fix $d = \log_\sigma^2 n$, Make a sparse (2D) suffix tree of sampled L-suffixes @DC across diagonals

Space = $O((n/\sqrt{d})\log n) = O(n \log \sigma)$ bits

Given two positions (i,j) and (i',j') for LCE, we know that there exists an $h < d$, such that
LCE of (i+h,j+h) & (i'+h,j'+h) are sampled positions are their LCE can be obtained in O(1) time

This means, we can run 1D-LCE queries previous algorithm just O(d) time and then jump to sparse ST

Time complexity: $O(d) = O(\log_\sigma^2 n)$

2D LCE in $\sim O(\log_\sigma n)$ time solution

Key Ideas:

Instead of taking one row/column at a time, we make slabs of sizes 1,2,4,8,... log n

However, we cannot maintain the corresponding 1D texts explicitly. So, we maintain the implicitly, and the following 1D LCE structure explicitly (we adjust "t" to keep the space low):

LCE can be answered in O(t) time using O(n/t) words [& text as some compressed structures]

The number of queries will be $O(\log \log n)$, but each query is now costly $O(\log_\sigma n)$

2D LCE in ~ $O(\log_\sigma^{2/3} n)$ time solution

Combining both Ideas:

We will maintain O(1) time 1D LCE structures for all rows and columns

Then LCE structures for selected slabs (optimized to keep the space/time small)

The final answer is obtained 3 queries

~ $O(\log_\sigma^{2/3} n)$ number of O(1) 1D-LCE queries on rows/columns
~ $O(\log\log_\sigma n)$ number of 1D-LCE queries on slabs (each costing $O(\log_\sigma^{2/3} n)$ time)
~ a single 2D LCE query on the sparse suffix tree

2D LCE in ~ $O(\log_\sigma^{2/3} n)$ time solution

Combining both Ideas:

We will maintain O(1) time 1D LCE structures for all rows and columns

Then LCE structures for selected slabs (optimized to keep the space/time small)

The final answer is obtained 3 queries

~ $O(\log_\sigma^{2/3} n)$ number of O(1) 1D-LCE queries on rows/columns
~ $O(\log\log_\sigma n)$ number of 1D-LCE queries on slabs (each costing $O(\log_\sigma^{2/3} n)$ time)
~ a single 2D LCE query on the sparse suffix tree

Summary!

Theorem: $O(N \log \sigma)$-bit structure for 2D LCE in $\sim O(\log^{2/3} n)$ time

Summary!

Theorem: $O(N \log \sigma)$-bit structure for 2D LCE in $\sim O(\log^{2/3} n)$ time

Corollaries

$O(N \log \sigma + N \log \log N)$

bits index for

ISA queries via O(log n)* LCE queries
Sub-linear time SA queries $O(N/poly(\sigma \log n))$
Pattern Matching $O(M + occ + N/poly(\sigma \log n))$

Summary!

Theorem: $O(N \log \sigma)$-bit structure for 2D LCE in ~ $O(\log^{2/3} n)$ time

Corollaries

$O(N \log \sigma + N \log \log N)$

bits index for

ISA queries via O(log n)* LCE queries
Sub-linear time SA queries $O(N/poly(\sigma \log n))$
Pattern Matching $O(M + occ + N/poly(\sigma \log n))$

We also have a REPETITION AWARE structure for 2D LCE queries

Thanks for your Listening!!!