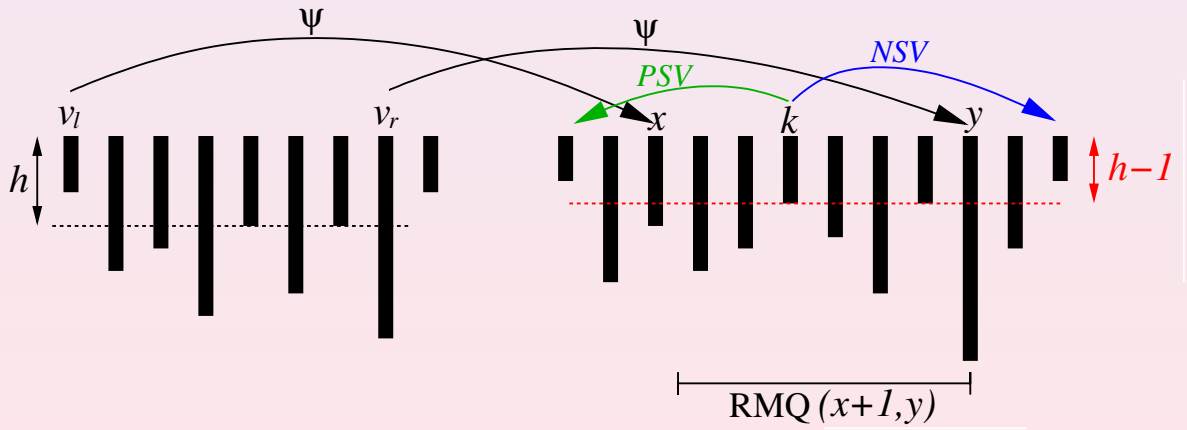
An(other) Entropy-Bounded Compressed Suffix Tree

Johannes Fischer — Univ. of Chile

Veli Mäkinen — Univ. of Helsinki

Gonzalo Navarro — Univ. of Chile

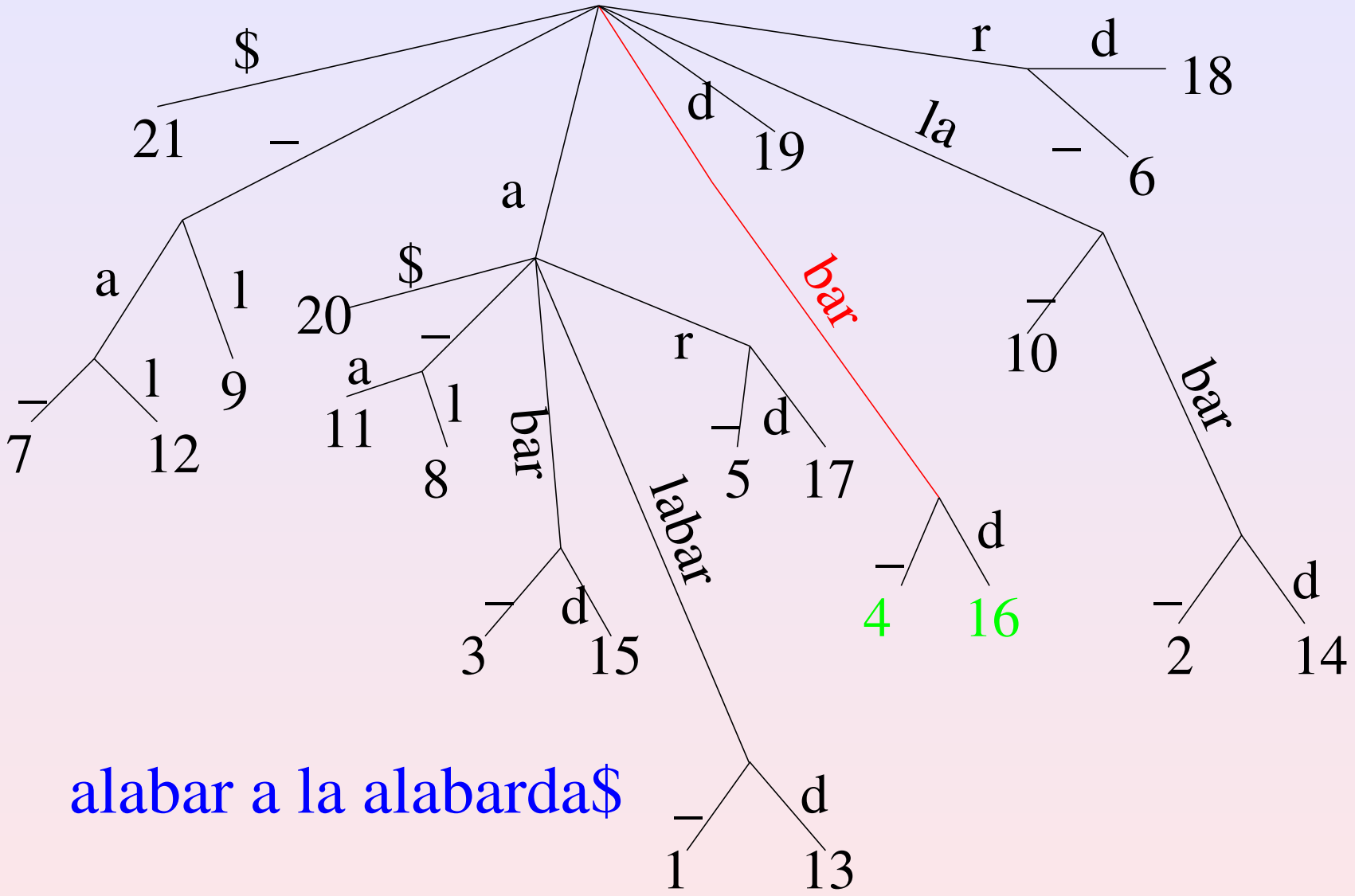


Why Compressed Suffix Trees?

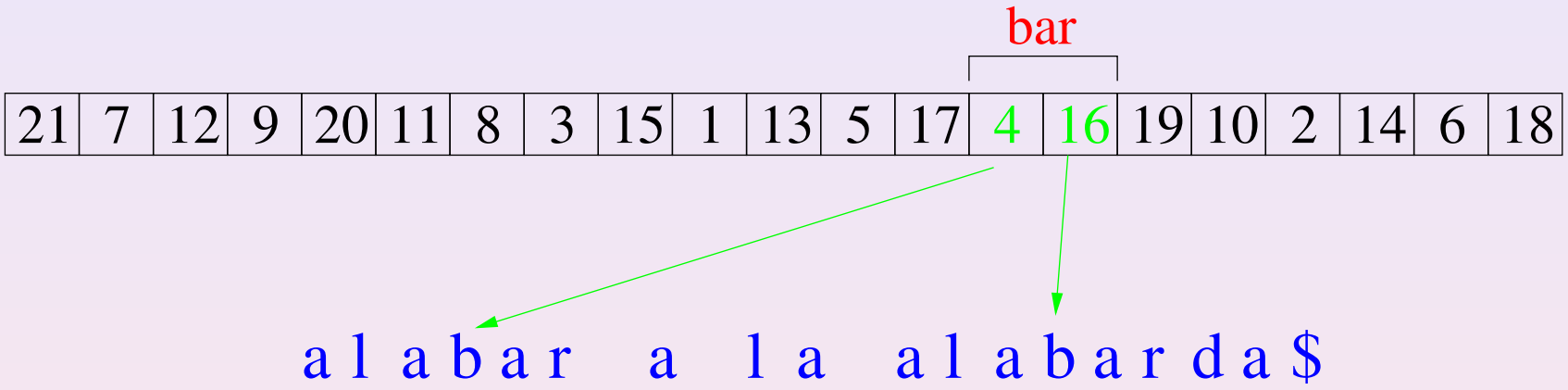
- Suffix trees are one of the most beautiful data structures ever invented.
- Loved by stringologists for their nice theoretical properties.
- Loved by bioinformaticians for their practical applications.
- Hated by everyone for their huge space consumption.
- E.g. the Human Genome, 3×10^9 base pairs, easily fits in a desktop PC.
- But its suffix tree requires 30 GB to 60 GB of memory.
- Much slower on disk, no matter how smart you are.
- A compressed suffix tree might require many more operations than a classical one, but it will anyway run much faster if it fits in RAM.

Quick Reminder

- A suffix tree is a compact trie storing all the suffixes of a text $T[1, n]$.
- Many combinatorial algorithms on strings are solved via traversing the suffix tree in different ways.
- The leaves of the suffix tree point to all text suffixes in lexicographic order.
- The array of those pointers is called the suffix array.
- Every suffix tree subtree corresponds to a suffix array interval.
- The suffix array alone can simulate some of the suffix tree traversals, but not all.
- There has been much work on compressing suffix arrays, but not much on suffix trees.



alabar a la alabarda$

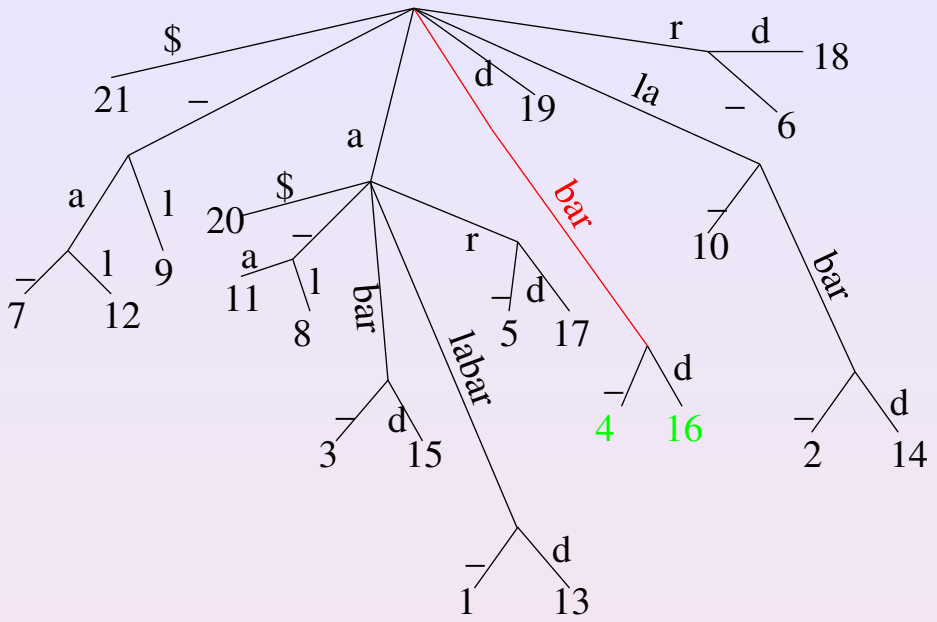

bar
21 7 12 9 20 11 8 3 15 1 13 5 17 4 16 19 10 2 14 6 18
a l a b a r a l a a l a b a r d a $

First Compressed Suffix Tree: Sadakane's CST

- A tree structure plus a suffix array.
- Tree shape is compressed to $4n$ bits using parentheses.
- The suffix array is compressed separately (Sadakane's Compressed Suffix Array (CSA)).
- Other $2n$ bits are used to represent longest common prefix (LCP) information.
- Combined with the Ψ function provided by the CSA, they can simulate any traversal.

$$\Psi(i) = A^{-1}[A[i]+1]$$

- Total space: $|CSA| + 6n = \frac{1}{\epsilon}H_0 + O(n \log\log\sigma)$ bits.
- Time for operations: most $O(1)$, yet common ones like child and letter take $O(\log\sigma \log^{\epsilon} n)$ and $O(\log^{\epsilon} n)$.



(()((()())())(()(()())(()())(()())(()())(()()))(()())()(()(()())(()())))

alabar a la alabarda$

bar

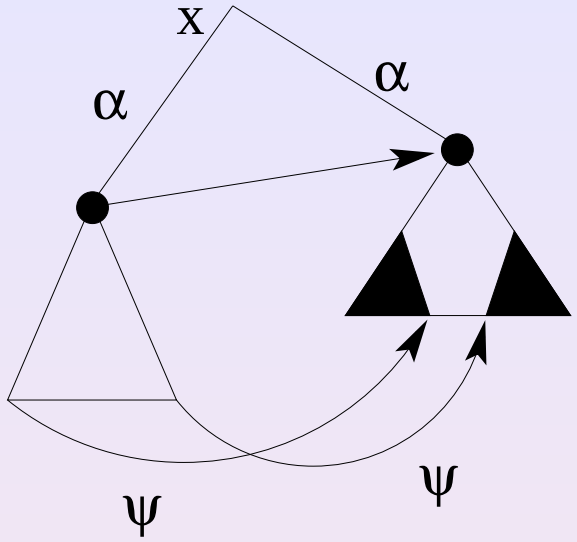
21	7	12	9	20	11	8	3	15	1	13	5	17	4	16	19	10	2	14	6	18

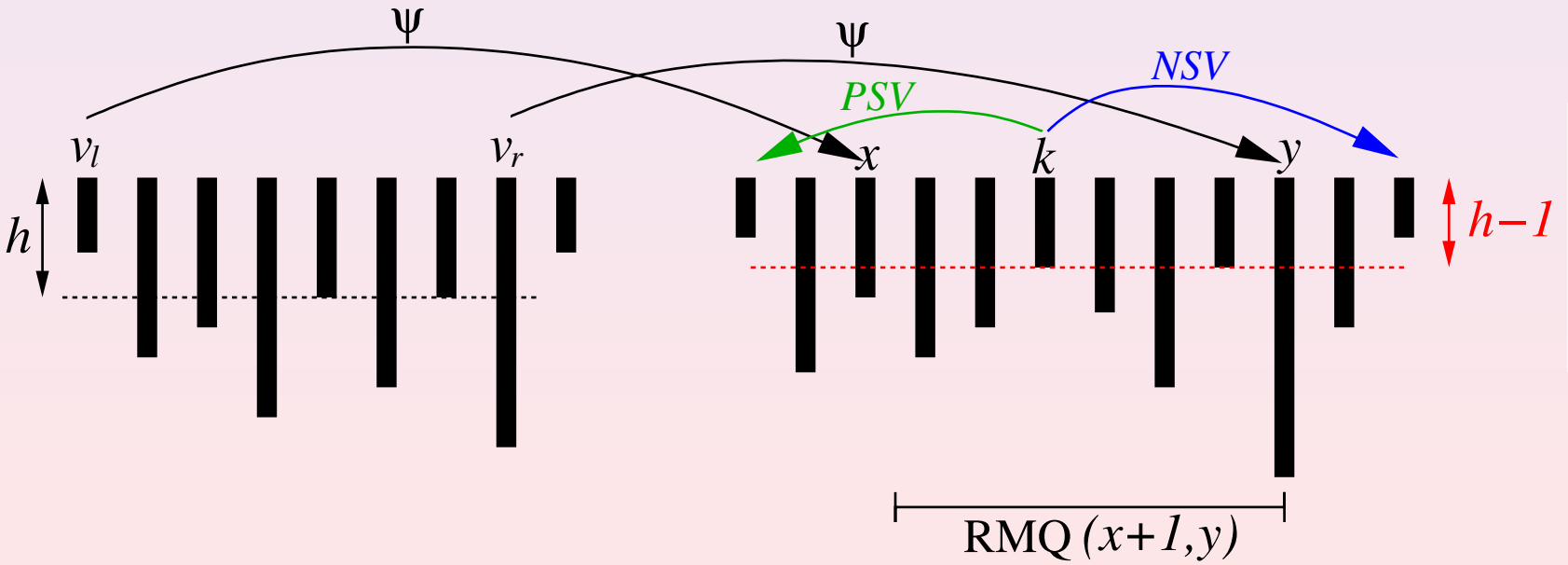
First Fully-Compressed Suffix Tree: Russo et al.'s

- Goal: get rid of the $\Theta(n)$ term in the space complexity.
- Parentheses are not represented.
- Instead, some suffix tree nodes are sampled.
- Operations on non-sampled nodes are carried out using suffix links towards a sampled node, and back.
- A compressed suffix array (an FM-index) provides Ψ, needed to emulate suffix links.
- Total space: $|FM| + o(n \log \sigma) = nH_k + o(n \log \sigma)$, "optimal".
- Time for most operations: polylogarithmic, $\omega(\log n)$.

Ours: Fully-Compressed, Sublogarithmic Times

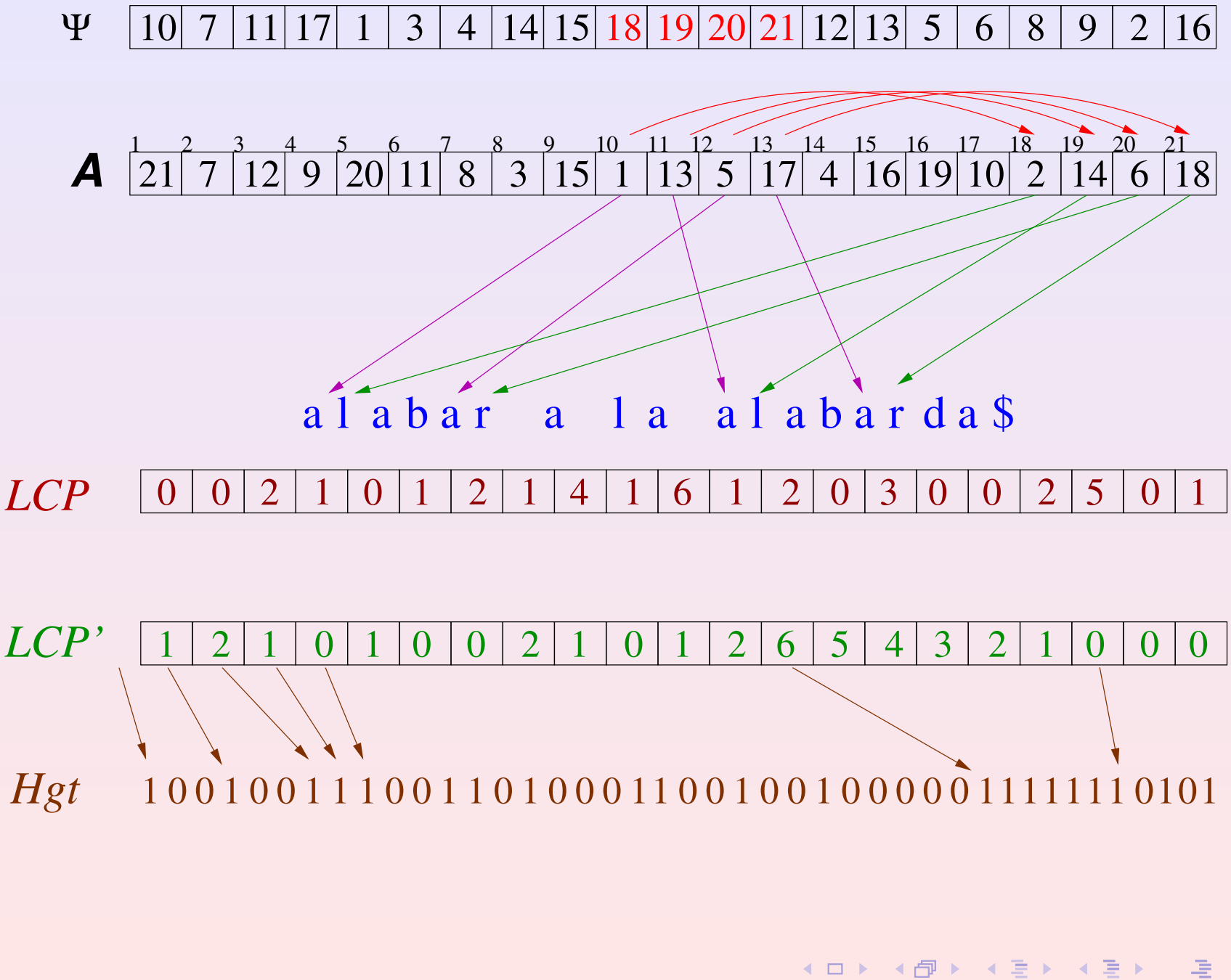
- We get rid of the tree at all.
- We use suffix array intervals to represent suffix tree nodes.
- We store a CSA (we need Ψ) as usual...
- ... and a longest common prefix (LCP) array.
- Over LCP, we need efficient range minimum queries (RMQ) and a novel previous/next smaller value (PSV/NSV) query.
- Sadakane's CST also used LCP (compressed to $2n$ bits) and RMQs on it.
- We show that this can be compressed, and solve RMQ and PSV/NSV within sublinear extra space.
- Total space: $nH_k(2\log\frac{1}{H_k} + \frac{1}{\epsilon} + O(1))$.
- Time for operations: $O(\log^{\epsilon} n)$.
- Several interesting ideas and byproducts.


x
α
α
ψ
ψ


ψ
ψ
NSV
PSV
v_l
v_r
x
k
y
h
h−1
RMQ (x+1,y)

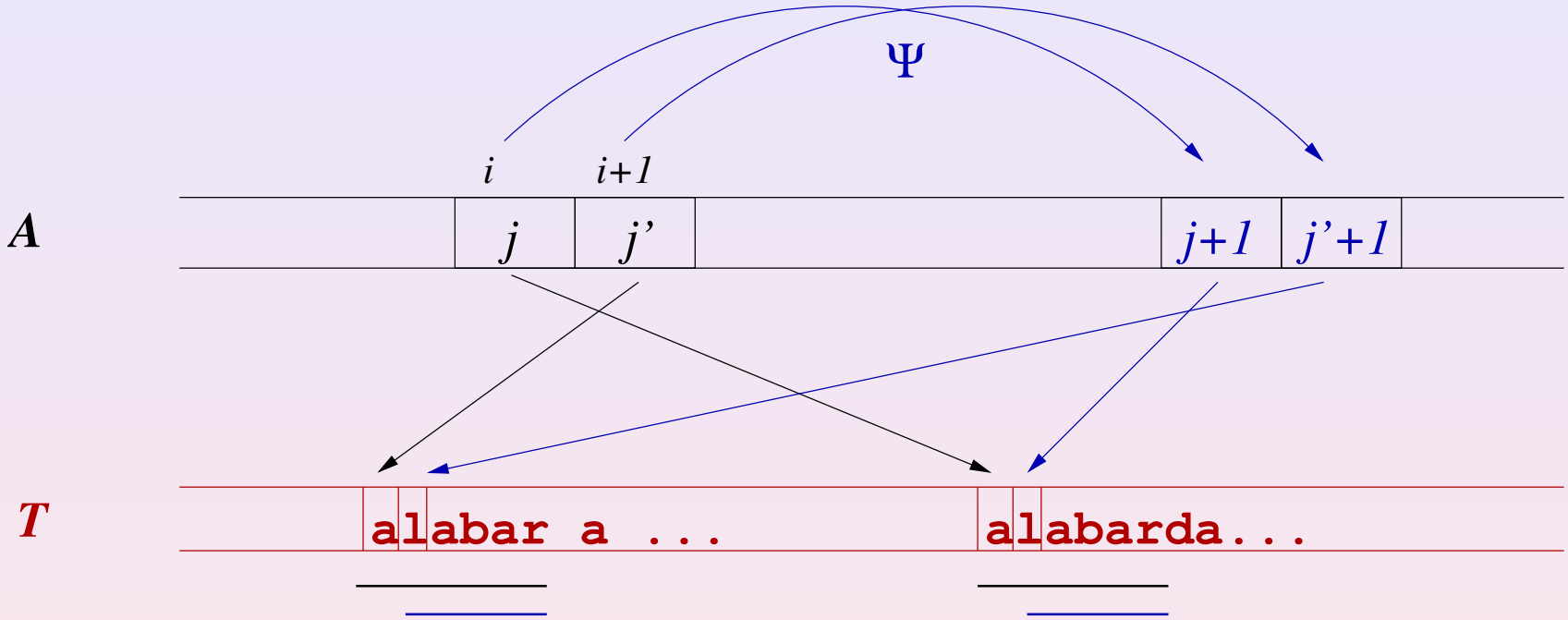
Compressing LCP

- Sadakane already noticed that LCP could be compressed to $2n$ bits.
- Because $LCP[A^{-1}[j+1]] \geq LCP[A^{-1}[j]] - 1$
- he represented LCP in text order, $LCP'[j+1] \geq LCP'[j] - 1$
- using bitmap $Hgt[1, 2n]$ where $Hgt[LCP'[j-1] + 2j - 1] = 1$ for $1 \leq j \leq n$
- so that $LCP[i] = LCP'[j = A[i]] = select(Hgt, j+1) - 2j + 1$.


Ψ
10 7 11 17 1 3 4 14 15 18 19 20 21 12 13 5 6 8 9 2 16
A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
21 7 12 9 20 11 8 3 15 1 13 5 17 4 16 19 10 2 14 6 18
a l a b a r a l a a l a b a r d a $
LCP
0 0 2 1 0 1 2 1 4 1 6 1 2 0 3 0 0 2 5 0 1
LCP'
1 2 1 0 1 0 0 2 1 0 1 2 6 5 4 3 2 1 0 0 0
Hgt
1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1

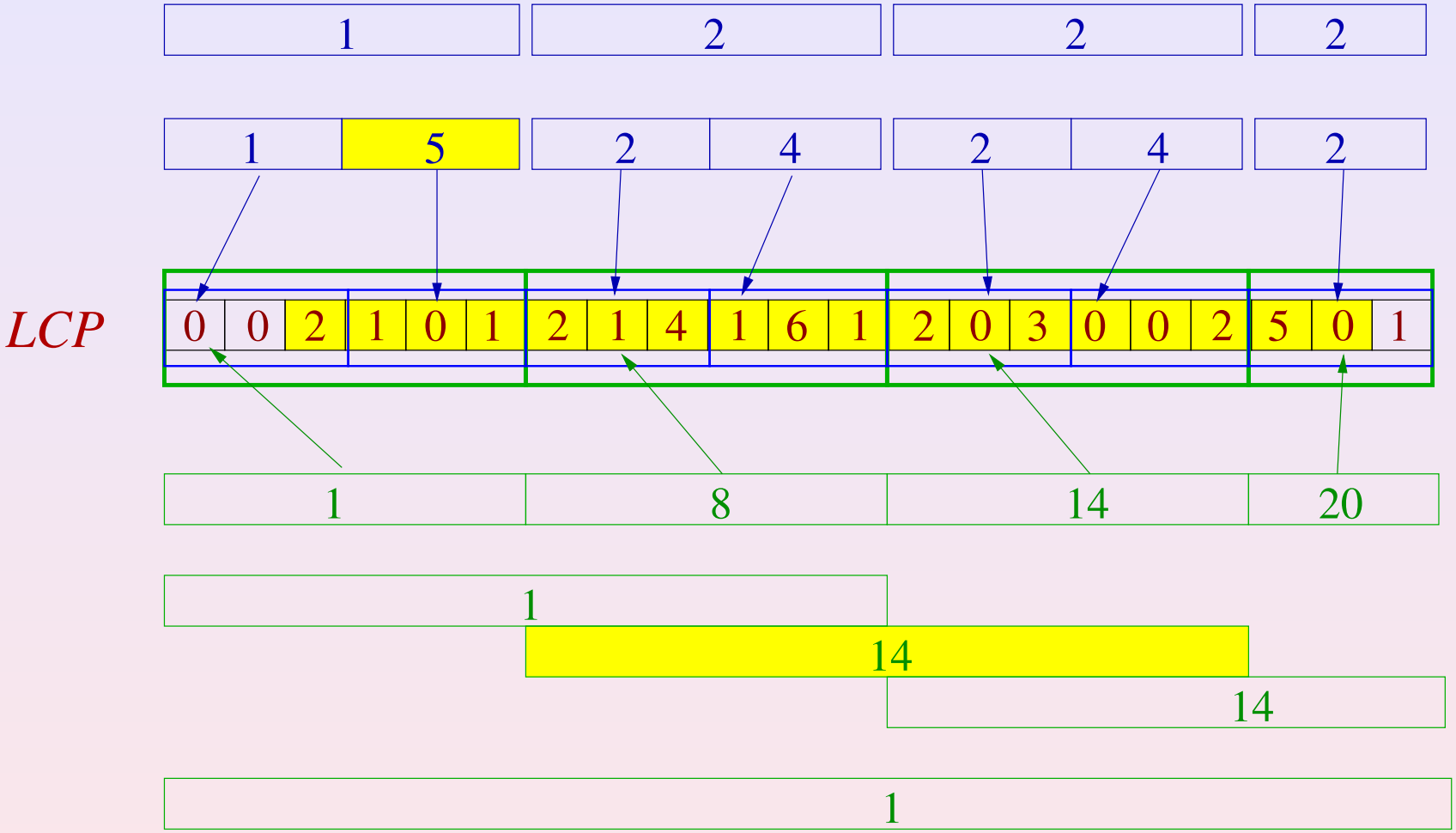
Compressing LCP

- Let us now consider a run in Ψ, that is, an area where $\Psi(i+1) = \Psi(i) + 1$.
- And consider also a chain in Ψ, $\Psi^j(i+1) = \Psi^j(i) + 1$ for consecutive j's.
- It is known that Ψ can be decomposed into $R \leq n \cdot \min(1, H_k) + \sigma^k$ runs/chains [M. and N. NJC 2005].
- Within each chain in Ψ, it holds $LCP'[j+1] = LCP'[j] - 1$.
- And hence we have have a run of 1s in Hgt.
- Thus there are at most $2R + 1$ 0/1 runs in Hgt...
- ... which can be represented using $2R \log \frac{n}{R} + O(R)$ bits...
- ... plus $o(n)$ to retain constant-time rank/select.


Ψ
i
i+1
A
j
j'
j+1
j'+1
T
alabar a ...
alabarda...

RMQ in Sublinear Space

- Classical solution achieves constant time but needs $\Omega(n)$ bits of space.
- It cuts the interval into superblocks and blocks.
- Superblocks are of $O(\log^2 n)$ positions and the answers spanning an range of 2^i integral superblocks are stored, in $O(n \log\log n / \log n)$ bits.
- Blocks span $O(\log n)$ positions, and the answers spanning 2^i integral blocks within each superblock are stored, in $O(n(\log\log n)^2 / \log n)$ bits.
- A query must consider the minimum between a range of superblocks, two ranges of blocks, and two ranges within blocks.
- A parentheses encoding for cartesian trees is used to find the minima within blocks in constant time, yet this takes $\Theta(n)$ extra space.


1
2
2
2
1
5
2
4
2
4
2
LCP
0 0 2 1 0 1 2 1 4 1 6 1 2 0 3 0 0 2 5 0 1
1
8
14
20
1
14
14
1

RMQ in Sublinear Space

- We replace the last level by a sequential traversal of the LCP values (via its compressed representation).
- We generalize the hierarchy to achieve different space/time tradeoffs.
- We find the best hierarchy $n > b_1 > b_2 \ldots$.
- For any $f(n) = \Omega(\underbrace{\log \ldots \log(n)}_{r})$...
- ... we can achieve $O(n/f(n))$ bits of space...
- ... and answer RMQ within $f(n)(\log f(n))^2)$ accesses to A.
- For example, $O(n/\log\log n)$ bits and $o((\log\log n)^2)$ time.

PSV/NSV in Sublinear Space

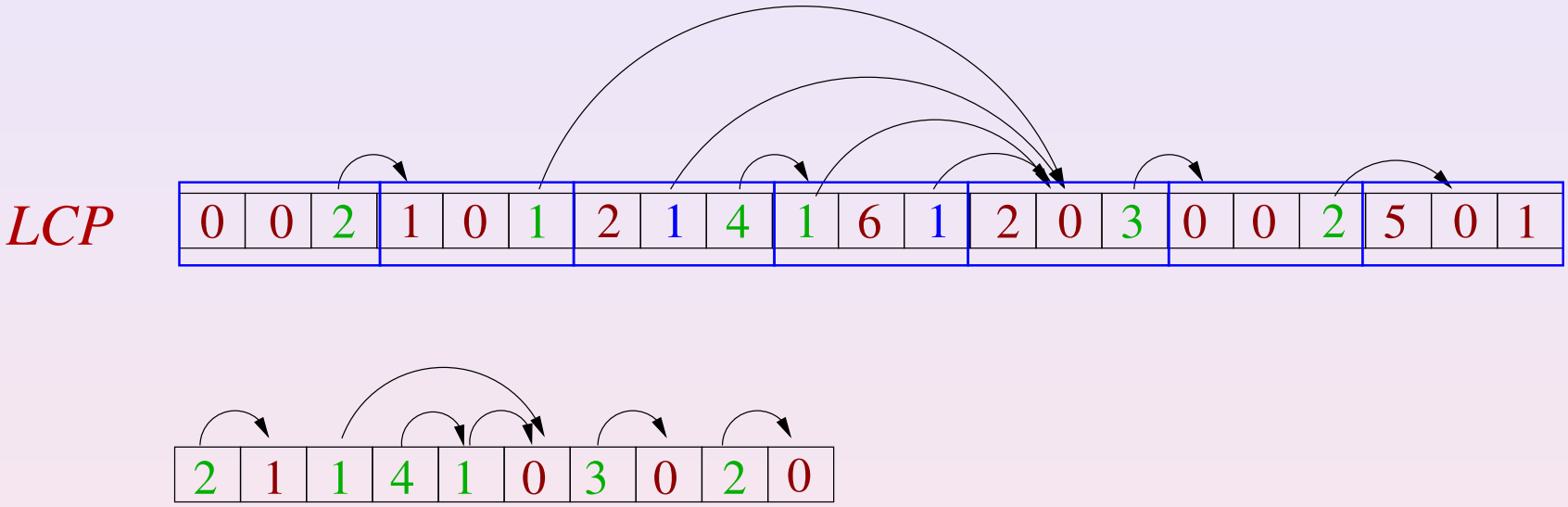
- Let us focus on Next Smaller Value queries.
- An idea resembling *findclose* in parentheses representations.
- We divide the sequence into blocks of size b.
- A position i is near if $NSV(i)$ is in its same block.
- We first scan the block looking for a near answer, solving near values in time $O(b)$.
- A far position i will be called a pioneer if the previous far position has its answer in a block different from that of i.
- There are $O(n/b)$ pioneers [Jacobson PhD thesis 1989].

PSV/NSV in Sublinear Space

- If far i is not a pioneer, then its answer is in the same block of that of its preceding pioneer.
- We mark pioneers in a compressed bitmap, $O(n/b \cdot \log b) + o(n)$ bits, and find the preceding pioneer in constant time using *rank*/*select*.
- We store pioneers' answers directly, $O(n/b \cdot \log n)$ bits.
- We scan the block of the previous pioneer from the beginning, looking for the first smaller value.
- This gives, e.g., $O(\frac{n}{\log \log n})$ bits and $O(\log n \log \log n)$ time.
- Not good enough...

PSV/NSV in Sublinear Space

- We do not store pioneers' information directly.
- We form a reduced sequence: pioneers and their answers.
- Now finding a pioneer's NSV can be translated into a similar NSV query in the reduced sequence.
- We handle the reduced sequence recursively.
- A bitmap R, compressible to $O(n/b \cdot \log b)$ bits, lets us map from the original to the reduced sequence and back.
- The P and R values of all the levels mark the positions with respect to the original LCP sequence, so that we can access any value at any level in constant time.
- For any $f(n) = O(\frac{\log n}{\log \log n})$...
- ... we can achieve $O(n/f(n))$ bits of space...
- ... and $O(f(n) \log \log n)$ time.
- E.g. $O(n/\log \log n)$ bits and $O((\log \log n)^2)$ time.


LCP
0 0 2 1 0 1 2 1 4 1 6 1 2 0 3 0 0 2 5 0 1
2 1 1 4 1 0 3 0 2 0

Final Result

- We get best results using Grossi, Gupta, and Vitter's CSA.
- This takes $(1+\frac{1}{\epsilon})nH_k + o(n\log\sigma)$ bits, for constant $\epsilon > 0$.
- It gives $O(1)$ time access to Ψ and $O(\log^{\epsilon} n \log^{1-\epsilon}\sigma)$ to A.
- On top of that we need space for the compressed LCP, $nH_k(2\log\frac{1}{H_k} + O(1)) + o(n)$ bits.
- And other $o(n)$ bits for *RMQ* and *NSV*/*PSV* queries.
- Let us fix the total bits to

$$nH_k(2\log(1/H_k) + 1/\epsilon + O(1)) + O(n/\log^{\epsilon'} n).$$

- Then the time for *RMQ* is $O(\log^{\epsilon'} n\,(\log\log n)^2)$ and *PSV*/*NSV* is $O(\log^{\epsilon'} n\,\log\log n)$ time.
- For example, we solve SDEPTH, FCHILD, NSIBLING, SLINK, and LCA, in time

$$O(\log^{\epsilon+\epsilon'} n\,(\log\log n)^2 \log^{1-\epsilon}\sigma).$$

An Example

$(\sigma = O(1), \frac{1}{\epsilon} H_k + O(\frac{n}{\log^{\epsilon'} n})$ extra bits)

Operation	Our CST	Sada's CST	Russo's CST
COUNT	1	1	1
ANCESTOR	1	1	1
LOCATE	$\log^{\epsilon} n$	$\log^{\epsilon} n$	$\log^{1+\epsilon'} n$
SDEPTH	$\log^{\epsilon+\epsilon'} n(\log\log n)^2$	$\log^{\epsilon} n$	$\log^{1+\epsilon'} n$
PARENT	$\log^{\epsilon+\epsilon'} n \log\log n$	1	$\log^{1+\epsilon'} n$
FCHILD	$\log^{\epsilon+\epsilon'} n(\log\log n)^2$	1	$\log^{1+\epsilon'} n$
NSIBLING	$\log^{\epsilon+\epsilon'} n(\log\log n)^2$	1	$\log^{1+\epsilon'} n$
SLINK, LCA	$\log^{\epsilon+\epsilon'} n(\log\log n)^2$	1	$\log^{1+\epsilon'} n$
SLINK^i	$\log^{\epsilon+\epsilon'} n(\log\log n)^2$	$\log^{\epsilon} n$	$\log^{1+\epsilon'} n$
CHILD	$\log^{\epsilon+\epsilon'} n(\log\log n)^2$	$\log^{\epsilon} n$	$\log^{1+\epsilon'} n \log\log n$
SLETTER	$\log^{\epsilon} n$	$\log^{\epsilon} n$	$\log^{1+\epsilon'} n$
LAQS	$\log^{1+\epsilon+\epsilon'} n(\log\log n)^2$	Not supported	$\log^{1+\epsilon'} n$

Conclusions

- A new compressed suffix tree.
- Tree structure totally removed, replaced by intervals.
- Free from $\Omega(n)$-bit terms in the space.
- Can achieve sublogarithmic navigation time.
- Independent interest solutions to *RMQ* and *NSV*/*PSV* problems.

Future Work

- Main challenge: a practical implementation.
  - Runs in Ψ also show up in LCP (shifted by 1).
  - A differential encoding of LCP could be compressed with Re-Pair.
  - Would achieve similar compression as González and N. [CPM 2007] for A, and fast access time.
  - Moreover, works well on incompressible texts (all $lcp \approx \log_\sigma n$).
  - Or representing LCP using [Ferragina and Venturini, SODA 2007].
  - Exploit relations between RMQ/NSV/PSV to reduce storage.
- Other (worse) challenges: dynamism, secondary memory.
- New problems: Can we do better for RMQ and PSV/NSV within $o(n)$ space?