Compressed Index with Construction in Compressed Space

Dmitry Kosolobov
St. Petersburg State University, Russia

Abstract

Suppose that we are given a string s of length n over an alphabet $\{0, 1, \ldots, n^{O(1)}\}$ and δ is the string complexity of s, a known compression measure. We describe an index on s with $O(\delta \log \frac{n}{\delta})$ space, measured in $O(\log n)$-bit machine words, which can search in s any string of length m in $O(m + (\mathrm{occ} + 1) \log^{\epsilon} n)$ time, where occ is the number of occurrences and $\epsilon > 0$ is any fixed constant (the big-O in the space bound hides factor $\frac{1}{\epsilon}$). Crucially, the index can be built in $O(n \log n)$ expected time by one left-to-right pass on the string s in a streaming fashion with $O(\delta \log \frac{n}{\delta})$ construction space. The index does not use the Karp–Rabin fingerprints, and the randomization in the construction time can be eliminated by using deterministic dictionaries instead of hash tables (with a slowdown). The search time matches currently best results and the space is almost optimal (the known optimum is $O(\delta \log \frac{n}{\delta\alpha})$, where $\alpha = \log_{\sigma} n$ and σ is the alphabet size, and it coincides with $O(\delta \log \frac{n}{\delta})$ when $\delta = O(n/\alpha^2)$). This is the first index that can be constructed within such space and with such time guarantees. To avoid uninteresting marginal cases, all above bounds are stated for $\delta \geq \Omega(\log \log n)$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases compressed index, pattern matching, string complexity, grammar, block tree

Digital Object Identifier 10.4230/LIPIcs.CPM.2026.25

Related Version *Full Version*: `https://arxiv.org/abs/2602.13735` [38]

Funding Supported by the Russian Science Foundation (RSF), project 24-71-00062.

1 Introduction

Given a string s of length n over the alphabet $\{0, 1, \ldots, \sigma - 1\}$ with $\sigma \leq n^{O(1)}$, a compressed index is a data structure that stores s in a compressed form supporting fast substring search. Currently best indexes use $O(\delta \log \frac{n}{\delta\alpha})$ space [34, 35, 32, 33], measured in $O(\log n)$-bit machine words, where $\alpha = \log_{\sigma} n$ and δ is a compression measure called string complexity [47], and they can search a substring of length m in $O(m + (\mathrm{occ} + 1) \log^{\epsilon} n)$ time, where occ is the number of occurrences and $\epsilon > 0$ is any fixed constant (the big-O in the space bound hides factor $\frac{1}{\epsilon}$). This space $O(\delta \log \frac{n}{\delta\alpha})$ is known to be tight for a wide range of parameters [32, 33] and it lowerbounds the space of most dictionary-based compressed indexes such as indexes on the Lempel–Ziv parsing [16, 52], run-length grammars [17, 31], Burrows–Wheeler transform [24], and string attractors [29] (see [43] for an overview).

There are essentially two primary methods to achieve $O(m \log n)$ search time within the space $O(\delta \log \frac{n}{\delta})$, which both enhance the basic idea of Claude and Navarro [18]: (1) a neat scheme [24] first outlined by Gagie et al. [23] that combines the Karp–Rabin hashing [22, 27] and z-fast tries [4, 5]; (2) a variant of the so-called locally consistent parsing [17, 19, 26, 40]. Known indexes that achieve $O(m + \log^{\epsilon} n)$ search time combine both approaches [17, 33]. Method (1) is applicable for many indexes; unfortunately, it has a serious drawback: the Karp–Rabin hashing must be collision-free for all substrings of length 2^k, for $k = 1, 2, \ldots$, and the only known algorithm that verifies this condition uses $O(n)$ space and $O(n \log n)$ time [10], which is prohibitive in use cases where the uncompressed data barely fits into memory. This issue hinders a wider usage of such indexes and it is the reason for the scarcity of experimental

studies that try to translate the theoretical state-of-the-art methods to practice (we note that there are many works on construction of indexes, e.g., [1, 20, 28, 37, 50, 51], but they never use the collision-free Karp–Rabin hashing as the best theoretical results do).

We make a step forward to solve this problem by presenting an index with $O(\delta \log \frac{n}{\delta})$ space and $O(m + (\mathrm{occ}+1)\log^{\epsilon} n)$ search time that can be built in $O(n \log n)$ expected time by one left-to-right pass in a streaming fashion with $O(\delta \log \frac{n}{\delta})$ construction space. We do not use the Karp–Rabin hashes and the randomization in the construction can be removed using deterministic dictionaries instead of hash tables (with a slowdown by an $O(\sqrt{\frac{\log n}{\log\log n}})$ factor, see [3, 48]). The result is purely theoretical due to large constants under the big-O but, we believe, its ideas might inspire practical designs. We conjecture that the index can be further improved to $O(\delta \log \frac{n}{\delta\alpha})$ space (also the construction space) and $O(m/\alpha + (\mathrm{occ}+1)\log^{\epsilon} n)$ time in the string packed model (see [11, 41]); this direction is left for a future work.

Throughout, s is the input string of length n with letters $s[0], s[1], \ldots, s[n-1]$ from an alphabet $\{0, 1, \ldots, n^{O(1)}\}$. For i, j, let $s[i\,..\,j]$ be the *substring* $s[i]s[i+1]\cdots s[j]$ (empty if $i > j$). For sequence $t = t_0, t_1, \ldots, t_{m-1}$ (maybe a string), denote by $\overleftarrow{t} = t_{m-1}, \ldots, t_1, t_0$ its *reversal*. Denote $|t| = m$, $s[i\,..\,j) = s[i\,..\,j-1]$, $[i\,..\,j] = \{i, i+1, \ldots, j\}$, $[i\,..\,j) = [i\,..\,j-1]$.

The string complexity, δ, for s is defined as $\delta = \max\{d_k(s)/k \colon k \in [1\,..\,n]\}$, where $d_k(s)$ denotes the number of distinct substrings of s of length k [35, 47].

We heavily use the concept of *blocks*: a block B for string s is an object attached to a certain substring $s[i\,..\,j]$; denote $\mathsf{b}(B) = i$ and $\mathsf{e}(B) = j$. The blocks are not "fragments": two distinct blocks B and B' can be attached to the same substring $s[i\,..\,j]$. If unambiguous, we treat a block B as a string, writing $|B|$, $B = s[i'\,..\,j']$ (with not necessarily $i' = \mathsf{b}(B)$), etc. Two blocks B and B' are *equal as strings* if $s[\mathsf{b}(B)\,..\,\mathsf{e}(B)] = s[\mathsf{b}(B')\,..\,\mathsf{e}(B')]$. When defining a new block, we usually use words like "generate", "create", "copy" (when the new block is a copy of an old one) and it must be clear to which substring the new block is attached. We sometimes treat the concept of blocks loosely but, hopefully, it will be clear from the context.

2 Main Ideas

What follows is a high level description of main ideas behind our index. Details and precise definitions follow. We first focus on a static index. Its construction algorithm is in Section 6.

Recent results on block trees [34, 44] showed that the space $O(\delta \log \frac{n}{\delta})$ for indexes is surprisingly easy to obtain by, roughly, the following scheme (or its variants): one constructs a hierarchy of blocks with $\log n$ levels such that the level-k blocks are $\lceil n/2^k \rceil$ substrings $s[i\cdot 2^k\,..\,(i+1)2^k)$ of length 2^k (the rightmost block might be shorter) and, for $k > 0$, each level-k block has two child blocks from level $k-1$: its prefix and suffix of length 2^{k-1}, respectively; then, each block $B = s[i \cdot 2^k\,..\,(i+1)2^k)$ for which the substring $\bar{B} = s[(i-1)2^k\,..\,(i+2)2^k)$ has an occurrence at a position smaller than $(i-1)2^k$ is encoded by a pointer to the leftmost occurrence of B in s and all descendants of B in the hierarchy are removed (for precise and detailed definitions, see [6, 7, 34, 42, 44]). The result is a block tree [6, 42].

After a closer inspection of the proof from [35] of the bound $O(\delta \log \frac{n}{\delta})$ for the block tree size, one can notice that the bound still holds if the block boundaries are slightly "jiggled". Our idea is to construct a "jiggy block tree", in which level-k blocks are not of exact size 2^k but somewhat close to this "on average", and each block might have up to $O(\log \log n)$ children. The variability of block lengths will be used to form a kind of locally consistent parsing, more specifically, we will use a "cut" version of the so-called deterministic coin tossing (DCT) [19] and the blocks will be built level by level bottom-up using this "cut DCT" (the idea of such level-by-level construction itself is not novel [46] but it is usually used with

the full DCT and it leads to a grammar of size $\Omega(\delta \log \frac{n}{\delta} \log^* n)$, not to a kind of block tree). Then, the same arguments as in [35] show that the resulting tree has $O(\delta \log \frac{n}{\delta})$ internal nodes. However, some nodes might have up to $\Theta(\log \log n)$ leaf children, blowing up the space by an $\Omega(\log \log n)$ factor, which seems unavoidable when using the DCT [12, 39, 40, 46, 49, 50].

We eliminate the factor $\Omega(\log \log n)$ by introducing a novel intermediate step in the level by level bottom-up construction: given a block B with $O(\log \log n)$ children $B_i, \ldots, B_j$, we greedily unite the children from left to right into larger "intermediate blocks" $B' = B_m \cdots B_{m+r-1}$ such that the sequence of blocks $B_m \cdots B_{m+r-1}$ occurs somewhere to the left in the block hierarchy; then, we store in the intermediate block B' a pointer to this earlier occurrence. Due to the property of "local consistency" of DCT, the pointers will be somewhat "aligned" on the block structure and our jiggly block tree will resemble a grammar with elements of the block tree structure (like a collage system [30] but more structured). There are many details in this scheme to make it efficient that are explained in the sequel.

For the pattern matching, we follow the standard approach based on the locally consistent parsing and z-fast tries [17, 33] but we replace the Karp–Rabin fingerprints [27] with new deterministic fingerprints that are derived from our locally consistent block structure.

3 Hierarchy of Blocks

At the core of our index is a *hierarchy of blocks* (see Fig. 1, the leftmost picture): it has $O(\log n)$ levels where the topmost level has one block equal to s and the bottom level 0 contains n one-letter blocks $s[0], \ldots, s[n-1]$; the concatenation of all blocks from one level equals s; for $\ell > 0$, each level-ℓ block B either is a copy of a level-$(\ell-1)$ block B' with $\mathsf{b}(B) = \mathsf{b}(B')$ and $\mathsf{e}(B) = \mathsf{e}(B')$ or is the concatenation of corresponding consecutive level-$(\ell-1)$ blocks. We define the hierarchy of blocks by a bottom-up construction process.

The bottom level-0 blocks are n one-letter blocks $s[0], s[1], \ldots, s[n-1]$. On a generic step with $k \geq 0$, we have a sequence of blocks $B_1, \ldots, B_b$ of level $2k$ whose concatenation is s itself. To produce new blocks of levels $2k+1$ and $2k+2$, the construction process will unite some adjacent blocks or will copy some blocks unaffected; in particular, all blocks of length $>2^k$ are copied to levels $2k+1$ and $2k+2$ unaffected. Every block B is assigned an $O(\log n)$-bit integer identifier $\mathsf{id}(B)$; the level-0 blocks have identifiers equal to the letters they represent (recall that the alphabet is $\{0, 1, \ldots, n^{O(1)}\}$). Blocks with equal identifiers are equal as strings but the converse is not necessarily true. We choose an arbitrary unambiguous method to encode any pair of $O(\log n)$-bit integers x, y into one $O(\log n)$-bit integer, denoted $\langle x, y \rangle$. The process maintains a counter to assign new unique identifiers to new blocks.

Level $2k+1$. In the level-$2k$ blocks $B_1, \ldots, B_b$, we identify all maximal "runs" of blocks $B_i, B_{i+1}, \ldots, B_{i+r-1}$ with equal identifiers such that $|B_j| \leq 2^k$, for $j \in [i \,..\, i+r)$, and $i = 1$ or $\mathsf{id}(B_{i-1}) \neq \mathsf{id}(B_i)$, and $i + r - 1 = b$ or $\mathsf{id}(B_{i+r}) \neq \mathsf{id}(B_{i+r-1})$. Each such run for which $r > 1$ is substituted by a new block $B = B_i B_{i+1} \cdots B_{i+r-1}$ whose identifier encodes the pair $\langle \mathsf{id}(B_i), r \rangle$. Hence, any two distinct runs of the same length $r > 1$ with the same identifiers $\mathsf{id}(B_i)$ of the run blocks are substituted by blocks with the same identifier $\langle \mathsf{id}(B_i), r \rangle$. Thus, the blocks for level $2k+1$ consist of copies of blocks from level $2k$ with length $>2^k$, copies of blocks that formed runs of length one, and the blocks substituted in place of longer runs.

Level $2k+2$. For any distinct integers $x, y \geq 0$, denote by $\mathsf{bit}(x, y)$ the index (0-based) of the lowest bit in which the bit representations of x and y differ; e.g., $\mathsf{bit}(8, 0) = 3$. Denote $\mathsf{vbit}(x, y) = 2 \cdot \mathsf{bit}(x, y) + a$, where a is the bit of x with index $\mathsf{bit}(x, y)$. Cole and Vishkin based their deterministic coin tossing (DCT) technique on the following observation.

▶ **Lemma 1** (see [19, 39]). *Given a string $a_0a_1\cdots a_m$ over an alphabet $[0\,..\,2^u)$ such that $a_{i-1}\neq a_i$ for any $i\in[1\,..\,m]$, the string $b_1b_2\cdots b_m$ such that $b_i=\mathsf{vbit}(a_{i-1},a_i)$, for $i\in[1\,..\,m]$, satisfies $b_{i-1}\neq b_i$, for any $i\in[2\,..\,m]$, and $b_i\in[0\,..\,2u)$, for any $i\in[1\,..\,m]$.*

Let $B_1,\ldots,B_b$ be all blocks on level $2k+1$. For each B_i, we assign two identifiers, denoted $\mathsf{id}'(B_i)$ and $\mathsf{id}''(B_i)$. If $|B_i|>2^k$, define $\mathsf{id}'(B_i)=\infty$. If $|B_i|\leq 2^k$, then $\mathsf{id}(B_{i-1})\neq\mathsf{id}(B_i)$ since the previous stage united all runs; define $\mathsf{id}'(B_i)=\mathsf{vbit}(\mathsf{id}(B_{i-1}),\mathsf{id}(B_i))$ if $i>1$ and $|B_{i-1}|\leq 2^k$, and $\mathsf{id}'(B_i)=\infty$ otherwise. Define $\mathsf{id}''(B_i)=\mathsf{vbit}(\mathsf{id}'(B_{i-1}),\mathsf{id}'(B_i))$ if $i>1$ and $\mathsf{id}'(B_{i-1})\neq\infty$ and $\mathsf{id}'(B_i)\neq\infty$, and define $\mathsf{id}''(B_i)=\infty$ otherwise. This computation of id'' corresponds to two first steps of the standard DCT, i.e., it is a "cut" variant of DCT.

For any maximal range of blocks $B_p,\ldots,B_q$ such that $|B_h|\leq 2^k$ for each $h\in[p\,..\,q]$ (i.e., $p=1$ or $|B_{p-1}|>2^k$, and $q=b$ or $|B_{q+1}|>2^k$), we have $\mathsf{id}''(B_p)=\mathsf{id}''(B_{p+1})=\infty$ (or just $\mathsf{id}''(B_p)=\infty$ if $p=q$) and, by Lemma 1, for each $h\in[p{+}2\,..\,q]$, we have $\mathsf{id}''(B_{h-1})\neq\mathsf{id}''(B_h)$ and $0\leq\mathsf{id}''(B_h)\leq O(\log\log n)$ (here we used the assumption $\mathsf{id}(B_h)\leq n^{O(1)}$). We will split $B_p,\ldots,B_q$ into ranges ending at points corresponding to local minima of id''. To this end, we mark each block B_i, with $i\in[1\,..\,b]$, for which one of the following conditions holds:

(a) $|B_i|>2^k$ or $|B_{i+1}|>2^k$ or $i=b$ (the last block on level $2k+1$);

(b) $i>2$ and $\infty>\mathsf{id}''(B_{i-2})>\mathsf{id}''(B_{i-1})$ and $\mathsf{id}''(B_{i-1})<\mathsf{id}''(B_i)<\infty$ (i.e., $\mathsf{id}''(B_i)$ is immediately preceded by a local minimum).

The marking splits all blocks $B_1,\ldots,B_b$ on level $2k+1$ into disjoint ranges as follows: each range $B_i,\ldots,B_j$ consists of unmarked blocks $B_i,\ldots,B_{j-1}$ and the marked block B_j and either $i=1$ or B_{i-1} is marked. For each such range $B_i,\ldots,B_j$: if $i=j$, the process copies B_i to level $2k+2$; if $i<j$, it creates a new block $B=B_i\cdots B_j$ for level $2k+2$ assigning to it a new identifier unless another block B' produced by the same sequence $\mathsf{id}(B_i),\ldots,\mathsf{id}(B_j)$ has already appeared earlier in the process (on any level), in which case we assign $\mathsf{id}(B)=\mathsf{id}(B')$. Since $\mathsf{id}''\leq O(\log\log n)$, local minima for id'' occur often and, thus, $j-i\leq O(\log\log n)$.

Properties of blocks. The main properties of the hierarchy are its "local consistency" and "local sparsity". The proof of the former is standard [35] and it was moved to Appendix A.

▶ **Lemma 2** (local consistency). *Fix $\ell=2k$ or $\ell=2k+1$. Let $B_1,B_2,\ldots,B_b$ denote all level-ℓ blocks. For any i, j and block B_h, if $s[i{-}2^{k+4}\,..\,j{+}2^{k+4}]=s[\mathsf{b}(B_h){-}2^{k+4}\,..\,\mathsf{e}(B_h){+}2^{k+4}]$, then there exists a level-ℓ block $B_{h'}$ such that $\mathsf{id}(B_{h'})=\mathsf{id}(B_h)$, $\mathsf{b}(B_{h'})=i$, and $\mathsf{e}(B_{h'})=j$.*

We call a position $h\in[0\,..\,n)$ a *block boundary* for level ℓ if $h=\mathsf{e}(B)$ for some level-ℓ block B. The "local sparsity" formalizes the intuition that blocks on levels $2k$ and $2k+1$ have length $\geq 2^k$ "on average". The proof is quite technical and it mostly repeats the arguments from the proof of [39, Lemma 11] word by word. The proof can be found in the full version of the paper [38, Appendix A].

▶ **Lemma 3** (local sparsity). *For any $i<j$, the number of block boundaries in $[i\,..\,j)$ on levels $2k$ and $2k+1$ is at most $2^6\lceil\frac{j-i}{2^k}\rceil$. In particular, there are $O(n/2^k)$ blocks on these levels.*

Lemma 3 implies that there are $O(\log n)$ levels in the hierarchy. We conclude the section with a technical lemma. Its proof is quite straightforward and has been moved to Appendix A.

▶ **Lemma 4.** *Any blocks B and B' with $\mathsf{id}(B)=\mathsf{id}(B')$ are equal as strings and both were created on the same level.*

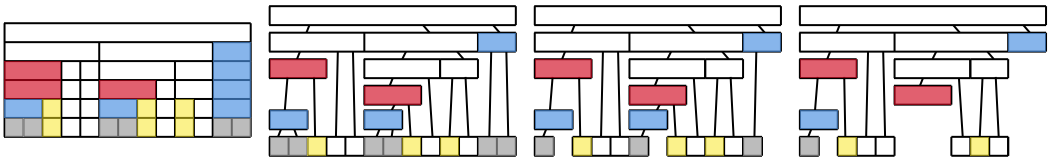

Figure 1 A schematic depiction of the hierarchy of blocks H. Colored rectangles with the same color depict blocks with the same id; white rectangles are blocks with any id. From left to right: the hierarchy H, H with rules 1–2 applied, H with rule 4 applied, H with rule 3 applied (i.e., it is $\hat{H}$).

4 Jiggly Block Tree

The described hierarchy of blocks for s naturally forms a tree H with blocks as vertices in which the root is the (only) topmost block and the parent of any level-ℓ block B with $B \neq s$ is the level-$(\ell+1)$ block B' such that $[\mathsf{b}(B') .. \mathsf{e}(B')] \supseteq [\mathsf{b}(B) .. \mathsf{e}(B)]$ (note that B and B' may be equal as strings). For our index, we create a tree $\hat{H}$ that is a copy of H with vertices removed according to the following rules 1–4 while it is possible to apply them (see Fig. 1):

1. if a level-0 block B has parent B' with $\mathsf{id}(B) = \mathsf{id}(B')$, merge B and B' by removing B' and connecting B to the parent of B';
2. if, for $\ell > 0$, a level-ℓ block B has parent B' with $\mathsf{id}(B) = \mathsf{id}(B')$, merge B and B' by removing B and connecting all children of B to B';
3. if, for $\ell > 0$ and $\ell' > 0$, a level-ℓ block B has a "preceding" level-ℓ' block B' such that $\mathsf{id}(B) = \mathsf{id}(B')$ and $\mathsf{b}(B') < \mathsf{b}(B)$, remove all descendants of B;
4. if, for $\ell > 0$, all children of a level-ℓ block B have equal id, remove them all except for the leftmost child (note that, by removing a child, we remove its descendants).

The idea of such construction is not novel (however, usually the full DCT is used, not the "cut" DCT as we do) and it leads to an index with $\omega(\delta \log \frac{n}{\delta})$ space [12, 39, 40, 46, 49, 50]. As it turns out, the obtained tree has $O(\delta \log \frac{n}{\delta})$ non-leaf vertices and the main issue with space is due to some vertices having up to $O(\log \log n)$ leaves from levels $2k+1$. To reduce the space, our idea is to parse the sequences of leaves by a specialized variant of LZ77 [52]. The construction is quite technical and requires some preparations to define it.

Let $B_1, \ldots, B_b$ be all blocks on a level $2k+1$. Recall that the block construction process has marked some of the blocks, thereby splitting them into ranges $B_i, \ldots, B_j$, where $B_i, \ldots, B_{j-1}$ are unmarked and B_j is marked. Fix such range $B_i, \ldots, B_j$ with $i < j$ such that $B_i, \ldots, B_j$ were not removed from $\hat{H}$ after the rules 1–4 above. Let B be the parent of $B_i, \ldots, B_j$. We are to parse $B_i, \ldots, B_j$ into subranges. To define the parsing, we need some notation.

For any block B_m and $\ell \geq 0$, choose the largest $h \leq \ell$ such that, for all $h' \in [m-h .. m)$, $B_{h'}$ is not marked; define $\mathsf{left}(m, \ell)$ as the sequence $\beta, \mathsf{id}(B_{m-h}), \mathsf{id}(B_{m-h+1}), \ldots, \mathsf{id}(B_{m-1})$, where β is empty (i.e., it is really not in the sequence) if $h = \ell$, and β is a special symbol \$ if $h < \ell$. Choose the largest $h \leq \ell$ such that, for all $h' \in [m .. m+h-1)$, $B_{h'}$ is not marked (but B_{m+h-1} may be marked); define $\mathsf{right}(m, \ell)$ as the sequence $\mathsf{id}(B_m), \mathsf{id}(B_{m+1}), \ldots, \mathsf{id}(B_{m+h-1})$.

We greedily parse $B_i, \ldots, B_j$ into subranges from left to right: suppose that $B_i, \ldots, B_{m-1}$ have already been parsed (initially $m = i$), let $B_m, \ldots, B_{m+r-1}$ be the longest subrange with $m+r-1 \leq j$ for which there is $m' < m$ such that $\mathsf{left}(m', 4r) = \mathsf{left}(m, 4r)$ and $\mathsf{right}(m', 5r) = \mathsf{right}(m, 5r)$ (note that $\mathsf{id}(B_m), \ldots, \mathsf{id}(B_{m+r-1})$ must be a prefix of $\mathsf{right}(m, 5r)$); let $r = 1$ if there is no such $B_m, \ldots, B_{m+r-1}$; once r is determined, we have obtained a new subrange $B_m, \ldots, B_{m+r-1}$ and we continue the parsing for $B_{m+r}, \ldots, B_j$, assigning $m := m + r$.

Define a new tree J, called the ***jiggly block tree***, that is a copy of $\hat{H}$ with the following modifications. For each level $2k+1$, we consider each range $B_i, \ldots, B_j$ as described above such that all $B_i, \ldots, B_j$ were not removed from $\hat{H}$ after the rules 1–4, and we parse $B_i, \ldots, B_j$

into subranges as described above; for each subrange $B_m, \dots, B_{m+r-1}$ with $r > 1$, we remove from J the blocks $B_m, \dots, B_{m+r-1}$ (which are children of B) and insert in their place a new block B' with parent B and no children; B' is called *intermediate*. We do not assign id to B'.

By a simple analysis, one can show that any "original" block B of H cannot disappear from J, always a copy of B remains, which is formalized as follows (the proof is in Appendix B).

▶ **Lemma 5.** *For $d \geq 0$, let B be the leftmost topmost block with $\mathsf{id}(B) = d$ and $|B| > 1$ in the hierarchy H, i.e., $\mathsf{b}(B)$ is minimal for B among all blocks with this id and B is such block with maximal level. The jiggly block tree J retains the block B and B has children in J.*

To support the basic navigation on the string s, the tree J is equipped as follows. Each non-intermediate block B in J stores $\mathsf{b}(B)$, $\mathsf{e}(B)$, $\mathsf{id}(B)$, pointers to children of B, and the lowest level $\ell(B)$ on which a block with identifier $\mathsf{id}(B)$ was created in H. For each non-leaf block B in J, two cases are possible: (1) B has children $B'_i, \dots, B'_j$ in J (some of which might be intermediate blocks inserted in place of old children of B from H) whose concatenation, if viewed as strings, equals $s[\mathsf{b}(B) \,..\, \mathsf{e}(B)]$ (i.e., no "gaps" between the children); (2) B has only one child B' and all removed children had the same identifiers $\mathsf{id}(B')$, i.e., B was a run block with exactly $|B|/|B'|$ children in H. If a block B in J with $|B| > 1$ has no children, then either B is intermediate or, by Lemma 5, there is a block B' with children in J such that $\mathsf{id}(B') = \mathsf{id}(B)$. In the latter case, we store in B a pointer to B'; observe that if B is from a level ℓ, then all children $B'_i, \dots, B'_j$ of B' are from levels $\leq \ell - 1$ since, by Lemma 4, B and B' originate from blocks created on the same level in the hierarchy H. Hence, when one moves from the level-ℓ block B to B' and, then, to a child of B', the level decreases to at most $\ell - 1$. Thus, if there are no intermediate blocks in J, one can read $s[\mathsf{b}(B) \,..\, \mathsf{e}(B)]$ for any block B in J in $O(|B|)$ time by naively descending in J as in grammar indexes [17].

Consider an intermediate block B in J from a level $2k + 1$. By construction, it has no children in J. Let B' be its parent. Denote by $B_1, \dots, B_b$ all blocks from level $2k + 1$ in the hierarchy H; we mark them according to conditions (a) and (b) from Section 3 so that we can use the notation left and right from now on. Suppose that $B' = B_i \cdots B_j$, where $B_i, \dots, B_j$ is the range of blocks from which B' was produced in H. Since B is intermediate, we have $B = B_m \cdots B_{m+r-1}$, for some m and r such that $[m \,..\, m{+}r) \subseteq [i \,..\, j]$. By construction, there is $m' < m$ with $\mathsf{left}(m', 4r) = \mathsf{left}(m, 4r)$ and $\mathsf{right}(m', 5r) = \mathsf{right}(m, 5r)$. Choose the smallest m'' such that $\mathsf{right}(m'', r) = \mathsf{right}(m, r)$. Obviously, $m'' \leq m'$. Let $B_{i'}, \dots, B_{j'}$ be a range of blocks in H such that $i' \leq m'' \leq j'$, and let $\hat{B}$ be a block on level $2k + 2$ produced from this range. Since $\mathsf{right}(m'', r) = \mathsf{right}(m, r)$, the definition of right implies that the blocks $B_{m''}, \dots, B_{m''+r-2}$ are unmarked and, therefore, $[m'' \,..\, m''{+}r) \subseteq [i' \,..\, j']$. Since m'' is the smallest such index, $\hat{B}$ must be the leftmost block in H from level $2k + 2$ produced from the sequence $\mathsf{id}(B_{i'}), \dots, \mathsf{id}(B_{j'})$, which, by Lemma 4, implies that $\hat{B}$ is the leftmost block with the identifier $\mathsf{id}(\hat{B})$ in the whole hierarchy H. By Lemma 5, the block $\hat{B}$ was retained in J and has children in J. We store in B a pointer to $\hat{B}$ and the following numbers required to identify the location of the copy of $B = B_m \cdots B_{m+r-1}$ among the children of $\hat{B}$: $\mathsf{off}(B) = m'' - i'$ and $\mathsf{r}(B) = r$. (Note that $\hat{B}$ might coincide with B', albeit it is unusual.)

Some subranges of blocks in the range $B_{i'}, \dots, B_{j'}$ could be replaced with intermediate blocks in J. We claim that any such intermediate block $\hat{B}'$ that replaced some of the blocks $B_{m''}, \dots, B_{m''+r-1}$ has $\mathsf{r}(\hat{B}') \leq r/2$. Suppose, to the contrary, that $\hat{B}' = B_{\hat{m}} \cdots B_{\hat{m}+\hat{r}-1}$ is such that $\hat{r} = \mathsf{r}(\hat{B}') > r/2$ and $[\hat{m} \,..\, \hat{m}{+}\hat{r}) \cap [m'' \,..\, m''{+}r) \neq \emptyset$. Since $4\hat{r} > r$, the concatenation of the sequences $\mathsf{left}(\hat{m}, 4\hat{r})$ and $\mathsf{right}(\hat{m}, 5\hat{r})$ contains the sequence $\mathsf{id}(B_{m''}), \dots, \mathsf{id}(B_{m''+r-1})$. Hence, by construction, there is $m''' < m''$ such that $\mathsf{right}(m''', r) = \mathsf{right}(m'', r)$, which contradicts the minimality of m''. This claim implies that, for an intermediate block B in J,

one can compute all r children of B from the original hierarchy of blocks H in $O(r)$ time by the simple traversal of pointers in J. Since, by Lemma 3 (local sparsity), any block B has $O(\sum_{k=1}^{\infty} |B|/2^k) = O(|B|)$ descendants in the hierarchy H, we obtain the following result.

▶ **Lemma 6.** *Given a block B in the jiggly block tree J, one can obtain all r children of B from the original hierarchy of blocks H in $O(r)$ time; further, one can compute all descendant blocks for B in H in $O(|B|)$ time.*

▶ **Theorem 7.** *The jiggly block tree for string s of length n has size $O(\delta \log \frac{n}{\delta})$, where δ is the string complexity of s, provided $\delta \geq \Omega(\log \log n)$.*

Proof (sketch). We are to estimate the number of blocks in J as $O(\delta \log \frac{n}{\delta})$. The basic argument is as in [35]: we will estimate the number of internal blocks on each level ℓ as $O(\delta)$, which implies that the number of internal blocks on $2 \log \frac{n}{\delta}$ lowest levels is $O(\delta \log \frac{n}{\delta})$, as required; then, by Lemma 3 (local sparsity), the number of blocks on levels $\ell \geq 2 \log \frac{n}{\delta}$ is at most $O(\sum_{k=\log(n/\delta)}^{\infty} n/2^k) = O(\delta)$. This scheme will be complicated by the subsequent counting of leaves since in J a block may have up to $O(\log \log n)$ leaf-children.

Fix $\ell = 2k+1$ or $2k+2$, for $k \geq 0$. Every non-leaf level-ℓ block B in J satisfies $|B| \leq 2^k$ and its corresponding substring $s[\mathsf{b}(B)-2^{k+4} \,..\, \mathsf{e}(B)+2^{k+4}]$ cannot occur at smaller positions since otherwise, due to Lemma 2 (local consistency), there is a copy of B to the left of B and, thus, all children of B should be removed and a pointer to this copy should be stored in B. We associate with every such B a group of $\Theta(2^k)$ substrings of length 2^{k+6}, each of which covers $s[\mathsf{b}(B)-2^{k+4} \,..\, \mathsf{e}(B)+2^{k+4}]$ and, hence, cannot occur at smaller positions, i.e., they are "leftmost substrings". Due to Lemma 3 (local sparsity), at most $O(1)$ level-ℓ blocks can be associated with one substring of length 2^{k+6}. Therefore, $\Theta(2^k) \cdot b \leq O(d_{2^{k+6}})$, where b is the number of non-leaf level-ℓ blocks B and $d_{2^{k+6}}$ is the number of distinct substrings of length 2^{k+6} (which is equal to the number of leftmost substrings of length 2^{k+6}). Thus, we obtain $b \leq O(d_{2^{k+6}}/2^{k+6}) \leq O(\delta)$. (The described argument is exactly the same as in [35].)

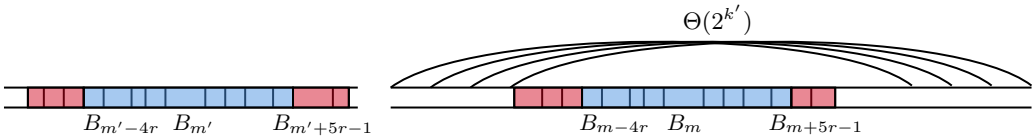

Figure 2 A schematic representation of a group of $\Theta(2^{k'})$ leftmost substrings of length $2^{k'}$ (depicted as arcs above) associated with a leaf block $B = B_m \cdots B_{m+r-1}$. The blue rectangles represent blocks $B_{m-4r}, \ldots, B_{m+5r-1}$ and their copies to the left, $B_{m'-4r}, \ldots, B_{m'+5r-1}$. The red rectangles represent that it is impossible to extend the subrange to $B_{m-4(r+1)}, \ldots, B_{m+5(r+1)-1}$.

If every block in J had $O(1)$ leaves, the total number of blocks would be $O(\text{number of non-leaf blocks})$. Unfortunately, some blocks may have up to $O(\log \log n)$ leaves. We analyze each level-ℓ leaf B that is not the rightmost child of its parent B'. B' was generated from a sequence of level-ℓ blocks: $B' = B_i \cdots B_j$ where a subrange $B_m, \ldots, B_{m+r-1}$ generated the leaf block B (intermediate, if $r > 1$, or not, if $r = 1$). See Fig. 2. The subranges were produced by a greedy parsing that tried to produce the subrange $B_m, \ldots, B_{m+r}$ instead of $B_m, \ldots, B_{m+r-1}$ but failed since there were no occurrences of the sequence $B_{m-4(r+1)}, \ldots, B_{m+5(r+1)-1}$ (the subrange with a "neighbourhood" of $4(r+1)$ blocks to its left and right) to the left in the hierarchy of blocks. Therefore, the substring $s[a \,..\, b] = s[\mathsf{b}(B_{m-4(r+1)})-2^{k+4} \,..\, \mathsf{e}(B_{m+5(r+1)-1})+2^{k+4}]$ cannot occur at smaller positions since, otherwise, by Lemma 2 (local consistency), there would be an earlier occurrence for the sequence. We associate with B a group of $\Theta(2^{k'})$ leftmost substrings of length $2^{k'+1}$ that cover $s[a \,..\, b]$, where $2^{k'}$ is the closest power of two to

the length of $s[a\,..\,b]$. We consider separately each number k' with all such groups of $\Theta(2^{k'})$ leftmost substrings, and the further analysis to bound the number of these groups by $O(\delta)$, for this k', is similar as above, in principle, but it has many subtle details and special cases.

All remaining details of the proof were moved to Appendix B. ◀

5 Substring Search

Let us augment the jiggly block tree J to support searching in s. Our approach is standard but with modifications since we will not use Karp–Rabin fingerprints [27]. Instead, we invent somewhat analogous *deterministic fingerprints*. Throughout, we use hash tables supporting queries in $O(1)$ time [9]. Let us fix a string t of length m, the pattern to search.

Parsing t. We create a compacted trie T_{id} and, for each non-run block B in J that was created from a range of blocks $B_i, \ldots, B_j$ from a level $2k+1$ of the hierarchy H, we store in T_{id} the sequence $\mathsf{id}(B_i), \ldots, \mathsf{id}(B_j)$ (treated as a string with letters $\mathsf{id}(B_i), \ldots, \mathsf{id}(B_j)$). As is usual for compacted tries, the edge labels in T_{id} are sequences of ids and there is a node x such that $\mathsf{str}(x) = \mathsf{id}(B_i), \ldots, \mathsf{id}(B_j)$, where $\mathsf{str}(x)$ denotes the sequence written on the root–x path; we store in x a pointer to B. The edges of T_{id} do not store their edge labels, only the lengths of these edge labels. Each internal node of T_{id} is augmented with a hash table that maps any block identifier to the child whose edge label starts with this identifier (if any). It follows from Theorem 7 that the size of T_{id} is $O(\delta \log \frac{n}{\delta})$.

Our search algorithm constructs a hierarchy of blocks for t analogous to H. Level 0 in the hierarchy for t contains the one-letter blocks $t[0], \ldots, t[m-1]$ whose identifiers are the corresponding letters; on a generic step of the algorithm, for $\ell \geq 0$, we have all level-ℓ blocks $B_1, \ldots, B_b$ and $B_1 \cdots B_b = t$. Suppose that $\ell = 2k$, for some $k \geq 0$. As in Section 3, we identify in $O(b)$ time all maximal runs of blocks $B_i, \ldots, B_{i+r-1}$ with equal identifiers (i.e., $\mathsf{id}(B_i) = \cdots = \mathsf{id}(B_{i+r-1})$) such that $|B_i| \leq 2^k$. If $r = 1$ or $|B_i| > 2^k$, we copy the block B_i to level $2k+1$. If $r > 1$, we substitute such run $B_i, \ldots, B_{i+r-1}$ with a new block B for level $2k+1$ with identifier $\langle \mathsf{id}(B_i), r \rangle$. Thus, all block for level $2k+1$ are constructed. Suppose that $\ell = 2k+1$, for some $k \geq 0$. As in Section 3, we assign in $O(b)$ time to each block B_i an identifier $\mathsf{id}''(B_i)$ that is either ∞ (for example, if $|B_i| > 2^k$) or an integer that is at most $O(\log \log n)$; we then produce new blocks for level $2k+2$ by identifying local minima for id'' and values of id'' equal to ∞; we omit details as they are the same as in Section 3. It remains to assign identifiers to new blocks. Suppose that a new such block B was created from blocks $B_i, \ldots, B_j$ from level $2k+1$. We descend from the root of the trie T_{id} reading the sequence $\mathsf{id}(B_i), \ldots, \mathsf{id}(B_j)$ and skipping sequences written on the edge labels; thus, we reach a node x that stores a pointer to a block B' in J and we retrieve all skipped edge labels in $O(j-i+1)$ time traversing J from B' as described in Lemma 6; we then assign $\mathsf{id}(B) = \mathsf{id}(B')$. If the described search has failed at any stage, we assign to $\mathsf{id}(B)$ a new unique identifier (for consistency, we also should maintain another trie analogous to T_{id} for new identifiers but, as it will be seen, this consistency is not very important).

The algorithm processes all b blocks on one level in $O(b)$ time. It then follows from Lemma 3 (local sparsity) that the overall time is $O(\sum_{k=0}^{\infty} m/2^k) = O(m)$.

Deterministic fingerprints. We view any substring $t[p\,..\,q)$ as the sequence of level-0 blocks $t[p], \ldots, t[q-1]$ and we define its deterministic fingerprint as $\mathsf{fin}(0, t[p], \ldots, t[q-1])$, where $\mathsf{fin}(\ell, B_1, \ldots, B_b)$ is the following recursive procedure that assigns a fingerprint to any contiguous subsequence of level-ℓ blocks $B_1, \ldots, B_b$ from the hierarchy of blocks of t.

1. The fingerprint $\mathsf{fin}(\ell, B_1, \dots, B_b)$ of the empty sequence ($b = 0$) is the empty sequence.
2. Case $\ell = 2k$, for $k \geq 0$. Choose maximal $i \geq 0$ such that $|B_i| \leq 2^k$ and $\mathsf{id}(B_i) = \mathsf{id}(B_h)$, for all $h \in [1\,..\,i]$, or $i = b$. Choose minimal $j \leq b$ such that $|B_{j+1}| \leq 2^k$ and $\mathsf{id}(B_{j+1}) = \mathsf{id}(B_h)$, for all $h \in (j\,..\,b]$, or $j = 0$. If $i = b$, the fingerprint is $\langle \mathsf{id}(B_1), b\rangle$. If $i \neq b$, the fingerprint is the sequence $\langle \mathsf{id}(B_i), i\rangle, \mathsf{fin}(2k+1, B'_1, \dots, B'_{b'}), \langle \mathsf{id}(B_{j+1}), b-j\rangle$, where the element $\langle \mathsf{id}(B_i), i\rangle$ is missing if $i = 0$, $\langle \mathsf{id}(B_{j+1}), b-j\rangle$ is missing if $j = b$, and $B'_1, \dots, B'_{b'}$ are all level-$(2k{+}1)$ blocks that are parents of the blocks $B_{i+1}, \dots, B_j$.
3. Case $\ell = 2k+1$, for $k \geq 0$. Redefine the identifiers id'' for $B_1, \dots, B_b$ as in Section 3. Choose the smallest i such that either $|B_{i+1}| > 2^k$ (where $0 \leq i < b$) or $\infty > \mathsf{id}''(B_{i-2}) > \mathsf{id}''(B_{i-1})$ and $\mathsf{id}''(B_{i-1}) < \mathsf{id}''(B_i) < \infty$ (where $3 \leq i \leq b$); let $i = b$ if there is no such i. Symmetrically, choose the largest j such that either $|B_j| > 2^k$ or $\infty > \mathsf{id}''(B_{j-2}) > \mathsf{id}''(B_{j-1})$ and $\mathsf{id}''(B_{j-1}) < \mathsf{id}''(B_j) < \infty$ (where $3 \leq j \leq b$); let $j = b$ if there is no such j. Note that $i \leq j$. The fingerprint is $\mathsf{id}(B_1), \dots, \mathsf{id}(B_i), \mathsf{fin}(2k+2, B'_1, \dots, B'_{b'}), \mathsf{id}(B_{j+1}), \dots, \mathsf{id}(B_b)$, where $B'_1, \dots, B'_{b'}$ are all level-$(2k{+}2)$ blocks that are parents of the blocks $B_{i+1}, \dots, B_j$.

The indices i and j in the procedure fin are chosen to be at block boundaries for the blocks $B'_1, \dots, B'_{b'}$ from level $\ell + 1$ so that all children of $B'_1, \dots, B'_{b'}$ are exactly the blocks $B_{i+1}, \dots, B_j$. Therefore, the concatenation of all blocks used in the fingerprint of $t[p\,..\,q)$ is equal to $t[p\,..\,q)$ (Fig. 3). Due to Lemma 3 (local sparsity), fin has at most $O(\log m)$ levels of recursion. Since id'' is at most $O(\log\log n)$ whenever it is not ∞, it follows from the discussion of Section 3 that case 3 in the recursion (when $\ell = 2k{+}1$) uses at most $O(\log\log n)$ blocks for the resulting fingerprint. Hence, the size of the fingerprint is $O(\log m \log\log n)$.

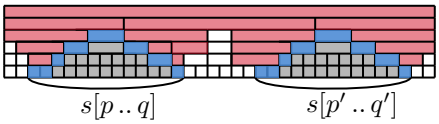

Figure 3 Here $s[p\,..\,q] = s[p'\,..\,q']$. The blocks depicted as blue form the fingerprints of these strings; the red blocks intersect the strings but can be "inconsistent" and are not in the fingerprints.

We store in each block B in the hierarchy built for t pointers to children/parent of B and to the first preceding block on the same level with different $\mathsf{id}(B)$. With this, one can retrieve the fingerprint for any substring of t in time $O(\log m \log\log n)$.

Analogously, we define the deterministic fingerprint for any substring of s.

▶ **Lemma 8.** *Two substrings of s or t coincide iff their deterministic fingerprints coincide.*

Proof. By construction, any two blocks in both hierarchies (for s and for t) that have equal identifiers are equal as strings. Therefore, two substrings with equal fingerprints must be equal. For converse, suppose that two substrings $s[p'\,..\,q')$ and $t[p\,..\,q)$ are equal. It suffices to show that the procedure fin computing the fingerprints for $s[p'\,..\,q')$ and $t[p\,..\,q)$ does the same calculations. Suppose that, on a generic step, a call to $\mathsf{fin}(\ell, B_1, \dots, B_b)$ occurs during the computation for $s[p'\,..\,q')$ and the same call $\mathsf{fin}(\ell, B_1, \dots, B_b)$ for $t[p\,..\,q)$; initially, for $\ell = 0$, $\mathsf{fin}(0, s[p'], \dots, s[q'{-}1])$ and $\mathsf{fin}(0, t[p], \dots, t[q{-}1])$ are called on the same sequences of level-0 blocks. Cases 2 or 3 in the description of fin obviously produce the same sequences of identifiers surrounding a recursive call to fin; the recursive call receives as its arguments a certain new sequence of blocks $B'_1, \dots, B'_{b'}$. In case 2 this sequence $B'_1, \dots, B'_{b'}$ is the same in both calls since the blocks $B'_1, \dots, B'_{b'}$ either are copies of blocks from $B_1, \dots, B_b$ or are runs of blocks to which we assigned the same identifiers of the form $\langle \mathsf{id}(B), r\rangle$ in both hierarchies of blocks. In case 3 the argument is more complicated but still straightforward since the

blocks $B'_1, \ldots, B'_{b'}$ were produced from local minima of the function id'' and we strategically skipped some blocks from the begin and end of the sequence $B_1, \ldots, B_b$ to make the local minima consistent, and, further, we rely on the fact that blocks created from the same sequences of blocks in both hierarchies are assigned with the same identifiers (recall that the trie T_{id} is used to guarantee this). The values of id'' for $B_1, \ldots, B_b$ during the computation of fin are exactly the same as they were in the definition of both hierarchies of blocks (for s and for t) except, possibly, for B_1 and B_2, as $\mathsf{id}''(B_1) = \mathsf{id}''(B_2) = \infty$ when id'' was computed in fin. But a simple case analysis shows that this nuance does not hurt the argument. ◀

By Lemma 6, the hierarchy of blocks H for s can be emulated using J, which suffices to get the fingerprint of any substring of s. But this approach is inefficient as it assembles the fingerprints top down, not bottom up as in the definition. We speed up the assembling by adding the following data structures. Let B be a block of J (intermediate or not) with $|B| > 1$. The block B must have been produced from a run/range/subrange of blocks $B_i, \ldots, B_j$ with $i < j$. We store in B a link to a block in J with identifier $\mathsf{id}(B_i)$, which exists in J by Lemma 5. The links naturally form a forest A_L on all blocks. Define a symmetric forest A_R for rightmost blocks (i.e., where the link from B refers to $\mathsf{id}(B_j)$). We equip A_L and A_R with the weighted ancestor structure [25, 36] that, for any B and any threshold w, can find the farthest ancestor $\hat{B}$ of B in A_L (or A_R) with $|\hat{B}| \geq w$ in $O(\log\log n)$ time. Using A_L and A_R, we obtain the following result; its tedious proof is very technical and it can be found in the full version of the paper [38, Appendix C].

▶ **Lemma 9** ($\mathsf{fingerprints}$). *Given a substring $s[p\,..\,q]$ and the lowest block B in the hierarchy of blocks H for s with $\mathsf{b}(B) \leq p < q \leq \mathsf{e}(B)$, if J retains B and B has children in J, then one can compute the fingerprint of $s[p\,..\,q]$ in $O(\log m \cdot (\log\log n)^2)$ time, where $m = q - p$.*

Indexing structures. We use a well-known scheme for pattern matching. The following description is sketchy in parts as it is mostly the same as in [17, 18]. Given $s[p\,..\,p+|t|) = t$, let B be the lowest block in J such that $\mathsf{b}(B) \leq p < p+|t|-1 \leq \mathsf{e}(B)$. If B has children in J, we call the occurrence *primary*; otherwise, *secondary*. We first find primary occurrences.

Let $\overleftarrow{T}$ and T be two compacted tries. Consider each block B in J. Let B be a non-run block with children $B_i, \ldots, B_j$ in J. For each $h \in [i\,..\,j)$, we store the string B_{h+1} in T and T has an explicit node x such that $\mathsf{str}(x) = B_{h+1}$; we store $\overleftarrow{B_i \cdots B_h}$ in $\overleftarrow{T}$ with an explicit node y such that $\mathsf{str}(y) = \overleftarrow{B_i \cdots B_h}$. Thus, we produce pairs of nodes (x, y) associated with B and indices h. Let B be a run block whose only child in J is B_1 and $r = |B|/|B_1|$. We store the string B_1 in T and $\overleftarrow{B_1}^{r-1}$ in $\overleftarrow{T}$, thus producing a pair of nodes (x, y) associated with B. The edges in T and $\overleftarrow{T}$ store only lengths of edge labels. By Theorem 7, the size of $\overleftarrow{T}$ and T and the number of produced pairs (x, y) is $O(\delta \log \frac{n}{\delta})$. Each node z in T stores a pointer to a block B in J such that $\mathsf{str}(z)$ is a prefix of B; each node z in $\overleftarrow{T}$ stores an index h and a pointer to B in J whose children are $B_i, \ldots, B_j$ such that $\mathsf{str}(z)$ is a prefix of $\overleftarrow{B_i \cdots B_h}$.

Define an order on nodes of T and $\overleftarrow{T}$: $x < x'$ iff $\mathsf{str}(x) < \mathsf{str}(x')$. Fix a constant $\epsilon > 0$. We store a range reporting structure R [15]: it contains all produced pairs of nodes (x, y) in $O(\delta \log \frac{n}{\delta})$ space (the big-O hides factor $\frac{1}{\epsilon}$) and, given any nodes a, b of T and c, d of $\overleftarrow{T}$, we can report all N pairs (x, y) such that $a \leq x \leq b$ and $c \leq y \leq d$ in $O((1+N)\log^{\epsilon} n)$ time.

We turn T into a z-fast trie by augmenting it with a hash table that, for each node x, maps the fingerprint of a certain prefix of $\mathsf{str}(x)$ to x. We use our deterministic fingerprints. Since each fingerprint takes $\omega(1)$ space, the hash table is organized as a compacted trie T_f

containing the fingerprints, treated as sequences of identifiers of length $O(\log m \log\log n)$; the edges of T_f do not store their edge labels, only lengths (as in the trie T_{id}). Let $t[q\,..\,m)$ be a string that can be read by descending from the root of T, assuming that the edge labels are somehow accessible by an "oracle". The z-fast trie can read it in $O(\log^2 m \log\log n)$ time by querying the hash on $O(\log m)$ fingerprints of prefixes of $t[q\,..\,m)$. If $t[q\,..\,m)$ cannot be read from the root of T, the z-fast trie returns an arbitrary node x of T. The found node x contains a pointer to a block B such that $\mathsf{str}(x)$ is a prefix of B. Using Lemma 9 (fingerprints) and relying on Lemma 8, we check in $O(\log m \cdot (\log\log n)^2)$ time whether the prefix of length $m-q$ of the string B is equal to $t[q\,..\,m)$ and, thus, we verify whether we indeed performed the descending correctly. We equip $\overleftarrow{T}$ with the same z-fast trie structure.

Searching for t. Let $s[p\,..\,p{+}|t|) = t$ be a primary occurrence. Let B be the lowest block in J such that $\mathsf{b}(B) \le p < p + |t| - 1 \le \mathsf{e}(B)$ and let $B_i, \ldots, B_j$ be all children of B in J. Choose the largest $h \in [i\,..\,j)$ such that $\mathsf{e}(B_h) < p + |t| - 1$. If B is not a run block, there is $q \in [0\,..\,m)$ such that $\mathsf{b}(B_{h+1}) = p + q$ and T contains the string $t[q\,..\,m)$ and $\overleftarrow{T}$ contains the string $\overleftarrow{t[0\,..\,q)}$; the range reporting structure R contains a pair of nodes (x, y) associated with B and index h. If B is a run block, there is $q \in [0\,..\,m)$ such that $m - q \le |B_1|$, $\mathsf{b}(B) + c|B_1| = p + q$, for some $c \ge 1$, and T contains $t[q\,..\,m)$ and $\overleftarrow{T}$ contains $\overleftarrow{t[0\,..\,q)}$; there is again an associated pair of nodes (x, y). Given the position q, using the fingerprints of prefixes of $\overleftarrow{t[0\,..\,q)}$ and $t[q\,..\,m)$ in the z-fast tries, one can find in $O(\log^2 m \cdot (\log\log n)^2)$ time the nodes a, b in T and c, d in $\overleftarrow{T}$ on which the range reporting structure R will report all pairs (x, y) that fit the above description and each such reported pair will be associated with a separate primary occurrence (note that a run block B will contain a group of primary occurrences at distances that are multiples of $|B_1|$ but the algorithm finds only the rightmost occurrence from each group; other group occurrences are easy to restore using periodicity).

To identify candidate positions $q \in [0\,..\,m)$ for which we perform the described search, we compute the fingerprint f of t, which is a sequence of $O(\log m \log\log n)$ block identifiers. Let $p_1, \ldots, p_b$ be the starting positions of these blocks from f such that $0 = p_1 < \cdots < p_b < p_{b+1} = m$, i.e., for $i \in [1\,..\,b]$, the ith block from f corresponds to $t[p_i\,..\,p_{i+1})$. By Lemma 8, each primary occurrence $s[p\,..\,p{+}|t|)$ of t has the same fingerprint and its respective blocks correspond to the substrings $s[p{+}p_i\,..\,p{+}p_{i+1})$, for $i \in [1\,..\,b]$. Let B be the lowest block in J such that $\mathsf{b}(B) \le p < p + |t| - 1 \le \mathsf{e}(B)$ and let $B_i, \ldots, B_j$ be all children of B in the hierarchy of blocks H for s (not in J!). Any substring $s[p + p_i\,..\,p + p_{i+1})$, for $i \in [1\,..\,b]$, that corresponds to a non-run block in the fingerprint must form a block in H, which must be a descendant of B in H. Any substring $s[p + p_i\,..\,p + p_{i+1})$ that corresponds to a run block with an identifier $\langle \mathsf{id}(B'_i), r\rangle$, for some block B'_i and $r > 1$, must form a sequence of blocks $s[p + p_i + c|B'_i|\,..\,p + p_i + (c+1)|B'_i|)$ in H with $c \in [0\,..\,(p_{i+1} - p_i)/|B'_i|)$, all of which must be descendants of B in H. These observations imply that any block boundary between $B_i, \ldots, B_j$ that is inside the range $[p\,..\,p{+}|t|)$ is equal either to $p + p_i$, for some $i \in [1\,..\,b]$, or to $p + p_{i+1} - |B'_i|$, where $t[p_i\,..\,p_{i+1})$ corresponds to a run block with an identifier $\langle B'_i, r\rangle$. We obtain $O(\log m \log\log n)$ candidate positions for q: all these p_i and $p_{i+1} - |B'_i|$.

Secondary occurrences. To find secondary occurrences, we reverse all pointers in the tree J and traverse the reversed pointers starting from each occurrence (including newly added secondary occurrences). More precisely, any non-intermediate leaf block B in J with $|B| > 1$ contains a pointer to another block $\hat{B}$ in J such that $\mathsf{id}(\hat{B}) = \mathsf{id}(B)$; any intermediate block B in J contains two numbers $h = \mathsf{off}(B) \ge 0$ and $r = \mathsf{r}(B) > 1$ and a pointer to a block $\hat{B}$ from a level $2k + 2$ such that $\mathsf{id}(B)$ was created from identifiers $\mathsf{id}(B_{i+h}), \ldots, \mathsf{id}(B_{i+h+r-1})$,

where $B_i, \ldots, B_j$ is the sequence of level-$(2k+1)$ blocks from which $\hat{B}$ was created and $i+h+r-1 \leq j$. The block $\hat{B}$ has children in J in both cases. Given a leaf B, secondary occurrences of t corresponding to B are exactly all occurrences of t from the substring $s[\mathsf{b}(B) \,..\, \mathsf{e}(B)]$. They may exist if the block $\hat{B}$ referred by B contains occurrences of t.

We construct on the tree J the marked ancestor data structure [2] that allows us to mark some nodes and to search the nearest marked ancestor of any node in $O(1)$ time. Then, we reverse all pointers: for each block $\hat{B}$, we store in $\hat{B}$ a link to each non-intermediate block B that has a pointer to $\hat{B}$ and we mark $\hat{B}$ in this case; further, if $\hat{B}$ is from a level $2k+2$ and $\hat{B}$ was created from level-$(2k+1)$ blocks $B_i, \ldots, B_j$, then we store in $\hat{B}$ a link to each intermediate block B that has a pointer to $\hat{B}$ and we store the range $[i+h \,..\, i+h+r)$ corresponding to B in an augmented binary search tree $R_{\hat{B}}$ associated with $\hat{B}$; for any $d \in [i \,..\, j]$, we mark the block B_d if B_d is a non-intermediate child of $\hat{B}$ in J and $d \in [i+h \,..\, i+h+r)$ for a range $[i+h \,..\, i+h+r)$ from $R_{\hat{B}}$. Note that the tree $R_{\hat{B}}$ contains at most $O((\log\log n)^2)$ distinct ranges but one range may correspond to many blocks B; thus, range searching operations on $R_{\hat{B}}$ can be performed in time $O(\log\log\log n) \leq O(\log^\epsilon n)$.

With this machinery, the search for secondary occurrences is just a traversal of the reversed pointers. We maintain a set of occurrences of t, which initially contains only primary occurrences, and each occurrence $s[p \,..\, p+|t|)$ is associated with a lowest block $\hat{B}$ in J such that $\mathsf{b}(\hat{B}) \leq p < p+|t|-1 \leq \mathsf{e}(\hat{B})$. Given such an occurrence $s[p \,..\, p+|t|)$ and the associated block $\hat{B}$, we use the tree $R_{\hat{B}}$ to check whether $s[p \,..\, p+|t|)$ lies inside some blocks $B_{i+h} \cdots B_{i+h+r-1}$ where $[i+h \,..\, i+h+r)$ is a range stored in $R_{\hat{B}}$ and $B_i, \ldots, B_j$ are blocks from which $\hat{B}$ was created in the hierarchy of blocks H for s. Each link associated with such range induces a new unique secondary occurrence of t that is added to the set of occurrences. We then loop through all links to all non-intermediate blocks B that have pointers to $\hat{B}$ and each such link analogously induces a new secondary occurrence. Then, we find the nearest marked ancestor B' of B (if any) and continue the same process recursively with B' in place of B, inducing new unique secondary occurrences. Each added secondary occurrence in the set is processed in the same way. The time for reporting all occ occurrences is $O(\mathrm{occ} \cdot \log^\epsilon n)$.

Time. We obtain $u = O(\log m \log\log n)$ candidate positions q at which we perform z-fast trie searches and range reporting: $O(\log^2 m \cdot (\log\log n)^2 + (1+\mathrm{occ}_q)\log^\epsilon n)$ time per q, where occ_q is the number of primary occurrences found for one q. We have $\mathrm{occ} \geq \sum_q \mathrm{occ}_q$. Thus, the total time is $O(m + u(\log^2 m \cdot (\log\log n)^2 + \log^\epsilon n) + \mathrm{occ} \cdot \log^\epsilon n) = O(m + \log^3 m \cdot (\log\log n)^3 + \log m \cdot \log\log n \cdot \log^\epsilon n + \mathrm{occ} \cdot \log^\epsilon n)$. Fix a constant $\epsilon' > \epsilon$. When $m \leq \log^6 n$, we estimate $\log m = O(\log\log n)$ and, thus, the sum $\log^3 m \cdot (\log\log n)^3 + \log m \cdot \log\log n \cdot \log^\epsilon n$ is bounded by $O(\log^{\epsilon'} n)$ and, after renaming ϵ' to ϵ, we obtain time $O(m + (1+\mathrm{occ})\log^\epsilon n)$. When $m > \log^6 n$, we bound this sum by $O(m)$ and obtain time $O(m + \mathrm{occ} \cdot \log^\epsilon n)$.

6 Construction

We build the following components: the compacted tries $\boldsymbol{T_{\mathsf{id}}}$, $\overleftarrow{\boldsymbol{T}}$, $\boldsymbol{T}$, $\boldsymbol{T_f}$, the range reporting structure $\boldsymbol{R}$ and all its pairs $\boldsymbol{(x, y)}$, the trees $\boldsymbol{A_L}$ and $\boldsymbol{A_R}$ with a weighted ancestor structure, and, for each block B, pointers to children (if any), the pointer to a copy of B (if B is a leaf), and $\mathsf{b}(\boldsymbol{B})$, $\mathsf{e}(\boldsymbol{B})$, $\boldsymbol{\ell(B)}$, $\mathsf{id}(\boldsymbol{B})$ (if any). In the end, we also construct in an obvious way all back links for secondary occurrences with a marked ancestor structure [2].

Our algorithm reads s from left to right and maintains for each level of the currently constructed hierarchy H a queue with at most $O(\log\log n)$ last built blocks from this level. The letters $s[0], s[1], \ldots$ are consecutively fed to the level 0. Each level ℓ receives new blocks

from the previous level $\ell-1$ or from s, in case of level 0, and puts them to the right of its queue (the queue elements are ordered from left to right and we dequeue elements from left); when a block is received, we may remove some blocks from the queue and may generate new blocks to feed for level $\ell+1$. If the level $\ell+1$ does not exist, we create it. After reading the whole s, we feed into level 0 a special letter \$ that forces the algorithm to complete all levels.

At any level ℓ, we assume that all level-ℓ blocks of the hierarchy H to the left of the leftmost block in the queue were already built together with their descendants and structures like A_L, A_R, T_{id}. In particular, we can use Lemma 9 (fingerprints) on these blocks. We maintain a global hash table L that maps any id to the leftmost block in J with this id (as in Lemma 5). By Lemma 5, such leftmost blocks have children in J and are never removed. Thus, once a block with new id is created and the corresponding information about it and its children is added to L, T_{id}, $\overleftarrow{T}$, T, T_f, A_L, A_R, this information will never be removed.

Time. We use hash tables for L, edge transitions in T_{id}, T, $\overleftarrow{T}$, T_f, for van Emde Boas structures from the weighted ancestor structure [25, 36] of A_L and A_R, and for other places. The query time in the tables is $O(1)$ and the insertion time is amortized $O(1)$ w.h.p. [9] It is the only component that uses randomization. From now on, we do not write explicitly that the time for modifying the tables is "amortized expected w.h.p.", assuming it by default.

For level $\ell=2k$ or $\ell=2k+1$, we will spend $O(\log^3|B|\cdot\log n)$ time on queue operations for each level-ℓ block B with $|B|\le 2^k$, $O(\log^3|B|\cdot\log n)$ time on each new block B that we will generate for level $\ell+1$, and $O(1)$ time on each level-ℓ block B with $|B|>2^k$ by simply feeding such B to level $\ell+1$. By Lemma 3, estimating $\log^3|B|=O(k^3)$ when $|B|\le 2^k$, the total time for all levels can be bounded by $O(\sum_{k=0}^{\infty}\frac{n}{2^k}k^3\log n)=O(n\log n\cdot\sum_{k=0}^{\infty}\frac{k^3}{2^k})=O(n\log n)$.

Level $\ell=2k$, for $k\ge 0$. We maintain at most one block B_1 in the queue and a number r such that the last r fed blocks had identifiers $\mathsf{id}(B_1)$. Suppose that a new block B_2 is fed to us. If $\mathsf{id}(B_2)=\mathsf{id}(B_1)$, increment r, done. Let $\mathsf{id}(B_2)\ne\mathsf{id}(B_1)$. We remove B_1 from the queue and process it as described below (if the queue is not empty), after which we finalize as follows: put B_2 in the queue seting $r:=1$, if $|B_2|\le 2^k$, or feed B_2 to level $\ell+1$, if $|B_2|>2^k$; done. Now let us describe how B_1 and r are processed before the finalization.

If $r=1$, feed B_1 to level $\ell+1$, finalize. If $r>1$, create a new block B with $\mathsf{id}(B)=\langle\mathsf{id}(B_1),r\rangle$, $\mathsf{b}(B)=\mathsf{b}(B_1)$, $\mathsf{e}(B)=\mathsf{b}(B)+r|B_1|-1$, $\ell(B)=\ell$ (which is correct by Lemma 4) and insert B in A_L and A_R in $O(\log\log n)$ time [36] with a link to the leftmost block with identifier $\mathsf{id}(B_1)$ found using L. If L finds a block B' with $\mathsf{id}(B')=\mathsf{id}(B)$, store in B a pointer to B', feed B to level $\ell+1$, finalize. If L finds no such B', we proceed as follows.

We insert B into L and set B_1 as the only child of B. We can search any prefix of B_1 in the z-fast trie T in $O(\log^2|B_1|\cdot(\log\log n)^2)$ time as in the search phase of Section 5, using Lemma 9 (fingerprints) for prefixes of B_1; thus, by the binary search on prefixes of B_1, we find the location in T at which the string B_1 should be inserted in $O(\log^3|B_1|(\log\log n)^2)$ time. We insert B_1 by creating a node x in T such that $\mathsf{str}(x)=B_1$ (if it did not exist) and, then, we modify the z-fast trie structure by inserting $O(1)$ new fingerprints into T_f: to insert a fingerprint in T_f, we descend from the root reading it in T_f, skipping edge labels, and then we restore the skipped labels from a reference to an appropriated substring whose fingerprint was stored in T_f, which takes $O(\log|B_1|(\log\log n)^2)$ time. Similarly, we insert the string $\overleftarrow{B_1}^{r-1}$ into $\overleftarrow{T}$, creating a node y, in $O(\log^3|B|\cdot(\log\log n)^2)$ time. Finally, we feed B to level $\ell+1$ and finalize. We add the pair (x,y) (together with a link to the block B) to the set of pairs for the range reporting structure R, which is built in the very end of the construction in $O(N\log N)$ time, for all N pairs [15]. Since $\delta\log\frac{n}{\delta}=O(n)$, this time is $O(n\log n)$.

Level $\ell = 2k+1$, for $k \geq 0$. Let $B_1, B_2, \ldots$ be all level-ℓ blocks from left to right, marked according to conditions (a)–(b) of Section 3: B_h is marked iff $|B_h| > 2^k$ or $|B_{h+1}| > 2^k$ or $\mathsf{id}''(B_{h-1})$ is a local minimum. Suppose that a new block B_j is fed to us. We compute $\mathsf{id}''(B_j)$ using the numbers $\mathsf{id}(B_{j-1})$, $\mathsf{id}'(B_{j-1})$. The queue contains the last contiguous sequence of unmarked blocks $B_i, \ldots, B_{j-1}$ fed to level ℓ (i.e., B_{i-1} is marked or $i = 1$). We determine whether B_{j-1} is marked after B_j is received. As in Section 3, we have $j - i = O(\log \log n)$.

If $|B_j| \leq 2^k$ and both B_{j-1} and B_j are not marked, put B_j in the queue, done. If $|B_j| \leq 2^k$ and B_j is marked (B_{j-1} cannot be marked in this case), we process $B_i, \ldots, B_j$ and unite them into a new block that is then fed to level $\ell + 1$; we then take $B_i, \ldots, B_j$ off the queue, done. If $|B_j| > 2^k$, we mark B_{j-1} and analogously process and unite $B_i, \ldots, B_{j-1}$ instead of $B_i, \ldots, B_j$; then, we move B_j to level $\ell + 1$, done (the queue is empty in the end). Let us describe how $B_i, \ldots, B_j$ are processed and united (for $B_i, \ldots, B_{j-1}$, it is analogous).

We read the sequence $\mathsf{id}(B_i), \ldots, \mathsf{id}(B_j)$ in the trie T_{id}: by descending and skipping edge labels, we reach a node z and restore the skipped labels using a block B' referred by z as described in Lemma 6. Then, if we have successfully verified that $\mathsf{str}(z)$ equals the sequence, we remove $B_i, \ldots, B_j$ and create a new childless block B with $\ell(B) = \ell$, $\mathsf{b}(B) = \mathsf{b}(B_i)$, $\mathsf{e}(B) = \mathsf{e}(B_j)$ and we store in B a pointer to B'. We insert B in A_L and A_R with links to leftmost blocks with identifiers $\mathsf{id}(B_i)$ and $\mathsf{id}(B_j)$. We then move B to level $\ell + 1$, done.

Suppose that we could not find the above block B' in T_{id}. We create a new block B with a new identifier $\mathsf{id}(B)$ and we insert the sequence $\mathsf{id}(B_i), \ldots, \mathsf{id}(B_j)$ into T_{id} in $O(\log \log n)$ time, creating a node z that refers to B (the algorithm is almost the same as reading this sequence in T_{id}). We insert B in A_L and A_R with links to leftmost blocks with identifiers $\mathsf{id}(B_i)$ and $\mathsf{id}(B_j)$. Then, we must greedily parse $B_i, \ldots, B_j$ into subranges: if $B_i, \ldots, B_{m-1}$ have already been parsed (for $m \geq i$), then $B_m, \ldots, B_{m+r-1}$ is the longest subrange with $m + r - 1 \leq j$ for which there is $m' < m$ such that $\mathsf{left}(m', 4r) = \mathsf{left}(m, 4r)$ and $\mathsf{right}(m', 5r) = \mathsf{right}(m, 5r)$; let $r = 1$ if there is no such $r > 0$. (We use the notation left and right here as in Section 3.) For each subrange $B_m, \ldots, B_{m+r-1}$ with $r > 1$, we form a new intermediate block $\tilde{B}$ in its place, removing the blocks $B_m, \ldots, B_{m+r-1}$, and we identify the smallest m'' such that $\mathsf{right}(m'', r) = \mathsf{right}(m, r)$; we store in $\tilde{B}$ certain numbers $\mathsf{off}(\tilde{B})$, $\mathsf{r}(\tilde{B}) = r$, and a pointer to the parent of $B_{m''}$ required to find the blocks $B_{m''}, \ldots, B_{m''+r-1}$ (see details in Section 4). After creating all intermediate blocks, the block B has children $\tilde{B}_{\tilde{\imath}}, \ldots, \tilde{B}_{\tilde{\jmath}}$ (intermediate or not) and we do almost the same computations as on level $2k$: for each $h \in [\tilde{\imath} \,..\, \tilde{\jmath})$, we store the string $\overleftarrow{\tilde{B}_{\tilde{\imath}} \ldots \tilde{B}_h}$ in $\overleftarrow{T}$, producing a node x, and $\tilde{B}_{h+1}$ in T, producing a node y, which also requires a modification of T_f; we add the pair (x, y) to R. All this, except the computation of subranges, takes $O(\log^3 |B| \cdot (\log \log n)^3)$ overall time. We then move B to level $\ell + 1$, done. It remains to describe how $B_i, \ldots, B_j$ are parsed into the subranges, which is subtle.

Parsing $B_i, \ldots, B_j$. We maintain two compacted tries $T_\circ$ and $\overleftarrow{T}_\circ$ defined as follows. Let B' be a block of J from a level $2k' + 2$ and $B'_{i'}, \ldots, B'_{j'}$ be the level-$(2k' + 1)$ blocks from which $B' = B'_{i'} \cdots B'_{j'}$ was created. As $B_i, \ldots, B_j$, the sequence $B'_{i'}, \ldots, B'_{j'}$ was also greedily parsed into subranges: for each such subrange $B'_{m'}, \ldots, B_{m'+r'-1}$ with $m' \in [i' \,..\, j']$, we store the sequence $\mathsf{id}(B'_{m'}), \mathsf{id}(B'_{m'+1}), \ldots, \mathsf{id}(B'_{m'+r'-1})$ in the trie $T_\circ$ and $T_\circ$ contains an explicit node x such that $\mathsf{str}(x)$ equals this sequence, and we store $\mathsf{id}(B'_{m'-1}), \mathsf{id}(B'_{m'-2}), \ldots, \mathsf{id}(B'_{i'}), \$$ in $\overleftarrow{T}_\circ$ with an explicit node y in $\overleftarrow{T}_\circ$ with $\mathsf{str}(y)$ equal to this sequence. All such pairs (x, y) are stored in a set P. The nodes of $T_\circ$ and $\overleftarrow{T}_\circ$ store pointers to (leftmost) blocks of J from which their edge labels can be restored. Each node $x \in \overleftarrow{T}_\circ$ (in $T_\circ$, analogously) stores an $O(\log n)$-bit number $\mathsf{v}(x)$ such that, for any $x, y \in \overleftarrow{T}_\circ$, we have $\mathsf{v}(x) < \mathsf{v}(y)$ iff $\mathsf{str}(x) < \mathsf{str}(y)$

(lexicographically); the numbers can be maintained with $O(1)$ amortized insertion time [8, 21] (such structure is called dynamic ordered linked list). Define $x < y$ iff $\mathsf{v}(x) < \mathsf{v}(y)$. We store in each node x the smallest and largest nodes, $\mathsf{L}(x)$ and $\mathsf{R}(x)$, in the subtree rooted at x.

We equip P with a dynamic range reporting structure [13, 45]: it uses $O(\log n)$ time for insertions and $O(\log n + t\frac{\log n}{\log\log n})$ for queries, where t is the number of reported points. This structure on P needs to know only the relative order of coordinates of its points (x, y), not the values $\mathsf{v}(x)$, $\mathsf{v}(y)$. By Theorem 7, the size of $T_\circ$, $\overleftarrow{T}_\circ$, P is $O(\delta \log \frac{n}{\delta})$ (the total number of pairs (x, y)). Once we have parsed $B_i, \ldots, B_j$ into subranges, the updates for $T_\circ$, $\overleftarrow{T}_\circ$, P are straightforward (similar to updates for T_{id} and R above) and take $O((j - i + 1)\log n)$ time.

Suppose that we have already parsed $B_i, \ldots, B_{m-1}$ into subranges. Let us find the largest r for which there is $m' < m$ with $\mathsf{left}(m', 4r) = \mathsf{left}(m, 4r)$ and $\mathsf{right}(m', 5r) = \mathsf{right}(m, 5r)$. Initially, set $r = 1$. We consecutively process the positions $h = i, \ldots, j$ as follows. We read the sequence $\mathsf{id}(B_h), \ldots, \mathsf{id}(B_{\min\{j, m+5(r+1)-1\}})$ by descending from the root of $T_\circ$, thus reaching a node z; set $z = \bot$ if we fail. We read the sequence $\mathsf{id}(B_{h-1}), \ldots, \mathsf{id}(B_{\max\{i-1, m-4(r+1)\}})$ from $\overleftarrow{T}_\circ$, thus reaching a node v, where we abuse notation by assuming that $\mathsf{id}(B_{i-1})$ equals the special symbol \$; set $v = \bot$ if we fail. If either $h \le m - 4(r + 1)$ and $z \ne \bot$, or $h \ge m + 5(r + 1)$ and $v \ne \bot$, then we increment r by 1 and repeat the processing of h again. Otherwise, we continue as follows. If $z = \bot$ or $v = \bot$, we proceed to the next h; otherwise, we perform the range query in P on $[\mathsf{L}(z) \mathinner{..} \mathsf{R}(z)] \times [\mathsf{L}(v) \mathinner{..} \mathsf{R}(v)]$: if it reports some points, we increment r by 1 and repeat the processing of h again, otherwise, we proceed to the next h.

The reading in $T_\circ/\overleftarrow{T}_\circ$ takes $O(\log\log n)$ time. The queries in P are performed only when $h \in [m-4(r+1) \mathinner{..} m+5(r+1))$. Hence, r was computed in time $O((j-i)\log\log n + r\log n) = O((\log\log n)^2 + r\log n) = O(r\log n)$. The algorithm is correct by the same reasons why our substring search was correct. Finally, we try to extend r by a naive pattern matching of the sequence $\mathsf{left}(m, 4(r+1)), \mathsf{right}(m, 5(r+1))$ in $\mathsf{id}(B_i), \ldots, \mathsf{id}(B_j)$ in $O(r\log\log n)$ time. Thus, r is found and it remains to compute the smallest m'' such that $\mathsf{right}(m'', r) = \mathsf{right}(m, r)$.

For each node $z \in T_\circ$, denote $P_z = P \cap [\mathsf{L}(z) \mathinner{..} \mathsf{R}(z)] \times [-\infty \mathinner{..} \infty]$ and $\mathsf{N}(z) = |P_z|$. See Fig. 4 in Appendix B. For each z with $\mathsf{N}(z) > \log\log n$, we order P_z according to the y-coordinates of its points and we split P_z into disjoint chunks $\{P_z^a\}_a$, each chunk P_z^a of size $\Theta(\log\log n)$, so that all y-coordinates in P_z^a are smaller than in $P_z^{a'}$ whenever $a < a'$. We store in z a van Emde Boas structure V_z: for each P_z^a, V_z stores the minimum and maximum of $\{\mathsf{v}(y) \colon (x, y) \in P_z^a\}$. We assign to each point $(x, y) \in P$ the position $p_{x,y}$ such that, at the time of insertion of (x, y) into P, the node x corresponded to a prefix of $s[p_{x,y} \mathinner{..} n)$ and y to a suffix of $s[0 \mathinner{..} p_{x,y})$. We store in z a dynamic linear-space RMQ structure Q_z from [14] that contains a linked list of all chunks P_z^a, where each P_z^a stores the number $\min\{p_{x,y} \colon (x, y) \in P_z^a\}$, and Q_z supports insertions/updates in $O(\frac{\log n}{\log\log n})$ time and, for any b and c, Q_z can compute $\min\{p_{x,y} \colon (x, y) \in \bigcup_{a \in [b \mathinner{..} c]} P_z^a\}$ in $O(\frac{\log n}{\log\log n})$ time. Note that we spent $O(1)$ space per chunk P_z^a in z.

To compute m'', we process each $h \in (m \mathinner{..} m+r)$ as follows: read the sequence $\mathsf{right}(h, m + r - h)$ in $T_\circ$ and $\mathsf{left}(h, h - m)$ in $\overleftarrow{T}_\circ$, thus reaching nodes z and v (as in the above analysis). If $\mathsf{N}(z) \le \log\log n$, we report all t points in the range $[\mathsf{L}(z) \mathinner{..} \mathsf{R}(z)] \times [\mathsf{L}(v) \mathinner{..} \mathsf{R}(v)]$ in P in $O(\log n + t\frac{\log n}{\log\log n})$ time, which is $O(\log n)$ as $t \le \mathsf{N}(z)$; each point corresponds to an occurrence of the sequence $\mathsf{id}(B_m), \ldots, \mathsf{id}(B_{m+r-1})$ and we choose the point (x, y) with minimal $p_{x,y}$. If $\mathsf{N}(z) > \log\log n$, we use V_z to find all chunks P_z^a such that $P_z^a \subseteq [\mathsf{L}(z) \mathinner{..} \mathsf{R}(z)] \times [\mathsf{L}(v) \mathinner{..} \mathsf{R}(v)]$ and we use Q_z to compute $\min p_{x,y}$ for all (x, y) from such chunks; two chunks P_z^a may partially intersect this range and, for each such P_z^a, we calculate all elements $P_z^a \cap [\mathsf{L}(z) \mathinner{..} \mathsf{R}(z)] \times [\mathsf{L}(v) \mathinner{..} \mathsf{R}(v)]$ using one range reporting query on P. This overall takes time $O(\log n + |P_z^a| \cdot \frac{\log n}{\log\log n}) = O(\log n)$.

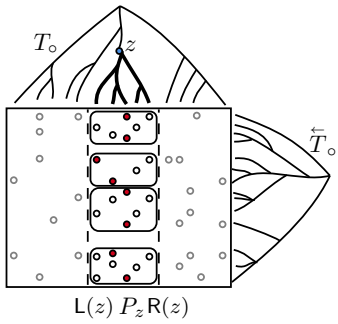

Figure 4 A schematic depiction of $T_\circ$ and $\overleftarrow{T}_\circ$ with the points P. The subset $P_z \subseteq P$ corresponding to a node $z \in T_\circ$ is depicted as split into chunks P_z^a. In each chunk P_z^a, its points with maximum and minimum y-coordinate are red. V_z stores the values $\mathsf{v}(y)$ for y-coordinates of exactly these red points ($O(|P_z|/\log\log n)$ values in total). Q_z stores, for each chunk P_z^a, the minimum of $p_{x,y}$ for all $(x,y) \in P_z^a$. The elements of each chunk P_z^a are not stored explicitly but can be retrieved in $O(\log n + |P_z^a|\frac{\log n}{\log\log n}) = O(\log n)$ time using one range reporting on P.

The total space is as follows. For a given node z, all its data structures take $O(|P_z|/\log\log n)$ space. Fix a depth ℓ in the trie $T_\circ$. The sets P_z for all nodes z with depth ℓ partition (a subset of) P and, therefore, in total occupy $O(|P|/\log\log n)$ space. Since the maximal depth of $T_\circ$ is $O(\log\log n)$, the total space is $O(|P|)$.

We maintain all V_z and Q_z as follows. Suppose that we insert a point (x,y) in P. We increment $\mathsf{N}(z)$ for all parents z of x. If $\mathsf{N}(z) = \log\log n$, we create V_z and Q_z for z with only one chunk. If $\mathsf{N}(z) > \log\log n$, we insert (x,y) into one chunk P_z^a found using V_z, possibly updating V_z in $O(\log\log n)$ time and Q_z in $O(\frac{\log n}{\log\log n})$ (if $p_{x,y} < \min\{p_{\tilde{x},\tilde{y}} \colon (\tilde{x},\tilde{y}) \in P_z^a\}$); if P_z^a overflows, we split it into two chunks with further updates to Q_z. To split P_z^a, we obtain all elements of P_z^a by one range reporting query on P in $O(\log n + |P_z^a|\frac{\log n}{\log\log n}) = O(\log n)$ time. Thus, the amortized time of insertion in P_z is $O(\frac{\log n}{\log\log n})$ since each chunk P_z^a, after its creation, can take $\Theta(\log\log n)$ more insertions before it is split. Hence, the amortized time of insertions in all parents of x in $T_\circ$ is $O(\frac{\log n}{\log\log n} \cdot \mathsf{height}(T_\circ)) = O(\log n)$.

When the value $\mathsf{v}(v)$ of $v \in \overleftarrow{T}_\circ$ changes, we should update all structures V_z containing it. To do it efficiently, we store in v as many copies of v as there are points of the form (z,v) in P, each with its own $\mathsf{v}(v)$ and they all are in the ordered linked list with all nodes of $T_\circ$. This does not change the algorithm or space bounds. Let v correspond to a point $(z,v) \in P$. In each parent z' of z in $T_\circ$, we update $V_{z'}$, which takes $O(\log\log n)$ time (to find whether the old value $\mathsf{v}(v)$ is present in $V_{z'}$ and to update it); $O((\log\log n)^2)$ time overall. Since the numbers $\mathsf{v}(\cdot)$ change at most $O(n)$ times in total, the time for all updates is $O(n(\log\log n)^2)$.

Removed blocks. Blocks $B_i, \ldots, B_j$ on level $2k+1$ or B_1 on level $2k$ can be removed. Non-leftmost blocks in J have no children and store pointers to their leftmost copies, which, by Lemma 5, cannot be removed. Hence, no "cleanup" is needed after removing blocks; A_L and A_R need no changes since their blocks contain links only to undeletable leftmost blocks.

Correctness. The described algorithm exactly corresponds to the construction of the hierarchy H from Section 3 and it applies all rules 1–4 whenever possible and parses ranges of children blocks into subranges whenever possible. Hence, it indeed produces J.

▶ **Theorem 10.** *Suppose that s is a string of length n over an alphabet $\{0,1,\ldots,n^{O(1)}\}$ and δ is its string complexity. Using $O(\delta\log\frac{n}{\delta})$ machine words, one can construct in one left-to-right pass on s in a streaming fashion an index of size $O(\delta\log\frac{n}{\delta})$ that can find in s all* occ *occurrences of any string of length m in $O(m + (\mathrm{occ}+1)\log^\epsilon n)$ time, where $\epsilon > 0$ is any fixed constant (the big-O in space hides factor $\frac{1}{\epsilon}$). The construction time is $O(n\log n)$ w.h.p.*

References

1 A. R. Adudodla and D. Kempa. Engineering fast and space-efficient recompression from SLP-compressed text. *arXiv preprint arXiv:2506.12011*, 2025.

2 S. Alstrup, T. Husfeldt, and T. Rauhe. Marked ancestor problems. In *Proc. 39th Annual Symposium on Foundations of Computer Science (Cat. no. 98CB36280)*, pages 534–543. IEEE, 1998. doi:10.1109/SFCS.1998.743504.

3 A. Andersson and M. Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM (JACM)*, 54(3):13–es, 2007. doi:10.1145/1236457.1236460.

4 D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna. Monotone minimal perfect hashing: searching a sorted table with O(1) accesses. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 785–794. SIAM, 2009. doi:10.1137/1.9781611973068.86.

5 D. Belazzougui, P. Boldi, and S. Vigna. Dynamic z-fast tries. In *Proc. SPIRE*, volume 6393, pages 159–172. Springer, 2010. doi:10.1007/978-3-642-16321-0_15.

6 D. Belazzougui, M. Cáceres, T. Gagie, P. Gawrychowski, J. Kärkkäinen, G. Navarro, A. Ordóñez, S. J. Puglisi, and Y. Tabei. Block trees. *Journal of Computer and System Sciences*, 117:1–22, 2021. doi:10.1016/j.jcss.2020.11.002.

7 D. Belazzougui, T. Gagie, P. Gawrychowski, J. Kärkkäinen, A. Ordónez, S. J. Puglisi, and Y. Tabei. Queries on LZ-bounded encodings. In *Proc. DCC*, pages 83–92. IEEE, 2015. doi:10.1109/DCC.2015.69.

8 M. A. Bender, R. Cole, E. D. Demaine, M. Farach-Colton, and J. Zito. Two simplified algorithms for maintaining order in a list. In *Algorithms-ESA 2002*, volume 2461 of *LNCS*, pages 152–164. Springer, 2002. doi:10.1007/3-540-45749-6_17.

9 M. A. Bender, M. Farach-Colton, J. Kuszmaul, and W. Kuszmaul. Modern hashing made simple. In *Proc. the 2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 363–373. SIAM, 2024. doi:10.1137/1.9781611977936.33.

10 P. Bille, I. L. Gørtz, M. B. T. Knudsen, M. Lewenstein, and H. W. Vildhøj. Longest common extensions in sublinear space. In *Proc. 26th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 9133 of *LNCS*, pages 65–76. Springer, 2015. doi:10.1007/978-3-319-19929-0_6.

11 P. Bille, I. L. Gørtz, and F. R. Skjoldjensen. Deterministic indexing for packed strings. In *Proc. CPM*, volume 78 of *LIPIcs*, pages 6:1–6:11, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2017.6.

12 O. Birenzwige, S. Golan, and E. Porat. Locally consistent parsing for text indexing in small space. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 607–626. SIAM, 2020. doi:10.1137/1.9781611975994.37.

13 G. E. Blelloch. Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In *SODA*, volume 8, pages 894–903, 2008.

14 G. S. Brodal, P. Davoodi, and S. S. Rao. Path minima queries in dynamic weighted trees. In *Workshop on Algorithms and Data Structures*, volume 6844 of *LNTCS*, pages 290–301. Springer, 2011. doi:10.1007/978-3-642-22300-6_25.

15 T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th Annual Symposium on Computational Geometry (SoCG)*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.

16 A. R. Christiansen and M. B. Ettienne. Compressed indexing with signature grammars. In *Proc. 13th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 10807 of *LNCS*, pages 331–345. Springer, 2018. doi:10.1007/978-3-319-77404-6_25.

17 A. R. Christiansen, M. B. Ettienne, T. Kociumaka, G. Navarro, and N. Prezza. Optimal-time dictionary-compressed indexes. *ACM Transactions on Algorithms (TALG)*, 17(1):1–39, 2020. doi:10.1145/3426473.

18 F. Claude and G. Navarro. Self-indexed grammar-based compression. *Fundamenta Informaticae*, 111(3):313–337, 2011. doi:10.3233/FI-2011-565.

19 R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

20 K. J. Conrad and P. R. Wilson. Grammatical Ziv-Lempel compression: Achieving PPM-class text compression ratios with LZ-class decompression speed. In *Proc. Data Compression Conference (DCC)*, page 586. IEEE, 2016. doi:10.1109/DCC.2016.119.

21 P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *Proc. 19th annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372. ACM, 1987. doi:10.1145/28395.28434.

22 M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger. Polynomial hash functions are reliable. In *Proc. 19th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 623 of *LNCS*, pages 235–246. Springer, 1992. doi:10.1007/3-540-55719-9_77.

23 T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proc. 11th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 8392 of *LNCS*, pages 731–742. Springer, 2014. doi:10.1007/978-3-642-54423-1_63.

24 T. Gagie, G. Navarro, and N. Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM (JACM)*, 67(1):1–54, 2020. doi:10.1145/3375890.

25 P. Gawrychowski, M. Lewenstein, and P.K. Nicholson. Weighted ancestors in suffix trees. In *Proc. 22nd Annual European Symposium on Algorithms (ESA)*, volume 8737 of *LNCS*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2_38.

26 A. Jeż. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016. doi:10.1016/j.tcs.2015.12.032.

27 R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of research and development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

28 D. Kempa and B. Langmead. Fast and space-efficient construction of AVL grammars from the LZ77 parsing. In *Proc. 29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 56:1–56:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.56.

29 D. Kempa and N. Prezza. At the roots of dictionary compression: string attractors. In *Proc. 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

30 T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003. doi:10.1016/S0304-3975(02)00426-7.

31 J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000. doi:10.1109/18.841160.

32 T. Kociumaka, G. Navarro, and F. Olivares. Near-optimal search time in δ-optimal space, and vice versa. In *Proc. 15th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 13568 of *LNCS*, pages 88–103. Springer, 2022. doi:10.1007/978-3-031-20624-5.

33 T. Kociumaka, G. Navarro, and F. Olivares. Near-optimal search time in δ-optimal space, and vice versa. *Algorithmica*, 86(4):1031–1056, 2024. doi:10.1007/s00453-023-01186-0.

34 T. Kociumaka, G. Navarro, and N. Prezza. Towards a definitive measure of repetitiveness. In *Proc. 14th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 12118 of *LNCS*, pages 207–219. Springer, 2020. doi:10.1007/978-3-030-61792-9_17.

35 T. Kociumaka, G. Navarro, and N. Prezza. Toward a definitive compressibility measure for repetitive sequences. *IEEE Transactions on Information Theory*, 69(4):2074–2092, 2022. doi:10.1109/TIT.2022.3224382.

36 T. Kopelowitz and M. Lewenstein. Dynamic weighted ancestors. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 565–574. ACM-SIAM, 2007. doi:10.5555/1283383.1283444.

37 D. Köppl, F. Kurpicz, and D. Meyer. Faster block tree construction. In *Proc. 31st Annual European Symposium on Algorithms (ESA)*, volume 274 of *LIPIcs*, pages 74:1–74:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.74.

38 D. Kosolobov. Compressed index with construction in compressed space. *arXiv preprint*, 2026. arXiv:2602.13735.

39 D. Kosolobov and N. Sivukhin. Construction of sparse suffix trees and LCE indexes in optimal time and space. In *Proc. 35th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 296 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CPM.2024.20.

40 K. Mehlhorn, R. Sundar, and C. Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997. doi:10.1007/BF02522825.

41 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In *Proc. 31st Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 161 of *LIPIcs*, pages 24:1–24:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2020.24.

42 G. Navarro. A self-index on block trees. In *Proc. 24th International Symposium on String Processing and Information Retrieval (SPIRE)*, volume 10508 of *LNCS*, pages 278–289. Springer, 2017. doi:10.1007/978-3-319-67428-5_24.

43 G. Navarro. Indexing highly repetitive string collections, part i: Repetitiveness measures. *ACM Computing Surveys (CSUR)*, 54(2):1–31, 2021. doi:10.1145/3434399.

44 Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theoretical Computer Science*, 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.

45 Y. Nekrich. Orthogonal range searching in linear and almost-linear space. *Computational Geometry*, 42(4):342–351, 2009. doi:10.1016/j.comgeo.2008.09.001.

46 T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda. Dynamic index and LZ factorization in compressed space. *Discrete Applied Mathematics*, 274:116–129, 2020. doi:10.1016/j.dam.2019.01.014.

47 S. Raskhodnikova, D. Ron, R. Rubinfeld, and A. Smith. Sublinear algorithms for approximating string compressibility. *Algorithmica*, 65(3):685–709, 2013. doi:10.1007/s00453-012-9618-6.

48 M. Ružić. Constructing efficient dictionaries in close to sorting time. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 5125 of *LNTCS*, pages 84–95. Springer, 2008. doi:10.1007/978-3-540-70575-8_8.

49 S. C. Sahinalp and U. Vishkin. Symmetry breaking for suffix tree construction. In *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 300–309. ACM, 1994. doi:10.1145/195058.195164.

50 Y. Takabatake, T. I, and H. Sakamoto. A space-optimal grammar compression. In *Proc. 25th Annual European Symposium on Algorithms (ESA)*, volume 87 of *LIPIcs*, pages 67:1–67:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2017.67.

51 F. Zhang, W. Wan, C. Zhang, J. Zhai, Y. Chai, H. Li, and X. Du. CompressDB: Enabling efficient compressed data direct processing for various databases. In *Proc. 2022 International Conference on Management of Data*, pages 1655–1669, 2022. doi:10.1145/3514221.3526130.

52 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

A Hierarchy of Blocks (Missing Proofs)

▶ **Lemma 2** (local consistency). *Fix $\ell = 2k$ or $\ell = 2k+1$. Let $B_1, B_2, \ldots, B_b$ denote all level-ℓ blocks. For any i, j and block B_h, if $s[i-2^{k+4} \mathinner{..} j+2^{k+4}] = s[\mathsf{b}(B_h)-2^{k+4} \mathinner{..} \mathsf{e}(B_h)+2^{k+4}]$, then there exists a level-ℓ block $B_{h'}$ such that $\mathsf{id}(B_{h'}) = \mathsf{id}(B_h)$, $\mathsf{b}(B_{h'}) = i$, and $\mathsf{e}(B_{h'}) = j$.*

Proof. The proof is by induction on ℓ. For $\ell = 0$, it is trivial. Assume $\ell > 0$. Let $\ell' = \ell - 1$. Denote $\tilde{\imath} = \mathsf{b}(B_h)$ and $\tilde{\jmath} = \mathsf{e}(B_h)$. Suppose that $B_h = \hat{B}_u \cdots \hat{B}_v$, where $\hat{B}_u, \ldots, \hat{B}_v$ are level-ℓ' blocks corresponding to B_h. By the inductive hypothesis, there are level-ℓ' blocks $\hat{B}_{u'}, \ldots, \hat{B}_{v'}$ such that $v' - u' = v - u$ and these blocks correspond to the substring $s[i\,..\,j]$ in the sense that $i = \mathsf{b}(\hat{B}_{u'})$ and $\mathsf{id}(\hat{B}_{u'}) = \mathsf{id}(\hat{B}_u), \ldots, \mathsf{id}(\hat{B}_{v'}) = \mathsf{id}(\hat{B}_v)$. Note that, for the inductive hypothesis to hold, it was sufficient to have a weaker assumption that $s[i{-}2^{k+3}\,..\,j{+}2^{k+3}] = s[\tilde{\imath}{-}2^{k+3}\,..\,\tilde{\jmath}{+}2^{k+3}]$.

Since the construction process copies blocks of length greater than 2^k from level ℓ' to level ℓ unaltered, all blocks $\hat{B}_u, \ldots, \hat{B}_v$ either have length at most 2^k, or $u = v$ and $|\hat{B}_u| > 2^k$. In the latter case, we have $u' = v'$, $\mathsf{id}(\hat{B}_{u'}) = \mathsf{id}(\hat{B}_u) = \mathsf{id}(B_h)$ and the block $\hat{B}_{u'}$ will be copied to level ℓ in the same way as B_u had been copied to level ℓ, which implies that the required level-ℓ block $B_{h'}$ is the copy of $\hat{B}_{u'}$.

From now on, assume that each of the blocks $\hat{B}_u, \ldots, \hat{B}_v$ has length at most 2^k. Observe that, since $0 < i \le j < |s| - 1$ and $0 < \tilde{\imath} \le \tilde{\jmath} < |s| - 1$, neither of the blocks $\hat{B}_u, \hat{B}_v, \hat{B}_{u'}, \hat{B}_{v'}$ is the first or last block on level ℓ'.

Let $\ell = 2k + 1$. We will prove the inductive step under the following weaker assumption: $s[i{-}2^{k+3}{-}2^k\,..\,j{+}2^{k+3}{+}2^k] = s[\tilde{\imath}{-}2^{k+3}{-}2^k\,..\,\tilde{\jmath}{+}2^{k+3}{+}2^k]$ (it will be useful in the analysis of the case $\ell = 2k{+}2$). As it was noted above, such weaker assumption still implies the existence of the blocks $\hat{B}_{u'}, \ldots, \hat{B}_{v'}$ corresponding to the substring $s[i\,..\,j]$. We have $\mathsf{id}(\hat{B}_u) = \cdots = \mathsf{id}(\hat{B}_v)$ and $\mathsf{id}(\hat{B}_{u-1}) \neq \mathsf{id}(\hat{B}_u)$ and $\mathsf{id}(\hat{B}_v) \neq \mathsf{id}(\hat{B}_{v+1})$. Since $\mathsf{id}(\hat{B}_{u'}) = \mathsf{id}(\hat{B}_u), \ldots, \mathsf{id}(\hat{B}_{v'}) = \mathsf{id}(\hat{B}_v)$, it remains to prove that $\mathsf{id}(\hat{B}_{u'-1}) \neq \mathsf{id}(\hat{B}_{u'})$ and $\mathsf{id}(\hat{B}_{v'}) \neq \mathsf{id}(\hat{B}_{v'+1})$. If $|\hat{B}_{u-1}| \le 2^k$, then the substring $\hat{B}_{u-1} = s[\tilde{\imath}{-}|\hat{B}_{u-1}|\,..\,\tilde{\imath}]$ is inside $s[\tilde{\imath}{-}2^{k+3}{-}2^k\,..\,\tilde{\jmath}{+}2^{k+3}{+}2^k]$ together with a "neighborhood" of length 2^{k+3} since $|\hat{B}_{u-1}| + 2^{k+3} \le 2^k + 2^{k+3}$. Therefore, by the inductive hypothesis, the substring $s[i{-}|\hat{B}_{u-1}|\,..\,i)$, which is equal to $\hat{B}_{u-1}$, must form a level-ℓ' block $\hat{B}_{u'-1}$ with $\mathsf{id}(\hat{B}_{u'-1}) = \mathsf{id}(\hat{B}_{u-1})$. Hence, we obtain $\mathsf{id}(B_{u'-1}) \neq \mathsf{id}(\hat{B}_{u'})$ since $\mathsf{id}(\hat{B}_{u-1}) \neq \mathsf{id}(\hat{B}_u)$. By the same reasoning, if $|\hat{B}_{u'-1}| > 2^k$, then $|\hat{B}_{u-1}| > 2^k$ and vice versa (due to symmetry). In case $|\hat{B}_{u'-1}| > 2^k$, we have $\mathsf{id}(\hat{B}_{u'-1}) \neq \mathsf{id}(\hat{B}_{u'})$ since $|\hat{B}_{u'}| \le 2^k$. Symmetrically, one can show that $\mathsf{id}(\hat{B}_{v'}) \neq \mathsf{id}(\hat{B}_{v'+1})$.

Let $\ell = 2k + 2$. To generate level-ℓ blocks (including B_h), the process marked some blocks from level $2k + 1$ using rules (a) and (b) described in Section 3. According to these rules, $\hat{B}_{u-1}$ and $\hat{B}_v$ must be marked and $\hat{B}_u, \ldots, \hat{B}_{v-1}$ must all be unmarked. It suffices to prove that $\hat{B}_{u'-1}$ and $\hat{B}_{v'}$ are marked and $\hat{B}_{u'}, \ldots, \hat{B}_{v'-1}$ are unmarked. Let us first derive a certain technical observation.

Suppose that, for some $t \in [1\,..\,7]$, each of the blocks $\hat{B}_{u-t}, \ldots, \hat{B}_{u-1}$ has length at most 2^k. Since $t \cdot 2^k + 2^{k+3} + 2^k \le 2^{k+4}$, each of these blocks is inside the substring $s[\tilde{\imath}{-}2^{k+4}\,..\,\tilde{\jmath}{+}2^{k+4}]$ together with a "neighborhood" of length $2^{k+3}{+}2^k$. As was proved in case $\ell = 2k{+}1$, it follows that the substrings $s[i{-}|\hat{B}_{u-t} \cdots \hat{B}_{u-1}|\,..\,i{-}|\hat{B}_{u-t+1} \cdots \hat{B}_{u-1}|), \ldots, s[i{-}|\hat{B}_{u-1}|\,..\,i)$, which are equal to $\hat{B}_{u-t}, \ldots, \hat{B}_{u-1}$, respectively, must form level-ℓ' blocks $\hat{B}_{u'-t}, \ldots, \hat{B}_{u'-1}$ such that $\mathsf{id}(\hat{B}_{u'-t}) = \mathsf{id}(\hat{B}_{u-t}), \ldots, \mathsf{id}(\hat{B}_{u'-1}) = \mathsf{id}(\hat{B}_{u-1})$. Symmetrically, if each of the blocks $\hat{B}_{u'-t}, \ldots, \hat{B}_{u'-1}$ has length at most 2^k, then also $\mathsf{id}(\hat{B}_{u'-t}) = \mathsf{id}(\hat{B}_{u-t}), \ldots, \mathsf{id}(\hat{B}_{u'-1}) = \mathsf{id}(\hat{B}_{u-1})$.

Suppose that $\hat{B}_{u-1}$ was marked because of rule (b), i.e., $\infty > \mathsf{id}''(\hat{B}_{u-3}) > \mathsf{id}''(\hat{B}_{u-2})$ and $\mathsf{id}''(\hat{B}_{u-2}) < \mathsf{id}''(\hat{B}_{u-1}) < \infty$. Hence, each of the five blocks $\hat{B}_{u-5}, \ldots, \hat{B}_{u-1}$ has length at most 2^k. Due to the above observation, we have $\mathsf{id}(\hat{B}_{u'-5}) = \mathsf{id}(\hat{B}_{u-5}), \ldots, \mathsf{id}(\hat{B}_{u'-1}) = \mathsf{id}(\hat{B}_{u-1})$. Therefore, the sequence $\mathsf{id}''(\hat{B}_{u-3}), \ldots, \mathsf{id}''(\hat{B}_v)$ is equal to the sequence $\mathsf{id}''(\hat{B}_{u'-3}), \ldots, \mathsf{id}''(\hat{B}_{v'})$. Hence, $\hat{B}_{u'-1}$ must be marked and $\hat{B}_{u'}, \ldots, \hat{B}_{v'-1}$ must be unmarked. We can analogously prove that, if $\hat{B}_v$ was marked due to rule (b), then $\hat{B}_{v'}$ is also marked.

Suppose that $\hat{B}_{u-1}$ was marked because of rule (a), i.e., $|\hat{B}_{u-1}| > 2^k$ or $|\hat{B}_u| > 2^k$. The latter is impossible since we assumed that $|\hat{B}_u| \leq 2^k$. Now, again due to the above observation, we have $|\hat{B}_{u'-1}| > 2^k$. Thus, $\mathsf{id}''(\hat{B}_{u'-1}) = \infty$ and, hence, $\hat{B}_{u'-1}$ is marked. Further, the sequences $\mathsf{id}''(\hat{B}_{u-1}), \ldots, \mathsf{id}''(\hat{B}_v)$ and $\mathsf{id}''(\hat{B}_{u'-1}), \ldots, \mathsf{id}''(\hat{B}_{v'})$ coincide, which implies that $\hat{B}_{u'}, \ldots, \hat{B}_{v'-1}$ are unmarked and, again, if $\hat{B}_v$ was marked due to rule (b), then $\hat{B}_{v'}$ is marked.

It remains to consider the case when $\hat{B}_v$ is marked due to rule (b) and to prove that $\hat{B}_{v'}$ must be marked too in this case. We assumed $|\hat{B}_v| \leq 2^k$ and, hence, the only option is that $|\hat{B}_{v+1}| > 2^k$. But we can repeat the proof of the above observation to deduce that $|\hat{B}_{v+1}| \leq 2^k$ iff $|\hat{B}_{v'+1}| \leq 2^k$. Therefore, we obtain $|\hat{B}_{v'+1}| > 2^k$ and, thus, $\hat{B}_{v'}$ is marked. ◀

▶ **Lemma 4.** *Any blocks B and B' with $\mathsf{id}(B) = \mathsf{id}(B')$ are equal as strings and both were created on the same level.*

Proof. The equality as strings is true by construction. Suppose, to the contrary, that the blocks B and B' were created on levels ℓ and ℓ', respectively, with $\ell < \ell'$ and ℓ is the smallest such level where the lemma fails. Let $\ell = 2k + 2$, for some k, and B was created from blocks $B_i, \ldots, B_j$ from level $2k + 1$ so that $B = B_i \cdots B_j$. Since $\mathsf{id}(B) = \mathsf{id}(B')$, there are blocks $B'_{i'}, \ldots, B'_{j'}$ on a level $<\ell'$ from which the block B' was created such that $j - i = j' - i'$ and $\mathsf{id}(B_i) = \mathsf{id}(B'_{i'}), \ldots, \mathsf{id}(B_j) = \mathsf{id}(B'_{j'})$. Since $2k + 1 < \ell$ and ℓ is the minimal level where the lemma fails, $B'_{i'} \cdots B'_{j'}$ were present on level $2k + 1$. The length of each of the blocks $B_i, \ldots, B_j$ is at most 2^k and, then, the same holds for $B'_{i'}, \ldots, B'_{j'}$. Thus, the latter blocks should have participated in a maximal range of blocks with lengths $\leq 2^k$. In such a range, all blocks are united into disjoint subranges, each of which contains at least two blocks, except, possibly, the last subrange, which can be formed by one block. Each subrange generates a block for level $2k + 2$. But this contradicts the assumption that the blocks $B'_{i'}, \ldots, B'_{j'}$ should have been transitioned unaffected to level $\ell' - 1 \geq \ell = 2k + 2$.

Let $\ell = 2k + 1$ for some k. One can analogously find decompositions of B and B' into blocks from level $2k$: $B = B_i \cdots B_j$ and $B' = B'_{i'} \cdots B'_{j'}$, where $j - i = j' - i'$ and $\mathsf{id}(B_i) = \mathsf{id}(B'_{i'}), \ldots, \mathsf{id}(B_j) = \mathsf{id}(B'_{j'})$. Since $\ell = 2k + 1$, we have $\mathsf{id}(B_i) = \cdots = \mathsf{id}(B_j)$ and B was produced by uniting these blocks into a "run". Then, $B'_{i'}, \ldots, B'_{j'}$ should have been similarly united on the level $\ell - 1$, a contradiction. ◀

B Jiggly Block Tree (Missing Proofs)

▶ **Lemma 5.** *For $d \geq 0$, let B be the leftmost topmost block with $\mathsf{id}(B) = d$ and $|B| > 1$ in the hierarchy H, i.e., $\mathsf{b}(B)$ is minimal for B among all blocks with this id and B is such block with maximal level. The jiggly block tree J retains the block B and B has children in J.*

Proof. Consider the process that constructs J by pruning the original hierarchy tree H applying, first, rules 1–4 and, then, introducing intermediate blocks. Suppose, to the contrary, that a certain rule either removed B itself or all children of B and assume that B is the first leftmost and topmost block with any id for which this happened. Let ℓ denote the level of B.

The rules 1–2 obviously could not do this (recall that $|B| > 1$). Rule 3 cannot be applied to B itself since B has no "preceding" blocks with equal id. Rule 4 applied to B itself does not remove all children and, thus, is not contradictory.

One can show by induction on levels that, for any blocks B' and B'', if $\mathsf{id}(B') = \mathsf{id}(B'')$, then, given $\ell' = 2k$ or $\ell' = 2k + 1$, any level-ℓ' block from the subtree rooted at B' in the tree H has a corresponding block with equal identifier in the subtree rooted at B''. It is then

straightforward that a rule 3 or 4 applied to a block cannot remove a block B' containing B as a descendant since, for such a rule, there is always an unaffected block B'' "preceding" B' with $\mathsf{id}(B'') = \mathsf{id}(B')$ and, hence, B should have been among the descendants of B'' if B were a descendant of B', which is impossible because B is already leftmost.

Suppose that B was pruned due to the introduction of an intermediate block B' in place of a subrange of blocks $B_m \cdots B_{m+r-1}$ with $r > 1$ from a level $2k+1$. By construction, there is $m' < m$ such that $\mathsf{id}(B_{m'+h}) = \mathsf{id}(B_{m+h})$, for all $h \in [0\,..\,r)$. If B were a descendant of B_{m+h}, for $h \in [0\,..\,r)$ (maybe $B_{m+h} = B$), then B must be a descendant of $B_{m'+h}$ in the tree H by the observation discussed above. This is a contradiction since B is leftmost. ◀

▶ **Theorem 7.** *The jiggly block tree for string s of length n has size $O(\delta \log \frac{n}{\delta})$, where δ is the string complexity of s, provided $\delta \geq \Omega(\log \log n)$.*

Proof (continuation). Let us detail the ideas described in the conceptual part of the proof from the main text.

Part i. Fix $\ell = 2k+1$ or $\ell = 2k+2$, for some k. Consider a level-ℓ block B in J that is not a leaf. By Lemma 3 (local sparsity), there are $O(1)$ such blocks B for which $\mathsf{b}(B) < 2^{k+6}$ or $\mathsf{e}(B) + 2^{k+6} \geq n$. Assume that these conditions do not hold for B and, hence, the substring $s[\mathsf{b}(B){-}2^{k+6}\,..\,\mathsf{e}(B){+}2^{k+6}]$ is well defined. We have $|B| \leq 2^k$ since otherwise B should have had a copy on level $\ell+1$ that is the parent of B in the original tree H and, hence, B could be removed by rule 2. Let us show that the string $s[\mathsf{b}(B){-}2^{k+4}\,..\,\mathsf{e}(B){+}2^{k+4}]$ has no occurrences at positions smaller than $\mathsf{b}(B) - 2^{k+4}$. Suppose, to the contrary, that it occurs at a position $i' - 2^{k+4}$ with $i' < \mathsf{b}(B)$. Then, due to Lemma 2 (local consistency), there is a block $B' = s[i'\,..\,i'{+}|B|)$ in the tree H such that $\mathsf{id}(B') = \mathsf{id}(B)$. By Lemma 5, the tree J contains the leftmost and topmost such block B'' with $\mathsf{id}(B'') = \mathsf{id}(B)$ and $\mathsf{b}(B'') < \mathsf{b}(B)$. But then all descendants of B could be removed by rule 3, which is a contradiction.

The length of $s[\mathsf{b}(B){-}2^{k+4}\,..\,\mathsf{e}(B){+}2^{k+4}]$ is $2 \cdot 2^{k+4} + |B| \leq 2^{k+5} + 2^k$. Hence, the strings $s[h\,..\,h{+}2^{k+6})$ with $h \in (\mathsf{b}(B){-}2^{k+4}{-}2^k\,..\,\mathsf{b}(B){-}2^{k+4}]$ all cover the substring $s[\mathsf{b}(B){-}2^{k+4}\,..\,\mathsf{e}(B){+}2^{k+4}]$ and, thus, must be distinct and each such string $s[h\,..\,h{+}2^{k+6})$ has no occurrences at positions smaller than h. We assign the block B to each of these 2^k strings $s[h\,..\,h{+}2^{k+6})$. We do the same analysis and assignments for every internal level-ℓ block in J. Due to Lemma 3 (local sparsity), we could have assign at most $O(1)$ blocks to any fixed substring $s[h\,..\,h{+}2^{k+6})$ with $h \in [0\,..\,n{-}2^{k+6})$. Denote by $d_{2^{k+6}}$ the number of distinct substrings of length 2^{k+6} and by b the number of internal blocks on level ℓ in J. We associate each distinct substring of length 2^{k+6} with its leftmost occurrence $s[h\,..\,h{+}2^{k+6})$. Thus, every internal block on level ℓ is assigned to exactly 2^k distinct substrings. We obtain $b \cdot 2^k \leq O(d_{2^{k+6}})$. Hence, $b \leq O(d_{2^{k+6}}/2^k) \leq O(\delta)$. So, the number of internal level-ℓ blocks in J is $O(\delta)$ and the total number of internal blocks in J is $O(\delta \log \frac{n}{\delta})$. Let us count leaves.

Part ii. First, let us restrict our analysis to only certain leaf blocks B. Denote by $\mathsf{par}(B)$ the parent of B. Since the total number of internal blocks is $O(\delta \log \frac{n}{\delta})$, the number of leaf blocks B such that either $2|B| \geq |\mathsf{par}(B)|$ or B is the first or last child of $\mathsf{par}(B)$ is at most $O(\delta \log \frac{n}{\delta})$ (across all levels). Therefore, it suffices to consider the case when $2|B| < |\mathsf{par}(B)|$ and B is neither first nor last child of $\mathsf{par}(B)$. Further, there are at most $O(\log n)$ blocks B' such that $\mathsf{b}(B') \leq 2^{11}|B'|$ and $\mathsf{e}(B') + 2^{11}|B'| \geq n$ and each such block B' contributes at most $O(\log \log n)$ leaf-children, i.e., $O(\log n \log \log n)$ in total, which is $O(\delta \log \frac{n}{\delta})$ since it was assumed that $\delta \geq \Omega(\log \log n)$. Thus, we can assume that $B' = \mathsf{par}(B)$, the parent of the leaf block B, is "far enough" from both ends of the string s so that the substring $s[\mathsf{b}(B'){-}2^{11}|B'|\,..\,\mathsf{e}(B'){+}2^{11}|B'|]$ is well defined. The block $B' = \mathsf{par}(B)$ is not a run block, due to rule 4; hence, B is from a level $2k+1$, for some k (B might be intermediate).

(⋆) Restriction: it suffices to count only leaf blocks B from levels $2k+1$ with $k \geq 0$ such that, for $B' = \mathsf{par}(B)$, we have $2|B| < |B'|$ and B is neither first nor last child of B', and $\mathsf{b}(B') > 2^{11}|B'|$ and $\mathsf{e}(B') + 2^{11}|B'| < n$.

Let B be a leaf block from level $2k+1$ as in the restriction and $B' = \mathsf{par}(B)$. Denote by $B_1, \ldots, B_b$ all blocks from level $2k+1$ in the hierarchy H; we mark them according to conditions (a) and (b) from Section 3 so that we can use the functions left and right from now on. Let $B_i, \ldots, B_j$ be a range of blocks from which $B' = B_i \cdots B_j$ was produced in H. Thus, $B = B_m \cdots B_{m+r-1}$ for some m and r such that $[m \,..\, m+r) \subseteq (i \,..\, j)$. We have $m + r \leq j$ since B is not the last child of B'. The block B is either intermediate (if $r > 1$) or an original block from H (if $r = 1$). Denote $x = \mathsf{b}(B)$ and $y = \mathsf{e}(B) + |B_{m+r}|$ and consider the substring $s[x \,..\, y] = B_m \cdots B_{m+r}$. The subrange $B_m, \ldots, B_{m+r-1}$ was distinguished by the greedy parsing procedure described in Section 4, according to which there are no indices $m' < m$ such that $\mathsf{left}(m', 4(r+1)) = \mathsf{left}(m, 4(r+1))$ and $\mathsf{right}(m', 5(r+1)) = \mathsf{right}(m, 5(r+1))$. Note that the length of each of the blocks $B_i, \ldots, B_j$ is at most 2^k. As $s[x \,..\, y] = B_m \cdots B_{m+r}$, one can show that the string $s[x-z \,..\, y+z]$ with $z = 4(r+2)2^k + 2^{k+4}$ cannot occur at any position $x'-z$ with $x' < x$ since, otherwise, Lemma 2 (local consistency) would imply that there is $m' < m$ such that $\mathsf{left}(m', 4(r+1)) = \mathsf{left}(m, 4(r+1))$ and $\mathsf{right}(m', 5(r+1)) = \mathsf{right}(m, 5(r+1))$. Any block B_h with $h \in [1 \,..\, b]$ is marked iff $|B_h| > 2^k$ or $|B_{h+1}| > 2^k$ or B_h is preceded by a local minimum $\mathsf{id}''(B_{h-1})$; to calculate the local minimum, we need the blocks $B_{h-4}, B_{h-3}, B_{h-2}, B_{h-1}$. If $\mathsf{left}(m, 4(r+1))$ does not contain the special symbol \$, then z is large enough to guarantee that at any occurrence $s[x'-z \,..\, y'+z]$ of $s[x-z \,..\, y+z]$ there are exactly the same $4(r+1)$ blocks $B_{m-4(r+1)}, \ldots, B_{m-1}$, each of length $\leq 2^k$, to the left of $s[x' \,..\, y']$ and, moreover, neither of them could be marked due to a local minimum since, if so, then the four blocks $B_{m-4(r+1)-4}, \ldots, B_{m-4(r+1)-1}$, because of which the local minimum appeared, should have occured at both positions by Lemma 2 (as z is large enough). The case when left contains \$ is analogous and the analysis of right is also analogous.

Denote by k' the smallest integer such that $k' \geq k+4$ and $2^{k'} > |B_m \cdots B_{m+r}|$. It follows from Lemma 3 (local sparsity) that $r + 2 \leq 2^6 2^{k'}/2^k$ and, hence, $2^{k'+6} \geq (r+2)2^k$ and $2^{k'+8} + 2^{k'} \geq 4(r+2)2^k + 2^{k+4} = z$. Therefore, $2^{k'+10} > 2 \cdot 2^{k'+8} + 2 \cdot 2^{k'} \geq 2z + |B_m \cdots B_{m+r}| + 2^{k'}$ and the strings $s[h \,..\, h+2^{k'+10})$ with $h \in (x-z-2^{k'} \,..\, x-z]$ all cover $s[x-z \,..\, y+z]$ and, thus, must be distinct and each such string $s[h \,..\, h+2^{k'+10})$ has no occurrences at positions smaller than h (note that all these strings are well defined since the block B' is "far enough" from both ends of the string s). We assign the block B to each of these $2^{k'}$ strings $s[h \,..\, h+2^{k'+10})$ with $h \in (x-z-2^{k'} \,..\, x-z]$. We do the same analysis and assignments for every leaf block B in J that satisfies the conditions from restriction (⋆).

Fix a substring $s[h \,..\, h+2^{k'+10})$ with $h \in [0 \,..\, n-2^{k'+10})$ and $k' > 0$. Let us show that $O(1)$ blocks were assigned to this substring in total. Let k be such that $k' = k+4$. Then, by Lemma 3 (local sparsity), at most $O(1)$ blocks were assigned from levels $\geq 2k$. Let k be such that $k' > k+4$. If a block $B = B_m \cdots B_{m+r-1}$ from level $2k+1$ was assigned to the string $s[h \,..\, h+2^{k'+10})$, where $B_m, \ldots, B_{m+r}$ are blocks from level $2k+1$ corresponding to B as in the description above, then we have $2^{k'} > |B_m \cdots B_{m+r}| \geq 2^{k'-1}$. We associate the block B with the interval $I(B) = [\mathsf{b}(B_m) \,..\, \mathsf{e}(B_{m+r})]$ of length between $2^{k'-1}$ and $2^{k'}$ that is a subset of $[h \,..\, h+2^{k'+10})$. For each block B assigned to $s[h \,..\, h+2^{k'+10})$ from every level $2k+1$ such that $k' > k+4$, we associate an interval $I(B)$ in this was. If intervals $I(B)$ and $I(B')$ intersect, then either they are nested (and the blocks B and B' are nested) or B and B' are from the same level. A given interval $I(B)$ may intersect at most two other intervals associated with blocks from the same level. If a block B' was assigned to the same string as block B and B' is an ancestor of B in J (they are nested), then for any ancestor B'' of

B', $|B''| > 2|B'| \geq 2|B_m \cdots B_{m+r}| \geq 2^{k'}$ and, hence, B'' could not be assigned to the same string $s[h\,..\,h{+}2^{k'+10})$. So, the system of intervals $I(B)$ associated with all blocks B assigned to the substring $s[h\,..\,h{+}2^{k'+10})$ contains $O(2^{k'+10}/2^{k'}) = O(1)$ intervals: to see this, count separately the intervals that are not nested in any intervals, and all remaining intervals.

Fix k'. Denote by $d_{2^{k'+10}}$ the number of distinct substrings of length $2^{k'+10}$ and by $\mathcal{B}_{k'}$ the set of blocks that were assigned to all strings $s[h\,..\,h{+}2^{k'+10})$ with $h \in [0\,..\,n{-}2^{k'+10})$. We associate each distinct substring of length $2^{k'+10}$ with its leftmost occurrence $s[h\,..\,h{+}2^{k'+10})$. Thus, every block from $\mathcal{B}_{k'}$ was assigned to exactly $2^{k'}$ such distinct substrings. We obtain $|\mathcal{B}_{k'}| \cdot 2^{k'} \leq O(d_{2^{k'+10}})$ and, hence, $|\mathcal{B}_{k'}| \leq O(d_{2^{k'+10}}/2^{k'}) \leq O(\delta)$. Therefore, one can estimate as $O(\delta \log \frac{n}{\delta})$ the number of leaf blocks that were assigned to substrings of length $2^{k'+10}$ with $k' \leq \log \frac{n}{\delta}$ as described above, i.e., $\sum_{k'=1}^{\log(n/\delta)} |\mathcal{B}_{k'}| \leq O(\delta \log \frac{n}{\delta})$.

Part iii. It remains to count the leaf blocks that were assigned to substrings of length $2^{k'+10}$ with $k' > \log \frac{n}{\delta}$. Fix such k'. Consider a leaf block B from a level $2k+1$ that was assigned to a substring of length $2^{k'+10}$. Let $B = B_m \cdots B_{m+r-1}$, where $B_m, \ldots, B_{m+r}$ are blocks from level $2k+1$ corresponding to B as in the description above. We associate B with the interval $I(B) = [\mathsf{b}(B_m)\,..\,\mathsf{e}(B_{m+r})]$. We do the same association to intervals for all blocks B assigned to substrings of length $2^{k'+10}$. As in the above discussion, one can show that all such intervals $I(B)$ are distinct, there is no "double nestedness" such as $I(B) \subset I(B') \subset I(B'')$, and an interval $I(B)$ can intersect at most two other intervals $I(B')$ and $I(B'')$ that are not nested in it and do not contain it and B' and B'' must be from the same level as B in this case. Since, for each such B, $|I(B)| \geq 2^{k'-1}$, one can estimate the number of intervals as $O(n/2^{k'})$ by counting separately the intervals that are not nested inside other intervals, and all remaining intervals. Therefore, the total number of blocks assigned to substrings of length $2^{k'+10}$ for all $k' > \log \frac{n}{\delta}$ is $O(\sum_{k'} n/2^{k'}) = O(n/2^{\log(n/\delta)}) = O(\delta)$. ◀