Data Structure Lower Bounds on Random Access to Grammar-Compressed Strings*

Elad Verbin and Wei Yu

Aarhus University
34 Åbogade, 8200 Aarhus N, Denmark
{eladv, yuwei}@cs.au.dk

Abstract. In this paper we investigate the problem of building a static data structure that represents a string s using space close to its compressed size, and allows fast access to individual characters of s. This type of data structures was investigated by the recent paper of Bille et al. [3]. Let n be the size of a context-free grammar that derives a unique string s of length L. (Note that L might be exponential in n.) Bille et al. showed a data structure that uses space $O(n)$ and allows to query for the i-th character of s using running time $O(\log L)$. Their data structure works on a word RAM with a word size of $\log L$ bits.

Here we prove that for such data structures, if the space is $\text{poly}(n)$, the query time must be at least $(\log L)^{1-\varepsilon}/\log \mathcal{S}$ where $\mathcal{S}$ is the space used, for any constant $\varepsilon > 0$. As a function of n, our lower bound is $\Omega(n^{1/2-\varepsilon})$. Our proof holds in the cell-probe model with a word size of $\log L$ bits, so in particular it holds in the word RAM model. We show that no lower bound significantly better than $n^{1/2-\varepsilon}$ can be achieved in the cell-probe model, since there is a data structure in the cell-probe model that uses $O(n)$ space and achieves $O(\sqrt{n \log n})$ query time. The "bad" setting of parameters occurs roughly when $L = 2^{\sqrt{n}}$. We also prove a lower bound for the case of not-as-compressible strings, where, say, $L = n^{1+\epsilon}$. For this case, we prove that if the space is $O(n \cdot \text{polylog}(n))$, the query time must be at least $\Omega(\log n/\log\log n)$.

The proof works by reduction from communication complexity, namely to the LSD (Lopsided Set Disjointness) problem, recently employed by Pătraşcu and others. We prove lower bounds also for the case of LZ-compression. All of our lower bounds hold even when the strings are over an alphabet of size 2 and hold even for randomized data structures with 2-sided error.

* The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed. Part of the work was done while the authors were working in IIIS, Tsinghua University in China.

J. Fischer and P. Sanders (Eds.): CPM 2013, LNCS 7922, pp. 247–258, 2013.

1 Introduction

In many modern databases, strings are stored in compressed form. Many compression schemes are grammar-based, in particular Lempel-Ziv [6,11,12] and its variants, as well as Run-Length Encoding.

A natural desire is to store a text using space close to its compressed size, but to still allow fast access to individual characters: can we do something faster than simply extracting the whole text each time we need to access a character? This question was recently answered in the affirmative by Bille et al. [3] and by Claude and Navarro [5]. These two works investigate the problem of storing a string that can be represented by a small CFG (context-free grammar) of size n, while allowing some basic stringology operations, in particular random access to a character in the text. The data structure of Bille et al. [3, Theorem 1] stores the text in space linear in n, while allowing access to an individual character in time $O(\log L)$, where L is the text's *uncompressed* size.[1] But is that the best upper bound possible?

In this paper we show a $(\log L)^{1-\varepsilon}$ lower bound on the query time when the space used by the data structure is $\text{poly}(n)$, showing that the result of Bille et al. is close to optimal. Our lower bounds are proved in the cell-probe model of Yao [10], with word size $\log L$, therefore they in particular hold for the model studied by Bille et al. [3], since the cell-probe model is strictly stronger than the RAM model. Our lower bound is proved by a reduction from Lopsided Set Disjointness (LSD), a problem for which Pătraşcu has recently proved an essentially-tight randomized lower bound [8]. The idea is to prove that grammars are rich enough to effectively "simulate" a disjointness query: our class of grammars, presented in Section 3.1, might be of independent interest as a class of "hard" grammars for other purposes as well.

In terms of n, our lower bound is $n^{1/2-\varepsilon}$. The results of Bille et al. imply an upper bound of $O(n)$ on the query time, since $\log L \leq n$, therefore in terms of n there is a curious quadratic gap between our lower bound and Bille et al.'s upper bound. We show that this gap can be closed by giving a better data structure: we show a data structure which takes space $O(n)$ and has query time $O(\sqrt{n \log n})$, showing that no significantly better lower bound is possible. This data structure, however, comes with a big caveat – it runs in the highly-unrealistic cell-probe model, thus serving more as an impossibility proof for lower bounds than as a reasonable upper bound. The question remains open of whether such a data structure exists in the more realistic word RAM model.

Our lower bound holds for a particular, "worst-case", dependence of L on n. Namely, L is roughly $2^{\sqrt{n}}$. It might also be interesting to explicitly limit the range of allowed parameters to other regimes, for example to non-highly-compressible text; in such a regime it might be that $L = n^{1+\epsilon}$. The above result does not imply any lower bound for this case. Furthermore, we show in another result that for any data structure in that regime, if the space is $n \cdot \text{polylog}\, n$,

[1] The result of Bille et. al. also allows other query operations such as pattern matching; we do not discuss those in this paper.

the query time must be $\Omega(\log n / \log\log n)$. This lower bound holds, again, in the cell probe model with words of size $\log n$ bits, and is proved by a reduction from two-dimensional range counting (which, once again, was lower bounded by a reduction from LSD [8]).

2 Preliminaries

In this paper we denote $[N] = \{1, \ldots, N\}$. All logarithms are in base 2 unless explicitly stated otherwise.

Our lower bounds are proved in Yao's cell-probe model [10]. In the cell-probe model, the memory is an array of cells, where each cell consists of w bits each. The query time is measured as the number of cells read, while all computations are free. This model is strictly stronger than the word RAM, since in the word RAM the operations allowed on words are restricted, while in the cell-probe model we only measure the number of cells accessed. The cell-probe model is widely used in proving data structure lower bounds, especially by reduction from communication complexity problems [7]. In this paper we prove our result by a reduction from the Blocked-LSD problem introduced by Pătraşcu [8].

An SLP (straight line program) is a collection of n derivation rules, defining the symbols $g_1, \ldots, g_n$. Each rule is either of the form $g_i \to `\sigma'$, i.e. g_i is a terminal, which takes the value of a character σ from the underlying alphabet, or of the form $g_i \to g_j g_k$, where $j < i$ and $k < i$, i.e. g_j and g_k were already defined, and we define the nonterminal symbol g_i to be their concatenation. The symbol g_n is the *start symbol*. To derive the string we start from g_n and follow the derivation rules until we get a sequence of characters from the alphabet. The length of the derived string is at most 2^n. W.l.o.g. we assume it is at least n. As the same in Bille et al. [3], we also assume w.l.o.g. that the grammars are in fact SLPs and so on the righthand side of each grammar rule there are either exactly two variables or one terminal symbol. In this paper SLP, CFG and grammar all mean the same thing.

The grammar random access problem is the following problem.

Definition 1 (Grammar Random Access Problem). *For a CFG* G *of size n representing a binary string of length L, the problem is to build a data structure to support the following query: given $1 \le i \le L$, return the i-th character (bit) in the string.*

We study two other data structure problems, which are closely related to their communication complexity counterparts.

Definition 2 (Set Disjointness, SD_N). *For a set $Y \subseteq [N]$, the problem is to build a data structure to support the following query: given a set $X \subseteq [N]$, answer whether $X \cap Y = \emptyset$.*

Given a universe $[BN] = \{1, \ldots, BN\}$, a set X is called *blocked with cardinality* N if when we divide the universe $[BN]$ into N equal-sized consecutive blocks, X contains exactly one element from each of the blocks while Y could be arbitrary.

Definition 3 (Blocked Lopsided Set Disjointness, $BLSD_{B,N}$). *For a set $Y \subseteq [BN]$, the problem is to build a data structure to support the following query: given a blocked set $X \subseteq [BN]$ where $|X| = N$ containing 1 element from each size B block, answer whether $X \cap Y = \emptyset$.*

For proving lower bound for near-linear space data structures, we also need reductions from a variant of the range counting problem.

Definition 4 (Range Counting). *The range counting problem is a static data structure problem. We need to preprocess a set of n points on a $[n] \times [n^\epsilon]$ grid. A query (x, y) asks to count the number of points in a dominance rectangle $[1, x] \times [1, y]$ (a rectangle contains the lower left corner $(1, 1)$). Return the answer* modulo 2.

Note that the above problem is "easier" than the classical 2D range-counting problem, since it is a dominant query problem, it is a grid $n \times n^\epsilon$, and it is modulo 2. However, the (tight) lower bound that is known for the general problem, given by Pătraşcu [8], could be generalized for the problem we define.

3 Lower Bound for Grammar Random Access

In this section we prove the main lower bound for grammar random access. In Section 3.1 we show the main reduction from SD and BLSD. In Section 3.2 we prove lower bounds for SD and BLSD, based on reductions to communication complexity (these are implicit in the work of Pătraşcu [8]). Finally, in Section 3.3 we tie these together to get our lower bounds.

3.1 Reduction from SD and LSD

In this section we show how to reduce the grammar access problem from SD or BLSD, by considering a particular type of grammar. The reductions tie the parameters n and L to the parameters B and N of BLSD (or just to the parameter N of SD). In Section 3.3 we show how to choose the relation between the various parameters in order to get our lower bounds. We remark that the particular multiplicative constants in the lemmas below will not matter, but we give them nonetheless, for concreteness.

These reductions might be confusing for the reader, but they are in fact almost entirely tautological. They just follow from the fact that the communication matrix of SD is a tensor product of the 2 by 2 communication matrices for the coordinates, i.e., it is just a N-fold tensor product of the matrix $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$. For BLSD, the communication matrix is the N-fold tensor product of the $(2^B) \times B$ communication matrix for each block (for example, for $B = 3$ this matrix is $\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$). We do not formulate our arguments in the language of

communication matrices and tensor products, since this would hide what is really going on. To aid the reader, we give an example after each of the two constructions.

Lemma 1 (Reduction from SD_N). *For any set $Y \subseteq [N]$, there is a grammar G_Y of size $n = 2N + 1$ deriving a binary string s_Y of length $L = 2^N$ such that for any set $X \subseteq [N]$, it holds that $s_Y[X] = 1$ iff $X \cap Y = \emptyset$.*

Note that in this lemma we have indexed the string s by *sets*: there are 2^N possible sets X, and the length of the string s_Y is also 2^N – each set X serves as an index of a unique character. The indexing is done in lexicographic order: the set X is identified with its *characteristic vector*, i.e., the vector in $\{0,1\}^N$ whose i-th coordinate is '1' if $i \in X$, and '0' otherwise, and the sets are ordered according to lexicographic order of their characteristic vectors. For example, here is the ordering for the case $N = 3$: $\emptyset, \{1\}, \{2\}, \{1,2\}, \{3\}, \{1,3\}, \{2,3\}, \{1,2,3\}$.

Proof. We now show how to build the grammar G_Y. The grammar has N symbols for the strings $0, 0^2, 0^4, \ldots, 0^{2^{N-1}}$, i.e., all strings consisting solely of the character '0', of lengths which are all powers of 2 up to 2^{N-1}. Then, the grammar has $N+1$ additional symbols $g_0, g_1, \ldots, g_N$. The terminal g_0 is equal to the character 1. For any $1 \leq i \leq N$, we set g_i to be equal to $g_{i-1}g_{i-1}$ if $i \notin Y$, and to be equal to $g_{i-1}0^{2^{i-1}}$ if $i \in Y$. The start symbol of the grammar is g_N.

We claim that the string derived by this grammar has the property that $s_Y[X] = 1$ iff $X \cap Y = \emptyset$. This is easy to prove by induction on i, where the induction claim is that for any i, g_i is the string that corresponds to the set $Y \cap \{1, \ldots, i\}$ over the universe $\{1, \ldots, i\}$. □

Example 1. Consider the universe $N = 4$. Let $Y = \{1, 3\}$. The string s_Y is 1010000010100000. The locations of the 1's correspond exactly to the sets that don't intersect Y, namely to the sets $\emptyset$, $\{2\}$, $\{4\}$ and $\{2, 4\}$, respectively.

We now show the reduction from blocked LSD. It follows along the same general idea, but the grammar is slightly more complicated.

Lemma 2 (Reduction from $BLSD_{B,N}$). *For any set $Y \subseteq [BN]$, there is a grammar G_Y of size $n = 2BN + 1$ deriving a binary string s_Y of length $L = B^N$ such that for any blocked set $X \subseteq [BN]$ of cardinality N, it holds that $s_Y[X] = 1$ iff $X \cap Y = \emptyset$.*

Recall that by "a blocked set $X \subseteq [BN]$ of cardinality N" we mean a set such that the universe $[BN]$ is divided into N equal-sized blocks, and X contains exactly one element from each of these blocks.

Note that in this lemma we have again indexed the string s by *sets*: there are B^N possible sets X and the length of the string is B^N. The indexing is done in *lexicographic order*, this time identifying a set X with a length-N vector whose i-th coordinate is chosen according to which element it contains in block i, and the sets are ordered according to lexicographic order of their characteristic vectors. For example, here is the ordering for the case $N = 2, B = 3$: $\{1,4\}, \{2,4\}, \{3,4\}, \{1,5\}, \{2,5\}, \{3,5\}, \{1,6\}, \{2,6\}, \{3,6\}$.

The construction in this reduction is similar to that in the case of SD, but instead of working element by element, we work block by block.

Proof. We now show how to build the grammar G_Y. The grammar has N symbols for the strings $0, 0^B, 0^{B^2}, 0^{B^3}, \ldots, 0^{B^{N-1}}$, i.e., all strings consisting solely of the character '0', of lengths which are all powers of B up to B^{N-1}. We cannot simply obtain the symbols directly from each other: e.g., to obtain 0^{B^2} from 0^B, we need to concatenate 0^B with itself B times. Thus we use BN rules to derive all of these symbols. (In fact, $O(N \log B)$ rules can suffice but this does not matter).

Then, beyond these, the grammar has $N+1$ additional symbols $g_0, g_1, \ldots, g_N$, one for each block. The terminal g_0 is equal to the character 1. For any $1 \leq 1 \leq N$, g_i is constructed from g_{i-1} according to which elements of the i-th block are in Y: we set g_i to be a concatenation of B symbols, each of which is either g_{i-1} or $0^{B^{i-1}}$. In particular, g_i is the concatenation of $g_i^{(1)}, \ldots, g_i^{(B)}$, where g_i^j is equal to g_{i-1} if the j-th element of the i-th block is not in Y, and it is equal to $0^{B^{i-1}}$ if the j-th element of the i-th block is in Y. To construct these symbols we need at most BN rules, because we need $B-1$ concatenation operations to derive g_i from g_{i-1}. (Note that here we cannot get down to $O(N \log B)$ rules – $\Theta(BN)$ seem to be necessary.) The start symbol of the grammar is g_N.

We claim that the string produced by this grammar has the property that $s_Y[X] = 1$ iff $X \cap Y = \emptyset$. This is easy to prove by induction on i, where the induction claim is that for any i, g_i is the string that corresponds to the set $X \cap \{1 \ldots, iB\}$ over the universe $\{1, \ldots, iB\}$. □

Example 2. Consider the values $B = 3$ and $N = 3$. Let $Y = \{1, 3, 5, 9\}$. The string s_Y is "010000010 010000010 000000000"[2]. The locations of the 1's correspond exactly to the blocked sets that don't intersect Y, namely to the sets $\{2, 4, 7\}$, $\{2, 6, 7\}$, $\{2, 4, 8\}$ and $\{2, 6, 8\}$, respectively. A brief illustration for this example is in Figure 1.

$$g_2 = 010000010$$

$$g_3 = 010000010 \quad 010000010 \quad 000000000$$

$$\{7\} \cap Y = \emptyset \quad \{8\} \cap Y = \emptyset \quad \{9\} \cap Y = \{9\}$$

Fig. 1. An illustration of Example 2

3.2 Lower Bounds for SD and BLSD

In this subsection we show lower bounds for SD and BLSD that are implicit in the work of Pătraşcu [8]. Recall the notations from Section 2: in particular, in all of the bounds, w, $\mathcal{S}$, and t denote the word size (measured in bits), the

[2] The spaces are just for easier presentation.

size of the data structure (measured in words) and the query time (measured in number of accesses to words), respectively.

Theorem 1. *For any 2-sided-error data structure for SD_N, $t \geq \Omega(N/(w + \log S))$.*

Note that this theorem does not give strong bounds when $w = O(\log L)$, but it is meaningful for bit-probe ($w = 1$) bound and a warm-up for the reader.

Theorem 2. *Let $\varepsilon > 0$ be any small constant. For any 2-sided-error data structure for $BLSD_{B,N}$,*

$$t \geq \Omega\left(\min\left(\frac{N \log B}{\log \mathcal{S}}, \frac{B^{1-\varepsilon} N}{w}\right)\right) . \tag{1}$$

The proofs follow by standard reductions from data structure to communication complexity, using known lower bounds for SD and BLSD (the latter is one of the main results in [8]).

We now cite the corresponding communication complexity lower bounds:

Lemma 3 (See [2,9,1]). *Consider the communication problem where Alice and Bob each receive a subset of $[N]$, and they want to decide whether the sets are disjoint. Any randomized 2-sided-error protocol for this problem uses communication $\Omega(N)$.*

Lemma 4 (See [8], Lemma 3.1). *Let $\varepsilon > 0$ be any small constant. Consider the communication problem where Bob gets a subset of $[BN]$ and Alice gets a blocked subset of $[BN]$ of cardinality N, and they want to decide whether the sets are disjoint. In any randomized 2-sided-error protocol for this problem, either Alice sends $\Omega(N \log B)$ bits or Bob sends $B^{1-\varepsilon} N$ bits. (The Ω-notation hides a multiplicative constant that depends on ε.)*

The way to prove the data structure lower bounds from the communication lower bounds is by reductions to communication complexity: Alice and Bob execute a data structure query; Alice simulates the querier, and Bob simulates the data structure. Alice notifies Bob which cell she would like to access; Bob returns that cell, and they continue for t rounds, which correspond to the t probes. At the end of this process, Alice knows the answer to the query. Overall, Alice sends $t \log \mathcal{S}$ bits and Bob sends tw bits. The rest is calculations, which we include here for completeness:

Proof (Lemma 3 $\Rightarrow$ Theorem 1). We know that the players must send a total of $\Omega(N)$ bits, but the data structure implies a protocol where $t \log \mathcal{S} + tw$ bits are communicated. Therefore $t \log \mathcal{S} + tw \geq \Omega(N)$ so $t \geq \Omega(N/(\log \mathcal{S} + w))$. □

Proof (Lemma 4 $\Rightarrow$ Theorem 2). We know that either Alice sends $\Omega(N \log B)$ bits or Bob sends $B^{1-\varepsilon} N$ bits. Therefore, either $t \log \mathcal{S} \geq \Omega(N \log B)$ or $tw \geq B^{1-\varepsilon} N$. The conclusion follows easily. □

3.3 Putting It Together

We now put the results of Section 3.1 and 3.2 together to get our lower bounds. Note that in all lower bounds below we freely set the relation of n and L in any way that gives the best lower bounds. Therefore, if one is interested in only a specific relation of n and L (say $L = n^{10}$) the lower bounds below are not guaranteed to hold. The typical "worst" dependence in our lower bounds (at least for the case where $w = \log L$ and $\mathcal{S} = \text{poly}(n)$) is roughly $L = 2^{\sqrt{n}}$.

Theorem 1 together with Lemma 1 immediately give:

Theorem 3. *For any 2-sided-error data structure for the grammar random access problem, $t \geq \Omega(n/(w+\log \mathcal{S}))$. And in terms of L, $t \geq \Omega(\log L/(w+\log \mathcal{S}))$.*

When setting $w = 1$ and $\mathcal{S} = poly(n)$ (polynomial space in the bit-probe model), we get that $t \geq \Omega(n/\log n)$. And in terms of L, $t \geq \Omega(\log L/\log\log L)$.

Proof. Trivial, since $n = \Theta(N)$ and $L = 2^{\Theta(N)}$. □

Theorem 2 together with Lemma 2 give:

Theorem 4. *Assume $w = \omega(\log \mathcal{S})$. Let $\varepsilon > 0$ be any arbitrarily small constant. For any 2-sided-error data structure for the grammar random access problem, $t \geq n/w^{\frac{1+\varepsilon}{1-\varepsilon}}$. And in terms of L, $t \geq \frac{\log L}{\log \mathcal{S} \cdot w^{\frac{\varepsilon}{1-\varepsilon}}}$.*

When setting $w = \log L$ and $\mathcal{S} = \text{poly}(n)$ (polynomial space in the cell-probe model with cells of size $\log L$), there is another constant δ such that $t \geq n^{1/2-\delta}$. And in terms of L, $t \geq (\log L)^{1-\delta}$.

The condition $w = \omega(\log \mathcal{S})$ is a technical condition, which ensures that the value of B we choose in the proof is at least $\omega(1)$. For $w \leq \log \mathcal{S}$ one gets the best results just by reducing from SD, as in Theorem 3.

Proof. For the first part of the theorem, substitute $B = (w/\log \mathcal{S})^{1/(1-\varepsilon)} \log(w/\mathcal{S})$, $N = n/B$, $L = B^N$ into (1). For the second part of the theorem, substitute $N = \frac{B^{1-\varepsilon} \log n}{\log^2 B}$, $n = BN$ and $L = B^N$. And for the result, set $\delta = \frac{2\varepsilon}{1-\varepsilon}$. □

4 Lower Bound for Less-Compressible Strings

In the above reduction, the worst case came from strings that can be compressed superpolynomially. However, for many strings we expect to encounter in practice, superpolynomial compression is unrealistic. A more realistic range is polynomial compression or less. In this section we discuss the special case of strings of length $O(n^{1+\epsilon})$. We show that for this class of strings, the Bille et al. [3] result is also (almost) tight by proving an $\Omega(\log n/\log\log n)$ lower bound on the query time, when the space used is $O(n \cdot \text{polylog}\, n)$. This is done by reduction from the range counting problem on a 2D grid. We have the following lower bound for the range counting problem (see Definition 4 for details). Due to lack of space, we omit the proof. A similar proof for a problem with slightly different parameters could be found in [8, Section 2.1+Appendix A].

Lemma 5. *Any data structure for the 2D range counting problem for n points on a grid of size $[n] \times [n^{\varepsilon}]$ using $O(n \operatorname{polylog} n)$ space requires $\Omega(\log n / \log\log n)$ query time in the cell probe model with cell size $\log n$.*

Recall that the version of range counting we consider is actually dominance counting modulo 2 on the $n \times n^{\epsilon}$ grid. The main idea behind our reduction is to consider the length-$n^{1+\epsilon}$ binary string consisting of the answers to all $n^{1+\epsilon}$ possible dominance range queries (in the natural order, i.e. row-by-row, and in each row from left to right); call this the *answer string* of the corresponding range counting instance. We prove that the answer string can be derived using a grammar of size $O(n \log n)$. The reduction follows obviously, since a dominance prange query can be answered by querying one bit of the answer string.

Lemma 6. *For any range counting problem in 2D, the answer string can be derived by a grammar of size $O(n \log n)$.*

The idea behind the proof of is to simulate a sweep of the point set from top to bottom by a dynamic one-dimensional range tree. The symbols of the grammar will correspond to the nodes of the tree. With each new point encountered, only $2 \log n$ new symbols will be introduced. Since there are n points, the grammar is of size $O(n \log n)$.

Proof. Assume w.l.o.g.p that n is a power of 2. It is easy to see that the answer string could be built by concatenating the answers in a row-wise order, just as illustrated in Figure 2.

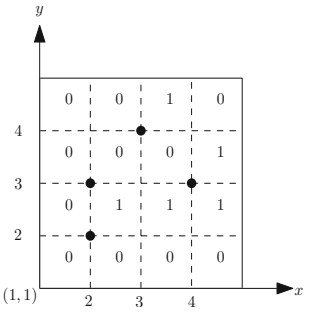
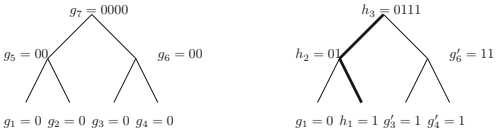


Fig. 2. The answer string for this instance is 0000 0111 0001 0010. The value in the grids are the query results for queries falling in the corresponding cell, including the bottom and left boundaries, excluding the right and top boundaries.

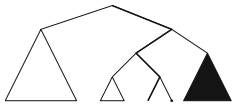
We are going to build the string row by row. Think of a binary tree representing the grammar built for the first row of the input. The root of the tree derives the first row of the answer string, whose two children respectively represent the answer string for the left and the right half of the row. In this way the tree is built recursively. The leaves of the tree are terminal symbols from $\{0, 1\}$. Thus there are $2n - 1$ symbols in total for the whole tree. At the same time we also

maintain the *negations* of the symbols in the tree, i.e., making a new symbol g'_i for each symbol g_i in the tree, where $g'_i = 1 - g_i$ if g_i is a terminal symbol; or $g'_i = g'_j g'_k$ if $g_i = g_j g_k$.

The next row in the answer string will be derived by changing at most $2p \log n$ symbols in the grammar of the previous row, where p is the number of new points in the row. We process the new points one by one. For each point, the new symbols needed all lie in a path from a leaf to the root of the tree. Assuming the update introduced by the point is the path $h_1, h_2, \ldots, h_{\log n}$, the new tree will contain an update of $h_1, h_2, \ldots, h_{\log n}$. Also, all the right children of these nodes will be switched to their negations (this switching step does not actually require introducing any new symbols). An intuitive picture of the process is given in Figure 3. The first row has a grammar $g_7 = g_5 g_6$, $g_5 = g_1 g_2$, $g_6 = g_3 g_4$ and $g_1 = g_2 = g_3 = g_4 = 0$, as well as rules for g'_i when $1 \leq i \leq 7$. The second row has a grammar with new rules $h_3 = h_2 g'_6$, $h_2 = g_1 h_1$, $g'_3 = g'_4 = h_1 = 1$.



(a) The trees built for the first and second rows for the example in Figure 2.



(b) The general process illustrated by picture. The black parts stands for the negations of corresponding symbols.

Fig. 3. Examples for building answer strings

It is easy to see for each new point, $2 \log n$ additional rules are created. $\log n$ of them are the new symbols $(h_1, \ldots, h_{\log n})$, and another $\log n$ of them are their negations $(h'_1, \ldots, h'_{\log n})$. After all, we use $2(2n-1) + n \cdot 2 \log n = O(n \log n)$ symbols to derive the whole answer string. □

By using the above lemma, we have the lower bound of the grammar random access problem when $L = n^{1+\epsilon}$.

Theorem 5. *Fix $\epsilon > 0$, any data structure using space $O(n \operatorname{polylog} n)$ for the grammar random access problem with n rules on strings of length $\Omega(n^{1+\epsilon})$ requires $\Omega(\log n / \log \log n)$ query time.*

Proof. For inputs of the range counting problem, we compress the answer string to a grammar of size $O(n \log n)$ according to Lemma 6. After that we build a data structure for the random access problem on this grammar using Lemma 8. For any query (x, y) of the range counting problem, we simply pass the query result on the index $(y-1)n + x - 1$ on the answer string as an answer. Assuming there is a data structure using $O(n \log n)$ space and query time t, then it will also solve the range counting problem. According to Lemma 5 the lower bound for range counting is $\Omega(\log n / \log \log n)$ for $O(n \operatorname{polylog} n)$ space, thus $t = \Omega(\log n / \log \log n)$. □

Note that natural attempt is to replace the 1D range tree that we used above by a 2D range tree and perform a similar sweep procedure, but this does not work for building higher dimensional answer strings.

5 LZ-Based Compression

In this section we discuss about what the lower bound means for LZ-based compression, which is a typical case for grammar-based compression by Lempel-Ziv [11,6]. First we look at LZ77. For LZ77 we have the following lemma.

Lemma 7 (Lemma 9 of [4]). *The length of the LZ77 sequence for a string is a lower bound on the size of the smallest grammar for that string.*

The basic idea of this lemma is to show that each rule in the grammar only contribute one entry for LZ77. Since LZ77 could compress any string with small grammar size into a smaller size, it can also compress the string s_Y in Lemma 1 and the answer string in Theorem 5 into a smaller size. Thus the both lower bounds for grammar random access problem also holds for LZ77.

The reader might also be curious about what will happen for the LZ78 [12] case. Unfortunately the lower bound does not hold for LZ78. This is because LZ78 is a "bad" compression scheme that even the input is 0^n of all 0's, LZ78 can only compress the string to length of $\sqrt{n}$. But a random access on an all 0 string is trivially constant with constant space. So we are not able to have any lower bounds for this case.

6 Optimality

In this section, we show that the upper bound in Bille et al. [3] is nearly optimal, for two reasons. First, it is clear that by Theorem 5, the upper bound in Lemma 8 is (almost) optimal, when the space used is $O(n \operatorname{polylog} n)$.

Lemma 8. *There is a data structure for the grammar random access problem with $O(n)$ space and $O(\log L)$ time. This data structure works in the word RAM with words of size $\log L$.*

Second, in the cell-probe model with words of size $\log L$ we also have the following lemma by Bille et al. [3].

Lemma 9. *There is a data structure for the grammar random access problem with $O(n)$ space and $O(n \log n / \log L)$ time.*

Proof. This is a trivial bound. The number of bits to encode the grammar is $O(n \log n)$ since each rule needs $O(\log n)$ bits. The cell size is $O(\log L)$, so in $O(n \log n / \log L)$ time the querier can just read all of the grammar. Since computation is free in the cell-probe model, the querier can get the answer immediately. □

Thus, by using Lemma 8 when $n = \Omega(\log^2 L/\log\log L)$ and Lemma 9 in the case $n = O(\log^2 L/\log\log L)$, we have the following corollary. This corollary implies that our lower bound of $\Omega(n^{1/2-\varepsilon})$ is nearly the best one can hope for in the cell-probe model.

Corollary 1. *Assuming $w = \log L$, there is a data structure in the cell-probe model with space $O(n)$ and time $O(\sqrt{n \log n})$.*

Acknowledgement. We thank Travis Gagie and Pawel Gawrychowski for helpful discussions.

References

1. Babai, L., Frankl, P., Simon, J.: Complexity classes in communication complexity theory. In: FOCS, pp. 337–347 (1985)
2. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. Journal of Computer and System Sciences 68(4), 702–732 (2004)
3. Bille, P., Landau, G.M., Raman, R., Rao, S., Sadakane, K., Weimann, O.: Random access to grammar compressed strings. In: SODA (2011)
4. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Transactions on Information Theory 51(7), 2554–2576 (2005)
5. Claude, F., Navarro, G.: Self-indexed text compression using straight-line programs. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 235–246. Springer, Heidelberg (2009)
6. Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Transactions on Information Theory 22(1), 75–81 (1976)
7. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. In: STOC, p. 111. ACM (1995)
8. Patrascu, M.: Unifying the Landscape of Cell-Probe Lower Bounds. SIAM Journal on Computing 40(3) (2011)
9. Razborov, A.A.: On the distributional complexity of disjointness. Theoretical Computer Science 106(2), 385–390 (1992)
10. Yao, A.C.C.: Should tables be sorted? Journal of the ACM 28(3), 615–628 (1981)
11. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on information theory 23(3), 337–343 (1977)
12. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory 24(5), 530–536 (1978)