PREPRINT. In Proc. 2nd Annual European Symposium on Algorithms, pages 82-93. Springer Verlag, 1994.

Faster Searching in Tries and Quadtrees— An Analysis of Level Compression*

Arne Andersson Stefan Nilsson
Department of Computer Science, Lund University,
Box 118, S-221 00 Lund, Sweden

Abstract

We analyze the behavior of the level-compressed trie, LC-trie, a compact version of the standard trie data structure. Based on this analysis, we argue that level compression improves the performance of both tries and quadtrees considerably in many practical situations. In particular, we show that LC-tries can be of great use for string searching in compressed text.

Both tries and quadtrees are extensively used and much effort has been spent obtaining detailed analyses. Since the LC-trie performs significantly better than standard tries, for a large class of common distributions, while still being easy to implement, we believe that the LC-trie is a strong candidate for inclusion in the standard repertoire of basic data structures.

1 Introduction

A fundamental, well known, and very well studied technique for storing and retrieving data is to use *tries* [9, 10, 14]. In its original form, the trie is a data structure where a set of strings from an alphabet containing m symbols is stored in a natural way in an m-ary tree where each string corresponds to a unique path. This data structure is used in many different settings, including string matching, approximate string matching, compression schemes, and genetic sequences. The results presented in this paper also apply to *quadtrees* [17], which are widely used in computer graphics, image processing, geographic information systems, and robotics.

The expected average depth of a trie containing n independent random strings is $\Theta(\log n)$ [5]. A common method to decrease the size of a trie is to use a *path compression* method known as Patricia compression. A path compressed trie is often called a *Patricia tree* [10]. However, this compression technique does not give an asymptotic improvement. The expected average depth is still $\Theta(\log n)$ [13, 14].

We recently proposed another compression method [3]. The main idea is that the i highest complete levels of a trie can be replaced—without losing any relevant information—by a single node of degree m^i, the replacement being made top-down. This data structure is called a *level-compressed trie*, or *LC-trie* and it performs much better than a conventional trie. The expected average depth of an LC-trie is $\Theta(\log^* n)$ for uniformly distributed data.[1] We also want to point out that for a reasonable machine model the time spent at each node in an LC-trie will be constant and hence the decrease in average depth reflects a real improvement in search time in practical applications.

*Published in *Proc. of Second Annual European Symp. on Algorithms*, pp. 82–93, 1994.

[1] The function $\log^* n$ is the iterated logarithm function, which is defined as follows. $\log^* 1 = 1$. For any $n > 1$, $\log^* n = 1 + \log^*(\lceil \log n \rceil)$.

The trie has been the subject of thorough theoretic analysis [5, 14, 15, 16, 19]; an extensive list of references can be found in Handbook of theoretical computer science [19]. Using simple analytic methods we show how level compression improves the search time in a trie for a wide variety of common statistical models.

When analyzing the behavior of trie structures used for text retrieval, it is common to assume that the input consists of independent random strings from a Bernoulli-type process [7, 8]. For this kind of input, the expected search cost in an LC-trie will be $O(\log\log n)$ which is significantly better than $\Theta(\log n)$, achieved by the conventional trie. Another frequently used model, particularly pertinent when studying quadtrees, is independent random variables with common density. We show that for a large class of densities, the expected search cost of a level-compressed trie structure is $\Theta(\log^* n)$. Once again, this complexity is significantly better than $\Theta(\log n)$, achieved by a conventional trie or quadtree.

The LC-trie does not only have a good asymptotic behavior, it is also easy to implement, and it can be used in a wide range of different applications. We therefore consider the LC-trie to be a strong candidate for inclusion in the standard repertoire of basic data structures.

2 Preliminary definitions

To keep the notation simple we will first and foremost consider binary strings and binary trie structures. The generalization to m-ary strings is straightforward. We say that a string v of length i is the *i-prefix* of a string u if there is a string w such that $u = vw$. The string w is called the *i-suffix* of u. To make things rigorous we give the following definition of a (binary) trie:

Definition 1 *A* trie *containing n strings is*

if $n = 0$: an empty leaf;

if $n = 1$: a leaf containing the string;

if $n > 1$: an internal node of degree 2. For each character b there is a child, which is a trie, containing the 1-suffixes of all strings starting with b.

Note that the strings are assumed to be prefix-free, i.e. no string is a prefix of another string. In particular this implies that there can't be any duplicate strings. However, if all strings to be stored in the trie are unique, it is easy to ensure that the strings are prefix-free by using a special string terminator. Instead of using a special terminator we can append the string "100..." to the end of each string.

Path compression, or Patricia compression, is the most common method to decrease the size of a trie. A trie where all internal nodes with only one child has been removed is called a *Patricia tree*. At each internal node of the Patricia tree an index is used to indicate the character used for branching at this node. Observe that it is not possible, in general, to retrieve the full strings from a Patricia tree; some information is lost when internal nodes are removed. Therefore it is often necessary to use an auxiliary data structure to store the complete strings. As pointed out above the Patricia tree does not offer an asymptotic improvement in search time as compared to the plain trie. In practice, however, it is an important size reduction technique [4].

Another natural way to decrease the search cost in a trie is to use more than one digit for branching. In this way we obtain a *multi-digit trie* as defined below.

Definition 2 *A* multi-digit *trie containing n strings is*

if $n = 0$: an empty leaf;

if $n = 1$: a leaf containing the string;

if $n > 1$: an internal node of degree 2^i, $i \geq 1$. For each possible i-prefix v there is a child, which is a multi-digit trie, containing the i-suffixes of all strings starting with v.

The next step is to chose the outdegrees efficiently. On the one hand we want the outdegree to be as large as possible, on the other hand we do not want to introduce a large number of superfluous leaves.

Definition 3 *A* level-compressed trie, LC-trie, *is a multi-digit trie with the following properties:*

- *the degree of the root is 2^i, where i is the smallest number such that at least one of the children becomes a leaf;*
- *each child is an LC-trie.*

The LC-trie may be viewed as a binary trie where the i highest complete levels are replaced by a single node of degree 2^i; the replacement being made top-down. Path compression and level compression can be combined. We will consider the case where we first apply level compression and then path compression to the trie.

Definition 4 *A* path compressed LC-trie *is a trie to which first level compression and then path compression has been applied.*

This is a straightforward procedure; each node in the LC-trie that only has one child is removed and an index is used at each node to indicate the bit used for branching.

Let $X_1, \ldots, X_n$ be infinite binary strings. We will study the trie formed by the smallest prefixes of $X_1, \ldots, X_n$ that are pairwise different. Let D_{ni} be the depth of X_i in this trie. The average depth is defined as

$$A_n = \frac{1}{n} \sum_{i=1}^{n} D_{ni}.$$

This parameter is proportional to the average successful search time. Let

$$A_n^{\text{trie}}, \; A_n^{\text{LC}}, \text{ and } A_n^{\text{path-LC}}$$

be the average depth in a trie, an LC-trie, and a path compressed LC-trie, respectively.

3 Independent strings from a Bernoulli-type process

Since tries are typically used for storing textual data, it is natural to consider input from a Bernoulli-type process. This model can be considered a first approximation of textual data, where we disregard all dependencies between letters and only consider the different probabilities of the letters. The behavior of a trie under this model has been thoroughly studied [15, 16]. In this section we show that the LC-trie performs significantly better. The expected average depth of an LC-trie is shown to be $O(\log \log n)$, which should be compared to $\Theta(\log n)$ for a trie.

Consider independent binary strings from a Bernoulli-type process, i.e. each string $v = (v_i)_{i \geq 1}$ is a sequence of independent identical random variables with values in the alphabet $S = \{0, 1\}$, $P(v_i = s) = p(s) > 0$, $s \in S$, $i \geq 1$.

Theorem 1 *For an independent random sample from a Bernoulli-type process, with character probabilities $p(s)$ not all equal we have $E(A_n^{LC}) = \Theta(\log\log n)$. If the probabilities are equal $E(A_n^{LC}) = \Theta(\log^* n)$.*

Proof: If all characters have the same probability the strings will be uniformly distributed and hence $E(A_n^{\mathrm{LC}}) = \Theta(\log^* n)$ as was shown in our original paper [3].

Now consider the case where $\alpha = min_{s\in S} p(s) < 1/|S|$. Let $k = -\log n/\log\alpha$, and let q be a number such that $0 < q < 1$. Without loss of generality we can assume that $p(0) = \alpha$. We will give a lower bound for the probability $P_{\mathrm{all}}(n)$ that all possible strings of length qk will occur at least once as a prefix among the n strings. It can be shown that $P_{\mathrm{all}}(n)$ is larger than the probability that the string consisting of qk zeroes occurs at least 2^{qk} times among the prefixes of the strings. In fact, to prove this we may assume without loss of generality that the probabilities of the prefixes are equal. Now consider all possible sequences of length n of prefixes of length qk. We just need to check that the number of different sequences that contain at least 2^{qk} prefixes consisting of only zeroes is not greater than the number of sequences that contain at least one copy of each possible prefix. This is a straightforward exercise in combinatorics; the details are omitted. Let A be the number of prefixes consisting of qk zeroes among the n input strings.

$$P_{\mathrm{all}}(n) \geq P(A \geq 2^{qk}) = P(A \geq 2^{-q\frac{\log n}{\log\alpha}}) = P(A \geq n^{-(1+q(\frac{1}{\log\alpha}-1))}n^{1-q}).$$

A obeys the binomial distribution $B(n, \alpha^{qk}) = B(n, n^{-q})$. And in particular $E(A) = n^{1-q}$. Using the following Chernoff bound

$$P(A \geq (1-\epsilon)E(A)) \geq 1 - e^{-\epsilon^2 E(A)/2}$$

we see that $P_{\mathrm{all}}(n) \to 1$ as $n \to \infty$ if $1+q(\frac{1}{\log\alpha}-1) > 0$, or equivalently $q < \frac{\log\alpha}{\log\alpha - 1}$. In particular, $P_{\mathrm{all}}(n) \to 1$ for all α such that $0 < \alpha < 1/2$, if $q < 1/2$. Clearly $E(A_n^{\mathrm{LC}}) \leq E(A_n^{\mathrm{trie}})$, but $E(A_n^{\mathrm{trie}}) = O(\log n)$ [16] and hence we get the following recursion inequality for $D(n) = E(A_n^{\mathrm{LC}})$.

$$D(n) \leq 1 + P_{\mathrm{all}}(n) \cdot D(n/2^{qk}) + (1 - P_{\mathrm{all}}(n)) \cdot O(\log n)$$

It is easy to check that for $q = 1/4$, $(1 - P_{\mathrm{all}}(n))\log n \to 0$ as $n \to \infty$ and hence

$$D(n) \leq O(1) + D(n/2^{qk}) = O(1) + D(n^{1+q/\log\alpha}).$$

But $-1/4 \leq q/\log\alpha < 0$ and hence $E(A_n^{\mathrm{LC}}) = O(\log\log n)$.

The lower bound can be established in a similar way. Choose α as above and let Q be a number such that $1 < Q < -\log\alpha$. Denote by $P_{\mathrm{none}}(n)$ the probability that there will be no string with a prefix consisting of Qk zeroes among the n strings. Assuming that $min_{s\in S} p(s) < 1/|S|$ we make the following two observations. $P_{\mathrm{none}}(n) = \left(1-\alpha^{Qk}\right)^n = \left(1-\frac{1}{n^Q}\right)^n \to 1$ as $n \to \infty$ and $n/2^{Qk} = n^{1+Q/\log\alpha}$, where $-1 < Q/\log\alpha < 0$. We have the following inequality

$$D(n) \geq 1 + (1 - P_{\mathrm{none}}(n)) \cdot 0 + P_{\mathrm{none}}(n) \cdot D\left(\frac{n}{2^{Qk}}\right)$$

and hence $E(A_n^{\mathrm{LC}}) = \Omega(\log\log n)$ in this case. □

Observe that the assumption that the probability of each character is positive is necessary. For example, consider a ternary alphabet a, b, c where $P(a) = 0$. No level compression will take place in this case since every node in the trie will have at most two children.

3.1 Compressed strings

Pattern matching in compressed text documents is a new and interesting problem that has been investigated by Amir et al. [1, 2, 6]. In this section we argue that the LC-trie could be particularly useful for this kind of applications. We show that under certain assumptions the LC-trie actually behaves better when the input strings have been compressed. Finally, we indicate how this fact could be used in a database application.

Most modern model-based compression schemes use either Huffman coding [12] och Arithmetic coding [20] to code a message with respect to a statistical model of the text. Arithmetic coding is the more efficient of the two. The only drawback being that it is slightly more difficult to implement.

In arithmetic coding a message is represented by a subinterval of $[0, 1)$. Successive characters of the text reduce the size of the interval according to the character probability generated by the statistical model. Consider the following example. The text contains three different characters a, b, and c with fixed probabilities 0.2, 0.3, and 0.5, respectively. We want to encode the text *cab*. To each of the characters we assign an interval whose size is proportional to the probabilities. In this example the intervals $[0, 0.2)$, $[0.2, 0.5)$, and $[0.5, 1]$ are assigned to the characters a, b, and c, respectively. After the first character (c) has been processed the interval is reduced to $[0.5, 1)$, the second character (a) reduces the interval to $[0.5, 0.6)$, and after the third character (b) we get $[0.52, 0.55)$.

Theorem 2 *For an independent random sample from a Bernoulli-type process where each string has been arithmetically coded,* $E(A_n^{LC}) = O(\log^* n)$.

Proof: The strings from the Bernoulli-type process will be uniformly distributed after having been arithmetically coded and hence the result follows immediately from Theorem 4. In fact, consider any subinterval I of $[0, 1)$ and an arithmetically coded infinite string s from a Bernoulli-type process. It follows from the definition of arithmetic coding that $P(s \in I)$ equals the size of I. □

Huffman coding also has the effect of smoothing the input, but to a lesser degree.

Theorem 3 *For an independent random sample from a Bernoulli-type process where each string has been Huffmann-coded,* $E(A_n^{LC}) = O(\log \log n)$.

Proof: In this case the distribution will not be uniform and the bits will not be independent. The probability of a bit being either 0 or 1 depends on the preceding bits. However, we can make the following observation. At each node r of the Huffman tree there will be a positive probability $P_r(0)$ that the next character occurring will be 0. In fact, $P_r(0) \geq \min_{s \in S} P(s)$, since $P_r(0)$ is the sum of the weights of the leaves in the left subtree of r. Similarly, $P_r(1) \geq \min_{s \in S} P(s)$. The theorem can now be proved using the same technique as in the proof of Theorem 1. □

As a possible application we mention a large dictionary where each entry, starting with its keyword, is coded separately by arithmetic coding. This dictionary, together with an LC-trie representing the coded keywords, would require less space that the uncompressed document and the LC-trie would support fast location of keywords. When a keyword is found, the appropriate article could be decoded separately and presented to the user.

4 Independent strings with common density

Independent random strings with common density is perhaps the most frequently used statistical model and for many applications this model will be more natural

than a Bernoulli-type process. For example, geometric data can often be accurately modelled in this way and, as pointed out above, both path compression and level compression can be applied not only to tries by also to quadtrees, a data structure that is frequently used in applications that handle geometric data.

Assume that $X_1, \ldots, X_n$ are the binary representations of independent random variables with common density f on $[0, 1)$. The behavior of binary tries under this model has been analyzed by Devroye [5]. If the density $f \in L^2$, i.e. $\int_{x=0}^{1} f^2(x)\,dx < \infty$, then $E(A_n^{\text{trie}}) = O(\log n)$ else $E(A_n^{\text{trie}}) = \infty$ for all $n \geq 2$. Even for LC-tries the case $f \notin L^2$ is intractable, as seen by the following observation.

Observation 1 *For an independent random sample of size $n \geq 2$ taken from a distribution with density $f(x) \notin L^2$, $0 \leq x \leq 1$, $E(A_n^{LC}) = \infty$.*

Proof: We prove this using a simple relation between LC-tries and tries. In a trie, consider an element x with depth $n+h$. On the path of x there are at least h nodes that have an empty child. Hence, in the LC-trie containing the same set, the depth of x will be at least h. From this follows that the average depth of the elements in an LC-trie is not less than the average depth in a trie minus n. The result now follows since the expected depth of an element in a trie is infinite if $f \notin L^2$. □

We also note that the depth of an LC-trie will never be greater than that of a conventional trie. In fact, for a large number of densities the expected average depth will be much smaller as will be shown in the following theorems.

Theorem 4 *For an independent random sample taken from a distribution with density $f(x) \in L^2$, $0 < \alpha \leq f(x) \leq \beta$, $0 \leq x \leq 1$, $E(A_n^{LC}) = \Theta(\log^* n)$.*

Proof: We first consider the case $\alpha = \beta = 1$.

We choose the indices in such a way that $X_1 < \cdots < X_n$, and write $X_0 = 0$ and $X_{n+1} = 1$ and consider the random variable

$$M_n = \max_{0 \leq i \leq n} (X_{i+1} - X_i),$$

the maximal spacing between adjacent elements in the sample. We will use the following lemma by Slud [18]:

Lemma 1 *If δ is a positive constant, then*

$$P(|nM_n / \ln n - 1| > \delta) = O(n^{-\delta}).$$

First we determine an upper bound. We will show that the degree at the root is $\Omega(n/\log n)$ with high probability. This also holds true at the lower levels of the trie, since the elements in a subtree are a random sample of independently chosen points in an interval of some length 2^{-j}; thus, the analysis applies with due alteration of details.

Let x be an element and let T_x denote the number of elements that are stored in the same subtrie as x. Choose the integer i such that $\frac{2 \ln n}{n} < 2^{-i} \leq \frac{4 \ln n}{n}$ and split the interval $[0, 1)$ into subintervals of size 2^{-i}. Let I denote the subinterval into which x falls. Let y be any of the other $n-1$ elements and let p be the probability that y falls into I. Then, $p = 2^{-i} \leq \frac{4 \ln n}{n}$.

In the expected case, the effect of choosing i like this will be that $M_n \leq 2^{-i}$. Therefore every subinterval of size 2^{1-i} will contain at least two elements and hence the number of children of the root will be at least 2^i. This implies that the elements that are stored in the same subtrie as x will be a subset of the elements in I. From this we will deduce that for some constant C, $P(T_x > C \ln n)$ is small. This is

done by studying the random variables M_n and n_I, where n_I denotes the number of elements (not counting x) falling into I.

Clearly, n_I obeys the binomial distribution $B(n-1,p)$. Let $C = 2e$. We use a Chernoff bound [11] to get a tail estimate for the binomial function.

$$P(n_I > C \ln n) \leq e^{-(n-1)p} \left(\frac{e(n-1)p}{C \ln n} \right)^{C \ln n} = O(n^{-1}) \tag{1}$$

If $M_n \leq 2^{-i}$ then, as explained above, $T_x \leq n_I$. In particular

$$M_n \leq 2^{-i} \text{ and } n_I \leq C \ln n \Rightarrow T_x \leq C \ln n$$

and hence

$$P(M_n \leq 2^{-i} \text{ and } n_I \leq C \ln n) \leq P(T_x \leq C \ln n). \tag{2}$$

From Lemma 1 we have

$$P\left(M_n > \frac{\ln n}{n} (1+\delta) \right) = O(n^{-\delta}).$$

Hence

$$P(M_n > 2^{-i}) = P\left(M_n > \frac{\ln n}{n} \left(1 + \frac{n 2^{-i}}{\ln n} - 1 \right) \right) = O(n^{-\varepsilon}), \tag{3}$$

where $\varepsilon = \frac{n 2^{-i}}{\ln n} - 1 \geq 1$.

Combining these results, we get

$$\begin{aligned} P(T_x > C \ln n) &\leq P(\neg(M_n \leq 2^{-i} \text{ and } n_I \leq C \ln n)) \\ &\leq P(M_n > 2^{-i}) + P(n_I > C \ln n) \\ &= O(n^{-1}). \end{aligned} \tag{4}$$

The depth of an element in an LC-trie is never larger than that of the corresponding binary trie. It has been shown by Devroye [5] that the expected average depth of a binary trie is $O(\log n)$. Therefore, we can use the estimate $O(\log n)$ for the expected depth of an element in an LC-trie when $T_x > C \ln n$.

We inductively assert that the expected depth $E(A_n^{\mathrm{LC}})$ can be bounded by

$$E(A_n^{\mathrm{LC}}) \leq D \log^* n \tag{5}$$

for some constant D. This is obviously true for $n = 1$. For $n \geq 2$ there is a constant D' such that

$$\begin{aligned} E(A_n^{\mathrm{LC}}) &\leq 1 + \sum_{m=1}^{C \ln n} P(T_x = m) \cdot E(A_m^{\mathrm{LC}}) + P(T_x > C \ln n) \cdot O(\log n) \\ &\leq 1 + \sum_{m=1}^{C \ln n} P(T_x = m) \cdot D \log^*(C \ln n) + O(1) \\ &\leq D(\log^*(\log n)) + D'. \end{aligned} \tag{6}$$

It can be arranged that $D' \leq D$. Then the last expression can be bounded from above by $D(\log^* n - 1) + D' \leq D \log^* n$ and hence $E(A_n^{\mathrm{LC}}) = O(\log^* n)$.

The lower bound can be established in a similar way. By choosing the integer i such that $\frac{\ln n}{4n} < 2^{-i} \leq \frac{\ln n}{2n}$ we can ensure that $P(T_x \leq c \ln n) \cdot E(A_{c \ln n}^{\mathrm{LC}}) = o(1)$ for some positive constant c and the result follows by an induction argument similar to the the one in the upper bound analysis.

Now consider the case $0 < \alpha < \beta < \infty$. We make the following observations:

- Let Δ be an interval of size $|\Delta|$ and let n_Δ be the expected number of elements falling into Δ. Then $n_\Delta \in \mathrm{Bin}(n,p)$, where $p \leq \beta|\Delta|$.
- Let M_n^f be the maximum spacing for data taken from f and let M_n^u be the maximum spacing for data from the uniform distribution over the interval $[0,1)$, then $P(M_n^f > x) \leq P(M_n^u > \alpha x)$. This can be shown in the following way: Consider the density $g(x) = \alpha$, $0 \leq x \leq 1/\alpha$. Then, $P(M_n^g > x) = P(M_n^u > \alpha x)$. The fact now follows since the domain of f is a subinterval of the domain of g and $f(x) \geq g(x)$ when $x \in [0,1)$.

Using the first observation we see that Eq. 1 holds if C is chosen large enough. If we make the interval I larger by a factor of $\frac{1}{\alpha}$, i. e. we choose i such that $\frac{\ln n}{\alpha n} < 2^{-i} \leq \frac{2\ln n}{\alpha n}$, the second observation gives

$$\begin{aligned} P(M_n^f > 2^{-i}) &\leq P(M_n^u > \alpha 2^{-i}) \\ &= P\left(M_n^u > \frac{\ln n}{n}\left(1 + \frac{\alpha n 2^{-i}}{\ln n} - 1\right)\right) \\ &= O(n^{-\varepsilon}), \end{aligned}$$

where $\varepsilon = \frac{\alpha n 2^{-i}}{\ln n} - 1 > 0$. Hence, equation 6 still holds if we choose C large enough ($C = 2\beta e/\alpha$ will do) and we conclude that $E(A_n^{\mathrm{LC}}) = O(\log^* n)$ in this case also.

The lower bound can be established using the same technique. □

We now show that if path compression is applied to the LC-trie, the class of densities for which the expected search time is $O(\log^* n)$ can be further extended. Using this technique, we can efficiently handle a class of densities that is zero on a finite number of intervals as proved in the following lemma and theorem.

Lemma 2 *Given a constant r, $0 \leq r \leq 1$, and a density $f(x) \in L^2$,*

$$\begin{cases} 0 < \alpha \leq f(x) \leq \beta, & 0 \leq x \leq r \\ f(x) = 0 & \textit{otherwise.} \end{cases}$$

For an independent random sample taken from the distribution with density $f(x)$, $0 \leq x \leq 1$, $E(A_n^{\textit{path-LC}}) = O(\log^* n)$.

Proof: We say that a trie *covers* an interval if all elements that fall in this interval is stored in that trie. Let L be an LC-trie containing n elements from the distribution in the lemma. For the purpose of this proof we construct a trie T containing the same elements as L in the following way: If the interval covered by T contains r the root is made binary even if we could afford a higher degree. Otherwise, we choose the maximal branching factor according to Definition 3. This construction is repeated in the subtries.

It is not hard to show [3] that the external path length of T will not be smaller than the external path length of L. Hence, we will be done if we can prove that the expected average depth of T is $O(\log^* n)$.

Since we use Patricia compression the root of T has two nonempty subtries T_{left} and T_{right}. If r is contained in any of the intervals covered by these two subtries, it has to be the interval covered by T_{right}. Thus, we have that

1. In the interval covered by T_{left}, f satisfies $0 < \alpha \leq f(x) \leq \beta$. From Theorem 4 follows that this subtrie can be represented by an LC-trie with an expected average depth of $O(\log^* n)$. The expected number of elements in this interval is at least $\frac{\alpha}{2}n$. In particular, the number of elements is more than $\frac{\alpha}{4}n$ with very high probability.

2. In the interval covered by T_{right} we can repeat the analysis recursively.

This gives us the following recursion inequality for $D(n) = E(A_n^{\text{path-LC}})$:

$$\begin{cases} D(n) \leq 2 + \left(1 - \frac{\alpha}{4}\right) D\left(n\left(1 - \frac{\alpha}{4}\right)\right) + \frac{\alpha}{4} O(\log^* n) \\ D(1) = 1 \end{cases}$$

This equation is of the form

$$\begin{cases} T(n) \leq a + bT(bn) + c \log^* n \\ T(1) = 1 \end{cases}$$

where a, b, and c are positive constants, $b < 1$. (The case when $b < 0$ is irrelevant, since in this case the lemma follows immediately). This equation is easily solved:

$$T(n) < (a + c \log^* n)(1 + b + b^2 + b^3 + \ldots) = O(\log^* n).$$

□

Theorem 5 *Let $\delta_1, \ldots, \delta_k$ be a number of disjoint subintervals of the unit interval, such that the length of the shortest interval is positive. Let $f(x) \in L^2$ be a function such that*

$$\begin{cases} f(x) = 0, & x \in \bigcup_{1 \leq i \leq k} \delta_i \\ 0 < \alpha \leq f(x) \leq \beta, & otherwise. \end{cases}$$

If an independent random sample taken from the distribution with density $f(x)$, $0 \leq x \leq 1$, $E(A_n^{path\text{-}LC}) = O(\log^* n)$.

Proof: Let $|\delta|$ be the length of the smallest interval. At each level we have a branching factor of at least two. Therefore, the function f restricted to the interval covered by a subtrie at level $\left\lceil \log \frac{1}{|\delta|} \right\rceil$ will be of the form described in Lemma 2. Thus, the expected average depth of an element will be at most $\left\lceil \log \frac{1}{|\delta|} \right\rceil + O(\log^* n)$. □

5 Conclusions

We note that the methods of path compression and level compression can be used to reduce the search time in two of the most prominent data structures, the trie and the quadtree.

The trie is perhaps most widely used in text processing applications, such as string matching, approximate string matching, and compression schemes. But it has also been used in algorithms for genetic sequences. The Bernoulli-type process is often a good first approximation for text and genetic sequences and since Theorem 3 shows that we get an asymptotic reduction in search time for data from this model, we conjecture that the LC-trie should give a significant improvement in many of these settings.

We also observe that the assumptions of Theorem 5 are likely to be met in many geometrical applications and hence the methods of path compression and level compression should give significant improvements in many applications in computer graphics, image processing, geographic information systems, and robotics, since quadtrees are often an integral part of these systems.

Furthermore, the LC-trie seems to behave even better for compressed strings as shown in Theorems 2 and 3. Consequently the LC-trie should be of interest in the recently initiated study of search algorithms for compressed text.

References

[1] A. Amir, G. Benson, and M. Farach. Efficient pattern matching with scaling. *Journal of Algorithms*, 13(1), 1992.

[2] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: pattern matching in z-compressed files. In *Proc. of 5th Symposium on Discrete Algorithms*, 1994.

[3] A. Andersson and S. Nilsson. Improved behaviour of tries by adaptive branching. *Information Processing Letters*, 46:295–300, 1993.

[4] A. Andersson and S. Nilsson. Efficient implementation of suffix trees. *Software—Practice and Experience*, 25(2):129–141, 1995.

[5] L. Devroye. A note on the average depth of tries. *Computing*, 28:367–371, 1982.

[6] T. Eilam-Tsoreff and U. Vishkin. Matching patterns in a string subject to multilinear transformation. In *Proc. of the International Workshop on Sequences, Combinatorics, Compression, Security and Transmission, Salerno, Italy*, June, 1988.

[7] Ph. Flajolet. On the performace evaluation of extendible hashing and trie searching. *Acta Informatica*, 20:345–369, 1983.

[8] Ph. Flajolet, M. Régnier, and D. Sotteau. Algebraic methods for trie statistics. *Ann. Discrete Math.*, 25:145–188, 1985.

[9] E. H. Fredkin. Trie memory. *Communications of the ACM*, 3:490–500, 1960.

[10] G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 1991.

[11] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Information Processing Letters*, 33(6):305–308, 1990.

[12] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. IRE*, volume 40, pages 1098–1101, 1952.

[13] P. Kirschenhofer and H. Prodinger. Some further results on digital search trees. In *Proc. 13th ICALP*, pages 177–185. Springer Verlag, 1986. Lecture Notes in Computer Science vol. 26.

[14] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

[15] B. Pittel. Asymptotical growth of a class of random trees. *The Annals of Probability*, 13(2):414–427, 1985.

[16] B. Pittel. Paths in a random digital tree: limiting distributions. *Advances in Applied Probability*, 18:139–155, 1986.

[17] H. Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2):187–260, 1984.

[18] E. Slud. Entropy and maximal spacings for random partitions. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 41:341–352, 1978.

[19] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science, vol. A*. Elsevier Science Publishers B.V., 1990. ISBN 0-444-88071-2.

[20] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540K, 1987.