On the Lexicographical Generation of Compressed Codes

Markus E. Nebel

Technische Universität Kaiserslautern, Fachbereich Informatik, Gottlieb-Daimler-Straße, 67663 Kaiserslautern, Germany

Abstract

A certain class of algorithms for the lexicographical generation of combinatorial objects can be considered as working on the code tree representation of the objects processed. Then the strategy used by the algorithms in order to find lexicographical successors corresponds to a special kind of tree traversal. If the encoding used is redundant in the sense that the code tree has nodes with only one successor, compression becomes possible which allows for a speed-up in the lexicographical generation. In this note we analyze the average running time saved when compression is applied. For this purpose we consider random code trees within the model of simply generated trees together with the compression as used for the trie and the PATRICIA data structure. We prove general results which quantify the average savings only depending on the generator Θ and the size of the family under consideration. As an example, those results are applied to consider random encodings over an s-ary alphabet. Finally, we comment on connections of our findings to the problem of ranking words of a given language.

Key words: Analysis of algorithms, combinatorial problems, lexicographical generation

1 Introduction

It is common practice to generate combinatorial objects by using bijective methods, e.g. by encoding the objects in a formal language $\mathcal{L}$ [2]. In such a case we can enumerate all objects of a given size by generating all words in $\mathcal{L}$ lexicographically. In [8] R. Kemp has presented a quite simple class of algorithm for this purpose which allows for a detailed average-case analysis in a unified way. In order to compute the lexicographical successor $\mathsf{ls}(w) \in \mathcal{L}$ of word $w \in \mathcal{L}$ Kemp's algorithms proceed in the following way:

(1) w is scanned from right to left letter by letter until the first symbol of the shortest suffix v with $w = uv$ and $\mathsf{ls}(w) = uv'$,

Email address: nebel@informatik.uni-kl.de (Markus E. Nebel).

$v \neq v'$, has been found.

(2) The new suffix v' of $\mathsf{ls}(w) = uv'$ is computed and attached to the right of u.

One example of such an algorithms is the well-known method of Zaks [17] for generating t-ary ordered trees. Assuming that no knowledge on u is needed in order to compute the suffix v', the running time of such an algorithm is always proportional to the lengths of the suffixes to be changed. Furthermore, assuming that all words to be generated are of the same length (i.e. we are considering a block code) and are equally likely, it turns out that all the s-th moments about the origin of the random variable describing the cost of a lexicographical generation can be expressed by means of $|\mathcal{L}|$ and the number of different prefixes of length k of the words in $\mathcal{L}$ [8]. Beside other applications, this general result allowed for the analysis of the well-known algorithms by Ruskey [16] and Zaks for the lexicographical generation of trees.

In this note we will shed more light on this class of algorithms by taking a different perspective: The set of words $\mathcal{L}$ to be generated lexicographically is assumed to be represented as a code tree with $|\mathcal{L}|$ leaves. Each edge of this tree is marked by a single symbol; we assume a left-to-right ordering of edges according to the order on the alphabet which is used to lexicographically order the language, small edges (symbols) left. Concatenating the symbols on a path from the root to a leaf yields a word in $\mathcal{L}$. Using this representation the above strategy for finding the lexicographical successor of a word $w \in \mathcal{L}$ is emulated by traversing the unique path form w's leaf towards the root up to the first internal node v where the edge e used by our traversal is not the rightmost edge of v. We then traverse the edge right to e heading for the leaves always continuing with the leftmost edge until we reach a leaf. This leaf represents the successor in question. Of course, the algorithms as proposed by Kemp do not work on code trees but have to identify the suffix to be changed by scanning a word from right to left, computing the words new suffix (which makes the word to become the successor) based on the knowledge that was collected during this scanning. However, having the image of a code tree in mind allows for an elegant analysis of all algorithms working this way. The key observation for this analysis is the following: If we generate all words in $\mathcal{L}$ (represented by a code tree) proceeding in the way just described, each edge will be traversed exactly twice. Based on this observation it will be possible to analyze the expected amount of work to be saved by compressing the code trees in two different ways:

(1) Trie compression: Each leaf becomes a child of its first predecessor with a degree larger than 1 (which is equivalent to the deletion of linear lists at the leaves). The labels of the deleted edges are concatenated and attached to the leaf's ingoing edge. This compression transforms the code tree into a trie structure [9,14].

(2) PATRICIA compression: All unary nodes of the tree are deleted, collecting labels at the remaining edges. By this compression the code tree becomes equivalent to a PATRICIA search-tree [9].

In both cases we will assume that the code tree to be compressed is randomly taken from a family of simply generated trees. We will prove precise formulæ for the expected amount of work saved (which is obviously given by twice the expected number of edges deleted during compression). In any case the results can be expressed by means of the number $t(\alpha, \beta)$ of trees in the family having α nodes of which β are leaves.

We shall mention that according to our point of view compression of codes is not an algorithmic task but is left to the designer of an algorithm. For instance when working with tree-like structures an easy and well-known encoding is given by the semi Dyck-language, representing a tree by means of a bracket term [17]. However, the code tree for the semi Dyck-language has many unary nodes giving rise to notable speedups by compression. Thus, instead of using the first encoding at hand, the designer should think of compressing the code according to the trie or PATRICIA compression strategy before preparing her algorithm.

Obviously, PATRICIA will never result in a code tree worse than the one generated by the trie compression. However, for the algorithm used for the lexicographic generation (which do not work on code trees) it might be much easier to only omit unary nodes at the leaves of a code tree than to omit them all. Therefore it makes sense to analyze both compression methods and to compare their influences.

According to the author's knowledge, this is the first paper proposing an improvement of Kemp's general algorithm or more precisely of the way we should apply Kemp's general concept. However, there are follow up papers to [8] like e.g. [4,6] or [12] which in some sense also take advantage of some kind of code trees namely in order to generate random objects of a given class (the latter is an intensively studied subject, see [2,5,10,15] to get access to the related literature). There the main idea is to generate a path from the root to the leaves and thus the encoding of an object based on a randomly chosen rank which is used to decide the next edge to be taken at a time (given an ordered set $\mathcal{U}$ and a rank i, the process of computing the i-th object of $\mathcal{U}$ is called *unranking*). Decisions are based on subtree sizes which are determined e.g. by using decomposition rules like unions or concatenations (products) that apply to the encoding in use [12]. That way it becomes possible to automatically translate the decomposition rules of an encoding into an unranking algorithm used for the random generation of objects. Literature also deals with the reverse process called *ranking* which consists of computing the rank of a given object. In section 3.3 we will comment on a connection of our findings to the problem of ranking lexicographically ordered words of a given language.

2 Additive Weights of Simply Generated Trees

Let $T = (I, L, r)$ be an unlabeled rooted planar tree with the set of internal nodes I, the set of leaves L and the root $r \in I$; T is said to be a λ-tree if the root r is of degree λ. The weight $\omega(T)$ of the tree T is recursively defined as follows [7]:
Let $g, a_i, f_i : \mathbb{N}_0^3 \to \mathbb{R}$, $i \in \mathbb{N}_0$, be given mappings. If T is the one-node tree, then $\omega(t) := g(0, 1, 1)$; if $T = (I, L, r)$ is a λ-tree with the subtrees $T_i = (I_i, L_i, r_i)$, $1 \le i \le \lambda$, then

$$\begin{aligned}\omega(T) := & g(\lambda, n, m) \\ & + \sum_{1 \le i \le \lambda} (a_i(\lambda, n_i, m_i)\omega(T_i) \\ & \qquad + f_i(\lambda, n_i, m_i)),\end{aligned}$$

where $n := |I \cup L|$, $m := |L|$, $n_i := |I_i \cup L_i|$ and $m_i := |L_i|$, $1 \le i \le \lambda$. Many well-known parameters of trees can be expressed in this framework; prominent examples are the internal path length or the number of paths between leaves. This framework also allows for the representation of the number of edges deleted during our compressions and thus provides a representation for the amount of work to be saved during the lexicographical generation:
Trie compression: With $g(\lambda, n, m) := 2 \cdot \delta_{\lambda,1} \cdot \delta_{m,1}$, $a_i(\lambda, n, m) = 1$ and $f(\lambda, n, m) = 0$, for δ Kronecker's symbol, $\omega(T)$ is equal to twice the number of edges belonging to a linear list at a leaf of T. We will use $\omega_t(T)$ to denote this weight of tree T in the sequel.
PATRICIA compression: With $g(\lambda, n, m) := 2 \cdot \delta_{\lambda,1}$, $a_i(\lambda, n, m) = 1$ and $f(\lambda, n, m) = 0$, $\omega(T)$ is equal to twice the number of unary nodes in T and thus equal to twice the number of edges that are deleted during a PATRICIA compression of T; we will use $\omega_p(T)$ as notation for this weight throughout this paper.
According to [13] a family $\mathcal{F}$ of rooted planar trees is called *simply generated*, if the generating function $E(z) = \sum_{n \ge 1} t(n) z^n$ of the number $t(n) := |\mathcal{F}_n|$ of all trees $T \in \mathcal{F}$ with n nodes satisfies a functional equation of the form $E(z) = z \cdot \Theta(E(z))$. Here $\Theta(y) := 1 + \sum_{\lambda \ge 1} c_\lambda y^\lambda$ is a regular function of y when $|y| < R < \infty$. In [1] Aldous pointed out that there is a natural correspondence between simply generated random trees and Galton-Watson branching processes conditioned on size.
For families of simply generated trees the analysis of additive parameters as defined above is well understood. When only the total number of nodes in the trees is used as a parameter, [7] provides general results for the expected additive weight of simply generated trees which could be applied for the analysis of PATRICIA compression; the choice of g in case of the trie compression is not in the range of application of those results. Furthermore, using only a single parameter to specify the size of a tree makes no sense in our context. Since we have to allow unary nodes, fixing only the number of leaves of a tree would give rise for an infinite number of possibilities; fixing only the number of internal nodes or the total number of nodes makes no sense since those numbers are rather unrelated to the number of leaves of the

tree (which is the interesting parameter for our studies). Therefore we decided to consider both as parameters, the tree's total number of nodes α and it's number of leaves β. In this way it is possible to regard the quotient $\rho := \alpha/\beta$ which can be interpreted as kind of redundancy. To handle this setting, we are looking for bivariate generating functions in which all nodes are counted by one, all leaves by an additional variable. To make those generating function available we need a slight modification of the above definition: For $t(\alpha, \beta)$ the number of trees in $\mathcal{F}$ having α nodes in total of which β are leaves, we define the enumerator

$$E(z, v) := \sum_{\alpha \geq 1, \beta \geq 1} t(\alpha, \beta) z^\alpha v^\beta.$$

Now $\mathcal{F}$ is called simply generated if E fulfills

$$E(z, v) = z \cdot \Theta(E(z, v), v) \qquad (1)$$

for $\Theta(y, v) := v + \sum_{\lambda \geq 1} c_\lambda y^\lambda$. At this time we do not need any regularity condition for Θ; this becomes only necessary in order to derive asymptotic results.

3 The Expected Cost for a Lexicographical Generation

3.1 General Results

To analyze the cost for the lexicographical generation (measured by the number of edges traversed), we first observe that any code tree with α nodes has exactly $\alpha - 1$ edges, hence the number of edge-traversals needed to generate all β words represented by the tree is equal to $2(\alpha - 1)$ (and thus independent of β).
To investigate the effect of PATRICIA compression, let $\mathcal{F}$ be simply generated by $\Theta(y, v)$, let $E(z, v)$ be $\mathcal{F}$'s enumerator, and let $\omega_p(T)$ denote the weight counting twice the number of nodes in T deleted by PATRICIA compression. Denoting by $\mathcal{F}^{(\lambda)}$ all the λ-trees of family $\mathcal{F}$ we find for $C_p(z, v) := \sum_{T \in \mathcal{F}} \omega_p(T) z^{\eta(T)} v^{\ell(T)}$

$$\begin{aligned} C_p(z, v) &= \sum_{\lambda \geq 1} \sum_{T \in \mathcal{F}^{(\lambda)}} \omega_p(T) z^{\eta(T)} v^{\ell(T)} \\ &= \sum_{\lambda \geq 1} c_\lambda \sum_{T_1 \in \mathcal{F}} \cdots \sum_{T_\lambda \in \mathcal{F}} (2\delta_{\lambda,1} + \omega_p(T_1) + \\ &\ldots + \omega_p(T_\lambda)) z^{1+\eta(T_1)+\ldots+\eta(T_\lambda)} v^{\ell(T_1)+\ldots+\ell(T_\lambda)} \\ &= 2zE(z, v)c_1 + zC_p(z, v)\Theta'(E(z, v)). \end{aligned}$$

Hence we conclude

$$C_p(z, v) = \frac{2zE(z, v)\Theta'(0)}{1 - z \cdot \Theta'(E(z, v))}. \qquad (2)$$

Here[1] $\Theta'(y)$ is used to represent $\frac{\partial}{\partial y}\Theta(y, v)$ and $\eta(T)$ (resp. $\ell(T)$) denotes the number of nodes (resp. leaves) of T. From $E(z, v) = z \cdot \Theta(E(z, v), v)$ we can conclude that $E'(z, v) := \frac{\partial}{\partial z} E(z, v) = \frac{E(z,v)}{z}/(1 - z \cdot \Theta'(E(z, v)))$ holds. Thus (2) turns into

$$C_p(z, v) = 2z^2 \Theta'(0) E'(z, v). \qquad (3)$$

[1] Because v only occurs as an additive term within $\Theta(y, v)$, v always disappears when taking partial derivatives with respect to y.

For $\omega_t(T)$ the weight according to the trie compression and

$$C_t(z,v) := \sum_{T\in\mathcal{F}} \omega_t(T) z^{\eta(T)} v^{\ell(T)}$$

we find

$$\begin{aligned} C_t(z,v) &= \frac{2z^3\Theta'(0)v}{1-z\cdot\Theta'(0)} \cdot \frac{E'(z,v)}{E(z,v)} \\ &= \frac{2zv\cdot\Theta'(0)}{1-z\cdot\Theta'(0)} \frac{\partial}{\partial v} E(z,v). \end{aligned} \quad (4)$$

The first equality results from a decomposition of $C_t(z,v)$ according to the different root degrees λ as done for $C_p(z,v)$ above and the equalities $\frac{\partial}{\partial v}E(z,v) = \frac{z}{1-z\cdot\Theta'(E(z,v))} = \frac{z^2E'(z,v)}{E(z,v)}$ which are direct consequences of (1). The second equality results from those equalities too. Using the representations (3) and (4) of our generating functions we can prove:

Theorem 1 *Let $\mathcal{F}$ be a family of simply generated trees and let all trees in $\mathcal{F}$ with α nodes of which β are leaves be equally likely. The expected number of edges traversed for the lexicographical generation of a random language $\mathcal{L}$ represented by a random tree T in $\mathcal{F}$ (considered as a code tree for $\mathcal{L}$) is given by*

$$2(\alpha-1)$$

if T is not compressed, it is given by

$$2(\alpha-1) - \frac{2\beta}{t(\alpha,\beta)} \times \sum_{1\le i<\alpha} \Theta'(0)^i \cdot t(\alpha-i,\beta),$$

if T is compressed according to the trie compression, and by

$$2(\alpha-1) - \frac{2\Theta'(0)(\alpha-1)t(\alpha-1,\beta)}{t(\alpha,\beta)},$$

if T is compressed according to the PATRICIA compression.

Proof: From the discussion above it is obvious that for an uncompressed code tree T the expected number of edges in question is given by $2(\alpha-1)$. Thus, by linearity of expectations, it is sufficient to subtract twice the expected number of edges which are deleted during compression from $2(\alpha-1)$ in order to consider the effect of compressing the code trees. For the PATRICIA compression the total number of edges deleted in the trees in $\mathcal{F}$ having α nodes and β leaves is given by the coefficient at $z^\alpha v^\beta$ of $C_p(z,v)$. Using $[z^\alpha v^\beta]$ to represent the operator which determines this coefficient we find from (3)

$$\begin{aligned} &[z^\alpha v^\beta] C_p(z,v) = \\ &\quad 2\Theta'(0)[z^{\alpha-2}v^\beta]E'(z,v) \\ &= 2\Theta'(0)[z^{\alpha-2}v^\beta] \\ &\quad \times \sum_{\substack{n\ge1\\ m\ge1}} n\cdot t(n,m) z^{n-1} v^m \\ &= 2\Theta'(0)(\alpha-1)t(\alpha-1,\beta). \end{aligned}$$

Assuming a uniform distribution for the trees in $\mathcal{F}$ implies the denominator $t(\alpha,\beta)$ which shows up in the theorem in order to take expectations. For the trie compression we have to determine $[z^\alpha v^\beta]C_t(z,v)$ for which we find from (4)

$$[z^\alpha v^\beta] C_t(z,v)$$

$$= 2 \sum_{k \geq 1} \Theta'(0)^k [z^{\alpha-k} v^{\beta-1}] \frac{\partial}{\partial v} E(z, v)$$
$$= 2\beta \sum_{1 \leq i \leq \alpha} \Theta'(0)^i \cdot t(\alpha - i, \beta).$$

Here the first equality holds due to the expansion $z \cdot \Theta'(0)/(1 - z \cdot \Theta'(0)) = \sum_{k \geq 1} z^k \cdot \Theta'(0)^k$ and the denominator $t(\alpha, \beta)$ of the theorem results from taking expectations. □

3.2 Example of Use

We consider simply generated code trees over an s-ary alphabet. In this case we have $\binom{s}{k}$ choices for an internal node of degree k since the k outgoing edges have to be marked by k different but arbitrarily chosen symbols taken from an alphabet of size s. Accordingly, $\Theta(y, v) = v + \sum_{1 \leq k \leq s} \binom{s}{k} y^k = v + (1 + y)^s - 1$ is the corresponding *generator* where the term v represents the possibility of a leaf and each of the summands $\binom{s}{k} y^k$ stands for the $\binom{s}{k}$ different choices for an internal node of degree k. We apply Lagrange's inversion [3] in order to get an exact representation of the coefficient $[z^\alpha]E(z, v)$. In this way we find

$$[z^\alpha]E(z, v) = \frac{1}{\alpha} \sum_{k \geq 0} \binom{\alpha}{k} \binom{sk}{\alpha - 1} (v - 1)^{\alpha - k}.$$

It is now an easy task to extract the coefficient at v^β from this representation. In the case of $s = 2$ the resulting closed form representation of $t(\alpha, \beta)$ is given by $\frac{2^{\alpha-2\beta+1}}{\beta} \binom{\alpha-1}{2\beta-2} \binom{2\beta-2}{\beta-1}$. Consequently, the expected number of edges which needs to be traversed in case of a PATRICIA compressed code tree is given by

$$4\beta - 4.$$

In case of the trie compression we have to traverse

$$2 \frac{(\beta - 1)(2\beta - 1 + \alpha)}{2\beta - 1}$$

many edges on the average. For $\alpha = \beta\rho$, the quotient of both results (the case of PATRICIA compression taken as the denominator) evaluates to $\frac{1}{2} + \frac{\beta\rho}{2(2\beta-1)}$. As a consequence, both compressions lead to the same complexity for a lexicographical generation in case of a minimal redundancy of $\rho = (2\beta-1)/\beta$ (in that case both algorithms do not change the code tree's structure). As a function of ρ, this quotient (or graphically speaking the drawback of the trie compression) grows linearly with a slope of at least $1/4$ (where this lower limit results from taking $\beta \to \infty$).

At this point we know how much a compression of the different kinds would speedup our algorithm on average. For any special case it would now be necessary to adapt the algorithm to the changes of the encoding implied by the compression – which might by a complicated task.

3.3 Further Implications

A problem similar to the lexicographic generation is that of ranking words of a lexicographically ordered language. Having our code tree representation of languages in mind the

general pattern for computing the rank of $w \in \mathcal{L}$ as proposed by [11] is the following:

- Traverse $\mathcal{L}$'s code tree starting at the root; $r := 0$;
- For a node v on level k take the edge corresponding to the k-th symbol of w;
  if this is the i-th edge of v then set $r := r + s_{i-1}$;
  STOP when there is a unique path to a leaf.

Here, s_i denotes the size (number of leaves) of the node's i-th subtree; at the end of the computation r yields the rank of w (which for example might be used in order to store the corresponding combinatorial object in a database using default data types viz natural numbers).

A moment's reflection shows that the edges traversed for computing the rank of all words in $\mathcal{L}$ are those that remain in the code tree for $\mathcal{L}$ after trie compressing the tree. However, edges might be traverse several times while ranking the words in $\mathcal{L}$; in detail, an edge leading to k leaves is used exactly k times. Thus, considering the additive weight $\omega_r(T)$ with $g(\lambda, n, m) := m$, $a_i(\lambda, n_i, m_i) := 1$ and $f_i(\lambda, n_i, m_i) := 0$, $1 \leq i \leq \lambda$, together with ω_t could be use in order to study the expected cost for ranking a random simply generated tree: the weight $\omega_r(T)$ overestimates the price payed for ranking all words in $\mathcal{L}$ by $|\mathcal{L}|$ plus the number of edges deleted by a trie compression, which is given by $\frac{1}{2}\omega_t(T)$, T the code tree of $\mathcal{L}$.

4 Conclusions

In this note we have considered the lexicographical generation of languages represented by code trees. We have considered simply generated families of (code) trees in an extended framework assuming a uniform probability distribution to derive our results. It should be mentioned that it is also possible to extend this model to a non-uniform setting by means of the c_λ (see e.g. [1]). Furthermore, we have only used our general approach to derive exact representations of the expectations in question. However, based on the implicit function theorem and standard techniques for the computation of asymptotic representations for coefficients of multivariate generating functions, it is possible to extend well-known results for the original setting by Meier and Moon and thus to derive an asymptotic representation for $t(\alpha, \beta)$ only depending on $\Theta(y, v)$ and its derivatives.

We have shown connections of this work to ranking and unranking procedures used e.g. for the generation of random objects and sketched a way to use our results for an analysis of a general ranking strategy proposed by Liebehenschel.

Finally we want so stress that our results may additionally be interpreted as an analysis of the expected savings (assuming a combinatorial model) for the path length of search tree structures constructed based on the digital representation of keys implied by the *compression* as used for tries and PATRICIA.

Acknowledgements

I wish to thank the anonymous referee for comments and suggestions which helped to improve the quality of the presentation.

References

[1] D. J. Aldous, *The Continuum Random Tree II: An Overview*, Stochastic Analysis (Durham, 1990), London Math. Soc. Lecture Note Ser., **167**, Cambridge Univ. Press, 1991, 23-70.

[2] L. Alonso and R. Schott, *Random Generation of Trees*, Kluwer, 1995.

[3] L. Comtet *Advances Combinatorics*, Reidel Publishing Company, 1974.

[4] A. Denise and P. Zimmermann, *Uniform random generation of decomposable structures using floating-point arithmetic*, Theoretical Computer Science **218** (1999), 233-248.

[5] L. Devroye, *Non-Uniform Random Variate Generation*, Springer Verlag, 1986.

[6] P. Flajolet, É. Fusy and C. Pivoteau,
Boltzmann Sampling of Unlabelled Structures, submitted.

[7] R. Kemp, *The Expected Additive Weight of Trees*, Acta Informatica **26** (1989), 711-740.

[8] R. Kemp, *Generating words lexicographically: An average-case analysis*, Acta Informatica **35** (1998), 17-89.

[9] D. E. Knuth *The Art of Computer Programming*, Volume 3: Sorting and Searching, Second Edition, Addison Wesley Longman, 1998.

[10] D. E. Knuth *The Art of Computer Programming*, Volume 4: Combinatorial Algorithms, Addison Wesley Longman, 2005-6, Fascicles 2-4.

[11] J. Liebehenschel, *Ranking and Unranking of Lexicographically Ordered Words: An Average-Case Analysis*, Journal of Automata, Languages and Combinatorics **2** (1997), 227-268.

[12] C. Marínez and X. Molinero, *Generic Algorithms for the Generation of Combinatorial Objects*, MFCS 2003, 572-581.

[13] A. Meier und J. W. Moon, *On the Altitude of Nodes in Random Trees*, Can. J. Math **30** (1978), 997-1015.

[14] M. E. Nebel, *The Stack-Size of Combinatorial Tries Revisited*, Discrete Mathematics and Theoretical Computer Science **5** (2002), 1-16.

[15] A. Nijenhuis and H. Wilf, *Combinatorial Algorithms*, Academic Press, seconad edition, 1978.

[16] F. Ruskey, *Generating t-Ary Trees Lexicographically*, SIAM J. Comput. **9** (1977), 137-164.

[17] S. Zaks, *Lexicographic Generation of Ordered Trees*, Theoret. Comp. Science **10** (1980), 63-82.