Balanced Parentheses Strike Back

Hsueh-I Lu
National Taiwan University
and
Chia-Chi Yeh
National Taiwan University

An *ordinal tree* is an arbitrary rooted tree where the children of each node are ordered. Succinct representations for ordinal trees with efficient query support have been extensively studied. The best previously known result is due to Geary, Raman, and Raman [*SODA* 2004, pages 1–10]. The number of bits required by their representation for an n-node ordinal tree T is $2n + o(n)$, whose first-order term is information-theoretically optimal. Their representation supports a large set of $O(1)$-time queries on T. Based upon a balanced string of $2n$ parentheses, we give an improved $2n + o(n)$-bit representation for T. Our improvement is two fold: Firstly, the set of $O(1)$-time queries supported by our representation is a proper superset of that supported by the representation of Geary, Raman, and Raman. Secondly, it is also much easier for our representation to support new queries by simply adding new auxiliary strings.

Categories and Subject Descriptors: E.1 [**Data**]: Trees; E.4 [**Coding and Information Theory**]: Data compaction and compression; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms, Trees*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Dictionaries, Indexing methods*

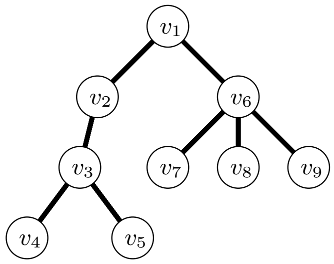
General Terms: Algorithm, Design, Theory

Additional Key Words and Phrases: Succinct data structures, XML document representation

1. INTRODUCTION

An *ordinal* tree (see, e.g., [Geary et al. 2004; Benoit et al. 2005]) is an arbitrary rooted tree where the children of each node are ordered. All trees in the paper are ordinal. The number of distinct n-node trees is $2^{2n-\Theta(\log n)}$ [Graham et al. 1989], so the information-theoretically minimum number of bits to differentiate these trees is $2n - \Theta(\log n)$. There are three major types of $2n$-bit representations for an n-node tree T:

Authors' Address: Department of Computer Science and Information Engineering, National Taiwan University. 1 Roosevelt Road, Section 4, Taipei 106, Taiwan, Republic of China. Emails: hil@csie.ntu.edu.tw, r93048@csie.ntu.edu.tw. Web: www.csie.ntu.edu.tw/∼hil/. This research is supported in part by NSC Grants 94-2213-E-002-126 and 95-2221-E-002-077.
The first author is the corresponding author, who is also affiliated with the Graduate Institute of Networking and Multimedia and the Graduate Institute of Biomedical Electronics and Bioinformatics, National Taiwan University.



Balanced parentheses: (((()()))(()()())).
LOUDS: 11010111011000000.
DFUDS: 11010110001110000.

Fig. 1. Three representations for the same tree.

—Balanced parentheses [Munro and Raman 2001; Chuang et al. 1998; He et al. 1999; Chiang et al. 2005; Munro and Rao 2004; Bonichon et al. 2006], a folklore encoding consisting of a balanced string of parentheses representing the counterclockwise depth-first traversal of T, where an open (respectively, closed) parenthesis denotes a descending (respectively, ascending) edge traversal. For technical reason, one usually adds a pair of enclosing parentheses to the above $2n-2$ parentheses, resulting in a representation consisting of $2n$ parentheses.

—Level order unary degree sequence (LOUDS) [Jacobson 1989], representing a node of degree d as a string of d copies of 1-bits followed by a 0-bit, where these nodes are represented in a level-order traversal of T.

—Depth first unary degree sequence (DFUDS) [Benoit et al. 2005], representing a node of degree d as a string of d copies of 1-bits followed by a 0-bit, where these nodes are represented in a depth-first traversal of T.

An example is shown in Figure 1.

Initiated by Jacobson [Jacobson 1989], succinct representations for trees with efficient query support have been extensively studied in the literature. Jacobson [Jacobson 1989] extended the LOUDS representation into a $\Theta(n)$-bit encoding to support the parent query and the rank and select queries for nodes in level-order traversal of T in $\Theta(\log n)$ time. Clark and Munro [Clark 1996; Clark and Munro 1996] squeezed Jacobson's encoding into a $3n+o(n)$-bit representation, from which the above queries and the subtree-size query can be supported in $O(1)$ time. Later succinct representations, all have $2n+o(n)$ bits, form the following trade-off between the choices of base representations and the sets of supported $O(1)$-time queries:

—Based upon balanced parentheses, Munro and Raman [Munro and Raman 2001] showed that an $o(n)$-bit auxiliary string suffices to support the following queries in $O(1)$ time: parent, depth, subtree-size, and the rank and select queries for nodes in pre-order and post-order traversal of T. Munro, Raman, and Rao [Munro et al. 2001] showed an $o(n)$-bit auxiliary string to support $O(1)$-time query for leaf-rank, leaf-select, and leaf-size. Chiang, Lin, and Lu [Chiang et al. 2005] showed an $o(n)$-bit auxiliary string to support $O(1)$-time degree query. Munro and Rao [Munro and Rao 2004] further gave an $o(n)$-bit auxiliary string to support $O(1)$-time level-ancestor query.

—Based upon the DFUDS representation, Benoit et al. [Benoit et al. 2005] gave an $o(n)$-bit auxiliary string that supports the following queries in $O(1)$ time: child-rank, child-select, degree, subtree-size, and node-rank and node-select in

	parentheses	DFUDS	Geary et al.	new
pre-order select and rank	$\vee$	$\vee$	$\vee$	$\vee$
post-order select and rank	$\vee$		$\vee$	$\vee$
child-select and child-rank		$\vee$	$\vee$	$\vee$
leaf-select, leaf-rank, and leaf-size	$\vee$			$\vee$
lowest common ancestor				$\vee$
subtree height				$\vee$
subtree size	$\vee$	$\vee$	$\vee$	$\vee$
level ancestor	$\vee$		$\vee$	$\vee$
distance				$\vee$
degree	$\vee$	$\vee$	$\vee$	$\vee$
depth	$\vee$		$\vee$	$\vee$

Table I. A summary for current $2n + o(n)$-bit encodings for an n-node tree: Parentheses [Munro and Raman 2001; Chiang et al. 2005; Munro and Rao 2004; Munro et al. 2001], DFUDS [Benoit et al. 2005], Geary et al. [Geary et al. 2004].

the pre-order traversal of T. However, such a choice of the base representation still does not provide $O(1)$-time support for the depth and level-ancestor queries, the node-rank and node-select queries in the post-order traversal of T, and the rank, select, and size queries for leaves.

Recently, Geary, Raman, and Raman [Geary et al. 2004] almost resolved the above trade-off by giving a $2n+o(n)$-bit encoding for T that supports in $O(1)$ time the aforementioned queries except those leaf-related ones [Munro et al. 2001]. Their approach differs from all previous work achieving $2n+o(n)$ bits in that their encoding does not consist of a $2n$-bit base representation for the topology of T plus an $o(n)$-bit auxiliary string. Instead, they decomposed T into several types of subtrees, whose topologies are represented in a hierarchical way, where different levels are composed of mixtures of different base representations and auxiliary strings. Such an involved structure seriously complicates the possibility of supporting additional queries using other stand-alone auxiliary strings. An implementation based upon a similar concept is studied in [Geary et al. 2004]. Very recently, Delpratt, Rahman, and Raman [Delpratt et al. 2006] showed that LOUDS-based representation can also be implemented to have competitive practical performance.

In the present paper, we give new $o(n)$-bit auxiliary strings for the $2n$-bit balanced string of parentheses representing T. Together with previous $o(n)$-bit auxiliary strings for balanced parentheses [Munro and Raman 2001; Chiang et al. 2005; Munro and Rao 2004], our $2n+o(n)$-bit encoding for T supports all of Geary et al.'s queries in $O(1)$ time. Consisting of a base representation plus $o(n)$-bit auxiliary strings, our encoding is better in the ease of supporting new queries by adding new $o(n)$-bit auxiliary strings. To demonstrate such an advantage, we also show how to handle $O(1)$-time queries currently unsupported by Geary et al.'s encoding, including (a) lowest common ancestor, (b) distance, and (c) subtree height. Table I summarizes the above discussion.

We follow the convention of unit-cost RAM model of computation with $\Theta(\log n)$-bit word size [van Emde Boas 1990], which is assumed in all the previous work except that of Jacobson [Jacobson 1989]. The rest of the paper is organized as follows. Section 2 gives the preliminaries. Section 3 shows our auxiliary strings for distance, subtree height, and lowest common ancestor. Section 4 shows our auxiliary strings for child-rank and child-select.

2. PRELIMINARIES

Let T be the input n-node tree. Let v_i denote the i-th node of T in the pre-order traversal of T. Let S be the balanced string of $2n$ parentheses for T. Let $S[i,j]$ denote the substring of S from index i to index j. Let $S[i] = S[i,i]$. Let ℓ_i be the index such that $S[\ell_i]$ is the i-th open parenthesis in S. Let r_i be the index such that $S[r_i]$ is the closed parenthesis that matches $S[\ell_i]$ in S. One can easily see that the correspondence between v_i and the matched parentheses $S[\ell_i]$ and $S[r_i]$: v_i is the parent of v_j if and only if $S[\ell_i]$ and $S[r_i]$ is the closest parenthesis pair that encloses $S[\ell_j]$ and $S[r_j]$. Let $w(i,j) = j - i + 1$. For the rest of the paper, all logarithms are of base 2. Let $B = \lceil \log^3 n \rceil$, $b = \lceil (\log \log n)^3 \rceil$, $n_B = \lceil \frac{2n}{B} \rceil$, and $n_b = \lceil \frac{2n}{b} \rceil$.

Lemma 2.1 (see [Bell et al. 1990; Elias 1975]). *For any $O(n)$-bit strings $S_1, S_2, \ldots, S_k$ with $k = O(1)$, there is an $O(\log n)$-bit auxiliary string α_{concat} such that, given the concatenation of $\alpha_{concat}, S_1, S_2, \ldots, S_k$ as input, the index of the first symbol of any given S_i in the concatenation is computable in $O(1)$ time.*

Let $S_1 \circ S_2 \circ \cdots \circ S_k$ denote the concatenation of $\alpha_{concat}, S_1, S_2, \ldots, S_k$ as in Lemma 2.1.

Lemma 2.2 (see [Munro and Raman 2001; Chiang et al. 2005]). *Let S be a length-$2n$ string of balanced parentheses that represents an n-node tree T. It takes $O(n)$ time to compute an $o(n)$-bit string α_{aux} such that the following queries for S can be determined from S and α_{aux} in $O(1)$ time: (a) the parent, degree, and depth of v_i in T, (b) the parenthesis that matches $S[i]$ in S, and (c) the rank and select queries for open and closed parentheses in S.*

By Lemma 2.2, given $S \circ \alpha_{aux}$, indices i, ℓ_i, and r_i can be determined from one another in $O(1)$ time. Our technique of dividing the input strings into multiple levels of blocks, which has been widely used in many succinct data structures, is inspired by Munro and Raman [Munro 1996; Munro and Raman 2001].

3. DISTANCE, SUBTREE HEIGHT, AND LOWEST COMMON ANCESTOR

Let L be the $2n$-element array such that each $L[i]$ is the number of open parentheses minus the number of closed parentheses in $S[1,i]$. Therefore, if $S[j]$ is the i-th open parenthesis in S, then $L[j]$ is the level of v_i in T. For any indices i and j with $i \le j$, let $index_{min}(L,i,j)$ (respectively, $index_{max}(L,i,j)$) denote the smallest index k with $i \le k \le j$ such that $L[k]$ equals the minimum (respectively, maximum) of $L[i], L[i+1], \ldots, L[j]$. As observed by Gabow, Bentley, and Tarjan [Gabow et al. 1984], the lowest-common-ancestor query can be reduced to the above range-minima query $index_{min}$. Similarly, our auxiliary string for supporting the queries of distance, subtree height, and lowest common ancestor is based on the lemma below. Observe that each $L[i]$ can be obtained from S in $O(1)$ time using the auxiliary string α_{aux} for the rank queries with respect to open and closed parentheses in S. Therefore, the following lemma does not require L in the encoding.

Let I be an array of m indices. Let $k_{min}(I,m,i,j)$ (respectively, $k_{max}(I,m,i,j)$) be the smallest index k with $i \le k \le j$ that minimizes (respectively, maximizes)

$L[I[k]]$. We first prove the following lemma using techniques extended from Section 3 of [Bender and Farach-Colton 2000].

LEMMA 3.1. *It takes $O(m \log m)$ time to compute an $O(m \log^2 m)$-bit string $\alpha_q(I, m)$ from which $k_{min}(I, m, i, j)$ and $k_{max}(I, m, i, j)$ for any indices i and j with $1 \le i \le j \le m$ can be determined from S, α_{aux}, and α_q in $O(1)$ time.*

PROOF. For each $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, \lceil \log m \rceil$, let $M_{min}[i][j]$ (respectively, $M_{max}[i][j]$) be the smallest index k with $i \le k < i + 2^j$ that minimizes (respectively, maximizes) $L[I[k]]$. Let $\alpha_q(I, m) = M_{min} \circ M_{max}$. Observe that $\alpha_q(I, m)$ takes $O(m \log^2 m)$ bits and can be computed from L and I in $O(m \log m)$ time using dynamic programming. Let $k_1 = M_{min}[i][k]$ and $k_2 = M_{min}[j-2^k+1][k]$, where $k = \lfloor \log(j - i) \rfloor$. It is not difficult to see that

$$k_{min}(I, m, i, j) = \begin{cases} k_1 & \text{if } L[I[k_1]] < L[I[k_2]] \\ k_2 & \text{otherwise.} \end{cases}$$

One can compute $k_{max}(I, m, i, j)$ from M_{max}, I, and L analogously in $O(1)$ time. □

LEMMA 3.2. *It takes $O(n)$ time to compute an $o(n)$-bit string α_{rmq} such that $index_{min}(L, i, j)$ and $index_{max}(L, i, j)$ for any indices i and j can be computed from S, α_{aux}, and α_{rmq} in $O(1)$ time.*

PROOF. First let I_B be the n_B-element array such that each $I_B[i]$ is the smallest index j with $(i - 1)B < j \le iB$ that minimizes $L[j]$. I_B takes $O(n_B \log B) = o(n)$ bits. Also, for each $i = 1, 2, \ldots, n_B$, let $I_b[i]$ be the $\lceil \frac{B}{b} \rceil$-element array such that each $I_b[i][j]$ is the smallest index t with $(j - 1)b < t \le jb$ that minimizes $L[(i - 1)B + t]$. I_b takes $O(n_B \lceil \frac{B}{b} \rceil \log b) = o(n)$ bits. Let $\alpha_{q1} = \alpha_q(I_B, n_B)$, and for each $i = 1, 2, \ldots, n_B$, let $\alpha_{q2}[i] = \alpha_q(I_b[i], \lceil \frac{B}{b} \rceil)$. By Lemma 3.1, both of α_{q1} and α_{q2} take $o(n)$ bits and can be obtained in $O(n)$ time. Finally, let α_{q3} be an $O(n)$-time obtainable table such that any $index_{min}(L, i, j)$ and $index_{max}(L, i, j)$ with $w(i, j) \le 2b$ can be computed from $S[i, j]$ and α_{q3} in $O(1)$ time. That is, let $\alpha_{q3}[S[i, i + 2b - 1]][j - i + 1] = (index_{min}(L, i, j) - i, index_{max}(L, i, j) - i)$ for any indices i and j with $w(i, j) \le 2b$. Since each entry takes $O(\log b)$ bits, the number of bits required by α_{q3} is $O(2^{2b} 2b \log b) = o(n)$. Let $\alpha_{rmq} = \alpha_{q1} \circ \alpha_{q2} \circ \alpha_{q3} \circ I_B \circ I_b$, which has $o(n)$ bits and is obtainable in $O(n)$ time.

To answer $index_{min}(L, i, j)$ from S, α_{aux}, and α_{rmq}, we can always decompose the interval $[i, j]$ into two (not necessarily disjoint) subintervals $[i_1, j_1]$ and $[i_2, j_2]$ whose union is $[i, j]$. Clearly $index_{min}(L, i, j)$ can be determined from $index_{min}(L, i_1, j_1)$ and $index_{min}(L, i_2, j_2)$ in $O(1)$ time. Consider the following cases.

—Case 1: $w(i, j) \le 2b$. We simply resort to $S[i, j]$ and α_{q3}.

—Case 2: $w(i, j) > 2b$ and $S[i, j]$ is in the same length-B block of S. Since $index_{min}(L, i, i + b - 1)$ and $index_{min}(L, j - b + 1, j)$ can be determined in $O(1)$ time using Case 1, it suffices to determine $index_{min}(L, i', j')$, where (a) i' is the smallest index with $i \le i'$ that is a starting index of a length-b block of S, and (b) j' is the largest index with $j' \le j$ that is an ending index of a length-b block of S. Since i' and j' are in the same length-B block of S, $index_{min}(L, i', j')$ can be determined from S, α_{aux}, and α_{q2} in $O(1)$ time.

—Case 3: $w(i,j) > 2b$ and $S[i,j]$ belongs to two or more consecutive length-B blocks of S. Let $i'-1$ be the ending index of the length-B block of S that contains i. Let $j'+1$ be the starting index of the length-B block of S that contains j. Since $index_{min}(L,i,i'-1)$ and $index_{min}(L,j'+1,j)$ can be determined in $O(1)$ time using Case 2, it suffices to determine $index_{min}(L,i',j')$ for the case that $i' \leq j'$. Since i' is a starting index of a length-B block of S and j' is an ending index of a length-B block of S, one can determine $index_{min}(L,i',j')$ from S, α_{aux}, and α_{q1} in $O(1)$ time.

It is not difficult to answer $index_{max}(L,i,j)$ from S, α_{aux}, and α_{rmq} analogously in $O(1)$ time. □

As pointed out by an anonymous reviewer, our data structure for lowest common ancestor is similar to that of Sadakane [Sadakane 2002] for suffix arrays.

THEOREM 3.3. *It takes $O(n)$ time to compute an $o(n)$-bit string α_{new1} such that the queries of distance, subtree height, and lowest common ancestor can be answered from S and α_{new1} in $O(1)$ time.*

PROOF. Let $\alpha_{new1} = \alpha_{aux} \circ \alpha_{rmq}$. By Lemmas 2.2 and 3.2, α_{new1} has $o(n)$ bits and can be computed from S in $O(n)$ time.

—The height of the subtree rooted at v_i is $L[index_{max}(L,\ell_i,r_i)]$ minus the depth of v_i in T.

—The lowest common ancestor v_k of v_i and v_j with $\ell_i < \ell_j$ can be determined as follows. If $r_i > r_j$, then $v_k = v_i$. Otherwise, $S[index_{min}(L,r_i,\ell_j)]$ has to be a closed parenthesis r_x such that v_x is a child of v_k, as observed by Bender and Farach-Colton [Bender and Farach-Colton 2000].

—The distance of v_i and v_j is exactly the depth of v_i plus the depth of v_j minus two times of the depth of v_k, where v_k is the lowest common ancestor of v_i and v_j.

By Lemmas 2.2 and 3.2, the above queries can all be answered from S and α_{new1} in $O(1)$ time. □

4. RANK AND SELECT FOR CHILDREN

Before solving rank and select for children, we introduce the following definition and its property. A non-root node v_i is *k-far* if $w(\ell_p,\ell_i) > k$ and $w(\ell_i,r_p) > k$, where v_p is the parent of v_i.

LEMMA 4.1. *If v_i and v_j are two k-far non-root nodes with $|w(\ell_i,\ell_j)| \leq k$, then v_i and v_j are siblings.*

PROOF. Without loss of generality, we assume $\ell_i < \ell_j$. Since v_i and v_j are k-far non-root nodes with $w(\ell_i,\ell_j) \leq k$, v_i cannot be an ancestor or descendant of v_j. Thus we have $r_i < \ell_j$. Assume for a contradiction that v_p (respectively, v_q) is the parent of v_i (respectively, v_j) and $v_p \neq v_q$. Observe that either $r_i < \ell_q$ or $r_p < \ell_j$ holds. Since v_j is k-far, $r_i < \ell_q$ implies $w(r_i,\ell_j) > k$. Since v_i is k-far, $r_p < \ell_j$ implies $w(r_i,\ell_j) > k$. Either case leads to a contradiction, so the lemma is proved. □

For presentational brevity, we classify non-root nodes into the following three disjoint classes: A node is

—*narrow* if it is not b-far;

—*medium* if it is b-far but not B-far; and

—*wide* if it is B-far.

4.1 Child rank

Let $child\_rank(S, v_k)$ denote the number c such that v_k is the c-th child of its parent. We have the following theorem.

THEOREM 4.2. *It takes $O(n)$ time to compute an $o(n)$-bit string α_{new2} such that $child\_rank(S, v_k)$ for each node v_k can be answered from S and α_{new2} in $O(1)$ time.*

PROOF. Let v_p be the parent of v_k. If $S[i, j]$ is a balanced string of parentheses, let $sibling(S, i, j)$ be the number of non-enclosed parenthesis pairs in $S[i, j]$. Observe that

$$\begin{aligned} child\_rank(S, v_k) &= sibling(S, \ell_p + 1, \ell_k - 1) + 1 \\ &= degree(S, v_p) - sibling(S, \ell_k, r_p - 1) + 1. \end{aligned}$$

Therefore, it remains to support each query $sibling(S, i, j)$ in $O(1)$ time.

If v_k is narrow, we only need to answer $sibling(S, i, j)$ with $w(i, j) \leq b$. We simply build an $O(n)$-time obtainable table M_1 to store the answers for any possible inputs. That is, let $M_1[S[i, i+b-1]][j-i+1] = sibling(S, i, j)$ for any indices i and j with $w(i, j) \leq b$. Since $sibling(S, i, j) \leq w(i, j)$, each entry requires $O(\log b)$ bits and M_1 takes $O(2^b b \log b) = o(n)$ bits.

If v_k is medium, we cannot afford to store all the answers of $sibling(S, i, j)$ with $w(i, j) \leq B$. We split S into length-b blocks. By Lemma 4.1, any two medium nodes v_i and v_j with $|w(\ell_i, \ell_j)| \leq b$ have the same parent, so for each block we save at most one medium node as a shortcut. Define tables M_2 and M_3 as follows. For each $t = 1, 2, \ldots, n_b$,

—let $M_2[t] = (\ell_i, sibling(S, \ell_p + 1, \ell_i - 1))$, where ℓ_i is the smallest index, if any, with $(t-1)b < \ell_i \leq tb$ such that v_i is a medium child of v_p with $w(\ell_p, \ell_i) \leq B$; and

—let $M_3[t] = (\ell_i, sibling(S, \ell_i, r_p - 1))$, where ℓ_i is the smallest index, if any, with $(t-1)b < \ell_i \leq tb$ such that v_i is a medium child of v_p with $w(\ell_i, r_p) \leq B$.

Note that M_2 and M_3 have n_b entries, each requiring $O(\log B)$ bits, so both of them take $O(n_b \log B) = o(n)$ bits. Therefore, for any medium child v_k of v_p, if $w(\ell_p, \ell_k) \leq B$, then

$$\begin{aligned} sibling(S, \ell_p + 1, \ell_k - 1) &= sibling(S, \ell_p + 1, \ell_i - 1) + sibling(S, \ell_i, \ell_k - 1) \\ &= m + M_1[S[\ell_i, \ell_i + b - 1]][\ell_k - \ell_i], \end{aligned}$$

where $(\ell_i, m) = M_2[\lceil \frac{\ell_k}{b} \rceil]$. Similarly, if $w(\ell_k, r_p) \leq B$, then

$$\begin{aligned} sibling(S, \ell_k, r_p - 1) &= sibling(S, \ell_i, r_p - 1) - sibling(S, \ell_i, \ell_k - 1) \\ &= m - M_1[S[\ell_i, \ell_i + b - 1]][\ell_k - \ell_i], \end{aligned}$$

function $\mathit{child\_rank}(S, v_k)$

let v_p be the parent of v_k;
if $w(\ell_p, \ell_k) \le b$, **then** return $M_1[S[\ell_p + 1, \ell_p + b]][\ell_k - \ell_p - 1] + 1$;
if $w(\ell_k, r_p) \le b$, **then** return $\mathit{degree}(S, v_p) - M_1[S[\ell_k, \ell_k + b - 1]][r_p - \ell_k]] + 1$;
if $w(\ell_p, \ell_k) \le B$, **then** let $(\ell_i, m) = M_2[\lceil \frac{\ell_k}{b} \rceil]$, and return $m + M_1[S[\ell_i, \ell_i + b - 1]][\ell_k - \ell_i] + 1$;
if $w(\ell_k, r_p) \le B$, **then** let $(\ell_i, m) = M_3[\lceil \frac{\ell_k}{b} \rceil]$, and return $\mathit{degree}(S, v_p) - m + M_1[S[\ell_i, \ell_i + b - 1]][\ell_k - \ell_i] + 1$;
let $(\ell_j, m) = M_5[\lceil \frac{\ell_k}{B} \rceil][\lceil \frac{\ell_k \bmod B}{b} \rceil]$, and return $M_4[\lceil \frac{\ell_k}{B} \rceil] + m + M_1[S[\ell_j, \ell_j + b - 1]][\ell_k - \ell_j] + 1$;

Fig. 2. An $O(1)$-time algorithm that computes $\mathit{child\_rank}(S, v_k)$.

where $(\ell_i, m) = M_3[\lceil \frac{\ell_k}{b} \rceil]$.

Similar tricks work for wide nodes, but they have to be applied in two levels. We first split S into length-B blocks. For each $t = 1, 2, \ldots, n_B$, let $M_4[t] = \mathit{sibling}(S, \ell_p + 1, \ell_i - 1)$, where ℓ_i is the smallest index, if any, with $(t-1)B < \ell_i \le tB$ such that v_i is a wide child of v_p. We further split each length-B block into length-b blocks. For each $t = 1, 2, \ldots, n_B$ and $u = 1, 2, \ldots, \lceil \frac{B}{b} \rceil$, let $M_5[t][u] = (\ell_j, \mathit{sibling}(S, \ell_p + 1, \ell_j - 1) - M_4[t])$, where ℓ_j is the smallest index, if any, with $(u-1)b < \ell_j - (t-1)B \le ub$ such that v_j is a wide child of v_p. Note that $\mathit{sibling}(S, \ell_p + 1, \ell_j - 1) - M_4[t] \le B$. One can easily verify that the number of bits required by M_4 is $O(n_B \log n) = o(n)$ and the number of bits required by M_5 is $O(n_B \lceil \frac{B}{b} \rceil \log B) = o(n)$. Thus, for any wide child v_k of v_p, we have

$$\begin{aligned} \mathit{sibling}(S, \ell_p + 1, \ell_k - 1) &= \mathit{sibling}(S, \ell_p + 1, \ell_j - 1) + \mathit{sibling}(S, \ell_j, \ell_k - 1) \\ &= M_4[\lceil \frac{\ell_k}{B} \rceil] + m + M_1[S[\ell_j, \ell_j + b - 1]][\ell_k - \ell_j], \end{aligned}$$

where $(\ell_j, m) = M_5[\lceil \frac{\ell_k}{B} \rceil][\lceil \frac{\ell_k \bmod B}{b} \rceil]$.

Finally, let $\alpha_{new2} = \alpha_{aux} \circ M_1 \circ M_2 \circ M_3 \circ M_4 \circ M_5$, which is an $o(n)$-bit string obtainable from S in $O(n)$ time. The $O(1)$-time algorithm for computing $\mathit{child\_rank}(S, v_k)$ is shown in Figure 2. □

4.2 Child select

First we need the following lemmas to handle the select query for children. For any node v_i, let $\mathit{index}_c(S, \ell_i, m, c) = \ell_j - \ell_i$, where v_j is a sibling of v_i with $w(\ell_i, \ell_j) \le m$ such that $\mathit{child\_rank}(S, v_j) = \mathit{child\_rank}(S, v_i) + c$. If such a v_j does not exist, $\mathit{index}_c(S, \ell_i, m, c) = \phi$.

Lemma 4.3. *It takes $O(n)$ time to compute an $o(n)$-bit string α_b such that $\mathit{index}_c(S, \ell_i, b^2, c)$ for any node v_i and index c can be computed from S and α_b in $O(1)$ time.*

Proof. We simply build an $O(n)$-time obtainable table α_b to store the answers for any possible inputs. That is, let $\alpha_b[S[\ell_i, \ell_i + b^2 - 1]][c] = \mathit{index}_c(S, \ell_i, b^2, c)$ for any node v_i and index c. Since each entry takes $O(\log b)$ bits, α_b requires $O(2^{b^2} b^2 \log b) = o(n)$ bits. □

Lemma 4.4. *Given a node v_i, it takes $O(B)$ time to compute an $o(B)$-bit string $\alpha_B(\ell_i)$ such that $\mathit{index}_c(S, \ell_i, B, c)$ for any index c can be computed from S, α_b, and*

$\alpha_B(\ell_i)$ *in* $O(1)$ *time.*

Proof. For each $t = 0, 1, ..., \lceil \frac{B}{b} \rceil - 1$, let $W_1[t] = index_c(S, \ell_i, B, tb)$. W_1 takes $O(\lceil \frac{B}{b} \rceil \log B) = o(B)$ bits. If $w(W_1[t], W_1[t+1]) > b^2$, we save the answers of $index_c(S, \ell_i, B, tb + z)$ for each $z = 0, 1, \ldots, b-1$ in W_2. W_2 takes at most $O(\lceil \frac{B}{b^2} \rceil b \log B) = o(B)$ bits. Otherwise, by Lemma 4.3 $index_c(S, \ell_i, B, tb + z)$ can be computed in $O(1)$ time using $W_1[t] + index_c(S, \ell_i + W_1[t], b^2, z)$. Let $\alpha_B(\ell_i) = W_1 \circ W_2$, which has $o(B)$ bits and is obtainable in $O(B)$ time. □

Given an array A of $\lceil \frac{m}{u} \rceil$ positive $\lceil \log u \rceil$-bit integers with $m \leq n$ and $u = \lceil \log^3 m \rceil$, let $index_{sum}(A, x)$ denote the largest index y with $\sum_{t=1}^{y} A[t] < x$.

Lemma 4.5. *It takes* $O(m)$ *time to compute an* $o(m)$*-bit string* $\alpha_A(A, m)$ *such that* $index_{sum}(A, x)$ *for any index* x *can be determined from* A *and* $\alpha_A(A, m)$ *in* $O(1)$ *time.*

Proof. This is a special case of the search query of the searchable partial sums problem [Raman et al. 2001; Hon et al. 2003]. Theorem 3 of [Hon et al. 2003] gave an $o(m)$-bit auxiliary string to support this query in $O(1)$ time, but it is unclear whether the preprocessing time is $O(m)$. Let us briefly prove this lemma as follows.

Let $d(x_1, x_2)$ denote $index_{sum}(A, x_2) - index_{sum}(A, x_1)$. For each $t = 0, \ldots, \lceil \frac{m}{u} \rceil - 1$, let $W_3[t] = index_{sum}(A, tu)$. W_3 needs $O(\lceil \frac{m}{u} \rceil \log m) = o(m)$ bits. If $d(tu, (t+1)u) > \lceil \log^2 u \rceil$, for each $z = 0, 1, \ldots, u-1$ we save the values of $d(tu, tu + z)$ in W_4. Because A is an array of positive integers, we have $d(tu, tu+z) \leq z$ and W_4 needs at most $O(\lceil \frac{m}{u \log^2 u} \rceil u \log u) = o(m)$ bits. Otherwise, let

$$W_5[A[index_{sum}(A, tu), index_{sum}(A, tu) + \lceil \log^2 u \rceil - 1]][z] = d(tu, tu + z)$$

for each $z = 0, 1, \ldots, u-1$. W_5 takes $O(2^{\log^3 u} u \log \log u) = o(m)$ bits and is obtainable in $O(m)$ time. Now, let $\alpha_A(A, m) = W_3 \circ W_4 \circ W_5$, which requires $o(m)$ bits and can be obtained in $O(m)$ time. To answer $index_{sum}(A, x)$ in $O(1)$ time, first let t and z be the integers with $x = tu + z$ and $0 \leq z < u$, and then find the values of $index_{sum}(A, tu)$ and $d(tu, tu + z)$ from $\alpha_A(A, m)$. The answer is $index_{sum}(A, tu) + d(tu, tu + z)$. □

Let $child\_select(S, v_p, c)$ denote the index ℓ_k such that v_k is the c-th child of v_p. We have the following theorem.

Theorem 4.6. *It takes* $O(n)$ *time to compute an* $o(n)$*-bit string* α_{new3} *such that* $child\_select(S, v_p, c)$ *for each node* v_p *and* c *can be answered from* S *and* α_{new3} *in* $O(1)$ *time.*

Proof. We say that nodes in a set D are *d-disjoint* [Chiang et al. 2005] if

—$w(\ell_i, r_i) > d$ holds for any node v_i in D; and

—any two nodes v_i and v_j in D satisfy at least one of $|w(\ell_i, \ell_j)| > d$ and $|w(r_i, r_j)| > d$.

Let X be a $2\lceil \frac{2n}{d} \rceil$-element array. For each $t = 1, 2, \ldots, \lceil \frac{2n}{d} \rceil$, we store v_i in $X[2t-1]$, where ℓ_i is the smallest index, if any, with $(t-1)d < \ell_i \leq td$ such that v_i is in D; and also store v_j in $X[2t]$, where r_j is the largest index, if any, with $(t-1)d < r_j \leq td$ such that v_j is in D. Then, every node v_i in D takes at least one slot in X, and can be easily verified using ℓ_i and r_i. We simply say that X *has* v_i if and only if v_i takes at least one of $X[2\lceil\frac{\ell_i}{d}\rceil - 1]$ or $X[2\lceil\frac{r_i}{d}\rceil]$. For notational brevity, let $X[v_i]$ denote the element taken by v_i.

The preprocessing is under the following traversal procedure: first traverse each node v_p of T in prefix order, and for each v_p traverse every child v_i of v_p in counter-clockwise order. Since selecting and matching a parenthesis on S takes $O(1)$ time, and each node is traversed at most two times, one as v_p and the other as v_i, the whole procedure takes $O(n)$ time. The discussion below focuses on nodes v_p and v_i in each iteration of the aforementioned traversal.

—Case 1: v_i is a wide child of v_p. Let *counter* denote the number of wide nodes discovered before each iteration. It is not difficult to see that the parents of wide nodes are B-disjoint. Let X_1 be the $2n_B$-element array with $X_1[v_p] = (\mathit{before}_p, \mathit{first}, \mathit{last})$, where before_p is the value of *counter* before we get v_p, and *first* (respectively, *last*) is the rank of the first (respectively, last) wide child of v_p. Then we partition S into length-B blocks. Let Y_1 be the n_B-element array with $Y_1[t] = (\mathit{before}_i, \ell_i)$, where ℓ_i is the smallest index in a block such that v_i is wide, before_i is the value of *counter* before we get v_i, and t is the first empty entry of Y_1. Both of X_1 and Y_1 take $O(n_B \log n) = o(n)$ bits.

—Case 2: v_i is a medium child of v_p. First we partition S into length-B blocks. If $w(\ell_p, \ell_i) \leq B$, we say that v_i *belongs to* the $\lceil\frac{\ell_p}{B}\rceil$-th block, otherwise the $\lceil\frac{r_p}{B}\rceil$-th block. For each $t = 1, 2, \ldots, n_B$, let $\mathit{counter}[t]$ denote the number of medium nodes belonging to the t-th block before each iteration. Note that at most B medium nodes belong to a block. Similarly, one can verify that the parents of medium nodes are b-disjoint. Let X_2 be the $2n_b$-element array with $X_2[v_p] = (\mathit{before}_L, \mathit{first}_L, \mathit{last}_L, \mathit{before}_R, \mathit{first}_R, \mathit{last}_R)$, where

  —before_L (respectively, before_R) is the value of $\mathit{counter}[\lceil\frac{\ell_p}{B}\rceil]$ (respectively, the value of $\mathit{counter}[\lceil\frac{r_p}{B}\rceil]$) before we get v_p,

  —first_L (respectively, first_R) is the rank of the first medium child of v_p belonging to the $\lceil\frac{\ell_p}{B}\rceil$-th (respectively, $\lceil\frac{r_p}{B}\rceil$-th) block, and

  —last_L (respectively, last_R) is the rank of the last medium child of v_p belonging to the $\lceil\frac{\ell_p}{B}\rceil$-th (respectively,$\lceil\frac{r_p}{B}\rceil$-th) block.

  Note that $1 \leq \mathit{first}_L \leq \mathit{last}_L \leq B$ and $\mathit{degree}(S, v_p) - B \leq \mathit{first}_R \leq \mathit{last}_R \leq \mathit{degree}(S, v_p)$. We further partition each length-B block into length-b blocks. For each $t = 1, 2, \ldots, n_B$, let $Y_2[t]$ be the $\lceil\frac{B}{b}\rceil$-element array with $Y_2[t][u] = (\mathit{before}_i, \ell_i)$, where ℓ_i is the smallest index in a length-b block such that v_i is a medium node belonging to the t-th length-B block, *before* is the value of $\mathit{counter}[t]$ before we get v_k, and u is the first empty entry of $Y_2[t]$. Observe that X_2 needs $O(n_b \log B) = o(n)$ bits and Y_2 needs $O(n_B\lceil\frac{B}{b}\rceil \log B) = o(n)$ bits.

For each $t = 1, 2, \ldots, n_B$, let $\alpha_{B1}[t] = \alpha_B(\ell_i)$ with $(\mathit{before}_i, \ell_i) = Y_1[t]$. By Lemma 4.4, α_{B1} takes $o(n)$ bits and is obtainable in $O(n)$ time. Let A_1 be the n_B-element array such that $\sum_{t=1}^{u} A_1[t] = \mathit{before}_i$ with $(\mathit{before}_i, \ell_i) = Y_1[u]$ holds for each $u = 1, 2, \ldots, n_B$. Note that $0 < A_1[t] \leq B$ holds for any index t, so A_1 takes $O(n_B \log B) = o(n)$ bits. Also, for each $t = 1, 2, \ldots, n_B$, let $A_2[t]$ be the $\lceil\frac{B}{b}\rceil$-element array such that $\sum_{u=1}^{x} A_2[t][u] = \mathit{before}_i$ with $(\mathit{before}_i, \ell_i) = Y_2[t][x]$ holds

function $child\_select(S, v_p, c)$
if X_1 has v_p **then**
  let $(before_p, first, last) = X_1[v_p]$;
  if $first \leq c \leq last$ **then**
    let $z = before_p + c - first + 1$ and $(before_i, \ell_i) = Y_1[index_{sum}(A_1, z)]$;
    return $\ell_i + index_c(S, \ell_i, B, z - before_i)$;
  end if
end if
if X_2 has v_p **then**
  let $(before_L, first_L, last_L, before_R, first_R, last_R) = X_2[v_p]$;
  if $first_L \leq c \leq last_L$ **then**
    let $t = \lceil \frac{\ell_p}{B} \rceil$, $z = before_L + c - first_L + 1$, and $(before_i, \ell_i) = Y_2[t][index_{sum}(A_2[t], z)]$;
    return $\ell_i + index_c(S, \ell_i, b^2, z - before_i)$;
  end if
  if $first_R \leq c \leq last_R$ **then**
    let $t = \lceil \frac{r_p}{B} \rceil$, $z = before_R + c - first_R + 1$, and $(before_i, \ell_i) = Y_2[t][index_{sum}(A_2[t], z)]$;
    return $\ell_i + index_c(S, \ell_i, b^2, z - before_i)$;
  end if
end if
if $index_c(S, \ell_p + 1, b^2, c) \neq \phi$, **then** return $\ell_p + 1 + index_c(S, \ell_p + 1, b^2, c)$;
else return $r_p - F[S[r_p - b + 1, r_p]][degree(S, v_p) - c]$;

Fig. 3. An $O(1)$-time algorithm that computes $child\_select(S, v_p, c)$.

for each $x = 1, 2, \ldots, \lceil \frac{B}{b} \rceil$. Observe that $0 < A_2[t][u] \leq b$ holds for any indices t and u, so A_2 takes $O(n_B \lceil \frac{B}{b} \rceil \log b) = o(n)$ bits. Let $\alpha_{A1} = \alpha_A(A_1, n)$, and for each $t = 1, 2, \ldots, n_B$, let $\alpha_{A2}[t] = \alpha_A(A_2[t], B)$. By Lemma 4.5, both of α_{A1} and α_{A2} take $o(n)$ bits and are obtainable in $O(n)$ time. At last, we construct an $O(n)$-time obtainable table F with $F[S[r_p - b + 1, r_p]][degree(S, v_p) - c] = r_p - \ell_i$, where v_i is the c-th child of v_p with $w(\ell_i, r_p) \leq b$. Note that $degree(S, v_p) - c \leq b$, so F takes $O(2^b b \log b) = o(n)$ bits.

To implement child select in $O(1)$ time, let $\ell_k = child\_select(S, v_p, c)$. v_k is wide if and only if X_1 has v_p and $first \leq c \leq last$, where $(before_p, first, last) = X_1[v_p]$. Moreover, letting $z = before_p + c - first + 1$, v_k is the z-th wide node discovered during the traversal procedure. Let $(before_i, \ell_i) = Y_1[index_{sum}(A_1, z)]$, so v_k is a sibling of v_i with $w(\ell_i, \ell_k) \leq B$ such that $child\_rank(S, v_k) = child\_rank(S, v_i) + z - before_i$. By Lemma 4.4, we can locate v_k using $\ell_k = \ell_i + index_c(S, \ell_i, B, z - before_i)$.

v_k is medium if and only if X_2 has v_p and at least one of $first_L \leq c \leq last_L$ and $first_R \leq c \leq last_R$ is satisfied, where $(before_L, first_L, last_L, before_R, first_R, last_R) = X_2[v_p]$. If $first_L \leq c \leq last_L$, let $t = \lceil \frac{\ell_p}{B} \rceil$ and $z = before_L + c - first_L + 1$. If $first_R \leq c \leq last_R$, let $t = \lceil \frac{r_p}{B} \rceil$ and $z = before_R + c - first_R + 1$. Then, v_k is the z-th medium node belonging to the t-th length-B block discovered during the traversal procedure. Let $(before_i, \ell_i) = Y_2[t][index_{sum}(A_2[t], z)]$, so v_k is a sibling of v_i with $w(\ell_i, \ell_k) \leq b$ such that $child\_rank(S, v_k) = child\_rank(S, v_i) + z - before_i$. By Lemma 4.3, we can locate v_k using $\ell_k = \ell_i + index_c(S, \ell_i, b^2, z - before_i)$.

If v_k is neither wide nor medium, it must be narrow. If $index_c(S, \ell_p + 1, b^2, c) \neq \phi$, then we have $\ell_k = \ell_p + 1 + index_c(S, \ell_p + 1, b^2, c)$. Otherwise, $\ell_k = r_p - F[S[r_p - b + 1, r_p]][degree(S, v_p) - c]$.

Finally, let $\alpha_{new3} = \alpha_{aux} \circ \alpha_b \circ \alpha_{B1} \circ X_1 \circ Y_1 \circ X_2 \circ Y_2 \circ A_1 \circ \alpha_{A1} \circ A_2 \circ \alpha_{A2} \circ F$, which takes $o(n)$ bits and can be computed from S in $O(n)$ time. The $O(1)$-time algorithm for computing $child\_select(S, v_p, c)$ is shown in Figure 3. □

Acknowledgments

We thank Kai-min Chung for helpful discussion. We also thank the anonymous reviewers for their helpful comments.

REFERENCES

Bell, T. C., Cleary, J. G., and Witten, I. H. 1990. *Text Compression*. Prentice-Hall, Englewood Cliffs, NJ.

Bender, M. A. and Farach-Colton, M. 2000. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, G. H. Gonnet, D. Panario, and A. Viola, Eds. Lecture Notes in Computer Science 1776. Springer, Punta del Este, Uruguay, 88–94.

Benoit, D., Demaine, E. D., Munro, J. I., Raman, R., Raman, V., and Rao, S. S. 2005. Representing trees of higher degree. *Algorithmica 43,* 4, 275–292.

Bonichon, N., Gavoille, C., Hanusse, N., Poulalhon, D., and Schaeffer, G. 2006. Planar graphs, via well-orderly maps and trees. *Graph and Combinatorics 22*, 1–18.

Chiang, Y.-T., Lin, C.-C., and Lu, H.-I. 2005. Orderly spanning trees with applications. *SIAM Journal on Computing 34,* 4, 924–945.

Chuang, R. C.-N., Garg, A., He, X., Kao, M.-Y., and Lu, H.-I. 1998. Compact encodings of planar graphs via canonical ordering and multiple parentheses. In *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming*, K. G. Larsen, S. Skyum, and G. Winskel, Eds. Lecture Notes in Computer Science 1443. Springer-Verlag, Aalborg, Denmark, 118–129.

Clark, D. R. 1996. Compact PAT trees. Ph.D. thesis, University of Waterloo.

Clark, D. R. and Munro, J. I. 1996. Efficient suffix trees on secondary storage. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM/SIAM, Atlanta, Georgia, 383–391.

Delpratt, O., Rahman, N., and Raman, R. 2006. Engineering the LOUDS succinct tree representation. In *Proceedings of the 5th International Workshop on Experimental Algorithms*. Lecture Notes in Computer Science 4007. Springer-Verlag, Cala Galdana, Menorca, Spain, 134–145.

Elias, P. 1975. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory IT-21*, 194–203.

Gabow, H. N., Bentley, J. L., and Tarjan, R. E. 1984. Scaling and related techniques for geometry problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. ACM Press, New York, NY, USA, 135–143.

Geary, R. F., Rahman, N., Raman, R., and Raman, V. 2004. A simple optimal representation for balanced parentheses. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching*, S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusöz, Eds. Lecture Notes in Computer Science 3109. Springer-Verlag, Istanbul, Turkey, 159–172.

Geary, R. F., Raman, R., and Raman, V. 2004. Succinct ordinal trees with level-ancestor queries. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, J. I. Munro, Ed. SIAM, New Orleans, Louisiana, USA, 1–10.

Graham, R. L., Knuth, D. E., and Patashnik, O. 1989. *Concrete Mathematics*. Addison-Wesley, Reading, Massachusetts.

He, X., Kao, M.-Y., and Lu, H.-I. 1999. Linear-time succinct encodings of planar graphs via canonical orderings. *SIAM Journal on Discrete Mathematics 12,* 3, 317–325.

Hon, W.-K., Sadakane, K., and Sung, W.-K. 2003. Succinct data structures for searchable partial sums. In *Proceedings of the 14th Symposium on Algorithms and Computation*, T. Ibaraki, N. Katoh, and H. Ono, Eds. Lecture Notes in Computer Science 2906. Springer-Verlag, Kyoto, Japan, 505–516.

Jacobson, G. 1989. Space-efficient static trees and graphs. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*. IEEE, Research Triangle Park, North Carolina, 549–554.

Munro, J. I. 1996. Tables. In *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science*. Lecture Notes in Computer Science 1180. Springer-Verlag, Hyderabad, India, 37–42.

Munro, J. I. and Raman, V. 2001. Succinct representation of balanced parentheses, static trees and planar graphs. *SIAM Journal on Computing 31,* 3, 762–776.

Munro, J. I., Raman, V., and Rao, S. S. 2001. Space efficient suffix trees. *Journal of Algorithms 39,* 2, 205–222.

Munro, J. I. and Rao, S. S. 2004. Succinct representations of functions. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science 3142. Springer-Verlag, Turku, Finland, 1006–1015.

Raman, R., Raman, V., and Rao, S. S. 2001. Succinct dynamic data structures. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, F. K. H. A. Dehne, J.-R. Sack, and R. Tamassia, Eds. Lecture Notes in Computer Science 2125. Springer-Verlag, Providence, RI, USA, 426–437.

Sadakane, K. 2002. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM/SIAM, San Francisco, 225–232.

van Emde Boas, P. 1990. Machine models and simulations. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. A. Elsevier, Amsterdam, Chapter 1, 1–60.

Received Month Year; revised Month Year; accepted Month Year