Journal of Discrete Algorithms 17 (2012) 103–108

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Succinct representations of weighted trees supporting path queries ☆

Manish Patil, Rahul Shah, Sharma V. Thankachan *

Department of Computer Science, Louisiana State University, 291 Coates Hall, Baton Rouge, LA 70803, USA

ARTICLE INFO

Article history:
Received 28 May 2012
Received in revised form 3 August 2012
Accepted 16 August 2012
Available online 28 August 2012

Keywords:
Succinct data structures

ABSTRACT

We consider the problem of succinctly representing a given vertex-weighted tree of n vertices, whose vertices are labeled by integer weights from $\{1, 2, \ldots, \sigma\}$ and supporting the following path queries efficiently:

- *Path median query*: Given two vertices i, j, return the median weight on the path from i to j.
- *Path selection query*: Given two vertices i, j and a positive integer k, return the kth smallest weight on the path from i to j.
- *Path counting/reporting query*: Given two vertices i, j and a range $[a, b]$, count/report the vertices on the path from i to j whose weights are in this range.

The previous best data structure supporting these queries takes $O(n \log n)$ bits space and can perform *path median/selection/counting* in $O(\log \sigma)$ time and *path reporting* in $O(\log \sigma + occ \log \sigma)$ time, where *occ* represents the number of outputs [M. He, J.I. Munro, G. Zhou, Path queries in weighted trees, in: International Symposium on Algorithms and Computation, 2011, pp. 140–149]. We present a succinct data structure taking $n \log \sigma + 6n + o(n \log \sigma)$ bits space that can perform the above mentioned queries in $O(\log \sigma \log n)$ and $O(\log \sigma \log n + occ \log \sigma)$ time respectively.

1. Introduction and previous work

We consider the problem of succinctly representing a given vertex-weighted tree $\mathcal{T}$ of size n, whose vertices are labeled by integer weights from $\{1, 2, \ldots, \sigma\}$ and supporting the following path queries efficiently:

- *Path median query*: Given two vertices i and j, return the median weight on the path from i to j. If the number of vertices on this path is m, then the median weight is the $\lceil m/2 \rceil$th smallest weight among the weights of those vertices.
- *Path selection query*: Given two vertices i, j and a positive integer k, return the kth smallest weight on the path from i to j. If k is greater than the number of vertices on this path, return NULL. Thus path selection query is a generalization of path median queries for which $k = \lceil m/2 \rceil$.
- *Path counting/reporting query*: Given two vertices i, j and a range $[a, b]$, count/report the vertices on the path from i to j whose weights are in this range.

These queries are fundamental to numerous applications in computer science as different data types can be modeled and represented as weighted trees [15,3,1,8]. The path median problem was introduced by Krizanc et al. [15], where they

☆ This work is supported in part by US NSF Grants CCF-0621457 and CCF-1017623 (Rahul Shah).
* Corresponding author. Tel.: +1 225 578 4355; fax: +1 225 578 1465.
E-mail addresses: mpatil@csc.lsu.edu (M. Patil), rahul@csc.lsu.edu (R. Shah), thanks@csc.lsu.edu (S.V. Thankachan).

1570-8667/$ – see front matter
http://dx.doi.org/10.1016/j.jda.2012.08.003

proposed two solutions with the following space–time trade-offs: $O(n\log^3 n)$ bits space with $O(\log n)$ query time and $O(n\log_b n\log n)$ bits space with $O(b\log^3 n/\log b)$ query time, where $2 \leqslant b \leqslant n$. By choosing $b = n^{\epsilon}/\log n$ for some small positive constant ϵ, the index space can be made $O(n\log n)$ bits, where as the query time will be $O(n^{\epsilon})$. Recently He et al. [10] improved both the above results simultaneously by achieving $O(\log\sigma)$ query time in $O(n\log n)$ bits space. Here we note that $\sigma \leqslant n$.

The path counting problem was introduced by Chazelle in [3], where he proposed an $O(n\log n)$ bits data structure with $O(\log n)$ query time,[1] and the best known solution is by He et al. [10] with $O(\log\sigma)$ query time using the same space requirements. Note that when σ is significantly smaller than n, this new structure performs better. The path reporting problem was introduced by He et al. [10], where they proposed an $O(n\log n)$ bits data structure answering these queries in $O(\log\sigma)$ and $O(\log\sigma + occ\log\sigma)$ time respectively, where occ represents the size of output.

While the previous best data structure by He et al. [10] achieves faster query times, its space requirement is $O(n\log n)$ bits, which can be much larger than the optimal $n\log\sigma + 2n - O(\log n)$ bits needed for encoding $\mathcal{T}$, when σ is significantly smaller than n. In this paper, our objective is to present a more space efficient solution to this problem, which occupies $n\log\sigma + 6n + o(n\log\sigma)$ bits space and can answer *path median/selection/counting* queries in $O(\log\sigma\log n)$ time and *path reporting* queries in $O(\log\sigma\log n + occ\log\sigma)$ time respectively.

2. Preliminaries

2.1. Bit-vectors with rank/select support

The rank and select operations on a given bit-vector $\mathcal{B}$ of length n corresponding to $b = \{0, 1\}$, are defined as follows:

- $rank_b(\mathcal{B}, i)$ returns the number of b's $\mathcal{B}[1 \ldots i]$.
- $select_b(\mathcal{B}, j)$ returns the position of jth b in $\mathcal{B}$.

Lemma 1. *(See [13,4].) The rank and select operations can be performed in constant time by maintaining the bit-vector $\mathcal{B}$ in $n + o(n)$ bits.*

2.2. Succinct representation of ordinal trees

An ordinal (ordered) tree is a rooted tree in which the children of a vertex are ordered and specified by their rank.

Lemma 2. *(See [14,18,2,16].) Any n-vertex ordered tree can be maintained in $2n + o(n)$ bits, such that many operations including the following can be supported in constant time ($vertex\ i$ refers to the vertex with pre-order rank i):*

- parent(i) *returns the parent of vertex i,*
- lca(i, j) *returns the lowest common ancestor of two vertices i and j,*
- depth(i) *returns the numbers of vertices in the path from root to vertex i,*
- left-child(i)/right-child(i) *returns the pre-order rank of left/right child of vertex i.*

2.3. Heavy path

Let $\mathcal{T}$ be a tree with n vertices. We define the *size* of an internal vertex v to be the number of leaves in the subtree rooted at v. Then the *heavy path* of the tree $\mathcal{T}$ is the path starting with the root vertex, with each vertex v on the path being the largest-size child of its parent. The *heavy path decomposition* of the tree $\mathcal{T}$ is the operation where we decompose each off-path subtree of the heavy path recursively; as a result, the edges in $\mathcal{T}$ will be partitioned into disjoint heavy paths.

Lemma 3. *(See [9,5].) The path from the root of $\mathcal{T}$ to any vertex v traverses at most $O(\log n)$ heavy paths.*

2.4. Wavelet tree

Let $A[1 \ldots n]$ be an array of length n, where each element $A[i]$ is a symbol drawn from a set Σ of size σ. The *wavelet tree* (WT) [7] for A is an ordered balanced binary tree on Σ, where each leaf is labeled with a symbol in Σ, and the leaves are sorted alphabetically from left to right. Each internal vertex u represents an alphabet set Σ_u, and is associated with a bit-vector B_u. In particular, the alphabet set of the root is $\Sigma_{root} = \Sigma$, and the alphabet set of a leaf is the singleton set containing its corresponding symbol. Each vertex partitions its alphabet set among the two children (almost) equally, such

[1] In the problem formulated by Chazelle, the weights are associated to edges instead of vertices. The tree can be rooted with root chosen arbitrarily. Then, edge-weights and vertex-weights are equivalent as each vertex (except root) can be associated with a unique edge and vice versa.

that all symbols represented by the left child are lexicographically (or numerically) smaller than those represented by the right child. For the vertex u, let A_u be a subsequence of A by retaining only those symbols that are in Σ_u. Then B_u is a bit-vector of length $|A_u|$, such that $B_u[i] = 0$ if and only if $A_u[i]$ is a symbol represented by the left child of u. Indeed, the subtree from u itself forms a wavelet tree of A_u. To reduce space requirement, A_u's are not stored explicitly in the wavelet tree. Instead, we only store the bit-vectors B_u, each of which is augmented with Raman et al.'s scheme [13,4] to support constant-time bit-rank and bit-select operations. WT takes $n \log \sigma (1 + o(1))$ bits space and can answer the following queries efficiently:

Lemma 4. *(See [17, Section 2], [12, Theorem 1].) Given two intervals $[s, e]$ and $[a, b]$ as an online query, a wavelet tree on A can count the number of $A[x]$'s such that $s \leqslant x \leqslant e$ and $a \leqslant A[x] \leqslant b$ in $O(\log \sigma)$ time, and can report all such occ $A[x]$'s in $O((1 + occ) \log \sigma)$ time.*

Lemma 5. *(See [6, Theorem 2].) Given an interval $[s, e]$ and a parameter k, a wavelet tree on A can compute the kth smallest element in $A[s \ldots e]$ in $O(\log \sigma)$ time.*

Proof. The algorithm is recursive and takes $O(\log \sigma)$ steps. We may represent the query as a quadruple *(array, start point, end point, selection parameter)* and it is initialized to (A_u, s, e, k), where u is the root of WT. Let u' be the left and u'' be the right child of u. Then the range $[s, e]$ in A_u can be translated to $[s', e']$ in $A_{u'}$ and $[s'', e'']$ in $A_{u''}$ in constant time as follows: $s' = 1 + rank_0(B_{u'}, s - 1)$, $e' = rank_0(B_{u'}, e)$, $s'' = 1 + rank_1(B_{u''}, s - 1)$ and $e'' = rank_1(B_{u''}, e)$. Now if $k \leqslant e' - s' + 1$, then the desired answer corresponds to a leaf in the sub-tree of u' and we recurse the algorithm on $(A_{u'}, s', e', k)$, else we recurse on $(A_{u''}, s'', e'', k - (e' - s' + 1))$. The algorithm terminates when it reaches a leaf vertex. Since the tree depth is $O(\log \sigma)$ and at each level, it takes only constant time for *rank* operations, the query time can be bounded as $O(\log \sigma)$. We refer to [6] for a detailed description of the algorithm with an example.[2] □

The following is a generalization of Lemma 5.

Lemma 6. *Given t disjoint intervals $[s_1, e_1], [s_2, e_2], \ldots, [s_t, e_t]$ and a parameter k, a wavelet tree on A can compute the kth smallest element in $A[s_1, e_1] \cup A[s_2, e_2] \cup \cdots \cup A[s_t, e_t]$ in $O(t \log \sigma)$ time.*

Proof. The above described algorithm can be easily modified to handle the general case. Starting with $u = root$ and u', u'' as its left and right vertices: we translate the ranges $[s_i, e_i]$ in A_u to $[s'_i, e'_i]$ in $A_{u'}$ and $[s''_i, e''_i]$ in $A_{u''}$, for $i = 1, 2, \ldots, t$, in $O(t)$ time. Now if $k \leqslant \sum_{i=1}^{t} (e'_i - s'_i + 1)$, then recurse in $A_{u'}$ with the corresponding translated ranges, else recurse in $A_{u''}$ with the corresponding translated ranges and with $k - \sum_{i=1}^{t} (e'_i - s'_i + 1)$ instead of k. The algorithm terminates when it reaches a leaf vertex. It takes $O(t)$ time at each level of wavelet tree, and the tree depth is $O(\log \sigma)$. □

3. The data structure

This section will present the proposed data structure and query algorithm for answering different path queries efficiently along with their theoretical guarantees. Henceforth, given a tree $\mathcal{T}$, we refer to the vertices of the tree by their pre-order rank i.e. vertex i refers to the vertex with pre-order rank i in tree $\mathcal{T}$. Also we assume pre-order numbering begins with 1. We begin by introducing the notion of "reference" in tree $\mathcal{T}$, which has been subjected to heavy path decomposition.

- We say that vertex j refers to vertex i in tree $\mathcal{T}$, represented by $ref(j) = i$ if: (a) vertex i is on the same heavy path as that of vertex j, (b) depth of vertex i (as defined in Section 2.2) is minimum among all the vertices in the heavy path as that of vertex j. Therefore, i is the vertex closest to the root and is on the same heavy path as that of j.
- We define the reference count of vertex i, represented by rc_i, as the number of vertices in tree $\mathcal{T}$ that refers to vertex i i.e. number of vertices j with $ref(j) = i$. We note that for a given vertex i, $rc_i > 0$ if $ref(i) = i$, otherwise $rc_i = 0$, i.e., in a heavy path, the reference count of all the vertices except the shallowest one is 0.

3.1. Structure

We are now ready to describe our structure for supporting path queries. Given the tree $\mathcal{T}$ of n weighted vertices, we first perform the heavy path decomposition of $\mathcal{T}$ and then construct various components of proposed data structure described below.

1. $2n + o(n)$ bits representation of the structure of $\mathcal{T}$ supporting the basic navigational operations (Lemma 2) in constant time.

[2] The space bound of Theorem 2 in [6] is written as $O(n \log \sigma)$ bits, but it is indeed $n \log \sigma + o(n \log \sigma)$ bits.

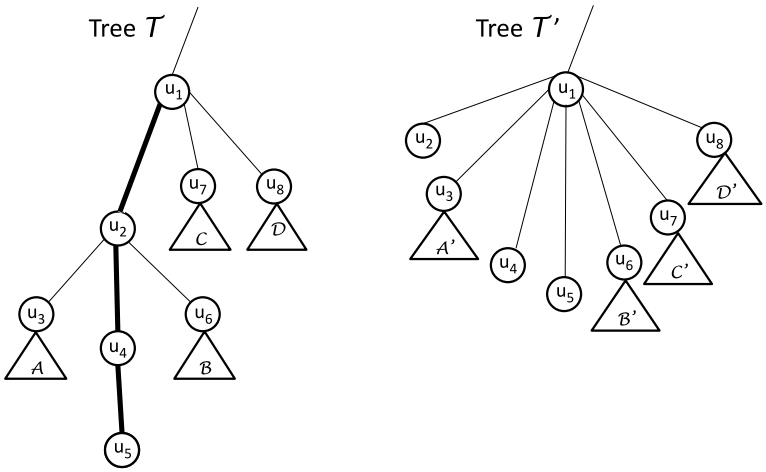

Fig. 1. Transforming $\mathcal{T}$ to $\mathcal{T}'$: The contiguous thick edges in $\mathcal{T}$ represent a heavy path.

2. A bit-vector $\mathcal{B} = 10^{rc_1}10^{rc_2}10^{rc_3}\ldots10^{rc_n}$ augmented with constant time rank/select data structures. As $\sum_{i=1}^{n} rc_i = n$, length of bit-vector $\mathcal{B}$ is $2n$ and augmented rank/select data structures only takes $o(n)$ bits space (Lemma 1). For any vertex i, its reference count can now be computed in constant time as: $rc_i = select_1(\mathcal{B}, i+1) - select_1(\mathcal{B}, i) - 1$.
3. To efficiently compute (constant time) $ref(i)$ for a given vertex i, without storing all the references explicitly, we maintain a tree $\mathcal{T}'$. Such a tree is obtained by simple transformation of $\mathcal{T}$ as follows (see Fig. 1 for an example).
For every vertex i (except root), we set $parent_{\mathcal{T}'}(i) = ref(parent_{\mathcal{T}}(i))$. Furthermore all children of a particular vertex in the transformed tree $\mathcal{T}'$ are arranged in the increasing order of their pre-order rank in the original tree $\mathcal{T}$ from left to right.
Note that this transformation guarantees the following properties:
• The pre-order rank of a vertex in $\mathcal{T}$ remains unchanged in $\mathcal{T}'$.
• For any vertex i in $\mathcal{T}$ with $ref(i) \neq i$, $ref(i)$ is the pre-order rank of parent of vertex i in $\mathcal{T}'$ i.e. $ref(i) = parent_{\mathcal{T}'}(i)$.
Therefore, for any given vertex i in $\mathcal{T}$, its reference $ref(i)$ can be computed in constant time as follows.

$$ref(i) = \begin{cases} i & \text{if } rc_i > 0, \\ parent_{\mathcal{T}'}(i) & \text{otherwise,} \end{cases}$$

where, reference count of vertex i (rc_i) can be computed using rank/select operations on $\mathcal{B}$ in constant time (explained in item 2). Just as we did with the original tree $\mathcal{T}$, we maintain the transformed tree $\mathcal{T}'$ using $2n + o(n)$ bits along with support for navigational operations.
4. Finally, we maintain the weights of vertices in the tree $\mathcal{T}$ as an array $\mathcal{C}[1\ldots n]$. Let $\mathcal{C}_i$ be the weight array for the vertex i, then array $\mathcal{C}$ is the concatenation of all weight arrays in the order of pre-order rank of vertices. i.e., $\mathcal{C} = \mathcal{C}_1\mathcal{C}_2\mathcal{C}_3\ldots\mathcal{C}_n$. For a vertex i, its weight array $\mathcal{C}_i$ is the collection of weights of all vertices j with $ref(j) = i$ ordered by their pre-order rank. Thus by definition $\mathcal{C}_i$ is empty if $rc_i = 0$, otherwise $\mathcal{C}_i = \mathcal{C}[1 + select_1(\mathcal{B}, i) - i, \ldots, select_1(\mathcal{B}, i) - i + rc_i]$.[3] We maintain array $\mathcal{C}$ as a wavelet tree, occupying $n\log\sigma + o(n\log\sigma)$ bits space.

Lemma 7. *The weight of a given vertex j is given by the following equation, where $i = ref(j)$.*

$$weight(j) = C\big[1 + select_1(B, i) - i + depth(j) - depth(i)\big].$$

Proof. Note that within $\mathcal{C}_i$, the weights are maintained from left to right in the increasing order of pre-order rank of the corresponding vertex. Hence the weight of i comes first (i.e., $weight(i) = \mathcal{C}_i[1] = \mathcal{C}[1 + select_1(\mathcal{B}, i) - i]$), followed by the weight of its child which is on the same heavy path and so on. Hence an entry corresponding to a vertex j comes in $\mathcal{C}_i$ only after the entries corresponding to all its ancestors which is on the same heavy path as that of j, and the number of such ancestors is given by $depth(j) - depth(i)$. □

Lemma 8. *The total space of our structure can be bounded by $n\log\sigma + 6n + o(n\log\sigma)$ bits.*

Proof. The data structure consists of $\mathcal{B}$ augmented with rank/select structures ($2n + o(n)$ bits), succinct encoding of $\mathcal{T}$ ($2n + o(n)$ bits), succinct encoding of $\mathcal{T}'$ ($2n + o(n)$ bits), and the wavelet tree of $\mathcal{C}$ ($n\log\sigma + o(n\log\sigma)$ bits). By putting all together, we obtain the above lemma. □

[3] The term $-i$ is to eliminate $1's$ from the bit-vector $\mathcal{B}$ corresponding to which there are no entries in $\mathcal{C}$.

3.2. Query answering

Before we describe how different path queries can be answered using the proposed data structure, we highlight an important property of array C which is used by the query algorithm.

- Given two vertices h, j in $\mathcal{T}$ such that vertex h is an ancestor of vertex j and both vertices belong to the same heavy path, weights of all the vertices on the path from h to j are maintained in a contiguous range $[l, \ldots, r]$ of array $\mathcal{C}$ with $i = ref(h) = ref(j)$, $l = select_1(\mathcal{B}, i) - i + depth(h) - depth(i)$ and $r = select_1(\mathcal{B}, i) - i + depth(j) - depth(i)$.

Let z and x be the two vertices in $\mathcal{T}$ such that vertex z is ancestor of vertex x. The path from vertex z to vertex x in $\mathcal{T}$ traverses at most $O(\log n)$ heavy paths (using Lemma 3). Now using the property described above, we can obtain a set S of $O(\log n)$ ranges over array $\mathcal{C}$ such that (a) any two ranges in S are disjoint, and (b) ranges in S together contain weight of only those vertices which appear on the path from vertex z to x in $\mathcal{T}$.

Obtaining such a set S in $O(\log n)$ time is the core part of query answering. We list the steps to be performed to achieve this objective.

- Initialize $j = x$ and $h = ref(j)$.
- Obtain a range for weights of vertices in the path from h to j using the property above, add it to our answer list.
- Update $j = parent(h)$ and $h = ref(j)$. Note that by this, we are jumping to another heavy path which overlaps with the path from z to x.
- Repeat the previous step until h is in the subtree of z i.e. $h \geqslant z$.
- Finally if $z \leqslant j$, add to our answer list a range covering the vertices on the path from z to j and exit.

As tree navigation operations as well as the operations involved in obtaining a range for given vertex pair (h, j) on the same heavy path i.e. *select*, *depth*, *ref* can be done on constant time, time complexity of the described routine can be bounded by $O(\log n)$.

Let i and j be the input vertices for path queries. If vertex i is an ancestor of vertex j in tree $\mathcal{T}$ or vice versa, we can obtain $O(\log n)$ ranges over array $\mathcal{C}$ covering weights of all the vertices on the path from i to j as described earlier. Otherwise we compute vertex $z = lca(i, j)$ and obtain two sets of ranges, one for vertex pair (z, i), the other for vertex pair (z, j) and merge them. Therefore different path queries on a tree can be converted to corresponding queries on an array $\mathcal{C}$ in $O(\log n)$ time, where the input consists of $t = O(\log n)$ disjoint intervals. Note that for the path median query, the number of vertices on the path from i to j is given by $depth(i) + depth(j) - 2 \times depth(lca(i, j)) + 1$. Combining these results with Lemmas 4 and 6, we have the following theorem.

Theorem 1. *There exists a succinct representation of $\mathcal{T}$ taking $n \log \sigma + 6n + o(n \log \sigma)$ bits of space which can perform path median/selection/count queries in $O(\log \sigma \log n)$ time and path report query in $O(\log \sigma \log n + occ \log \sigma)$ time, where occ is the number of outputs.* □

4. Concluding remarks

In this paper, we proposed a succinct representation of vertex-weighted trees supporting path queries efficiently. The main technique used here is heavy paths based tree transformation, by which the vertex weights are maintained in an array, such that weights on any tree path fall in $O(\log n)$ contiguous regions of this array. In previous best result [10], He et al. used a different approach, where the weight set is recursively partitioned and a tree structure is maintained for each of its subset. An input query is then answered by querying on at most $O(\log \sigma)$ tree structures. A very recent (concurrent) work by He et al. [11] provides an $nH(W_T) + o(n \log \sigma)$ bits solution, where $H(W_T)$ is the entropy of the multi-set of the weights of the vertices in $\mathcal{T}$. This index can support path counting queries in $O(\log \sigma / \log \log n + 1)$, path median/selection queries in $O(\log \sigma / \log \log \sigma + 1)$ and path reporting queries in $O((1 + occ) \log \sigma / \log \log n)$ time. Their result is optimal, for space as well as query time, when $\sigma = \Omega(n / poly \log n)$.

References

[1] N. Alon, B. Schieber, Optimal preprocessing for answering on-line product queries, Tech. Rep., Tel Aviv University, 1987.
[2] D. Benoit, E.D. Demaine, J.I. Munro, R. Raman, V. Raman, S.S. Rao, Representing trees of higher degree, Algorithmica 43 (4) (2005) 275–292.
[3] B. Chazelle, Computing on a free tree via complexity-preserving mappings, Algorithmica 2 (1987) 337–361.
[4] D.R. Clark, J.I. Munro, Efficient suffix trees on secondary storage (extended abstract), in: ACM–SIAM Symposium on Discrete Algorithms, 1996, pp. 383–391.
[5] R. Cole, M. Farach-Colton, R. Hariharan, T.M. Przytycka, M. Thorup, An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees, SIAM J. Comput. 30 (5) (2000) 1385–1404.
[6] T. Gagie, S.J. Puglisi, A. Turpin, Range quantile queries: another virtue of wavelet trees, in: International Conference on String Processing and Information Retrieval, 2009, pp. 1–6.
[7] R. Grossi, A. Gupta, J.S. Vitter, High-order entropy-compressed text indexes, in: ACM–SIAM Symposium on Discrete Algorithms, 2003, pp. 841–850.
[8] T. Hagerup, Parallel preprocessing for path queries without concurrent reading, Inform. and Comput. 158 (1) (2000) 18–28.

[9] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (2) (1984) 338–355.
[10] M. He, J.I. Munro, G. Zhou, Path queries in weighted trees, in: International Symposium on Algorithms and Computation, 2011, pp. 140–149.
[11] M. He, J.I. Munro, G. Zhou, Succinct data structures for path queries, in: European Symposium on Algorithms, 2012.
[12] W.-K. Hon, R. Shah, J.S. Vitter, Ordered pattern matching: towards full-text retrieval, Purdue University Tech. Rept. CSD TR 06-008, 2006.
[13] G. Jacobson, Space-efficient static trees and graphs, in: IEEE Symposium on Foundations of Computer Science, 1984, pp. 549–554.
[14] J. Jansson, K. Sadakane, W.-K. Sung, Ultra-succinct representation of ordered trees, in: ACM–SIAM Symposium on Discrete Algorithms, 2007, pp. 575–584.
[15] D. Krizanc, P. Morin, M.H.M. Smid, Range mode and range median queries on lists and trees, Nordic J. Comput. 12 (1) (2005) 1–17.
[16] H.-I. Lu, C.-C. Yeh, Balanced parentheses strike back, ACM Trans. Algorithms 4 (3) (2008), Article No. 28.
[17] V. Mäkinen, G. Navarro, Position-restricted substring searching, in: Latin American Symposium on Theoretical Informatics, 2006, pp. 703–714.
[18] J.I. Munro, V. Raman, Succinct representation of balanced parentheses and static trees, SIAM J. Comput. 31 (3) (2001) 762–776.