JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering, Vol. 13, No. 1, March 2019, pp. 11-16

Enumerating Subsequences of a Sequence and Paths of a Tree in the Order of Weight

Sung Kwon Kim*

School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea

skkim@cau.ac.kr

Abstract

The following two enumeration problems are addressed: (i) given a sequence A of n real numbers, the subsequences of A in the order of their weight (or sum), and (ii) given an n-vertex edge-weighted tree T, the paths of T in the order of their weight. We show that both enumerations (i) and (ii) can be done in $O(n^2 \log n)$ time and the data structures for (i) can be constructed in $O(n \log n)$ time using $O(n)$ space, and data structures for (ii) can be built in $O(n \log n)$ time using $O(n \log n)$ space.

Category: Algorithms and Complexity

Keywords: Algorithm; Enumeration; Subsequences of sequence; Paths of tree

I. INTRODUCTION

We assumed $A = (a_1,\ldots, a_n)$ and $B = (b_1,\ldots, b_n)$ as two sequences of length n with $a_i > 0$ and $b_i > 0$ for $1 \le i \le n$. Chen et al. [1] introduce the subsequence sum problem of two sequences: Given A, B, and a number $M > 0$, find all possible pairs of subsequences $(a_i,\ldots, a_j)$ and $(b_k,\ldots, b_l)$ for $i \le j$ and $k \le l$ such that their sum $a_i +\ldots + a_j + b_k + \ldots + b_l = M$. For bioinformatic application of this problem, refer to [1].

To develop algorithms for the problem, as in [1], the subsequences of A need to be enumerated in non-increasing order of their sum (in addition to those of B in non-decreasing order). Since A consists of positive numbers only, it is rather easy to develop an algorithm to enumerate the subsequences of A. The algorithm assumes $O(n^2 \log n)$ time using $O(n)$ space.

We now generalize the enumeration problem of the subsequences of A by allowing zeroes or negative numbers in A. Let $A = (a_1,\ldots, a_n)$ be a sequence of n real numbers. For $i \le j$, $(a_i,\ldots, a_j)$ is a *subsequence* of A and its *weight* is $w(i, j) = a_i + \ldots + a_j$. A problem we want to address is: enumeration of the subsequences of A in non-increasing order of their weight. In other words, we want to output the triplets in the multiset $[<i, j, w(i, j) > 1 \le i \le j \le n]$ one by one in the non-increasing order of $w(i, j)$ ([ ] denotes a multiset in which multiple instances of an element are allowed).

One simple solution is to sort the triplets based on their weight in non-increasing order and to output them starting from the first to the last. This solution obviously takes $O(n^2 \log n)$ time and $O(n^2)$ space as we need to store the multiset. Our problem is closely related to the problem of sorting $X + Y$ [2], for which finding an algorithm with time complexity $O(n^2 \log n)$ is open. Hence, as discussed in [3], reducing the time complexity to $O(n^2 \log n)$ seems quite difficult. We show instead that the space complexity can be reduced to $O(n)$. Our

Open Access http://dx.doi.org/10.5626/JCSE.2019.13.1.11 http://jcse.kiise.org

Received 15 January 2019; **Accepted** 17 February 2019

*Corresponding Author

algorithm takes $O(n \log n)$ time and $O(n)$ space to prepare data structures and, using them, we can enumerate the subsequences in $O(n^2 \log n)$ time.

We now extend the problem from sequences to trees. Assuming $T = (V, E)$ as an edge-weighted tree with $V = \{1,\ldots, n\}$, each edge $e \in E$ is associated with a real number w_e. For two vertices $u, v \in V$ with $u \neq v$, assuming $\pi(u, v)$ as the path connecting them, we defined its $w(u, v) = \sum_{e\in\pi(u,v)} w_e$.

We enumerated the paths of T in non-increasing order of their weight, i.e., to output the triplets in the multiset $[<u, v, w(u, v)> \mid u, v \in V, u < v]$ in the non-increasing order of weight. Only the case $u < v$ is considered $w(u, v) = w(v, u)$. As in the case of sequence, if we simply sort the weights in the multiset and output them in the sorted order, it again takes $O(n^2 \log n)$ time and $O(n^2)$ space. We show that data structures can be constructed in $O(n \log n)$ time using $O(n \log n)$ space, and demonstrate the enumeration using them in $O(n^2 \log n)$ time.

Enumeration of subsequences in a sequence is discussed in Section II, and enumeration of paths of a tree is presented in Section III. We concentrate on explaining how to enumerate the weights in the sorted order only. The subsequences or paths themselves can be enumerated via slight modification of our algorithms.

II. ENUMERATING SUBSEQUENCES OF A SEQUENCE

Given a sequence $A = (a_1,\ldots, a_n)$ of n real numbers, we want to enumerate the subsequences of A in the non-increasing order of their weight. As mentioned in Section I, we focus on describing an algorithm for the enumeration of weights in the sorted order. We defined $B = [w(i, j) \mid 1 \leq i \leq j \leq n]$ and $B_i = [w(i, j) \mid i \leq j]$ for $1 \leq i \leq n$, where B_i contains the weights of the subsequences of A whose left end is fixed at index i. To enumerate the weights in B, we employed the selection tree [4], which is used to merge k sorted lists into a single sorted list in $O(N \log k)$ time, where N is the total number of elements in the k lists.

Let H be a selection tree with n leaves, labelled $1,\ldots, n$. Each leaf i is associated with B_i for $1 \leq i \leq n$. In our selection tree, for each internal vertex, we compare the values of its two derivatives and stored the larger one as its value. Hence, the root contains the maximum values in the leaves.

To use the selection tree in enumerating the weights in B by combining $B_1,\ldots, B_n$, we need to overcome the following two obstacles:

- Each of $B_1,\ldots, B_n$, is unsorted.
- Storing all of $B_1,\ldots, B_n$ explicitly requires $O(n^2)$ space.

Therefore, we need to devise a method to generate the weights in each of $B_1,\ldots, B_n$ in sorted order using $O(n)$ space only.

```
nextweight($i$)
{
  while($p_i \leq n$)
  {
    $p_i = p_i + 1$;
    if $i \leq \alpha_{p_i}$, return $c_{\alpha_{p_i}} - c_{i-1}$;
  }
}
```

Fig. 1. Function nextweight(i).

We defined prefix sums of A, $c_0 = 0$ and $c_i = c_{i-1} + a_i$ for $1 \leq i \leq n$. Then, $w(i, j) = c_j - c_{i-1}$ for $1 \leq i \leq j \leq n$, sorted the prefix sums $(c_1,\ldots, c_n)$ in the non-increasing order and generated $(c_{\alpha_1},\ldots, c_{\alpha_n})$ as a sorted list. Note that $c_{\alpha_1} \geq \ldots \geq c_{\alpha_n}$ and $(\alpha_1,\ldots, \alpha_n)$ is a permutation of $(1,\ldots, n)$.

Since $B_i = [w(i, i),\ldots, w(i, n)] = [c_i - c_{i-1},\ldots, c_n - c_{i-1}] = [c_{\alpha_j} - c_{i-1} \mid 1 \leq j \leq n, i \leq \alpha_j]$, we have the lemma.

LEMMA 1. For $1 \leq i \leq n$, $B_i = [c_{\alpha_j} - c_{i-1} \mid 1 \leq j \leq n, i \leq \alpha_j]$.

Based on Lemma 1, we enumerate the weights in B_i in non-increasing order, while scanning $(\alpha_1,\ldots, \alpha_n)$ from $j = 1$ to n we determined if $i \leq \alpha_j$ and, if yes, the output $c_{\alpha_j} - c_{i-1}$. The weight output in this order exactly correspond to the weights in B_i in non-increasing order.

More specifically, let $\alpha_{n+1} = n + 1$ and $c_{\alpha_{n+1}} = -\infty$. Initially, $p_i = 0$. p_i points to the current position while scanning $(\alpha_1,\ldots, \alpha_n)$ for B_i. With $p_i = 0$, calling nextweight(i) in Fig. 1 increments of p_i until $i \leq \alpha_{p_i}$ and returns $c_{\alpha_{p_i}} - c_{i-1}$, which corresponds to the maximum of B_i. Another call to nextweight(i) finds the second maximum of B_i. By calling nextweight(i) repeatedly, we can determine the output of the weights in B_i in non-increasing order. Finally, when $p_i = n + 1$, nextweight(i) returns $-\infty$, which indicates that no weight remains in B_i. A call to nextweight(i) always returns the next weight in non-increasing order from B_i.

The selection tree H and the function nextweight(i) are employed to enumerate the weights in B in non-increasing order. Note that each leaf i of H is associated with B_i for $1 \leq i \leq n$. After setting $p_i = 0$ for all $1 \leq i \leq n$, we call nextweight(i) for each $1 \leq i \leq n$ to determine the maximum of B_i and assign it to leaf i of H. With all of the leaves of H now assigned weights, we compute the overall winner of the tournament of H, which corresponds to the maximum of B, as the output. Finding the maximum of B takes $O(n^2)$ time. If the winner comes from leaf i_0, we call nextweight(i_0) to obtain the next maximum from B_{i_0} and assign it to leaf i_0. The winner of this new tournament (all remain unchanged but leaf i_0 has a new weight), which corresponds to the second maximum of B, can be located in $O(\log n)$ time as in [4]. Thus, we enumerated the weights of B in non-increasing order as if we merged the n sorted lists. This takes $O(n^2 \log n)$ time as there are $O(n^2)$ weights in B.

The spaces needed are A, $(c_1,\dots, c_n)$, $(\alpha_1,\dots, \alpha_n)$, $(p_1,\dots, p_n)$ and H, each of which requires $O(n)$ space. Every data structure can be built in $O(n)$ time except that it takes $O(n \log n)$ time in sorting $(c_1,\dots, c_n)$ to get $(\alpha_1,\dots, \alpha_n)$.

THEOREM 1. *Given a sequence A of n real numbers, the subsequences of A can be enumerated in non-increasing order of their weight in $O(n^2 \log n)$ time. Our data structures for enumeration can be constructed in $O(n \log n)$ time using $O(n)$ space.*

III. ENUMERATING PATHS OF A TREE

Let $T = (V, E)$ be an edge-weighted tree with $V = \{1,\dots, n\}$ for $n \geq 3$. We want to enumerate the paths of T in the non-increasing order of their weight. Before describing our algorithm, we introduce a problem and its solution for the design of our algorithm. The *path weight query* problem is as follows: Preprocess T such that, after preprocessing, for any query pair of vertices $u, v \in V$, the path weight $w(u, v)$ can be answered quickly. Solutions for the problem with $O(n)$ preprocessing time and $O(1)$ query answering time are given by Harel and Tarjan [5], and Schieber and Vishkin [6]. So, we may assume that $w(u, v)$ for any $u, v \in V$ can be obtained in $O(1)$ time.

Again, we concentrate on explaining our process of enumeration of the weights of the paths in the sorted order. Let $B = [w(u, v) \mid u, v \in V, u < v]$, and let $B_u = [w(u, v) \mid u < v, v \in V]$ for $u \in V$. B_u contains the weights of paths whose one end vertex is fixed at u. Note that $\cup_{u\in V} B_u = B$.

To present an algorithm for enumerating the weights in B, we first introduce centroid decomposition of T, which hierarchically decomposes T into edges in a balanced manner. This hierarchical decomposition was used to compute canonical subsets of V, which are called canonical in the sense that $V - \{u\}$ for any $u \in V$ can be partitioned into $O(\log n)$ canonical subsets of V. This partition allows us to express the weights from a given vertex u to all vertices in $V - \{u\}$ as a union of $O(\log n)$ disjoint sorted lists (to be detailed later), from which we enumerated the weights in B_u using a selection tree in non-increasing order.

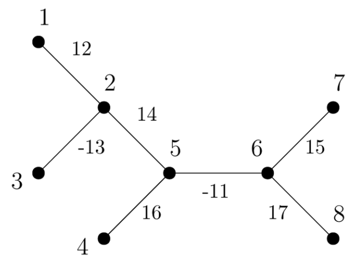

Fig. 2. An example tree T. T has only one centroid, 5. A centroid decomposition divides T into T_l and T_r such that T_l (T_r) is the subtree of T that has vertex set $V_l = \{1, 2, 3, 4, 5\}$ ($V_r = \{5, 6, 7, 8\}$).

Deleting a vertex and its adjacent edges from T leaves several connected components of T. A vertex is called a centroid of T if its deletion results in connected components of T, each of which has a maximum of $n/2$ vertices. A tree has either one or two centroids and if there are two centroids, they are adjacent [7]. A *centroid decomposition* of T consists of finding a centroid c of T and dividing T into two subtrees $T_l = (V_l, E_l)$ and $T_r = (V_r, E_r)$ such that $V_l \cap V_r = V$, $V_l \cap V_r = \{c\}$, $E_l \cup E_r = E$, and $\frac{n+2}{3} \leq |V_l| \leq \frac{2n+1}{3}$. A centroid decomposition of T can be computed in $O(n)$ time [8, 9] (Fig. 2).

We applied the centroid decomposition procedure recursively to subtrees T_l and T_r, until the subtree contains two vertices.

Remark: Though recursion ended here and no further decomposition was needed, for a later purpose a centroid of the two-vertex subtree was designated. One of the two vertices represented is a leaf of T, and selected as the centroid. As shown in Fig. 3, the centroid decompositions were applied at all levels of recursion to T in Fig. 2.

As shown in Fig. 3, the centroid decompositions of T can be modelled as a binary tree $CD(T)$. Each node (i.e., it is used in $CD(T)$ to distinguish it from vertex in T) q of $CD(T)$ represents a subtree of T, denoted $T(q)$, and a centroid of $T(q)$, denoted $c(q)$, and its left and right children represent two subtrees (corresponding to T_l and T_r) of $T(q)$ generated by the centroid decomposition of $T(q)$ with respect to $c(q)$. If we ignore all of the edges of the subtrees in Fig. 3, $CD(T)$ can be abstracted as in Fig. 4, in which each node q is associated with a vertex set $V(q)$ and a centroid $c(q)$ such that $V(q)$ is the vertex set of $T(q)$.

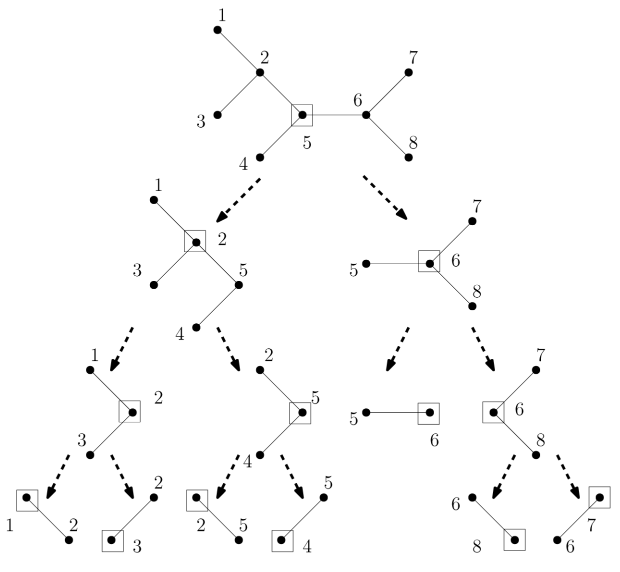

Fig. 3. Centroid decompositions applied to the tree in Fig. 2, modelled as binary tree $CD(T)$. Vertices in squares denote centroids.

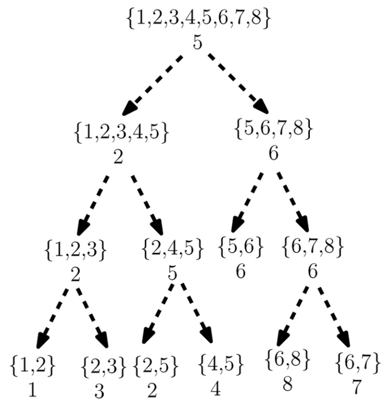

Fig. 4. Abstracted version of *CD*(*T*) in Fig. 3. Each node *q* is associated with a vertex set *V*(*q*) and a centroid *c*(*q*).

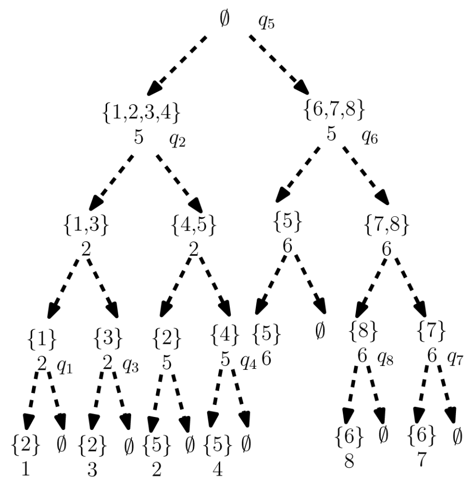

Fig. 5. Binary tree *CD'*(*T*) with canonical subsets of *V*. *W*(*q*) and *d*(*q*) only are shown. All q_u's are also given.

Notation: For a node q, p(q), l(q), r(q) and s(q) denote the parent, left child, right child and sibling node of q, respectively.

We copied $CD(T)$ and modified it to construct another binary tree. For each non-root node q of $CD(T)$, the following values were set: $d(q) = c(\mathrm{p}(q))$ and $W(q) = V(q) - \{d(q)\}$. For the root q of $CD(T)$, we set $W(q) = \emptyset$. For each leaf node q of $CD(T)$, two child nodes were created: $\mathrm{l}(q)$ and $\mathrm{r}(q)$ and set $d(\mathrm{l}(q)) = c(q)$, $W(\mathrm{l}(q)) = V(q) - \{d(\mathrm{l}(q))\}$ and $W(\mathrm{r}(q)) = \emptyset$. $d(q)$ is undefined for nodes q with $W(q) = \emptyset$. Assuming that $CD'(T)$ denote this new binary tree, $CD'(T)$ can be constructed in $O(n \log n)$ time and $O(n \log n)$ space because its height is $O(\log n)$. In Fig. 5, $CD'(T)$ is shown.

$W(q)$'s are canonical subsets of V mentioned earlier. Later, $d(q)$ serves as a source vertex in computing the weights of all destination vertices in $W(q)$. From the definition of $V(q)$ and $W(q)$ we have the following lemma.

Lemma 2. *If q is a non-root node of $CD'(T)$, then $W(q) \uplus V(\mathrm{s}(q)) = V(\mathrm{p}(q))$.*

$\uplus$ is the disjoint union operator that unions two disjoint sets.

We start explaining how $V - \{u\}$ for any vertex $u \in V$ can be expressed as a union of $O(\log n)$ disjoint canonical subsets of V. Each vertex of T appears as a centroid in $CD'(T)$. Since our centroid decompositions, as shown in Fig. 3, we started with T and finally decomposed it into edges of T, each non-leaf vertex of T must be used as a centroid of a centroid decomposition. As enforced in Remark, each leaf vertex of T was selected as a centroid at a leaf node of $CD(T)$.

Lemma 3. *For each vertex $u \in V$, there is a node q of $CD'(T)$ such that $u = c(q)$.*

For each vertex $u \in V$, q_u is defined as the node q of $CD'(T)$ such that $u = c(q)$, i.e., u is a centroid of $T(q)$. If there are two or more nodes q such that $u = c(q)$, we selected q_u as the one that is the closest to the root of $CD'(T)$ in terms of the number of edges between them in $CD'(T)$. All q_u for $u \in V$ are shown in Fig. 5.

Considering a path from q_u to the root of $CD'(T)$, $(p_1^u, \ldots, p_{m_u}^u)$ represents the path such that $p_1^u = q_u$, $p_i^u = \mathrm{p}(p_{i-1}^u)$ for $2 \le i \le m_u$, $p_{m_u}^u$ is the root of $CD'(T)$, and m_u is the number of nodes involved. In Fig. 5, $m_3 = 4$ and $m_6 = 2$.

By Lemma 2, we have

$$\begin{aligned} V(p_2^u) &= V(p_1^u) \uplus W(\mathrm{s}(p_1^u)), \\ V(p_3^u) &= V(p_2^u) \uplus W(\mathrm{s}(p_2^u)), \\ &\vdots \\ V(p_{m_u}^u) &= V(p_{m_u-1}^u) \uplus W(\mathrm{s}(p_{m_u-1}^u)). \end{aligned}$$

Since $V = V(p_{m_u}^u)$, we obtain $V = V(p_1^u) \uplus W(\mathrm{s}(p_1^u)) \uplus \ldots \uplus W(\mathrm{s}(p_{m_u-1}^u))$. Note that $V(p_1^u) = W(\mathrm{l}(p_1^u)) \uplus W(\mathrm{r}(p_1^u)) \uplus \{u\}$. Thus, $V - \{u\} = W(\mathrm{l}(p_1^u)) \uplus W(\mathrm{r}(p_1^u)) \uplus W(\mathrm{s}(p_1^u)) \uplus \ldots \uplus W(\mathrm{s}(p_{m_u-1}^u))$.

For each $u \in V$, define

$$W_1^u = W(\mathrm{l}(p_1^u)) \cup W(\mathrm{r}(p_1^u))$$

and for $2 \le i \le m_u$,

$$W_i^u = W(\mathrm{s}(p_{i-1}^u)).$$

In Fig. 5, if $u = 3$, then $W_1^3 = \{2\}$ as $W(\mathrm{l}(p_1^3)) = \{2\}$ and $W(\mathrm{r}(p_1^3)) = \emptyset$, $W_2^3 = \{1\}$, $W_3^3 = \{4, 5\}$, and $W_4^3 = \{6, 7, 8\}$; if $u = 6$, then $W_1^6 = \{5\} \cup \{7, 8\}$ and $W_2^6 = \{1, 2, 3, 4\}$.

Since $V - \{u\} = \uplus_{i=1}^{m_u} W_i^u$ and $m_u = O(\log n)$, we derived the following:

Lemma 4. *For each $u \in V$, $V - \{u\}$ can be expressed as a union of $O(\log n)$ disjointed canonical subsets.*

Lemma 4 allows us to develop a method to express the weights from a given vertex u to all the vertices $v \in V - \{u\}$

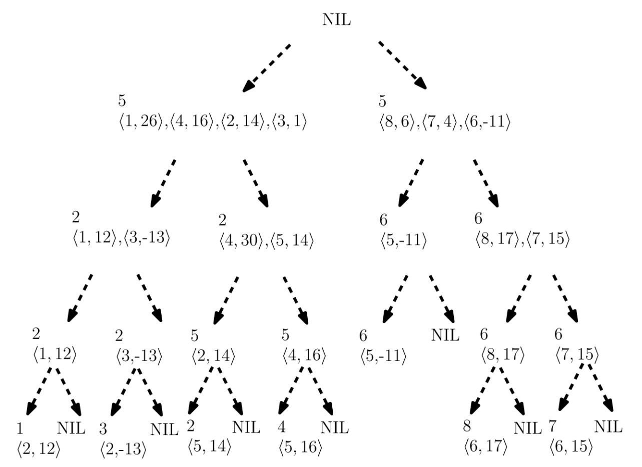

Fig. 6. Each node *q* is associated with *d*(*q*) and *L*(*q*). The nodes with *W*(*q*) = $\emptyset$ have NIL. Each node *v* ∈ *W*(*q*) appears in *L*(*q*) as an ordered pair <*v*, *w*(*d*(*q*), *v*)>.

as a union of *O*(log n) disjoint sorted lists.

For each node *q* of *CD*′(*T*) with *W*(*q*) ≠ $\emptyset$, the *w*(*d*(*q*), *v*) and the weight from *d*(*q*) to *v* were computed, and created an ordered pair <*v*, *w*(*d*(*q*), *v*)> for all vertices *v* ∈ *W*(*q*). Let *L*(*q*) = {<*v*, *w*(*d*(*q*), *v*)> | *v* ∈ *W*(*q*)} be a list of these ordered pairs sorted on *w*(*d*(*q*), *v*) in non-increasing order. *L*(*q*) = *NIL* if *W*(*q*) = $\emptyset$. Computing *L*(*q*) for all nodes *q* of *CD*′(*T*) can be done in *O*(*n* log *n*) time using *O*(*n* log *n*) space [10]. In Fig. 6, *d*(*q*) and *L*(*q*) for all nodes *q* of *CD*′(*T*) are shown.

For each *u* ∈ *V*, define

$$L_1^u = merge(L(\mathrm{l}(p_1^u)), L(\mathrm{r}(p_1^u)))$$

and for $2 \le i \le m_u$,

$$L_i^u = L(\mathrm{s}(p_{i-1}^u))$$

where the function *merge* merges the two sorted lists. Remember that in L_i^u we can find the weights from $d(p_i^u)$ to all other vertices *v* in W_i^u.

Given a vertex *u* ∈ *V*, consider a path *π*(*u*, *v*) from *u* to a vertex *v* ∈ *V* − {*u*}. Based on Lemma 4, we assume that $v \in W_i^u$ for some $1 \le i \le m_u$. The path starts from *u*, passes through $d(p_i^u)$, and arrives at *v*. In other words, *π*(*u*, *v*) is $\pi(u, d(p_i^u))$ followed by $\pi(d(p_i^u), v)$, and thus, $w(u, v) = w(u, d(p_i^u)) + w(d(p_i^u), v)$. For *i* = 1, $\pi(u, d(p_1^u)) = \emptyset$ as $u = d(p_1^u)$.

For example, if *u* = 3 and $v = 8 \in W_4^3$, then *π*(3, 8) = *π*(3, 5) ∪ *π*(5, 8) as $d(p_4^3) = 5$, and *w*(3, 8) = *w*(3, 5) + *w*(5, 8) = 1 + 6.

Since $w(d(p_i^u), v)$ for all $v \in W_i^u$ are stored in L_i^u in the sorted order and $w(u, d(p_i^u))$ is irrelevant to *v*, if we need an enumeration of the weights *w*(*u*, *v*) from *u* to all $v \in W_i^u$ in non-increasing order, we can obtain it by adding $w(u, d(p_i^u))$ to every weight $w(d(p_i^u), v)$ in L_i^u. Considering all of *i*, $1 \le i \le m_u$, we obtained the weights *w*(*u*, *v*) from *u* to all vertices *v* ∈ *V* − {*u*} in the form of m_u enumerations, one for each *i*, by Lemma 4. Table 1 shows the cases when *u* = 3 and *u* = 6.

Table 1. For *u* = 3 and *u* = 6, how X_i^u is related with M_i^u

	$d(p_i^u)$		*v*	$w(u, d(p_i^u))$		$w(d(p_i^u), v)$	*w*(*u*, v)
u = 3	3	X_1^3	2	*w*(3, 3) = 0	M_1^3	*w*(3, 2) = -13	-13
	2	X_2^3	1	*w*(3, 2) = -13	M_2^3	*w*(2, 1) = 12	-1
	2	X_3^3	4	*w*(3, 2) = -13	M_3^3	*w*(2, 4) = 30	17
			5			*w*(2, 5) = 14	1
	5	X_4^3	8	*w*(3, 5) = 1	M_4^3	*w*(5, 8) = 6	7
			7			*w*(5, 7) = 4	5
			6			*w*(5, 6) = -11	-10
u = 6	6	X_1^6	8	*w*(6, 6) = 0	M_1^6	*w*(6, 8) = 17	17
			7			*w*(6, 7) = 15	15
			5			*w*(6, 5) = -11	-11
	5	X_2^6	1	*w*(6, 5) = -11	M_2^6	*w*(5, 1) = 26	15
			4			*w*(5, 4) = 16	5
			2			*w*(5, 2) = 14	3
			3			*w*(5, 3) = 1	-10

We are ready to describe how to enumerate the weights in *B*. Remember that *B* = [*w*(*u*, *v*) | *u*, *v* ∈ *V*, *u* < *v*], and B_u = [*w*(*u*, *v*) | *u* < *v*, *v* ∈ *V*] for *u* ∈ *V*. We first show how to enumerate the weights in B_u for *u* ∈ *V*, and then explain a method of enumerating the weights in *B*. For *u* ∈ *V*, let H_u be a selection tree with m_u leaves, each of which is associated with L_i^u for $1 \le i \le m_u$. For each $1 \le i \le m_u$, we scan L_i^u in non-increasing order of weight to find the first pair $< \hat{v}_i, w(d(p_i^u), \hat{v}_i) >$ with $u < \hat{v}_i$, add $w(u, d(p_i^u))$ to its weight, and store the sum $w(u, d(p_i^u)) + w(d(p_i^u), \hat{v}_i) = w(u, \hat{v}_i)$ at leaf *i* of H_u.

Note that $w(u, \hat{v}_i)$ is the maximum of $\{w(u, v) \mid v \in W_i^u, u < v\}$.

With m_u weights stored at the leaves of H_u, we conducted a tournament competition to determine the winner and output, which corresponds to the maximum of B_u. If the winner comes from leaf i_0, we scan $L_{i_0}^u$ starting at the position where the previous pair was located in the next pair $< \hat{v}_{i_0}, w(d(p_{i_0}^u), \hat{v}_{i_0}) >$ with $u < \hat{v}_{i_0}$ and replace the weight stored at leaf i_0 of H_u by the sum $w(u, d(p_{i_0}^u)) + w(d(p_{i_0}^u), \hat{v}_{i_0})$. The winner of H_u was recomputed and outputted, which corresponds to the second maximum of B_u. Thus, with the help of H_u, we can enumerate all of the weights in B_u in non-increasing order.

We now build another selection tree *H* with *n* leaves, labeled 1,…, *n*, on top of the selection trees H_1,…, H_n, where each leaf *u* of *H* is linked to the root of H_u for *u* ∈ *V*. As described above, each H_u outputs the weights in B_u in non-increasing order. *H* can be used to enumerate the weights in *B* in non-increasing order by merging the *n* sorted lists that are the output from H_1,…, H_n. Since a weight in L_i^u at leaf i of H_u reaches the root of H in $O(\log n+\log m_u)$ time, the enumeration of B can be accomplished in $O(n^2 \log n)$ time.

For analysis of the constructed data structures, the computation of $CD'(T)$ and $(d(q), L(q))$ of all the nodes q of $CD'(T)$ can be done in $O(n \log n)$ time using $O(n \log n)$ space [10]. Building $H_1,\ldots, H_n$, and H also requires $O(n \log n)$ time and $O(n \log n)$ space.

Theorem 2. *Given an n-vertex edge-weighted tree T, the paths of T can be enumerated in a non-increasing order of their weight in $O(n^2 \log n)$ time. Our data structures for enumeration can be constructed in $O(n \log n)$ time using $O(n \log n)$ space.*

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. NRF-2015R1D1A1A01059937).

REFERENCES

1. T. Chen, J. D. Jaffe, and G. M. Church, "Algorithms for identifying protein cross-links via tandem mass spectrometry," in *Proceedings of the 5th Annual International Conference on Computational Biology*, Montreal, Canada, 2001, pp. 95-102.
2. The Open Problems Project, "Problem 41: Sorting X + Y (pairwise sums)," 2017; http://cs.smith.edu/~jorourke/TOPP/P41.html.
3. A. H. Barrera, "Finding an O(n^2 log n) algorithm is sometimes hard," in *Proceedings of the 8th Canadian Conference on Computational Geometry*, Ottawa, Canada, 1996, pp. 289-294.
4. E. Horowitz, S. Sahni, and S. Anderson-Freed, *Fundamentals of Data Structures in C*, 2nd ed. Summit, NJ: Silicon Press, 2008.
5. D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM Journal on Computing*, vol. 13, no. 2, pp. 338-355, 1984.
6. B. Schieber and U. Vishkin, "On finding lowest common ancestors: simplification and parallelization," *SIAM Journal on Computing*, vol. 17, no. 6, pp. 1253-1262, 1988.
7. D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms.* Reading, MA: Addison-Wesley, 1973.
8. N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran, "An O(n log^2 n) algorithm for the kth longest path in a tree with applications to location problems," *SIAM Journal on Computing*, vol. 10, no. 2, pp. 328-337, 1981.
9. B. Y. Wu, C. Kun-Mao, and C. Y. Tang, "An efficient algorithm for the length-constrained heaviest path problem on a tree," *Information Processing Letters*, vol. 69, no. 2, pp. 63-67, 1999.
10. G. N. Frederickson and D. B. Johnson, "Finding kth paths and p-centers by generating and searching good data structures," *Journal of Algorithms*, vol. 4, no. 1, pp. 61-80, 1983.

Sung Kwon Kim

He received his bachelor's degree from Seoul National University, Korea, his master's degree from Korea Advanced Institute of Science and Technology (KAIST), Korea, and his Ph.D. degree from University of Washington, Seattle, WA, USA. He is currently a professor at School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea. His research interest includes algorithms, computational geometry, and recommendation systems.