IEICE TRANS. INF. & SYST., VOL.E96–D, NO.3 MARCH 2013

LETTER Special Section on Foundations of Computer Science

Linear-Time Algorithm for the Length-Constrained Heaviest Path Problem in a Tree with Uniform Edge Lengths*

Sung Kwon KIM[†a)], ***Member***

SUMMARY Given a tree T with edge lengths and edge weights, and a value B, the length-constrained heaviest path problem is to find a path in T with maximum path weight whose path length is at most B. We present a linear time algorithm for the problem when the edge lengths are uniform, i.e., all one. This algorithm with slight modification can be used to find the heaviest path of length exactly B in T in linear time.

key words: *length-constrained paths, heaviest paths, uniform edge lengths*

1. Introduction

Let $T = (V, E)$ be an undirected tree with $|V| = n$. Each edge $e \in E$ is associated with two real numbers, $l(e)$ and $w(e)$, called its *length* and *weight*, respectively. Let $\pi(u, v)$ denote the path between two vertices $u, v \in V$. The *length* of $\pi(u, v)$ is the sum of the lengths of the edges in it, $l(\pi(u, v)) = \sum_{e\in\pi(u,v)} l(e)$, and the *weight* of $\pi(u, v)$ is the sum of the weights of the edges in it, $w(\pi(u, v)) = \sum_{e\in\pi(u,v)} w(e)$.

The *length-constrained heaviest path* problem is: Given an undirected tree T with edge lengths $l(e)$ and edge weights $w(e)$, and a real number B, find the heaviest path of length at most B in T, namely, the path π such that

$$w(\pi) = \max_{u,v\in V}\{w(\pi(u, v)) \mid l(\pi(u, v)) \le B\}.$$

Wu et al. [8] introduced the length-constrained heaviest path problem and solved it in $O(n \log^2 n)$ time, and later, Bhattacharyya and Dehne [1] improved the time complexity by showing an $O(n \log n)$ time algorithm.

We are interested in the case where the edge lengths are *uniform*. In the case of uniform edge lengths, $l(e) = 1$ for all $e \in E$ and B is a positive integer. We present an $O(n)$ time algorithm for finding the heaviest path of length at most B in a tree with uniform edge lengths. Both algorithms [1], [8] for the nonuniform edge length case are divide-and-conquer ones, and they cannot solve the uniform edge length case in linear time. A modified version of our algorithm can find the heaviest path of length exactly B in $O(n)$ time if such a path exists.

Manuscript received March 5, 2012.
Manuscript revised June 6, 2012.

[†]The author is with School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea.

[*]This research was supported by the Chung-Ang University research grant in 2013 and by Basic Science Research Program through the NRF funded by the Ministry of Education, Science and Technology (No. 2012008222).

a) E-mail: skkim@cau.ac.kr

DOI: 10.1587/transinf.E96.D.498

2. Preliminaries

We review data structures due to Frederickson [2], which play a critical role in designing our algorithm. Let T be a binary tree with n vertices. A *vertex cluster* (for short, *cluster*) of T is a set of vertices that induces a connected subtree of T. Two clusters are *adjacent* if there is an edge of T that connects them. The *external degree* of a cluster is the number of its adjacent clusters.

Let $2 \le z \le n$ be an integer. A *restricted partition of order* z for T is a partition of the vertex set V of T into clusters such that[**]

1. Each cluster in the partition has external degree at most 3.
2. Each cluster of external degree 3 contains exactly one vertex.
3. Each cluster of external degree 1 or 2 contains at most z vertices.
4. No two adjacent clusters can be combined, still satisfying the above.

Frederickson [2] showed that a restricted partition of order z for T consists of $\Theta(n/z)$ clusters and can be computed in $O(n)$ time.

A *restricted multilevel partition of order* z for T is a hierarchical series of partitions of V satisfying the following:

1. The clusters at level 0 form a restricted partition of order z of V.
2. Each cluster at level $i > 0$ is either (i) a copy of a cluster at level $i-1$ or (ii) the union of two clusters at level $i-1$.
3. There is precisely one cluster at the topmost level h, which contains all vertices of V.

To accomplish 2 above, contract each of the clusters at level $i - 1$ into a single vertex to get a "contracted" tree of T, and compute a restricted partition of order 2 for this tree. Each cluster in this restricted partition contains either one "contracted" vertex or two "contracted" vertices, which respectively correspond to (i) and (ii). Frederickson [2] proved that a restricted multilevel partition of order z for T can be constructed in $O(n)$ time and has $\Theta(\log(n/z))$ levels, i.e., $h = \Theta(\log(n/z))$.

A restricted multilevel partition for T naturally gives a rooted binary tree, called a *topology tree* for T, in which all leaf nodes are at the same depth, such that:

[**]We follow the definition given in [4].

1. A node[†] at level i[††] in the topology tree represents a cluster at level i in the restricted multilevel partition.
2. A node at level $i \geq 1$ has at most two children that represent the clusters at level $i-1$ whose union is the cluster it represents.

It is obvious that a topology tree for T is of height $\Theta(\log(n/z))$ and has $\Theta(n/z)$ leaf nodes.

Besides the Frederickson's data structures above, a solution for the following *path weight query* problem is employed in the design of our algorithm: For an edge-weighted tree T, preprocess T so that, after preprocessing, given a query pair of vertices u, v, the path weight $w(\pi(u, v))$ can be found in $O(1)$ time. A solution for the problem consists of two parts: preprocessing and query-answering.

The following solution is well-known, whose preprocessing works as follows: (1) Compute $w(\pi(x, v))$ for each vertex v of T, where x is the root of T. (2) Preprocess T so that, given a query pair of vertices u, v of T, their lowest common ancestor can be found in $O(1)$ time. Given two vertices u and v, the query-answering finds their lowest common ancestor y and computes $w(\pi(u, v)) = w(\pi(x, u)) + w(\pi(x, v)) - 2 \cdot w(\pi(x, y))$.

Since solutions with $O(n)$ preprocessing time and $O(1)$ query answering time for the lowest common ancestor problem are given in [3], [5], the path weight query problem can be solved within the same time complexity. Similarly, the *path length query* problem can be defined and it can also be solved in the same way as above.

3. The Algorithm

Since our algorithm is designed to handle binary trees only, we first show how to transform an undirected tree into a binary tree. Given an edge-weighted tree T_0 with uniform edge lengths, a binary tree T is obtained as follows [6]: Select an arbitrary vertex x of degree 1 of T_0 and regard T_0 as a rooted tree whose root is x. For each vertex of v of T_0 with $d \geq 3$ children, $v_1, \ldots, v_d$, replace v by a path $(v, u_2, \ldots, u_{d-1})$, where $u_2, \ldots, u_{d-1}$ are new vertices and $(v, u_2), (u_2, u_3), \ldots, (u_{d-2}, u_{d-1})$ are new edges. Each e of these new edges has $l(e) = w(e) = 0$. Replace the edges $\{(v, v_i) \mid 2 \leq i \leq d-1\}$ by the edges $\{(u_i, v_i) \mid 2 \leq i \leq d-1\}$ of corresponding weights, and replace the edge (v, v_d) by the edge (u_{d-1}, v_d) of corresponding weight. Each e of these edges has $l(e) = 1$.

Let $T = (V, E)$ be the resulting binary tree. The edge lengths in T are binary, i.e., $l(e) \in \{0, 1\}$. T has $O(|T_0|)$ vertices and edges [6], and every path in T_0 has a corresponding path in T with same length and weight, and vice versa [1].

To utilize topology tree for our purpose of computing the heaviest path of length at most B in T, compute a restricted multilevel partition of order $z = B + 1$ for T and build its topology tree. Let $T^* = (V^*, E^*)$ be the topology tree. (1) T^* is a rooted binary tree of height $h = \Theta(\log(n/B))$ and has $\Theta(n/B)$ leaf nodes. (2) Each leaf node of T^* represents a cluster with at most B vertices.

For each $p \in V^*$, let V_p be the vertex cluster it represents and let T_p be the subtree of T induced by V_p. If p has two children q and r in T^*, then $V_p = V_q \cup V_r$ and $T_p = T_q \cup T_r \cup \{(b, c)\}$, where (b, c) for $b \in V_q$ and $c \in V_r$ is the edge connecting T_q and T_r. b and c are called the *connectors* of T_q and T_r, respectively. If p has a single child q, then $V_p = V_q$ and $T_p = T_q$. In this case, we define the connector of T_q to be the connector of T_p. For every nonroot node $p \in V^*$, T_p has one connector.

For a nonroot node $p \in V^*$ and a vertex $a \in V$, define $A_p^a[\cdot]$ to be an array of length $B + 1$ such that for $0 \leq i \leq B$,

$$A_p^a[i] = \max_{v \in V_p}\{w(\pi(a, v)) \mid l(\pi(a, v)) = i\}.$$

In other words, $A_p^a[i]$ is the weight of the heaviest one among the paths of length i from a to the vertices of V_p. If there is no path of length i from a to any vertex of V_p, then $A_p^a[i] = -\infty$. For notational convenience, $A_p[\cdot]$ is used for $A_p^a[\cdot]$ if a is the connector of T_p.

Suppose for a while that the arrays $A_p[\cdot]$ for all nonroot nodes $p \in V^*$ have been computed and therefore are available for reference. We show that the heaviest path of length at most B in T can be found in $O(n)$ time. For simplicity of explanation, our algorithm is described to find only the weight of the heaviest path of length at most B, but not the path. An easy modification of our algorithm can find the path itself.

Traverse T^* in postorder and compute, for each $p \in V^*$, the weight w_p of the heaviest path of length at most B in T_p. Let α denote the root of T^*. Then, w_α is the weight we want to compute as $T_\alpha = T$.

If $p \in V^*$ is a leaf node, then find the heaviest path π (without any length constraint) in T_p and set $w_p = w(\pi)$. An algorithm for finding the longest path in trees (see, e.g., [7]) can be used to find π in $O(B)$ time as T_p has at most B vertices.

If p has a single child q in T^*, then set $w_p = w_q$.

Consider the case where p has two children q and r in T^*. Let $b \in V_q$ and $c \in V_r$ be the connectors of T_q and T_r, respectively, and let $e = (b, c)$, which is the edge that connects T_q and T_r. Since $T_p = T_q \cup T_r \cup \{e\}$, and w_q and w_r have already been computed, we need to compute w' only, the weight of the heaviest path of length at most B in T_p that contains e.

$A_q[\cdot]$ and $A_r[\cdot]$ are used in computing w'. Compute the "prefix maxima" array of $A_r[\cdot]$, defined as $\hat{A}[i] = \max\{A_r[j] \mid 1 \leq j \leq i\}$ for $0 \leq i \leq B$. If $l(e) = 0$, then $w' = \max\{A_q[i] + \hat{A}[B - i] \mid 0 \leq i \leq B\}$, and if $l(e) = 1$, then $w' = w(e) + \max\{A_q[i] + \hat{A}[B - i - 1] \mid 0 \leq i \leq B - 1\}$. In either case, it takes $O(B)$ time. Then, $w_p = \max\{w_q, w_r, w'\}$.

Since T^* has $\Theta(n/B)$ nodes and each computation of w_p for $p \in V^*$ takes $O(B)$ time, we can compute w_p for all nodes p of T^* in $O(n)$ time.

[†]To make a distinction, nodes are used for a topology tree.

[††]Assume that the leaf nodes are at level 0 and a node is at level $i > 0$ if its children are at level $i - 1$.

Lemma 1: If the arrays $A_p[\cdot]$ for all nodes p of T^* are available, then w_α can be computed in $O(n)$ time.

We now explain how to compute in $O(n)$ time the arrays $A_p[\cdot]$ for all $p \in V^*$ except the root. Again, T^* is traversed in postorder.

If p is a leaf in T^*, then T_p consists of at most B vertices and therefore $A_p[\cdot]$ can be computed in $O(B)$ time: (1) Set $A_p[i] = -\infty$ for $0 \le i \le B$. (2) Visit the vertices of T_p in preorder starting at a, the connector of T_p. (3) For each currently visited vertex v, set $A_p[l(\pi(a,v))] = \max\{w(\pi(a,v)), A_p[l(\pi(a,v))]\}$. Note that each $w(\pi(a,v))$ and each $l(\pi(a,v))$ can be found in $O(1)$ time after the preprocessings of the path weight query and path length query are done with respect to T in $O(n)$ time as explained in Sect. 2.

If p has a single child q in T^*, then set $A_p[\cdot] = A_q[\cdot]$.

Suppose that a nonroot node p has two children q and r in T^*. In this case, $T_p = T_q \cup T_r \cup \{e\}$, where $e = (b, c)$ such that b and c are the connectors of T_q and T_r, respectively. $A_p[\cdot]$ is to be computed in T_p with respect to a, the connector of T_p. At this point, the arrays for all descendants of p in T^* have been computed and are stored at their corresponding nodes. Without loss of generality, assume that $a \in V_q$. We compute $A^a_q[\cdot]$ and $A^a_r[\cdot]$ separately and then obtain $A_p[\cdot]$ by computing $A_p[i] = \max\{A^a_q[i], A^a_r[i]\}$ for $0 \le i \le B$.

We first show how to compute $A^a_r[\cdot]$ from $A_r[\cdot]$ that is stored at r. In this case, $a \notin V_r$. Let $\pi = \pi(a, c)$. Remember that c is the connector of T_r and $A_r[\cdot]$ has been computed with respect to c. Any path from a to a vertex in T_r contains π.

If $l(\pi) \ge B$, then set $A^a_r[i] = -\infty$ for all $0 \le i \le B$ as no vertex of V_r, possibly except c, can be reached from a using a path of length at most B. If $l(\pi) < B$, then set for $0 \le i \le B$

$$A^a_r[i] = \begin{cases} -\infty & \text{if } i \le l(\pi), \\ w(\pi) + A_r[i - l(\pi)] & \text{if } i > l(\pi), \end{cases}$$

Note that $l(\pi)$ and $w(\pi)$ can be found in $O(1)$ time. So, $A^a_r[\cdot]$ can be obtained in $O(B)$ time.

The computation of $A^a_q[\cdot]$ is done by recursively calling the function of computing $A^a_p[\cdot]$ with q replacing p. Note that a is in both T_p and T_q and $|T_q| < |T_p|$. Shown in Fig. 1 is our algorithm for computing $A^a_p[\cdot]$, which follows the explanation given so far. For p at level i in T^*, compArray(p, a) makes at most i recursive calls until the recursion reaches level 0.

Before analyzing the time complexity of computing $A_p[\cdot]$ for all nonroot nodes p, let us review some properties of T^*: T^* is a binary tree of height $h = \Theta(\log(n/B))$ and has $\Theta(n/B)$ leaf nodes. Frederickson [2] proved that in T^* the number of nodes at level $i > 0$ is at most $5/6$ the number of nodes at level $i - 1$. So, in T^*, if m is the number of leaf nodes, then the number of nodes at level i is at most $(5/6)^i m$.

Since compArray(p, a) for a node p at level i makes at most i recursive calls, the total number of recursive calls made by calling compArray(p, a) for all nonroot nodes p in T^* is at most

$$\sum_{i=1}^{h-1} i \cdot (\frac{5}{6})^i \cdot m,$$

which is less than $30m$ as $\sum_{i=1}^{\infty} i \cdot \theta^i = \theta/(1-\theta)^2$ for $0 < \theta < 1$. Since each call to compArray(p, a), except the recursive calls in it, takes $O(B)$ time and $m = \Theta(n/B)$, the arrays $A_p[\cdot]$ for all nonroot nodes p of T^* can be computed in $O(n)$ time.

Lemma 2: The arrays $A_p[\cdot]$ for all nodes p of T^* can be computed in $O(n)$ time.

Given an edge-weighted tree and a positive integer B, our algorithm for finding the heaviest path of length at most B in the tree works as follows:

1. Transform the tree into a binary tree.
2. Compute a restricted multilevel partition of order $B + 1$ for the binary tree and build a topology tree for the partition.
3. Compute $A_p[\cdot]$ for all nonroot nodes p of the topology tree in postorder.
4. Compute w_p for all nodes p of the topology tree in postorder.

Then, w_α is the weight of the heaviest path of length at most B in T. Steps 3 and 4 can be combined into one as w_p for a node p of the topology tree can be computed once the arrays for its children are obtained. Since each of the steps 1–4 can be executed in $O(n)$ time, we have proved the following theorem:

Theorem 1: Given an edge-weighted tree and a positive integer B, the heaviest path of length at most B in the tree can be found in $O(n)$ time.

The algorithm can be modified to find the heaviest path of length *exactly* B. The only part of the algorithm that needs to be modified is that of computing, for all $p \in V^*$, w_p, whose definition is now changed to the weight of the heaviest path of length exactly B in T_p if such a path exists and $-\infty$ otherwise.

If $p \in V^*$ is a leaf node, then set $w_p = -\infty$ as T_p has at most B vertices and contains no path of length B. If p has a single child q in T^*, then set $w_p = w_q$. If p has two children q and r in T^*, then we need to compute w', the weight of the heaviest path of length B in T_p that contains e, the connecting edge between T_q and T_r. w' can be computed using $A_q[\cdot]$ and $A_r[\cdot]$. If $l(e) = 0$, then $w' = \max\{A_q[i] + A_r[B - i] \mid 0 \le i \le B\}$, and if $l(e) = 1$, then

```
compArray(p, a)
  if (p is a leaf), then directly compute A^a_p[·] and return it.
  if (p has a single child q), then return A^a_q[·].
  // p has two children q, r and assume that a ∈ V_q. //
  compute A^a_r[·].
  A =compArray(q, a).
  for (i = 0 to B)
    A[i] = max{A[i], A^a_r[i]}.
  return A.
```

Fig. 1 Algorithm for computing $A^a_p[\cdot]$

$w' = w(e) + \max\{A_q[i] + A_r[B-i-1] \mid 0 \le i \le B-1\}$. Then, $w_p = \max\{w_q, w_r, w'\}$, where w_q and w_r are the weights of the heaviest paths of length exactly B in T_q and T_r, respectively. Since w_p for each $p \in V^*$ can be computed in $O(B)$ time, we have a lemma, which is similar to Lemma 1.

Theorem 2: Given an edge-weighted tree and a positive integer B, the heaviest path of length exactly B in the tree can be found in $O(n)$ time.

4. Conclusion

In this paper, we present a linear-time algorithm for the length-constrained heaviest path problem on a tree with uniform edge lengths. The algorithm can be modified to find the heaviest path of a specific length on a tree with uniform edge lengths. An interesting future work is to find other problems that can be solved in linear time using our method developed in this paper.

References

[1] B. Bhattacharyya and F. Dehne, "Using spine decompositions to efficiently solve the length-constrained heaviest path problem for trees," Inf. Process. Lett., vol.108, pp.293–297, 2008.
[2] G.N. Frederickson, "Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees," SIAM J. Comput., vol.26, pp.484–538, 1997.
[3] D. Harel and R.E. Tarjan, "Fast algorithms for finding nearest common ancestors," SIAM J. Comput., vol.13, no.2, pp.338–355, 1984.
[4] G.F. Italiano and R. Ramaswami, "Maintaining spanning trees of small diameter," Algorithmica, vol.2, pp.275–304, 1998.
[5] B. Schieber and U. Vishkin, "On finding lowest common ancestors: Simplification and parallelization," SIAM J. Comput., vol.17, pp.1253–1262, 1988.
[6] A. Tamir, "An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs," Oper. Res. Lett., vol.19, pp.59–64, 1996.
[7] R. Uehara and Y. Uno, "On computing longest paths in small graph classes," Int. J. Found. Comput. Sci., vol.18, no.5, pp.911–930, 2007.
[8] B.Y. Wu, K.M. Chao, and C.Y. Tang, "An efficient algorithm for the length-constrained heaviest path problem on a tree," Inf. Process. Lett., vol.69, pp.63–67, 1999.