Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 93 (2005) 275–279
Information Processing Letters
www.elsevier.com/locate/ipl

Finding a longest nonnegative path in a constant degree tree

Sung Kwon Kim [1]

Department of Computer Engineering, Chung-Ang University, 221 Huksuk-dong Dongjak-ku, Seoul 156-756, Republic of Korea

Received 16 July 2004; received in revised form 15 November 2004

Available online 4 January 2005

Communicated by F.Y.L. Chin

Abstract

A longest nonnegative path in an edge-weighted tree is a path such that the sum of edge weights on it is nonnegative and the number of edges on it is as large as possible. In this paper we show that if a tree has a constant degree, then its longest nonnegative path can be found in $O(n \log n)$ time, where n is the number of nodes. Previously known algorithms take $O(n \log^2 n)$ time.

Keywords: Algorithms; Nonnegative paths; Trees

1. Introduction

Let $A = \langle a_1, \ldots, a_n \rangle$ be an array of real numbers. For any $1 \leqslant i \leqslant j \leqslant n$, $\langle a_i, \ldots, a_j \rangle$ is called a *subarray* of A. Its *length* is $j - i + 1$, its *sum* $a_i + \cdots + a_j$, and its *average* $\frac{a_i + \cdots + a_j}{j-i+1}$. Given a threshold value θ, the problem of finding a longest subarray of A with its average at least θ has important applications in computational biology and bioinformatics, e.g., see [1,6].

The problem can be explained in another way as follows: As the average $\frac{a_i + \cdots + a_j}{j-i+1} \geqslant \theta$, this can be written as $(a_i - \theta) + \cdots + (a_j - \theta) \geqslant 0$. From this, finding a longest subarray of A whose average is at least θ now becomes finding a longest subarray of A' whose sum is nonnegative, where $A' = (a_1 - \theta, \ldots, a_n - \theta)$. Finding a longest nonnegative subarray can be accomplished in $O(n)$ time [1,6].

A generalization of this in a tree is following: A tree $T = (V, E)$ consists of a set V of nodes and a set E of edges. Each edge $e \in E$ is associated with a weight, $w(e)$, which is a (positive, negative, or zero) real number. There exists a unique path P between two different nodes in a tree. The *length* of P is the number of edges in P, and its *weight*, denoted by $w(P)$, is the sum of the weights of edges in P. That is, $w(P) = \sum_{e \in P} w(e)$. P is called *nonnegative* if $w(P) \geqslant 0$. Finding a longest nonnegative path in a tree can be done in $O(n \log^2 n)$ time where $n = |V|$ by the algorithm in [7].

E-mail address: skkim@cau.ac.kr (S.K. Kim).

[1] Supported by the ITRI of Chung-Ang University.

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2004.11.012

The *degree* of a node is the number of edges incident on the node, and the *degree* of a tree is the maximum among the degrees of the nodes in it. The algorithm of [7] takes $O(n \log^2 n)$ time regardless of the degree of trees. We are interested in whether it is possible to reduce time complexity if a tree has a constant degree. We answer the question positively by giving an algorithm with running time $O(n \log n)$.

Our algorithm (with a slight modification) can be applied to any tree with a constant degree at least three; however, for ease of presentation, our attention will be restricted to trees of degree three.

Note 1. Wu et al. [7] developed an algorithm for the problem of finding in a tree a *length-constrained heaviest* path, i.e., a maximum-weight path whose length is at most some predefined threshold. In [7], each edge is associated with two values, *length* and *weight*, both of which are real numbers (negatives are allowed). The length (weight) of a path is the sum of its edge lengths (weights). Their algorithm is a divide-and-conquer algorithm: split a tree into (at most) three subtrees, recursively compute subsolutions, and combine them to obtain the solution. The *combine* step requires a sorting of $O(n)$ real numbers, which correspond to lengths of *paths*, taking $O(n \log n)$ time. Therefore, the time complexity of the whole algorithm is $O(n \log^2 n)$. If the sorting in the combine step can be done in linear time, the time complexity will reduce to $O(n \log n)$. They actually claimed that if the *edge* lengths are all integers in the range $[1..O(n)]$, integer sorting algorithms, e.g., counting sort [2], can be used and the time complexity will be $O(n \log n)$. *Even though the edge lengths are all* $O(n)$*, some paths may have lengths beyond* $O(n)$. So, the sorting cannot be accomplished in $O(n)$ time for some inputs.

Note 2. The algorithm in [7] finds a length-constrained heaviest path, but it can find a *weight-constrained longest* path by switching the roles of lengths and weights. A longest nonnegative path, which is the focus of our algorithm, is a weight-constrained longest path as we are trying to find a longest path whose weight $\geqslant 0$. In our algorithm each edge is of length one. The algorithm in [7] cannot find a longest nonnegative path in $O(n \log n)$ time, as pointed in Note 1, even if the input tree has a constant degree. Notice that since the roles of lengths and weights have been switched, the sorting in the combine step is done with respect to path weights, not path lengths.

2. Structure of algorithm

Our algorithm is a recursive one based on a divide-and-conquer method. Before explaining the algorithm we need an important definition.

T is a tree of degree three. If a node v and its edges are removed from T, of degree three, then T is partitioned into (at most) three subtrees T_1, T_2 and T_3, depending on the degree of v. Node v is called a *centroid* of T if $|T_i| \leqslant |T|/2$ for $i = 1, 2, 3$, where $|T|$ denotes the number of nodes in T. A tree has either one or two centroids; and if there are two, they must be adjacent [5]. If a tree has two centroids, one of them is chosen as *the* centroid.

The centroid of T can be found in $O(|T|)$ time [3, 4]. A well-known method starts with converting T into a (rooted) binary tree by choosing a node of degree one as the root. Let $T(v)$ for node v be the subtree consisting of v and all of its descendants. Compute $|T(v)|$ for every node v of T by traversing (the binary tree version of) T in postorder. If v is a leaf, then $|T(v)| = 1$; otherwise, $|T(v)| = 1 +$ the number of nodes in the left and right subtrees of v. During this procedure, the first node v that satisfies $|T(v)| \geqslant |T|/2$ is the centroid of T.

Our algorithm, as mentioned before, runs in a divide-and-conquer fashion:

Input: A tree T of degree three.

Output: The length of the longest nonnegative path in T (the path itself can be found by slightly modifying the algorithm).

[*Divide*] If T has only one node, then returns 0. Otherwise, find the centroid c of T, and remove c and its edges from T. At most three subtrees T_1, T_2 and T_3 are left. Depending on the degree of c, T_2, T_3 or both may be empty.

[*Conquer*] Recursively find the length, denoted by L_1, of the longest nonnegative path in T_1. In similar ways, recursively find L_2 and L_3 in T_2 and T_3, respectively.

[*Combine*] Find the length, L_c, of the longest nonnegative path that is contained in T and passing