COMBINATORICA 5 (1) (1985) 57—65

LINEAR VERIFICATION FOR SPANNING TREES

J. KOMLÓS

Received 30 May 1984

Given a rooted tree with values associated with the n vertices and a set A of directed paths (queries), we describe an algorithm which finds the maximum value of every one of the given paths, and which uses only

$$5n+n\log\frac{|A|+n}{n}$$

comparisons.

This leads to a spanning tree verification algorithm using $O(n+e)$ comparisons in a graph with n vertices and e edges.

No implementation is offered.

0. Introduction

1. Finding the minimal spanning tree in an undirected network is a well-researched area of computer science. The classical algorithms of Kruskal and Prim have been modified and improved several times. For a study of several spanning tree algorithms, see [1].

The best known result has been the $O(|E|\log\log|V|)$ algorithm of Yao [6] until recently.

A few weeks ago, Fredman and Tarjan [2] developed a method which applies to both the shortest path and the spanning tree problems, leading to an $O(|E|\beta(|E|,|V|))$ algorithm for the latter one; where $\beta(m,n)=\min\{i|\log^{(i)}n\leqq m/n\}$ On the other hand, the only verification result we know of is the $O(|E|\alpha(|E|,|V|))$ algorithm of Tarjan [5]; (here α is the inverse Ackermann function).

Here we describe an algorithm which finds maxima over various paths of a tree, which leads to a minimal spanning tree verification algorithm with a *linear number of comparisons.*

We want to emphasize, however, that the only cost we deal with is the total number of comparisons made, for we could not find an effective implementation with a linear overhead cost. In other words, our result is of an information theoretical nature.

AMS subject classification (1980): 68 E 10

We remark that the problem is a particular instance of the following general question that is discussed in [3].

(*Q*) *Given an n element set* $E=(e_1, \ldots, e_n)$, *and a list of m subsets of* $\{1, 2, \ldots \ldots, n\}$, $L=(S_1, \ldots, S_m)$. *Find the maxima*

$$M_i = \max_{j \in S_i} e_j, \quad i = 1, 2, \ldots, m.$$

(If E is the set of edges of the spanning tree and the elements of S_i are the circuit edges created by the i-th outside edge, we get the tree verification problem.)

Of course, sorting the whole list E provides all the necessary information for finding all M_i (overhead is not counted!), but one would hope for an algorithm using only $O(m+n)$ comparisons.

Fredman proved (see [3]) that the number of possible outcomes is not more than $\binom{m+n-1}{n-1} < 2^{m+n}$, thus the above hope is realistic.

A family of paths on trees provides enough structure to make the problem easier to attack.

The general question (Q) is still unanswered.

2. Given an undirected network G (a graph with n vertices, e edges and real values associated with the edges) and a spanning tree T of G, we want to test whether T is minimal among all spanning trees of G.

Any edge x of G not in T creates a unique circuit C_x with edges of T; and it is well-known that T is minimal if and only if, for any outside edge x, the value of x is not smaller than any value in C_x.

Thus, we only need to know, for all outside x, the maximum value on the path $C_x - x$, so that we can compare this maximum with the value of x. Note that $C_x - x$ consists entirely of edges of T.

Let us root T by a leaf of T, and consider it a directed tree with edges directed away from the root. Any path of T is the union of at most two directed paths, and so it is sufficient to find the maxima on the directed "half-paths" corresponding to the outside edges.

By reassigning the values of the edges to their lower endpoints (and deleting the root), we get a more attractive model, in which the values are associated with the vertices.

Whatever cost we obtain for this directed path problem, we only need $2(e-n+1)$ extra comparisons for the spanning tree verification problem.

Notation

T	is a rooted tree with edges directed away from the root.
$V(T)$	$=$ set of vertices of T
$T-U$	for a vertex set $U \subset V(T)$ is the graph T restricted to $V(T)-U$
$x \geqq y$	for $x, y \in V(T)$ means x is a predecessor of y on T
$x > y$	means $x \geqq y$ and $x \neq y$
deg(x)	is the number of outgoing edges from x
path	means directed path of T
$p(x, y)$	stands for the directed path $(u \mid x \geqq u \geqq y)$
$\log x$	is binary logarithm

Formulation of the problem

Given a rooted tree T with real values associated with the vertices $f\colon V(T)\to\mathbf{R}$ and a set A of directed paths of T (queries)

$$A\subset Q=\{p(x,y)\mid x,y\in V(T),\ x\geqq y\}.$$

Find the maximum

$$\max_{x\geqq u\geqq y} f(u)$$

for all $p(x,y)\in A$.

We will present a solution using less than

$$C=5n+n\log\frac{|A|+n}{n}$$

comparisons.

Note that $C\ll|A|$ for $|A|\gg n$, thus the linear term $O(e)$ for the tree verification problem ($|A|=e-n+1$) comes from the comparisons made between the maxima in A and the outside edges.

The paper is structured as follows. First we will describe two completely different algorithms for two particular cases: when T is a string (Section 1, this is even implementable) and when T is a full branching tree (Section 2), and then we will show how a general tree can be interpreted as a mixture of these two extremes.

1. When T is a string

In other words, we have an array, $[f(i);\ 1\leqq i\leqq n]$, and want to find maxima over intervals $[f(i);\ s\leqq i\leqq t]$. Although there are $\binom{n+1}{2}$ such intervals, we give an algorithm that uses less than $2n$ comparisons, and still can find the answers *for all* cn^2 queries (with a bounded overhead per query).

This easy part of our algorithm may be folklore, but we could not trace it in the literature.

Symmetric order heaps

Given an array $[f(i);\ 1\leqq i\leqq n]$, we construct a binary tree SH on n nodes with the following properties:

1. $f(i)$ is assigned to node v_i of SH,
2. SH is a heap; i.e. $v_i>v_j$ implies $f(i)\geqq f(j)$,
3. v_i are in symmetric order; i.e., if v_j belongs to the left (right) subtree of v_i then $j<i$ $(j>i)$.

Clearly, these properties uniquely determine SH, namely the root of SH must be $f(m)=\max_{1\leqq i\leqq n} f(i)$, and the elements $f(i)$, $1\leqq i<m$ $\left(f(i),\ m<i\leqq n\right)$ should form the left subtree (right subtree) of v_m; proceed recursively inside these subtrees.

Once the tree SH has been constructed, finding the maximum over a subarray $[f(i);\ s\leqq i\leqq t]$ reduces to determining the lowest common ancestor of v_s and v_t. Harel [4] has an ingenious algorithm which, after an $O(n)$ cost preprocessing, will process any number of lowest common ancestor queries for a constant cost each.

Construction of the tree SH

The binary tree SH will be respresented by the standard LEFTCHILD (LC) and RIGHTCHILD (RC) arrays.

Definition. The right shoulder of a binary tree is the (maximal) array $S=[S(0), S(1), \ldots, S(k)]$ of nodes of the tree in which $S(0)$ is the root of the tree, and $S(i)$ is the right child of $S(i-1)$.

For an easier formal description, we will use an auxiliary node v_0 with value $f(0)=+\infty$, and SH will be the right subtree of v_0. Starting with this v_0, we insert the elements of $f(i)$ into the tree one-by-one, by keeping track of not only the tree structure (LC, RC) but also the right shoulder array S. Note that S forms a monotone stack $\big(S(k)\geqq S(k-1)\geqq\ldots\geqq S(0)$ and $f(S(k))\leqq\ldots\leqq f(S(0))\big)$.

We compare the new value $f(i)$ with the stack values $f\big(S(j)\big)$, $j=k, k-1, \ldots$, until we find an $f\big(S(j)\big)\geqq f(i)$. Then we remove the subtree T with roots $S(j+1)$ from $S(j)$, add a new node v_i (with value $f(i)$) to the tree as the new right child of $S(j)$, and attach the above T back as the left subtree of v_i. It is clear that both the symmetric order and the heap property are preserved.

The formal description uses the arrays

$$[f(i);\ 0\leqq i\leqq n], \qquad f(0)=+\infty$$
$$[LC(i), RC(i);\ 0\leqq i\leqq n], \quad LC(0)=\Lambda$$
$$[S(i);\ 0\leqq i\leqq n], \qquad S(0)=0.$$

```
Procedure SYMHEAP ($n$, $f$, $RC$, $LC$)
        $S(0)=0$
        $RC(0)=\Lambda$
        $k=0$
        for $i=1$ to $n$ do
          $j \leftarrow k$
          while $f(i)>f(S(j))$ do
            $j \leftarrow j-1$
          repeat
          $k \leftarrow j+1$
          $LC(i) \leftarrow RC(S(j))$
          $RC(i) \leftarrow \Lambda$
          $RC(S(j)) \leftarrow i$
          $S(j+1) \leftarrow i$
        end for
        end SYMHEAP
```

2. When T is a full branching tree

A rooted tree is a full branching tree, if all leaves are on the same level and non-leaves have at least 2 children. For a full binary tree, processing all path-queries has a cost Θ $(n \log \log n)$, so here we have to restrict our attention to the set A of actual queries.

Let $A(y)$ be the set of actual queries which go through y:

$$A(y) = \{p(x, z) \in A | x \geqq y \geqq z\}$$

and $A^*(y)$ the set of restrictions of these paths to the interval [root (T), y].

Starting with the root, we go down level by level and successively find the maxima over all paths in the sets $A^*(y)$.

Assume that we know these maxima down to the i-th level, and let us find the maxima on paths in $A^*(y)$ for a y on level $i+1$. If $\bar{y}$ is the parent of y, then we know the maxima on the restrictions to [root, $\bar{y}$] of all paths in $A(y)$. It remains to compare $f(y)$ with these maxima. Since the maxima are known as an *ordered* list (maximum on $p(x, \bar{y})$ is at least as large as maximum on $p(x', \bar{y})$ if $x > x'$), we can simultaneously compare $f(y)$ with them using binary insertion. Cost$\leqq \lceil \log (A(y)+1) \rceil$.

Write L_i for the set of nodes on level i, and $l_i = |L_i|$. For a full branching tree, $\sum_{j<i} l_j < l_i$ and thus for the entropy

$$\sum_i (l_i/n) \log (n/l_i) < \sum_i i2^{-i} = 2.$$

Since L_i is an antichain, the sets $A(y)$, $y \in L_i$ are pairwise disjoint, thus Jensen's inequality leads to

$$\sum_{y \in L_i} \lceil \log (A(y)+1) \rceil < \sum_{y \in L_i} (1 + \log (A(y)+1)) \leqq l_i + l_i \log \frac{|A|+l_i}{l_i}.$$

Thus, we get for the total cost:

$$\text{Cost} < \sum_i \left(l_i + l_i \log \frac{|A|+l_i}{l_i} \right)$$

$$\leqq n + n \log \frac{|A|+n}{n} + \sum l_i \log \frac{n}{l_i}$$

$$\leqq 3n + n \log \frac{|A|+n}{n}.$$

3. General T

Scalping a rooted tree

The scalp of a tree T is defined as

$$S(T) = \{v \in V(T) | \deg(u) \leqq 1 \quad \text{for all} \quad u \leqq v\}.$$

The subgraph of T spanned by $S(T)$ splits into vertex-disjoint paths (of length $\geqq 0$) called hairs or fringes.

We set $T_0=T$ and inductively define the trees $T_{i+1}=T_i-S(T_i)$, $i=0, 1, \dots$ and write k for the index of the last non-empty T_i (hence $S(T_k)=T_k$). Note that every vertex $v\in T_i$ $(i\geqq 1)$ has a child in T_{i-1}.

For a given i, $1\leqq i\leqq k$, and a vertex $v\notin T_i$, we define the *root of v in T_i* as the element

$$R_i(v) = \min \{u\in T_i | u > v\}.$$

For $v\in S(T_{i-1})$, the "natural" root $R_i(v)$ will be denoted by $R^*(v)$. We will write R for the set of all roots:

$$R = \{u | R_i(v) = u \quad \text{for some} \quad i, v\}.$$

Clearly, $R=\{u|\deg(u)\geqq 2\}$. (Note that the root of the whole tree in the traditional sense need not be in R.)

The restriction of A (actual queries) to T_i is defined in the natural way (by restricting the paths in A to T_i), and is denoted by A_i. We also define $A^*(y)=A_i(y)$ for $y\in S(T_i)$, and $A^+(y)$ as the set of all queries in A, whose restriction to T_i ends at $y\in S(T_i)$. (I.e., for $y\in S(T_i)$, the restrictions to T_i of queries in $A^+(y)$ form the set $A^*(y)$.) (The rest of the section is not needed for the description of the algorithm, only for the cost analysis.) In other words, a query $p(x, z)\in A^+(y)$ (where $y\in S(T_i)$) iff $R_i(z)=y$. But then $R(v)=y$ for *all* v, $y\geqq v\geqq z$. Hence, if $y_1\in S(T_i)$, $y_2\in S(T_j)$, $i>j$, and a query $p(x, z)$ belongs to both $A^+(y_1)$ and $A^+(y_2)$, then $R_i(y_2)=y_1$.

Thus, we obtained the following lemma.

Lemma 1. *If Y is a set of vertices such that no element in Y is a root of another, then the sets $A^+(y)$, $y\in Y$, are pairwise disjoint.* ▌

For an element $u\in S(T_i)$, the set $C(u)=\{v|R_i(v)=u\}$ is called the court of u. Now we are going to partition R by coloring its elements with k colors. The elements of $S(T_1)\cap R$ get color 1. Having colored the elements of

$$\bigcup_{1\leqq j<i} [S(T_j)\cap R]$$

we color the elements of $S(T_i)\cap R$ as follows: u gets the smallest color (smallest positive integer) that does not appear in its court $C(u)$. Clearly, in the obtained coloring an element in $S(T_i)\cap R$ gets a color not exceeding i. We will write R_i for the set of $u\in R$ with color i, R_0 for the set of leaves of T_0, r_i for $|R_i|$, and r for $|R|$.

The following crucial lemma will be proved at the end of the paper.

Lemma 2. A) *For any fixed $i\geqq 1$, the sets $A^+(y)$, $y\in R_i$, are disjoint.*

B) *For $i\geqq 0$, we have the exponential decay*

$$\sum_{j>i} r_j < r_i$$

whence $r<n/2$, and

$$\sum_{j>i} r_j < r2^{-i}, \quad i = 1, 2, \dots$$

consequently, for the entropy

$$\sum_{i\geqq 1} (r_i/r)\log(r/r_i) < \sum_{i\geqq 1} i2^{-i} = 2.$$

In other words, the coloring defined above provides a finite-entropy decomposition of R into antichains in the partial order $u\gg v$ if $u=R_i(v)$ for some i.

The Algorithm

We start with the decomposition

$$V(T) = \sum_{i=0}^{k} S(T_i) \quad \text{(disjoint union).}$$

Since every scalp $S(T_i)$ represents a disjoint union of paths, $V(T)$ is decomposed into disjoint sets, each of which spans a (directed) path of T.

Perform SYMHEAP on each of these sets. Total cost is less than $2n$. Next, starting with A_k, we will inductively find the answers for all queries in A_i, $i=k, k-1, \dots, 0$ ($A_0=A$ is the original set of queries).

For $i=k$, the tree T_k is but a string, and SYMHEAP provided us with the maximum on all possible paths of T_k.

Having answered all queries in A_{i+1}, we proceed to get the answers for queries in $A_i - A_{i+1}$. For a given vertex $y \in S(T_i)$ there are $|A_i(y)|$ queries in A_i that end at y. For a particular query $p(x,y) \in A_i$ we already know an index m_{xy} such that

$$f(m_{xy}) = \max_{x \geqq u \geqq R^*(y)} f(u)$$

(for $x < R^*(y)$ we may interpret $f(m_{xy})=0$), and from SYMHEAP, we know an index m_y such that

$$f(m_y) = \max_{R^*(y) > u \geqq y} f(u).$$

It remains to compare $f(m_y)$ with $f(m_{xy})$. For fixed y, we know the order of the values $f(m_{xy})$, since $x_1 > x_2$ implies $f(m_{x_1y}) \geqq f(m_{x_2y})$. Thus, we can make the comparisons of $f(m_y)$ with the various values $f(m_{xy})$ simultaneously, by merging the value $f(m_y)$ into the ordered sequence $f(m_{x_1y}) \geqq f(m_{x_2y}) \geqq \dots$. Cost is at most $\lceil \log(|A_i(y)|+1) \rceil = \lceil \log(|A^*(y)|+1) \rceil$ for every $y \in S(T_i)$.

Cost Analysis

Theorem. *The obtained total cost*

$$C = 2n + \sum_y \lceil \log(|A^*(y)|+1) \rceil$$

is less than

$$5n + n \log \frac{|A|+n}{n}.$$

Proof. We will separately handle terms with $y \notin R$ and those with $y \in R$.

By Lemma 1, the sets $A^+(y)$, $y \notin R$, are pairwise disjoint, thus applying the Jensen inequality

$$\sum_{i=1}^{t} \log x_i \leqq t \log \frac{\sum x_i}{t}$$

we get

$$\sum_{y \notin R} \lceil \log(|A^*(y)|+1) \rceil < \sum \left[1 + \log(|A^*(y)|+1)\right] \leqq m + m \log \frac{|A|+m}{m}$$

where $m = n - r$ is the number of non-roots.

It remains to estimate the sum

$$C^1 = \sum_{i=1}^{k} \sum_{y \in R_i} \lceil \log(|A^*(y)|+1) \rceil .$$

Here we may deal with different restrictions of the same path, so multiplicities may occur. By part A of Lemma 2, however, we can use the above estimation for the inner sums:

$$C^1 < \sum_{i=1}^{k} \left[r_i + r_i \log \frac{|A|+r_i}{r_i} \right].$$

The exponential decay of the sizes r_i does the rest (part B of Lemma 2):

$$C^1 < r + r \log \frac{|A|+r}{r} + \sum_{i \geq 1} r_i \log \frac{r}{r_i} < 3r + r \log \frac{|A|+r}{r}.$$

Now $m+r=n$, thus (using $r<n/2$ and the log-sum inequality

$$x \log \frac{a}{x} + y \log \frac{b}{y} \leq (x+y) \log \frac{a+b}{x+y}, \quad x, y > 0 \Big)$$

we get for the total cost

$$C < 2n + m + 3r + m \log \frac{|A|+m}{m} + r \log \frac{|A|+r}{r} < 5n + n \log \frac{|A|+n}{n}$$

as stated. ▌

Proof of Lemma 2. The sets R are clearly antichains of T in the partial order $\gg$, so part A follows from Lemma 1. To establish the exponential decay of $r_i = |R_i|$, we need two more lemmas:

Lemma 3. *Leaves of* T_i $(i \geq 1)$ *have color* i. *Furthermore, for* $i \geq 2$ *and* $u \in R_i$, *the court* $C(u)$ *contains exactly the colors* $\{1, \ldots, i-1\}$.

Lemma 4. *Every node* $u \in \bigcup_{j<i} R_j$ *must have at least two children in* T_i.

To get the exponential decay, it remains to apply the following simple fact: In any rooted tree, the number of vertices of degree ≥ 2 is less than the number of leaves. ▌

Proof of Lemma 3. For $i=1$ the statement is trivial. Assuming its validity for all values less than a certain i, let us prove it for i.

We know that a leaf u of T_i (actually any vertex of T_i) has a child in T_{i-1}, thus there is a leaf v of T_{i-1} such that $R_i(v)=u$. Since (by induction) the color of v is $i-1$, and $C(v)$ (a subset of $C(u)$) contains all colors $\{1, 2, \ldots, i-2\}$, u must have color $\geq i$. But $u \in S(T_i)$, so its color cannot exceed i, thus it is i.

Furthermore, any vertex $u \in R_i$ has (by definition) a vertex $v \in R_{i-1}$ in its court $C(u)$. By induction, $C(v)$ contains all colors $\{1, \ldots, i-2\}$, thus the relation $C(v) \subset C(u)$ implies the second statement of the lemma. ▌

Proof of Lemma 4. If $u \in \bigcup_{j>i} R_j$, then u has a vertex $v \in R_i$ in its court $C(u)$. $R_i \subset T_i$ implies $v \in T_i$. Restricting ourselves to the tree T_i, we found a vertex v to which u is a root. Thus, the degree of u within T_i must be at least 2. ▌

References

[1] D. Cheriton and R. E. Tarjan, Finding Minimum Spanning Trees, *SIAM J. on Computing*, **5** (1976), 724—742.
[2] M. Fredman and R. E. Tarjan, *private communication, December 1983.*
[3] R. L. Graham, A. C. Yao, and F. F. Yao, Information Bounds are Weak in the Shortest Distance Problem, *JACM*, **27** (1980), 428—444.
[4] D. Harel, A Linear Time Algorithm for the Lowest Common Ancestors Problem, *Proc. 21st Annual Symp. on Foundations of Computer Science*, (1980), 308—319.
[5] R. E. Tarjan, Application of Path Compression on Balanced Trees, *JACM*, **26** (1979), 690—715.
[6] A. C. Yao, An $O(|E| \log \log |V|)$ Algorithm for Finding Minimum Spanning Trees, *Information Processing Letters*, **4** (1975), 21—23.

J. Komlós

Mathematical Institute of the
Hungarian Academy of Sciences
Budapest, P.O.B. 127
1364, Hungary
and
University of California, San Diego
La Jolla, CA 92093, U.S.A.