Linear Algorithms for Finding the Jordan Center and Path Center of a Tree

S. MITCHELL HEDETNIEMI

University of Oregon, Eugene, Oregon

E. J. COCKAYNE

University of Victoria, Victoria, British Columbia

and

S. T. HEDETNIEMI

University of Oregon, Eugene, Oregon

This paper contains linear algorithms for finding the Jordan center, the weighted Jordan center and the path center of a tree. All of these algorithms take advantage of an efficient data structure for representing a tree called a canonical recursive representation. As a result, the first two algorithms are different from and faster than similar algorithms in the literature. This paper also defines a new parameter of a graph called the path center, and shows that the path center of a tree T consists of a unique subpath of T.

INTRODUCTION

The notion of centrality in a graph G with vertex set $V = V(G)$ and edge set $E = E(G)$ has an abundance of real world applications. These typically involve problems of finding optimum locations on a network for such things as medical centers, warehouses, stores, communication centers, schools or police stations. In general, a "center" is a vertex (or a minimum set of vertices) which minimizes some function involving the distance between an arbitrary vertex and the vertex in the center.

Transportation Science
Vol. 15, No. 2, May 1981

0041-1655/81/1502-0098 $01.25

For example, one may want to find a vertex x which minimizes a sum such as

$$f(x) = \Sigma d(v, x),\ v \in V(G)$$

or a weighted sum such as

$$g(x) = \Sigma a(v)d(v, x),\ v \in V(G), \quad \text{or}$$

$$h(x) = \Sigma(a(v) + d(v, x)),\ v \in V(G),$$

where $a(v)$ is a non-negative, real-valued weight associated with a vertex v and $d(v, x)$ is the distance in G from v to x.

Alternately, one might want to find a minimax, i.e. a vertex (or set of vertices) which minimizes a maximum such as

$$\max\{d(v, x)\},$$

$$\max\{a(v)d(v, x)\}, \quad \text{or}$$

$$\max\{a(v) + d(v, x)\}, \quad \text{for all } v \in V(G).$$

A representative sample of these notions of centrality can be found in [1]–[9], [11]–[13], and [14].

In this paper we will present algorithms for finding three different "centers" of trees, where a *tree* is a connected, acyclic graph. The first is an improvement over existing algorithms in [4], [8] and [9]; the second is a weighted version of the first algorithm. The third algorithm introduces a new notion of centrality, the path center of a tree, which can be viewed as a generalization of the "center" considered in the first two algorithms.[1] These three algorithms are different from those previously considered in that they involve an explicit data structure for representing a tree (a canonical recursive representation) which simplifies the computer implementation of these algorithms and makes them more efficient and faster.

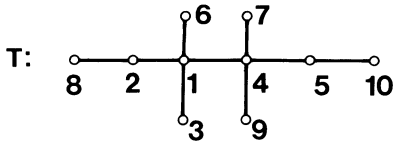
The restriction from general graphs to trees, which is also found in many of the cited references, is made mainly for the sake of "simplest-case" tractability, but also accurately models situations in which economy demands a *minimal* network connecting the given vertex-set.

1. RECURSIVE REPRESENTATIONS OF TREES

AN ENDVERTEX u in tree T (called a "tip" in [4]) is a vertex of degree one. A vertex v adjacent to an endvertex u is called a *remote vertex*. A *pendant edge* (u, v) (called a "quill" in [4]) is an edge between an endvertex u and a remote vertex v.

A tree T having M vertices, labeled $1, 2, \cdots, M$, is *recursively labeled*[12]

[1] A recent paper by SLATER[15] also studies this generalization.



	1	2	3	4	5	6	7	8	9	10
C	0	1	1	1	4	1	4	2	4	5

(a) CANONICAL REPRESENTATION OF T

U	4	4	5	4	1	1	4	1	2
V	5	7	10	1	3	2	9	6	8

(b) AN ARBITRARY RECURSIVE REPRESENTATION OF T

Fig. 1. Recursive representations of trees.

if either $M = 1$ or $M > 1$ and T was iteratively constructed by joining the vertex with label i to one of the $i - 1$ previous vertices, for every i, $2 \leq i \leq M$. It is easy to see that a tree with M vertices labeled 1, 2, $\cdots$, M is recursively labeled if and only if every vertex except the vertex labeled '1' is adjacent to exactly one vertex with a smaller label.

Given a recursively labeled tree T one can easily form a linear sequence, $C(1)$, $C(2)$, $\cdots$, $C(M)$, called the *canonical* (*recursive*) *representation of* T, which uniquely represents T, where $C(j)$ is the unique vertex adjacent to vertex j having a smaller label than j, i.e., $C(j) < j$; we assume that $C(1) = 0$. In Figure 1(a) we illustrate this with a recursively labeled tree T and its canonical representation.

More generally, a *recursive representation of a tree* T is a sequence $(u_{i_1}, v_{i_1})\ (u_{i_2}, v_{i_2}) \cdots (u_{i_{M-1}}, v_{i_{M-1}})$ of the edges of T, such that for every j, $1 \leq j \leq M - 1$, the initial subsequence

$$(u_{i_1}, v_{i_1})(u_{i_2}, v_{i_2}) \cdots (u_{i_j}, v_{i_j})$$

is a recursive representation of a subtree T_j in which (u_{i_j}, v_{i_j}) is a pendant edge. In Figure 1(b) we present a typical recursive representation of T. In general, any procedure which iteratively deletes pendant edges from T and records the sequence of deleted edges, right to left, in a linear sequence, produces a recursive representation of T.

It can be shown,[13] however, that any recursive representation of T can be mapped uniquely into the canonical representation of a (relabeled) tree T' which is isomorphic to T. Hence, the following algorithms are designed to operate on the simpler, canonical representation of trees.

2. THE JORDAN CENTER OF A TREE

FOLLOWING HARARY[10] we present the following definitions. The *distance* $d(u, v)$ between two vertices u and v in a graph G is the length of a shortest path between them. A shortest path between any two vertices is called a *geodesic*. The *diameter* of a graph G, $d(G)$, is the length of a longest geodesic, i.e. $d(G) = \max\{d(u, v)\}$, $u, v \in V(G)$. The *eccentricity* $e(v)$ of a vertex v in a connected graph G is the length of a longest geodesic from v, i.e. $e(v) = \max\{d(u, v)\}$, $u \in V(G)$. The *radius* $r(G)$ of G is the minimum eccentricity of any vertex in G, i.e. $r(G) = \min\{e(v)\}$. A *central vertex* is a vertex v for which $e(v) = r(G)$, and the *center of* G is the set of all central vertices in G. With these definitions it can be seen that the center of G consists of all vertices which minimize the value

$$e(v) = \max\{d(u, v)\}, u \in V(G).$$

In 1869, JORDAN[11] presented the following result.

THEOREM 1 (Jordan). *The center of every tree consists either of one vertex or two adjacent vertices.*

The proof of Jordan's theorem suggests an easy algorithm for finding the center of any tree. The principal idea in the proof is that if T' is obtained from T by deleting all endvertices, then the center of T' is the same as the center of T. Hence, an algorithm which iteratively deletes endvertices in layers until either a single vertex, or two adjacent vertices remain, will find the center of T.

Such an algorithm could be implemented by maintaining a list of current endvertices. As each of these is deleted, a check can be made to see if the corresponding remote vertex becomes an endvertex; if it does then it can be added to the end of the list of endvertices. Such a procedure is suggested by HALFIN.[8]

A different and faster algorithm for finding the center of a tree T can be designed using recursive representations. This algorithm in effect makes one pass from right to left over the canonical representation of T and computes the diameter $d(T)$. A partial second pass, from left to right, then locates the center using the observation that the center of T occurs at the midpoint(s) of any longest path in T. The correctness of this algorithm is based on the next theorem.

Let T be an undirected tree and let T^u be the directed tree which

results from rooting T at an arbitrary vertex u (i.e. orienting all edges of T away from u). For any vertex v in T^u, the set of vertices w for which there exists a directed path from v to w (including v itself) defines a subtree T_v^u which is rooted at v.

The *height of a tree T rooted at v*, $h(T^v)$, is the length of a longest path from v to an endvertex of T^v. The *diameter of a tree T rooted at v*, $d(T^v)$, is the length of a longest path in T^v, where the edges are considered to be undirected.

THEOREM 2. *Let T^v be a tree rooted at a non-endvertex v and let $v_1, v_2, \ldots, v_k$, $k \geq 2$, be the vertices adjacent to v in T^v. Then*

$$d(T^v) = \max\{\{d(T^v_{v_i})\} \cup \{h(T^v_{v_i}) + h(T^v_{v_j}) + 2\}\}, \quad 1 \leq i \leq k, \ 1 \leq i < j < k$$

and

$$h(T^v) = 1 + \max\{h(T^v_{v_i})\}, \quad 1 \leq i \leq k.$$

Proof. A longest path in T^v either passes through the root v or it does not. If it does, then it must enter v from some vertex adjacent to v, say v_i, and it must leave v through some other adjacent vertex, say v_j. The length of this path in $T^v_{v_i}$ and in $T^v_{v_j}$ must equal $h(T^v_{v_i})$ and $h(T^v_{v_j})$, respectively. It follows that

$$d(T^v) = 2 + h(T^v_{v_i}) + h(T^v_{v_j}) = \max\{h(T^v_{v_i}) + h(T^v_{v_j}) + 2\}.$$

If, on the other hand, this path does not pass through v, then it must lie entirely within one of the subtrees, say $T^v_{v_i}$. But then

$$d(T^v) = d(T^v_{v_i}) = \max\{d(T^v_{v_i})\}.$$

In either case,

$$d(T^v) = \max\{\{d(T^v_{v_i})\} \cup \{h(T^v_{v_i}) + h(T^v_{v_j}) + 2\}\}.$$

This result immediately suggests an algorithm, for computing the diameter of any rooted tree T^v, which removes endvertices one at a time. Associate with each vertex w a variable HEIGHT(w), which records the length of a longest path encountered so far from w to an endvertex. When a given vertex w becomes an endvertex, the current value of HEIGHT(w) will equal the height of the subtree T_w^v rooted at w, and this value can be used to update the current HEIGHT of the remote vertex v adjacent to w. The value of HEIGHT(v) + HEIGHT(w) + 1 can then be used as a new candidate for the diameter of T. The largest value encountered for HEIGHT(v) + HEIGHT(w) + 1, for an endvertex w and its remote vertex v, will equal $d(T)$.

The following algorithm, and those given later in the paper, are presented in an ALGOL-like language. In this language the instruction "set $A \leftarrow B$" means "assign value B to variable A." The symbol "od" marks the end of the instruction begun by the most recent appearance of

"do," and similarly for "fi" and "if." A statement in square brackets explains the purpose of the next group of instructions.

Algorithm JORDAN-CENTER

To find the DIAMETER and CENTER of a tree T having M vertices labeled 1, 2, ..., M, described by the canonical representation $C(1)$, $C(2), \ldots, C(M)$.

```
Step 0. [Initialize]
        Set DIAMETER ← 0
        for I ← 1 to M do set HEIGHT(I) ← 0 od
Step 1. [Determine diameter]
        For END ← M downto 2 do
          [determine remote vertex]
          set REMOTE ← C(END)
          [update height and diameter]
          set NEWHT ← HEIGHT(END) + 1
          set NEWDIAM ← HEIGHT(REMOTE) + NEWHT
          if NEWHT > HEIGHT(REMOTE)
            then set HEIGHT(REMOTE) ← NEWHT
            fi
          if NEWDIAM > DIAMETER
            then set DIAMETER ← NEWDIAM
            fi
          od
Step 2. [Determine radius]
        Set RADIUS ← ⌊DIAMETER/2⌋
Step 3. [Determine center]
        For NODE ← 1 to M do
          if HEIGHT(NODE) = RADIUS
            then if DIAMETER is odd
                   then set CENTER ← {NODE, C(NODE)}
                   else set CENTER ← {NODE}
                   fi
                 STOP
            fi
          od
```

As mentioned earlier, Theorem 2 is sufficient to prove that Algorithm JORDAN-CENTER correctly computes the diameter of T. It only remains to show that Steps 2 and 3 correctly find the center of T. But this follows from the following observations:

1. Any vertex (vertices) at the midpoint(s) of any longest path in a tree T is in the center of T.
2. Algorithm JORDAN-CENTER assigns the value ⌊DIAMETER/2⌋

to the height of exactly one vertex. At the time that the final value of DIAMETER is assigned, the statement "DIAMETER ← NEWDIAM" is executed. If HEIGHT(END) = HEIGHT(REMOTE) at the time this statement is executed, then one statement earlier the value of HEIGHT(REMOTE) is updated to HEIGHT(END) + 1. Thus, vertex END, and not REMOTE, is assigned the value ⌊DIAMETER/2⌋.

If HEIGHT(END) ≠ HEIGHT(REMOTE) at the time this statement is executed, then let v denote the vertex (either END or REMOTE) with greater HEIGHT, i.e. HEIGHT(v) > ⌊DIAMETER/2⌋. There must be a path from v to an endvertex of T, down which the value of HEIGHT decreases by one at each successive vertex. Thus, exactly one vertex on this path will have a HEIGHT of ⌊DIAMETER/2⌋. In either case, therefore, at least one vertex is assigned a HEIGHT of ⌊DIAMETER/2⌋.

It is easy to show that two vertices u and v cannot be assigned a HEIGHT of ⌊DIAMETER/2⌋, since this would imply the existence of a path of length greater than DIAMETER.

Since it can be shown that the unique vertex which is assigned a HEIGHT of ⌊DIAMETER/2⌋ also lies on a path of length DIAMETER, this vertex must lie in the center of T. It only remains to show that the one other vertex which Algorithm JORDAN-CENTER might place in the center actually belongs there. We omit these details.

A simple complexity analysis of Algorithm JORDAN-CENTER demonstrates its linearity. The initialization in Step 0 requires $0(M)$ time. Since the test END ≥ 2 in Step 1 is performed $M - 1$ times, each step within Step 1 is also executed $0(M)$ times. Each of these steps requires at most constant time. Step 2 involves a constant amount of time. The test in Step 3 must be performed a maximum of M times, hence, each step within Step 3 must be performed a maximum of M times. Since each of these steps requires at most constant time, Algorithm JORDAN-CENTER can be executed in $0(M)$ time.

3. A WEIGHTED CENTER OF A TREE

THE ALGORITHM in the previous section can easily be generalized to determine a center of a tree in which the vertices and/or edges are assigned weights. Let us associate with each vertex u a weight $a(u)$, and with each edge (u, v) a length $l(u, v)$. If, for example, the graph represents the main nodes and main links of a more extensive transport network, then $a(u)$ might represent the average length of local travel between true trip-ends in the vicinity of u and the nearest main node (namely, u) at which those trips join the main network. Since we are using a canonical representation of a tree, the edges of T are of the form $(i, C(i))$. Thus the length of an edge can simply be denoted $l(i)$.

In a graph G with weighted edges and vertices, let us define the *weighted distance* $wd(u, v)$ between two vertices u and v to equal $wd(u, v) = a(u) + d(u, v) + a(v)$, where $d(u, v)$ equals the sum of the lengths of the edges on a shortest path between u and v. The *weighted diameter* is defined as $wd(G) = \max\{wd(u, v)\}$, $u, v \in V(G)$. The *weighted eccentricity*, *weighted radius* and *weighted center* are defined similarly as:

$$we(u) = \max\{a(u) + d(u, v)\}, u \in V(G),$$

$$wr(G) = \min\{we(u)\}, u \in V(G),$$

and the weighted center consists of all vertices having minimum weighted eccentricity. Thus a vertex v in the weighted center is one which minimizes the value

$$\max\{a(u) + d(u, v)\}, u \in V(G).$$

The correctness of the following algorithm for finding the weighted center of a tree is based on immediate generalizations of the proof for Algorithm JORDAN-CENTER. In the interest of brevity these are omitted.

Algorithm WEIGHTED-JORDAN-CENTER

To find the weighted diameter WDIAM and weighted center WCTR of a tree T, having M vertices labeled 1, 2, ..., M, described by the canonical representation $C(1)$, $C(2)$, ..., $C(M)$, where each vertex i has a weight $A(i)$, and each edge has length $L(i)$.

Step 0. [Initialize]
Set WDIAM ← 0
for $I \leftarrow 1$ *to* M *do set* WHT(I) ← $A(I)$ *od*
Step 1. [Determine weighted diameter]
For END ← M *downto* 2 *do*
[Determine remote vertex]
set REMOTE ← C(END)
[update weighted height and weight diameter]
set HTEND ← WHT(END) + L(END)
set NEWDIAM ← HTEND + WHT(REMOTE)
if HTEND > WHT(REMOTE)
then set WHT(REMOTE) ← HTEND
fi
if NEWDIAM > WDIAM
then set WDIAM ← NEWDIAM
fi
od

```
Step 2. [Determine weighted center]
        Set WRADIUS ← WDIAM/2
Step 3. [Determine weighted center]
        For NODE ← M downto 2 do
          [determine remote vertex]
          set REMOTE ← C(END)
          [is edge (END, REMOTE) in the center?]
        if WHT(END) + L(END) = WHT(REMOTE) and
                WHT(REMOTE) ≥ WRADIUS
          then set DIFFE ← WRADIUS − WHT(END)
               set DIFFR ← WHT(REMOTE) − WRADIUS
               if DIFFE = DIFFR
                 then set WCTR ← {END, REMOTE}
                 else if DIFFE < DIFFR
                        then set WCTR ← {END}
                        else set WCTR ← {REMOTE}
                        fi
                 fi
               STOP
          fi
        od
```

Algorithm WEIGHTED-JORDAN-CENTER proceeds, as in the previous algorithm, to make one pass over the canonical representation (right-to-left) in order to compute the weighted diameter WDIAM. It then makes a second partial, right-to-left pass over the representation to find an edge (END, REMOTE) such that the value WDIAM/2 is less than or equal to WHT(REMOTE). It is important to note that this edge satisfies the equality

$$\text{WHT(END)} + L(\text{END}) = \text{WHT(REMOTE)},$$

where $C(\text{END}) = \text{REMOTE}$; this equality is necessary in order that the edge (END, REMOTE) can lie on a weighted diameter.

4. THE PATH CENTER OF A TREE

THE JORDAN CENTER of a tree can be generalized in a natural way to include a larger collection of vertices. Let p be an arbitrary path in a tree T. For an arbitrary vertex v in $V(T)$ define $d(v, p)$ to be the shortest distnce between v and a vertex in p. If v is a vertex in p, then define $d(v, p) = 0$. Define the *eccentricity of a path* p as

$$e(p) = \max\{d(v, p)\},\ v \in V(T),$$

and define the *path radius of* T as

$$\text{pr}(T) = \min\{e(p)\},\ p \text{ in } T.$$

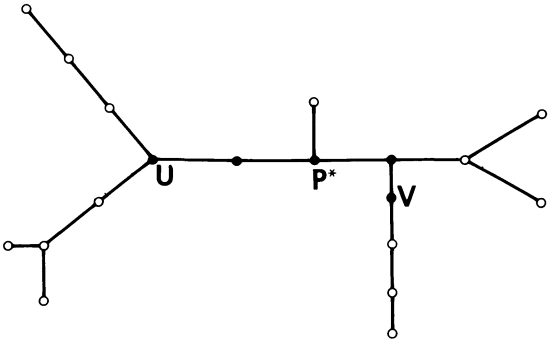


Fig. 2. The path center of a tree.

A path p is minimal with respect to its eccentricity if every path $p' \subset p$ has a greater eccentricity, i.e. $e(p') > e(p)$. A minimal path p^* whose path eccentricity satisfies

$$e(p^*) = \mathrm{pr}(T)$$

is called a *central path*; the *path center of* T consists of all central paths in T.

In Figure 2 we illustrate the path center with a tree T; its path center is the path p^* from u to v. Note that $e(p^*) = 3$, but no path in T has eccentricity of 2, i.e. no path is within distance two of every other vertex. The path p^* might be interpreted as a linear transit-vehicle route which is optimally chosen for accessibility from all vertices in T.

We next show that for any tree T the path center is unique. We then present an algorithm for finding this unique path in any tree.

THEOREM 3 (Uniqueness). *The path center of any tree contains exactly one path.*

Proof. Suppose that there exist two paths p_1 and p_2 in the path center. Then either p_1 and p_2 have vertices in common or they do not.

Case 1. Assume that p_1 and p_2 have no common vertex. Let a_i, $i = 1, 2$ be the initial and terminal vertices of the path p from p_1 to p_2 and T_i, $i = 1, 2$ be the component subtree containing p_i in the forest obtained by deleting all edges of p from T (cf. Figure 3).

Let $x_i \in V(T_i)$ satisfy $d(x_i, a_i) = \max\{d(x, a_i)\}$, $i = 1, 2$, $x \in V(T_i)$. We assert that the path p^* from x_1 to x_2 satisfies

$$e(p^*) < e(p_1) = e(p_2).$$

If $y \in V(T_1)$, then $d(y, p^*) \leq d(y, a_1) \leq d(x_1, a_1) < d(x_1, p_2) \leq e(p_2)$. Similarly, if $y \in V(T_2)$, then $d(y, p^*) < e(p_1)$. If z is a vertex whose path to p contains no vertex of T_1 or T_2, then

$$d(z, p^*) < d(z, p_i) \leq e(p_i), \quad \text{for} \quad i = 1, 2.$$

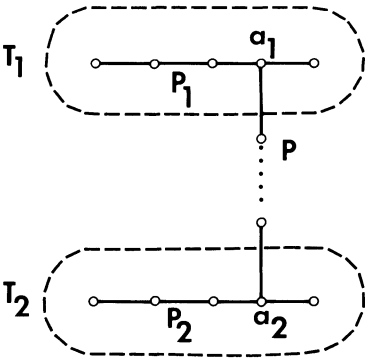


Fig. 3. Components created by removing path P.

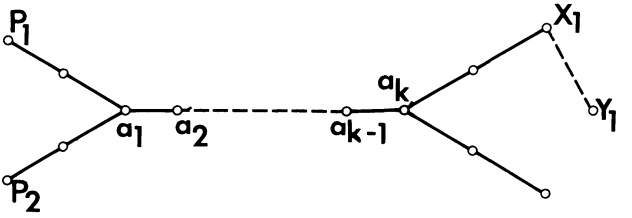


Fig. 4. Two paths with common vertices.

Finally, for vertices z of p, $d(z, p^*) = 0$. Hence,

$$e(p^*) < e(p_1) = e(p_2),$$

which is a contradiction.

Case 2. Suppose that p_1 and p_2 have vertices $a_1, \ldots, a_k$ in common (cf. Figure 4). It follows that $a_1, \ldots, a_k$ is a path in T. Suppose p_1 has an endvertex $x_1 \notin \{a_i \mid i = 1, \ldots, k\}$. Since p_1 is minimal with respect to its eccentricity, there exists a vertex $y_1 \in V(T)$ such that the shortest path from y_1 to p_1 meets p_1 at x_1 and $d(y_1, x_1) = e(p_1)$. However, $e(p_2) \geq d(y_1, p_2) > d(y_1, x_1) = e(p_1)$, which is a contradiction. Hence, p_1 has no endvertex which is not in $\{a_1, \ldots, a_k\}$; a similar argument establishes the same thing for p_2. Hence $p_1 = p_2$.

Algorithm PATH-CENTER works roughly as follows. Using Algorithm JORDAN-CENTER we find the Jordan center of T: either a single vertex $\{v\}$ or two adjacent vertices $\{u, v\}$. We then re-root T at the Jordan center v and proceed to "grow" the path center (PC) out from the center along a longest path (diameter) in T, adding two vertices at a time to PC until a stopping condition is reached. In order to know which two vertices

to add next to PC and to know when to stop, we need the following information:

LEFT, RIGHT	the vertices at the ends of the current path center;
ONE(LEFT), ONE(RIGHT), DIST	the length of a longest path, from LEFT/RIGHT to an endvertex which contains no other vertex in the current path center (note that ONE(LEFT) = ONE(RIGHT) = DIST);
NEXT(LEFT), NEXT(RIGHT)	the vertex adjacent to LEFT/RIGHT on a longest path from LEFT/RIGHT to an endvertex;
TWO(LEFT), TWO(RIGHT)	the length of a second-longest path, if any (edge-disjoint from the longest), from LEFT/RIGHT to an endvertex, which contains no other vertex in the current path center; initialized to zero for all vertices;
BRANCH	the length of a longest (branch) path, if any, from an interior vertex W in the path center to an endvertex, which contains no other vertex in the path center, where LEFT $\neq W \neq$ RIGHT; initialized to zero.
$JC(T)$	the Jordan center of T.

Each of the two possibilities for the Jordan center requires a different initialization procedure. If the Jordan center consists of a single vertex V, then we need to know the lengths y_1, y_2, y_3 of the three longest paths from V to an endvertex, where $y_1 = y_2 \geq y_3$, and possibly $y_3 = 0$. If y_1 is equal to y_3 then the path center consists of the vertex V alone, and the algorithm stops. Otherwise, we set $PC \leftarrow \{V\}$; LEFT $\leftarrow V$; and RIGHT $\leftarrow M + 1$, where $M + 1$ is a new vertex, a "copy" of V. We also set NEXT(RIGHT) to equal the next vertex on a second longest path from V to an endvertex (which may be empty). We then set TWO(LEFT) $\leftarrow y_3$ and TWO(RIGHT) $\leftarrow y_3$, and proceed to get-the-next-vertices.

If the Jordan center consists of two vertices U and V, then we set $PC \leftarrow \{U, V\}$; LEFT $\leftarrow U$ and RIGHT $\leftarrow V$. We let ONE be the length of a longest path, from U to an endvertex not passing through V, and we initialize the appropriate second longest path lengths TWO(LEFT) and TWO(RIGHT). In either case, BRANCH can be initialized to 0. We then proceed to get-the-next-vertices.

We get-the-next-vertices symmetrically about the path center by adding vertices LEFT and RIGHT to PC. A path p with endvertices LEFT and RIGHT is *symmetric in T* if the lengths of the longest paths from

LEFT and RIGHT to an endvertex of T, which contains no other vertices of p, are the same. The stopping condition occurs when the value of DIST is not greater than the maximum of BRANCH, TWO(LEFT), and TWO(RIGHT). The value of BRANCH is updated after each additional two vertices are added to the path center.

In Algorithm PATH-CENTER which follows, we assume that Algorithm JORDAN-CENTER has already been applied to the tree T and that T has been re-rooted at its center. Both of these processes are carried out in linear time.

Algorithm PATH-CENTER

To find the path center PC of a tree T which is rooted at its Jordan center; T is described by its canonical representation $C(1)$, $C(2)$, ..., $C(M)$, and a variable CTR, such that if $CTR = 1$ then the Jordan center consists of vertex 1; if $CTR = 2$ then the Jordan center consists of vertices 1 and 2; if $CTR = 1$ then THREE is the length of a third longest path from 1 to an endvertex; THREE = 0 if no such path exists.

```
Step 0. [Initialize]
        Set THREE ← 0
        if CTR = 2
          then set LAST ← 3
          else set LAST ← 2
          fi
        for I ← 1 to M do
          set ONE(I) ← 0
          set TWO(I) ← 0
          set NEXT(I) ← 0
          od
Step 1. [Determine the lengths of paths ONE and TWO
                 from each vertex]
        For END ← M downto LAST do
          [get remote vertex]
          set REMOTE ← C(END)
          [update ONE and TWO]
          if ONE(REMOTE) ≤ ONE(END) + 1
            then if CTR = 1 and REMOTE = 1
                   then set THREE ← TWO(REMOTE)
                        set NEXT(M + 1) ← NEXT(REMOTE)
                        fi
                 set TWO(REMOTE) ← ONE(REMOTE)
                 set ONE(REMOTE) ← ONE(END) + 1
                 set NEXT(REMOTE) ← END
```

```
            else if TWO(REMOTE) ≤ ONE(END) + 1
                    then if CTR = 1 and REMOTE = 1
                            then set THREE ← TWO(REMOTE)
                                 set NEXT(M + 1) ← END
                            fi
                            set TWO(REMOTE) ← ONE(END) + 1
                    fi
            fi
            od
Step 2. [Initialize PC]
        If CTR = 2 then set PC ← {1, 2}
                        set LEFT ← 1
                        set RIGHT ← 2
                   else set PC ← {1}
                        set LEFT ← 1
                        set RIGHT ← M + 1
                        set TWO(LEFT) ← THREE
                        set TWO(RIGHT) ← THREE
                        fi
        set DIST ← ONE(1)
        set BRANCH ← 0
Step 3. [Get-the-next-vertices]
        While DIST > max{TWO(LEFT), TWO(RIGHT),
              BRANCH} do
          set PC ← PC ∪ {NEXT(LEFT), NEXT(RIGHT)}
          set BRANCH ← max{TWO(LEFT), TWO(RIGHT),
                BRANCH}
          set LEFT ← NEXT(LEFT)
          set RIGHT ← NEXT(RIGHT)
          set DIST ← ONE(LEFT)
          od
        STOP
```

In order to prove the correctness of Algorithm PATH-CENTER we present the following results. The proof of Lemma 5 is obvious and is omitted.

LEMMA 4. *If vertex u is in the Jordan center of a tree T, then u is in the path center of T.*

Proof. Assume to the contrary that there exists a vertex $u \in JC(T)$ such that $u \notin PC(T) = p$. Let T_u be the component subtree containing u of the forest obtained by deleting all edges of the path q from u to p. Since $u \in JC(T)$, there exists an x in T_u such that $d(x, u) \geq e(u) - 1$; otherwise the vertex adjacent to u on q has smaller eccentricity than u.

Hence,

$$e(p) \geq d(x, p) > d(x, u) \geq e(u) - 1.$$

Thus, $e(p) \geq e(u)$.

But since p is the path center,

$$e(p) \leq e(u).$$

This contradicts the Uniqueness Theorem 3.

LEMMA 5. *The initialization (Step 2) in Algorithm PATH-CENTER produces a path which is symmetric in T and ends with DIST* $\geq$ *max* $\{BRANCH, TWO(LEFT), TWO(RIGHT)\}$.

LEMMA 6. *Let* $p \subseteq PC(T)$, *where p is symmetric in T and where* $DIST > max\{BRANCH, TWO(LEFT), TWO(RIGHT)\}$. *Then*

a) $p' = p \cup \{NEXT(LEFT), NEXT(RIGHT)\} \subseteq PC(T)$

b) $p \cup \{NEXT(LEFT), NEXT(RIGHT)\}$ *is symmetric in T, and if the parameters take new values* $DIST'$, $TWO(LEFT)'$, $TWO(RIGHT)'$, *and* $BRANCH'$ *in Step 3, then*

$$DIST \geq max\{TWO(LEFT)', TWO(RIGHT)', BRANCH'\}.$$

Proof. a) Since DIST > max{BRANCH, TWO(LEFT), TWO(RIGHT)}, $e(p)$ = DIST and $e(p')$ = DIST − 1. Moreover, if any neighbor of LEFT other than NEXT(LEFT) were in $PC(T) - p$, then the endvertex x of a maximum length path which defines NEXT(LEFT) satisfies $e(PC(T) \geq d(x, PC(T)) = d(x, \text{LEFT}) = \text{DIST} = e(p) > e(p')$, which contradicts the minimality of $e(PC(T))$.

Similarly, no neighbor of RIGHT other than NEXT(RIGHT) may be added to p as part of the path center. We conclude that

$$p' \subseteq PC(T).$$

b) Trivially, DIST′ = DIST − 1, i.e. p' is symmetric in T; also BRANCH′ = max{TWO(LEFT), TWO(RIGHT), BRANCH} ≤ DIST − 1 = DIST′. Also, by definition of NEXT(LEFT),

$$\text{TWO(LEFT)}' \leq \text{DIST} - 1 = \text{DIST}'.$$

Similarly, TWO(RIGHT)′ ≤ DIST′. Hence,

DIST′ ≥ max{TWO(LEFT)′, TWO(RIGHT)′, BRANCH′}, as required.

LEMMA 7. *Let* $p \subseteq PC(T)$, *where p is symmetric in T, and let* $DIST = max\{TWO(LEFT), TWO(RIGHT), BRANCH\}$. *Then* $p = PC(T)$.

Proof. Suppose to the contrary that $p \neq PC(T)$ and there exists a vertex w adjacent to LEFT which is in the path center of T. Three cases arise which satisfy the equality for DIST.

Case 1. DIST = BRANCH. Let x be the endvertex of T which defines BRANCH. Then

$$e(PC(T)) \geq d(x, PC(T)) = \text{BRANCH} = \text{DIST} = e(p).$$

But this contradicts the Uniqueness Theorem.

Case 2. DIST = TWO(LEFT). Let x be the endvertex of a longest path not including w from LEFT to an endvertex. Then,

$$e(PC(T)) \geq d(x, PC(T)) = d(x, \text{LEFT}) = \text{DIST} = e(p).$$

But this also contradicts the Uniqueness Theorem.

Case 3. DIST = TWO(RIGHT). The proof of this case is the same as that for Case. 2.

A simple complexity analysis of Algorithm PATH-CENTER demonstrates its linearity. The initialization in Step 0 can be performed in $0(M)$ time. The implied test that END $\geq$ LAST in Step 1 requires that each of the steps in Step 1 be executed $M - 2$ times, but each of these requires at most constant time. Step 3 involves the addition of vertices to the path center. Since no vertex is a link to more than one vertex, each vertex can be visited at most once. Therefore, Step 3 requires at most $0(M)$ time. Algorithm PATH-CENTER therefore requires $0(M)$ time for a tree T with M vertices.

ACKNOWLEDGMENTS

THE AUTHORS wish to thank the referee for his suggestions which helped to enhance the paper.

REFERENCES

1. A. ADAM, "The Centrality of Vertices in Trees," *Studia Sci. Math. Hungar.* **9,** 285–303 (1974).
2. N. CHRISTOFIDES, *Graph Theory: An Algorithmic Approach*, Academic Press, New York, 1975.
3. A. J. GOLDMAN, "Optimal Locations for Centers in Networks," *Trans. Sci.* **3,** 352–360 (1969).
4. A. J. GOLDMAN, "Minimax Location of a Facility in a Network," *Trans. Sci.* **6,** 407–408 (1972).
5. A. J. GOLDMAN, "Optimal Center Locations in Simple Networks," *Trans. Sci.* **5,** 212–221 (1971).
6. S. HAKIMI, "Optimum Location of Switching Centers and the Absolute Centers and Medians of a Graph I," *Opns. Res.* **12,** 450–459 (1964).
7. S. HAKIMI, "Optimum Distribution of Switching in a Communication Network and Some Related Graph Theoretic Problems," *Opns. Res.* **13,** 462 (1965).
8. S. HALFIN, "On Finding the Absolute and Vertex Centers of a Tree with Distances," *Trans. Sci.* **8,** 75–77 (1973).

9. G. HANDLER, "Minimax Location of a Facility in an Undirected Tree Graph," *Trans. Sci.* **7,** 287–293 (1973).
10. F. HARARY, *Graph Theory*, Addison-Wesley, Reading, Mass., 1969.
11. C. JORDAN, "Sur les assemblages de lignes," *J. Reine Angew. Math.* **70,** 185–190 (1869).
12. A. MEIR AND J. MOON, "Cutting Down Recursive Trees," *Math. Biosci.* **21,** 173–181 (1974).
13. S. MITCHELL, Linear Algorithms on Trees and Maximal Outerplanar Graphs: Design, Complexity Analysis and Data Structures Study, Ph.D. thesis, University of Virginia, 1977.
14. S. SINGER, "Multi-Centers and Multi-medians of a Graph with an Application to Optimal Warehouse Location," presented at the 16th TIMS Conf., 1968.
15. P. SLATER, "Locating Central Paths in a Graph," *Trans. Sci.* (in press).

Received April 1978