CS-TR-77-6

Shortest paths in trees

by

T. Beyer*

S. Hedetniemi*

S. Mitchell**

* Department of Computer Science
University of Oregon, Eugene, Oregon 97403

** Department of Applied Mathematics and Computer Science
University of Louisville, Louisville, Kentucky 40208

Abstract

This paper contains four algorithms involving shortest paths in trees; they determine:

A. the shortest path from one vertex to another (O(logM))

B. the shortest distances from one vertex to all other vertices (O(M))

C. the shortest distances from all vertices to all other vertices ($O(M^2)$)

D. the expected distance between two vertices (O(M))

The expected time complexities for a tree T with M vertices, for each of these algorithms is indicated above. Algorithm A first appeared in [8] but is included here for completeness; the remaining three algorithms are new.

Key Words: trees, shortest paths, algorithms, analysis

1. Introduction

There is an extensive literature on the subject of shortest paths in graphs, a comprehensive survey of which is found in [10]. There is also a growing literature on the subject of recursive representations of trees (also called father arrays for trees) (cf. [2]-[5],[7]-[9],[11],[12]). In these papers it has been shown that the use of recursive representations can considerably improve the speed, efficiency and compactness of many algorithms on trees. In this paper we show that the same economies can be obtained for algorithms for computing shortest paths in trees.

In particular, we present an algorithm for finding the shortest path from one vertex to another in a tree, whose worst case behavior is O(M), but whose expected performance is O(logM) for a tree T with M vertices. This algorithm has the attractive property of examining only those edges which appear on the shortest path, unlike typical depth-first search algorithms, which potentially examine every edge in a tree.

We also present an O(M) algorithm for computing the shortest distance from one vertex to every other vertex; an $O(M^2)$ algorithm for computing the shortest distances between every pair of vertices in a tree; and an O(M) algorithm for computing the expected distance between two vertices in a tree.

2. Recursive representations of trees

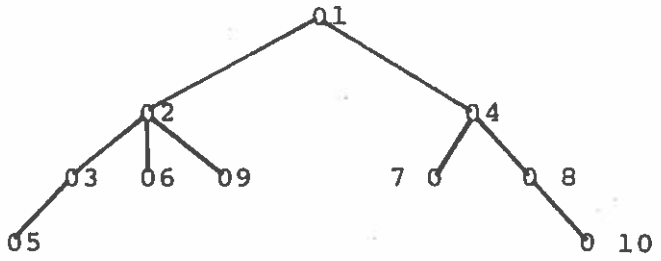
A tree T is a connected, acyclic graph. An endvertex U in a tree is a vertex of degree one. A vertex V adjacent to an endvertex U is called a remote vertex. A pendant edge (U,V) is an edge between an endvertex U and a remote vertex V.

The following definition of a recursive tree was presented by Meir and Moon in [6]. A tree T having M vertices labeled 1,2,...,M is recursive 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, $2 \leq I \leq M$. A canonical linear representation of a recursively labeled tree T with M vertices is an array C(1),C(2),...,C(M), where C(I) is the label of the vertex to which the I-th vertex was joined in the iterative construction of T. This is called the father array in [2] and [5]. A recursive tree is essentially a tree which is rooted at vertex "1". Figure 1 illustrates a recursive tree T and its canonical representation C. Note that C(1) is undefined.

Figure 1

3. The shortest path from one vertex to another vertex

The following compact algorithm, due to Cockayne, was originally presented in [8]. It finds the shortest path from one vertex U to another vertex V in a rooted tree T by marching toward the root of T from each vertex until a common



Recursively labeled tree T

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

The canonical representation of T

FIG. 1

'ancestor' is reached.

Algorithm U-TO-V To find the shortest path from a vertex U to a vertex V in a tree T with M vertices, which is given by a recursive representation C(1),C(2),...,C(M); the shortest path is recorded in the array FROMU(1),FROMU(2),...,FROMU(UI); the array FROMV(1),..., FROMV(VI) contains vertices on the shortest path starting from vertex V; the last vertex on the path from U (V) is UNEXT (VNEXT); NEXT, the larger of UNEXT and VNEXT, is the vertex from which the shortest path is next extended.

Step 0. [Initialize]

Set UNEXT ← U

Set VNEXT ← V

Set UI ← 1

Set VI ← 1

Set FROMU(UI) ← U

Set FROMV(VI) ← V

Step 1. [Proceed toward root from larger vertex]

While UNEXT ≠ VNEXT do

if UNEXT > VNEXT

then set UI ← UI + 1

set FROMU(UI) ← C(UNEXT)

set UNEXT ← C(UNEXT)

```
            else set VI ← VI + 1
                 set FROMV(VI) ← C(VNEXT)
                 set VNEXT ← C(VNEXT)
            fi
         od
```

Step 2. [Record remainder of path in FROMU]

```
      Set VI ← VI - 1
      While VI ≥ 1 do
         set UI ← UI + 1
         set FROMU(UI) ← FROMV(VI)
         set VI ← VI - 1
         od
      STOP
```

For a worst case and expected case complexity analysis of Algorithm U-TO-V the reader is referred to [6] and [8].

4. The shortest distance from one vertex to all others

We next present Algorithm U-TO-ALL which determines the distances between a given vertex U and all other vertices in a tree T. This algorithm extends the process used to determine the shortest path between two vertices, in Algorithm U-TO-V, in the following way. In a first pass, the distance from U to each vertex W on the path from U to the root vertex "1" is determined and recorded in an array DIST(W). Essentially,

vertex V in the previous algorithm has become vertex "1". The remainder of the vertices for which distances need to be calculated fall into one of two classes: either they lie in the branch of vertex 1 containing vertex U, or they do not. The latter distances can be calculated by adding the distance from the given vertex to vertex 1 to the distance from vertex U to vertex 1. The former distances can be calculated from the distances to the vertices on the path from U to 1. Because the tree is recursively labeled (rooted), the distance from vertex I to U equals 1 plus the distance of vertex C(I) to U.

Algorithm U-TO-ALL To find the shortest distance (DIST(VTX)) from one vertex (U) to every other vertex (VTX) in a tree with M vertices, which is described by the recursive representation C(1),C(2),...,C(M).

Step 0. [Initialize]

For I ← 1 to M do set DIST(I) ← -1 od

Set DIST(U) ← 0

Step 1. [Proceed to root from vertex U]

Set VTX ← U

While VTX ≥ 2 do

set DIST(C(VTX))← DIST(VTX) + 1

set VTX ← C(VTX)

od

```
Step 2. [Compute remaining distances]
        For VTX ← 2 to M do
           if DIST(VTX) = -1
              then set DIST(VTX) ← DIST(C(VTX)) + 1
              fi
           od
        STOP
```

It is easy to demonstrate that the worst-case time complexity of Algorithm U-TO-ALL is O(M) for a tree with M vertices. Step 0 requires O(M) time to initialize the array DIST. The while-loop in Step 1 requires the execution of two assignment statements M-1 times, and the for-loop in Step 2 requires the execution of one test M-1 times, plus an assignment statement no more than M-1 times.

5. The shortest distances from every vertex to all others

In this section we present Algorithm ALL-TO-ALL which computes the distances between every pair of vertices in a tree T. These distances are determined by proceeding from the root to the endvertices of T. The distance between a vertex and itself is assumed to be 0. The distance between vertices I and J is recorded in a distance matrix DISTAN in both DISTAN(I,J) and DISTAN(J,I). Although only the lower triangular entries are necessary, both entries are

stored in order to simplify the updating procedure. The correctness of Algorithm ALL-TO-ALL is based on the observation that if a new endvertex V is added to a tree T by adding an edge between V and an arbitrary vertex U in T, then the distance from V to an arbitrary vertex W in T equals one plus the (previously computed) distance from U to W.

Algorithm ALL-TO-ALL To find the shortest distances (DISTAN(I,J)) between every pair of vertices I,J in a tree with M vertices, which is described by the recursive representation C(1),C(2),...,C(M).

```
Step 0. [Initialize]
        Set DISTAN(1,1) ← 0
Step 1. [Compute DISTAN(I,J) for 1 ≤ J ≤ I]
        For I ← 2 to M do
           for J ← 1 to I-1 do
              set DISTAN(I,J) ← DISTAN(C(I),J) + 1
              set DISTAN(J,I) ← DISTAN(I,J)
              od
           set DISTAN(I,I) ← 0
           od
        STOP
```

Since $O(M^2)$ entries must be determined for the array DISTAN for a tree having M vertices, any algorithm to determine the distances between all pairs of vertices must have a

lower bound time complexity of $O(M^2)$. Algorithm ALL-TO-ALL achieves this best bound. The initialization in Step 0 requires constant time. Step 1 involves an O(M) iteration, during each of which O(K) operations are performed, for K = 1,2,...,M-1. Hence Step 1 requires $O(M \cdot M-1) = O(M^2)$ operations.

6. The expected distance between two vertices in a tree

The last algorithm which we will present calculates the expected, or mean, distance between two vertices in a tree. We could use Algorithm ALL-TO-ALL to determine this number by simply summing all the entries in the array DISTAN and dividing this sum by M(M-1). However this would require $O(M^2)$ time. Another algorithm for computing the expected distance is given in [1], however, this algorithm also requires $O(M^2)$ time.

The following algorithm calculates the expected distance in O(M) time. Algorithm EXPECT does not determine the individual length of each path, but simply the sum of the lengths of these paths. We illustrate this process with the tree T on M = 5 vertices in Figure 2.

Figure 2

We can see that the pendant edges (1,2) and (4,5) occur in exactly M-1 = 4 shortest paths. That is, each of

lower bound time complexity of $O(M^2)$. Algorithm ALL-TO-ALL achieves this best bound. The initialization in Step 0 requires constant time. Step 1 involves an O(M) iteration, during each of which O(K) operations are performed, for K = 1,2,...,M-1. Hence Step 1 requires $O(M \cdot M-1) = O(M^2)$ operations.

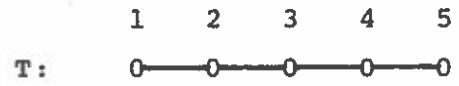
6. The expected distance between two vertices in a tree

The last algorithm which we will present calculates the expected, or mean, distance between two vertices in a tree. We could use Algorithm ALL-TO-ALL to determine this number by simply summing all the entries in the array DISTAN and dividing this sum by M(M-1). However this would require $O(M^2)$ time. Another algorithm for computing the expected distance is given in [1], however, this algorithm also requires $O(M^2)$ time.

The following algorithm calculates the expected distance in O(M) time. Algorithm EXPECT does not determine the individual length of each path, but simply the sum of the lengths of these paths. We illustrate this process with the tree T on M = 5 vertices in Figure 2.

Figure 2

We can see that the pendant edges (1,2) and (4,5) occur in exactly M-1 = 4 shortest paths. That is, each of



From	To	Shortest path	Edge	Occurences
1	5	1 2 3 4 5	(1,2)	4
1	4	1 2 3 4	(2,3)	6
1	3	1 2 3	(3,4)	6
1	2	1 2	(4,5)	4
2	5	2 3 4 5		
2	4	2 3 4		
2	3	2 3		
3	5	3 4 5		
3	4	3 4		
4	5	4 5		

FIG. 2

these edges contributes M-1 to the sum of shortest path lengths. The edges (2,3) and (3,4) each occur 2(M-2) = 6 times in a shortest path. Hence the sum of the lengths of all shortest paths in this tree is 2(M-1) + 2(2(M-2)) = 20, and the expected distance between two vertices in this tree is 20 / (M(M-1)/2) = 2.

In general the number of shortest paths in which an edge (U,V) occurs is given by S(M-S), where S is the number of vertices in the connected component of T - (U,V) which contains U. Algorithm EXPECT determines the number of times each edge occurs in a shortest path by proceeding from the endvertices to the root of a tree T.

Algorithm EXPECT To find the expected distance (EXPD) between two vertices in a tree T with M vertices, which is represented by the recursive representation C(1),C(2),..., C(M); SUBT(I) records the number of vertices in the subtree of T rooted at vertex I; TOTAL records the sum of the lengths of all paths in T.

Step 0. [Initialize]

```
For I ← 1 to M do set SUBT(I) ← 1 od
Set TOTAL ← 0
```

Step 1. [Compute subtree sizes and TOTAL]

```
For VTX ← M downto 2 do
   set SUBT(C(VTX)) ← SUBT(C(VTX)) + SUBT(VTX)
   set TOTAL ← TOTAL + SUBT(VTX)*(M-SUBT(VTX))
   od
```

Step 2. [Determine expected distance]

Set EXPD ← TOTAL/ (M*(M-1)/2)

STOP

The worst-case time complexity of Algorithm EXPD is clearly O(M) in the number of vertices M. Steps 0 and 1 require at most O(M) time, while Step 2 requires constant time.

7. Summary

In this paper we have presented four algorithms involving the computation of shortest paths in recursively labeled trees. Each algorithm is expressed very compactly and 'roughly speaking' achieves a factor of M improvement in speed over corresponding algorithms for arbitrary graphs. Although we have not done so, each algorithm can easily be modified to compute the same results in edge-weighted trees. In a subsequent paper we will attempt to design similar algorithms for more general classes of graphs which can also be recursively represented.

REFERENCES

[1] J.K. Doyle and J.E. Graver, Mean distances in a graph, Discrete Math. 17(1977),147-154.

[2] H.N. Gabow, Two algorithms for generating weighted spanning trees in order, SIAM J. Comput. 6(1977), 139-150.

[3] S. Goodman, S. Hedetniemi and R. Tarjan, B-matchings in trees, SIAM J. Comput. 5(1976),104-108.

[4] A. Kershenbaum and R.M. Van Slyke, Recursive analysis of network reliability, Networks 3(1973),81-94.

[5] D. Knuth, Fundamental Algorithms, Addison-Wesley, Reading, Mass.(1968),334-338.

[6] A. Meir and J. Moon, Cutting down recursive trees, Mathematical Biosciences 2](1974,173-181.

[7] S. Mitchell, Algorithms on trees and maximal outer-planar graphs: design, complexity analysis and data structures study, Ph.D. Thesis, Univ. of Virginia, 1977.

[8] S.L. Mitchell, E.J. Cockayne and S.T. Hedetniemi, Linear algorithms on recursive representations of trees, J. Comput. Systems Sci., to appear.

[9] F. Neville, The codifying of tree structure, Proc. Cambridge Philos. Soc. 49(1953),381-385.

[10] A.R. Pierce, Bibliography on algorithms for shortest path, shortest spanning tree, and related circuit routing, Networks 5(1975),129-149.

[11] A. Prüfer, Neur Beweiss Eines Satzes Über Permutationen, Archiv für Mathematik Physik 27(1918),122-142.

[12] R.C. Read, The coding of various kinds of unlabeled trees, in Graph Theory and Computing, (R.C.Read, Ed.) Academic Press, New York(1972),153-183.