Tree Isomorphism Algorithms: Speed vs. Clarity
Author(s): Douglas M. Campbell and David Radford
Source: *Mathematics Magazine*, Vol. 64, No. 4 (Oct., 1991), pp. 252-261
Published by: Mathematical Association of America
Stable URL: http://www.jstor.org/stable/2690833
Accessed: 15-12-2015 00:01 UTC

Your use of the JSTOR archive indicates your acceptance of the Terms & Conditions of Use, available at http://www.jstor.org/page/info/about/policies/terms.jsp

JSTOR is a not-for-profit service that helps scholars, researchers, and students discover, use, and build upon a wide range of content in a trusted digital archive. We use information technology and tools to increase productivity and facilitate new forms of scholarship. For more information about JSTOR, please contact support@jstor.org.

Mathematical Association of America is collaborating with JSTOR to digitize, preserve and extend access to *Mathematics Magazine*.

http://www.jstor.org

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

Tree Isomorphism Algorithms: Speed vs. Clarity

DOUGLAS M. CAMPBELL

DAVID RADFORD
Brigham Young University
Provo, UT 84602

No algorithm, other than brute force, is known for testing whether two arbitrary graphs are isomorphic. In fact it is still an open question [3] whether graph isomorphism is NP complete. But polynomial time isomorphism algorithms for various graph subclasses such as trees are known (see [3, p. 285 and p. 339] for a summary). In gene splicing, protein analysis, and molecular biology the chemical structures are often trees with millions of vertices. In such applications, the difference between $O(n)$, $O(n \log n)$, and $O(n^2)$ isomorphism algorithms is of practical not just theoretical importance. Readers of MATHEMATICS MAGAZINE will find it of interest to see how such algorithms evolve and are analyzed. As in an earlier article [2], we have chosen dialogue to capture the spirit of discovery and failure in the search for a fast and clear tree isomorphism algorithm.

Jill. I need a quick way to determine whether two trees, such as those of FIGURE 1, are isomorphic.

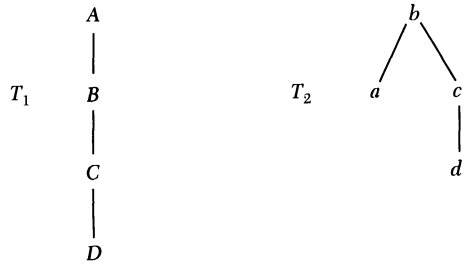

FIGURE 1

Peter. What do you mean! They are isomorphic under the trivial correspondence A–a, B–b, C–c, D–d.

Jill. Well, yes and no. That is a *graph* isomorphism. Formally, if E is a set of edges, V a set of vertices, and r a distinguished vertex called the root, then a *tree* $T = (E, V, r)$ is a connected, acyclic graph. A *tree isomorphism* must match roots, as well as be a graph isomorphism.

Peter. Providing a quick way to *see* a tree may provide a quick way to determine whether two trees are isomorphic. Assuming each edge of a tree has length one, let's assign to each vertex its distance to the root. This distance is the *level number of the vertex*. The root's level number is zero. Draw the tree, level by level, starting with the root at the top. A vertex with no descendants is a *leaf*. Allow me an observation.

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

Observation 1. *Since a tree isomorphism preserves root and edge incidence, the level number of a vertex (the number of edges between the root and the vertex) is a tree isomorphism invariant.* Here's the quick test you wanted:

CONJECTURE 1. *Two trees are isomorphic if and only if they have the same number of levels and the same number of vertices on each level.*

An array implementation is simple and fast. Start with an array $L[0..\|V\|]$ whose entries have been set to zero. Let $L[n]$ denote the number of vertices of level n. For each vertex, if the vertex has level number n, then increment $L[n]$ by one.

Conjecture 1 says we need only test if two arrays are equal, which takes time proportional to $\|V\|$.

Jill. You haven't added in the time it takes to determine the level numbers, nor the time to fill the array, but never mind. Your conjecture is false. For every $\|V\| > 4$, I can construct two non-isomorphic trees that have $\|V\|$ vertices, the same number of levels, and the same number of vertices at each level. Let me make an observation.

Observation 2. *Since a tree isomorphism preserves root and edge incidence, the number of paths from the root to the leaves is a tree isomorphism invariant.*

With observation 2 in mind, let T_1 be the tree of FIGURE 2 with $n-3$ vertices at level one, one of which has two descendants. Let T_2 be the tree of FIGURE 2 with $n-3$ vertices at level one, two of which have one descendant. Since T_1 has $n-2$ paths from root to leaves and T_2 has $n-3$ paths from root to leaves, they cannot be isomorphic.

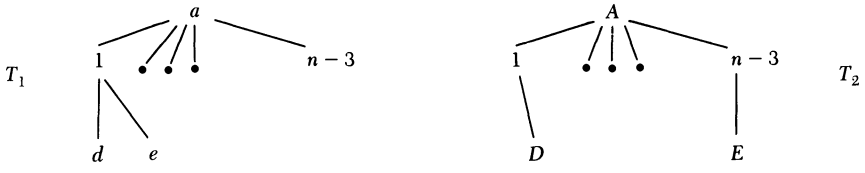

FIGURE 2

You didn't take into account the *degree spectrum of a tree*, the sequence of non-negative integers (d_j), where d_j is the number of vertices in T that have j descending edges when the tree is drawn from the root down. I conjecture:

CONJECTURE 2. *Two trees are isomorphic if and only if they have the same degree spectrum.*

An array implementation is simple and fast. Start with an array $D[0..\|V\|]$ whose entries are set to zero. Let $D[n]$ denote the number of vertices with n descending edges. For each vertex, if the vertex has n descending edges, increment $D[n]$ by one. Again, we are reduced to testing the equality of two arrays of size $\|V\|$, an operation requiring $2\|V\|$ time.

Peter. This time you failed to add the time it takes to compute the degree of each vertex. No point in doing that since your conjecture is false. Let me make an observation.

Observation 3. *Since a tree isomorphism preserves longest paths, the number of levels in a tree (the longest path) is a tree isomorphism invariant.*

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

With observation 3 in mind, I can construct two non-isomorphic trees of n vertices, $n > 6$, that have the same degree spectrum. Let T_1 be the tree of FIGURE 3 with a strand of $n-5$ segments from C. Let T_2 be the tree of FIGURE 3 with a strand of $n-5$ segments from b. Both have degree spectrum $(3, n-5, 2)$. But T_1's longest path has length $n-3$, while T_2's longest path has length $n-4$.

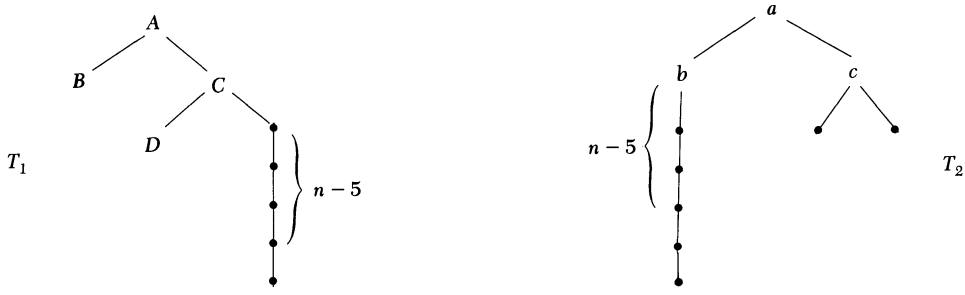

FIGURE 3

However, I conjecture:

CONJECTURE 3. *Two trees are isomorphic if and only if they have the same degree spectrum at each level.*

Jill. Tricky. If two trees have the same degree spectrum at each level, then they must automatically have the same number of levels, the same number of vertices at each level, and the same global degree spectrum!

Nevertheless, Conjecture 3 is false for all $n > 7$. Let me make an observation.

Observation 4. *The number of leaf descendants of a vertex and the level number of a vertex are both tree isomorphism invariants.*

With observation 4 in mind, I can construct two non-isomorphic trees of $m+8$ vertices, $m > 0$, whose degree spectrums are equal level by level. Indeed T_1 and T_2 of FIGURE 4 cannot be isomorphic since the number of leaf descendants of A, B, C, and D on level one are, respectively, 2, 2, 3, 1.

We must measure not only the degree spectrum at each level, but also the degree spectrum of each vertex's descendants.

Peter. Enough of this. Let's quote the AHU algorithm by Aho, Hopcroft, and Ullman [1] that purports to determine tree isomorphism in time $O(\|V\|)$ by keeping track of the history of the degree spectrum of the descendants of a vertex. The algorithm assigns numbers to the vertices of trees in such a way that trees T_1 and T_2 are isomorphic if and only if the same number is assigned to the root of each tree.

1. Assign to all leaves of T_1 and T_2 the integer 0.

2. Inductively, assume that all vertices of T_1 and T_2 at level $i-1$ have been assigned integers. Assume L_1 is a list of the vertices of T_1 at level $i-1$ sorted by non-decreasing value of the assigned integers. Assume L_2 is the corresponding list for T_2.

3. Assign to the non-leaves of T_1 at level i a tuple of integers by scanning the list L_1 from left to right and performing the following actions: For each vertex v on list L_1 take the integer assigned to v to be the next component of the tuple associated with the father of v. On completion of this step, each non-leaf w of T_1 at level i will have a tuple $(i_1, i_2, \ldots, i_k)$ associated with it, where $i_1, i_2, \ldots, i_k$ are the integers, in

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

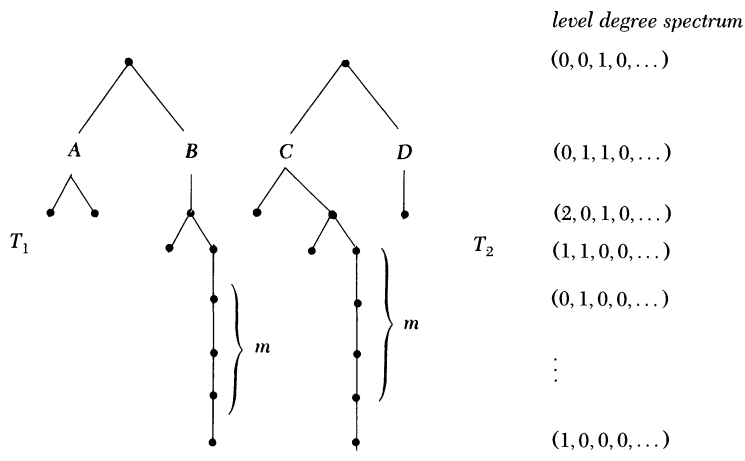

FIGURE 4

non-decreasing order, associated with the sons of w. Let S_1 be the sequence of tuples created for the vertices of T_1 on level i.

4. Repeat step 3 for T_2 and let S_2 be the sequence of tuples created for the vertices of T_2 on level i.

5. Sort S_1 and S_2 lexicographically. Let S_1' and S_2' respectively, be the sorted sequences of tuples.

6. If $S_{1'}$ and $S_{2'}$ are not identical then halt; the trees are not isomorphic. Otherwise, assign the integer 1 to those vertices of T_1 on level i represented by the first distinct tuple on S_1', assign the integer 2 to the vertices represented by the second distinct tuple, and so on. As these integers are assigned to the vertices of T_1 on level i, make a list L_1 of the vertices so assigned. Append to the front of L_1 all leaves of T_1 on level i. Let L_2 be the corresponding list of vertices of T_2. These two lists can now be used for the assignment of tuples to vertices at level $i + 1$ by returning to step 3.

7. If the roots of T_1 and T_2 are assigned the same integer, T_1 and T_2 are isomorphic.

Jill. Utterly opaque. Even on second or third reading. When an algorithm is written it should be clear, it should persuade, and it should lend itself to analysis.

Let's write an algorithm that is close to obvious, that lends itself to timing analysis, and that is clear. After we do those three things, *then* let's try to make it fast.

Peter. Perhaps we should begin by trying to understand the AHU algorithm.

Jill. The AHU algorithm associates with each vertex a tuple that describes the complete history of its descendants. In order to do that it uses a unique integer as a sort of shorthand. Since the unique integers are introduced for speed and not clarity, let's return to a simpler tuple version.

Peter. Let's begin by converting the tree of FIGURE 5 to the Knuth tupled tree [1] of FIGURE 6 using AHU's steps (1), (2), and (3). First, eyeball the tree and find the leaves D, E, F, and G. Assign them the tuple (0). Then fill in the vertices of level 1. Then fill in the root.

Unfortunately, the assignment has proceeded in a haphazard manner that appears to require multiple eyeball passes through the tree. Much worse, I have been forced to access the tree 'level by level' although the tree is generally stored 'parent-to-child'.

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

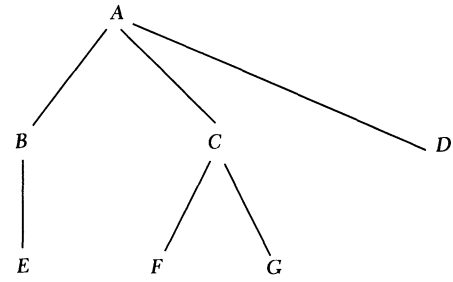

FIGURE 5

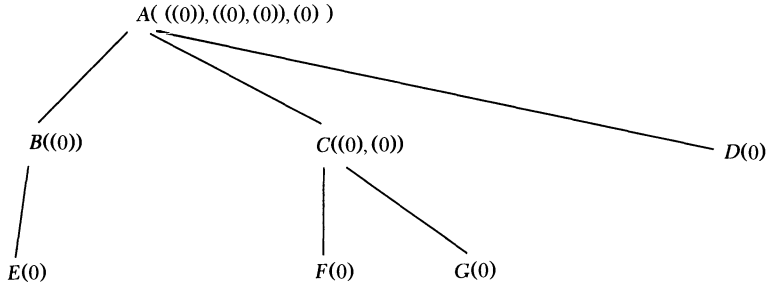

FIGURE 6

In fact, since we have both been loose on how the tree is stored, let's fix the realm of discourse. A tree *is* the root and the set of parent-child relationships. That is, if you give a tree black box the name of a vertex, it will respond with the names of the vertex's children, but cannot give the vertex's cousins, siblings, or parent! If you want to build and store *that* information, be my guest. But the time it takes to acquire that knowledge must be accounted for when analyzing the tree isomorphism algorithm.

A glance at the tree of FIGURE 5 shows that it reduces to the following facts: root A; $A:B, C, D$; $B:E$; $C:F, G$; $D:$; $E:$; $F:$; $G:$;. and nothing more!

Jill. That is an excellent point. Let me first prove that I can assign Knuth parenthetical tuples to all tree vertices in time proportional to $\|V\|$, whether the tree is a long, single strand or short and squat with multitudes of edges spilling off each vertex.

I claim the existence of a set of instructions whose execution causes every vertex of the tree to be visited exactly twice (Post Order Traversal). It is on the second visit to a vertex that the tuple name is assigned. The second visit to a vertex is made *after* all of the vertex's descendants have had their tuple name assigned. Here are the instructions.

```
Post_Order_Version_One(v: vertex);
Begin
    if v is childless then
      Give v the tuple name (0)
    else
     begin
      For each child w of v do
            Post_Order_Version_One(w);
```

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

```
      Concatenate the names of all the children of v to temp;
      Give v the tuple name (temp);
    end
end;
```

Peter. Aha! Recursion! How slick. A vertex doesn't need to know its level, only its children. If it has no children, it is automatically given the tuple name (0). If it has children, then after all of its children have been named, the children's names are concatenated, the concatenated name is enclosed in a pair of parentheses and assigned to the vertex.

Jill. Now that you understand this *simple* way to name vertices, I must explain why it is insufficient for an isomorphism test. Although the two trees of FIGURE 7 are isomorphic, the roots have been assigned the different names ((0) ((0))) and (((0)) (0)) because we didn't insist on concatenating the children's names in some fixed order.

FIGURE 7

Peter. But how can you order children's names? Children's names have no order! Children can be stored in any order. What rhyme or reason should make us order (0)(0) before or after ((0))?

Jill. Easier than you think. First, drop the 0's from names; they are superfluous. Each name becomes just a string of ('s and)'s. Interpreting each '(' as a 1 and each ')' as a 0 lets us interpret each string of ('s and)'s either as a *name* or as a *binary number*. Interpreting a string as a number lets us use the ordinary ordering for binary numbers!

Peter. Although your name's number isn't *natural*, I see that it does allow an ordering. No doubt you will use this ordering as follows. Take a vertex's children, order the children's names, concatenate the names in order, put a pair of parentheses around the concatenated names, and assign this new name to the vertex. Doesn't the cost of putting the names in order before they are concatenated significantly add to the cost of executing Post_Order_Version_One?

Jill. Yes. First let's present the modified version.

```
Post_Order_Version_Two(v: vertex);
Begin
  if v is childless then
    Give v the tuple name "10"
  else
  begin
    For each child w of v do
        Post_Order_Version_Two(w);
```

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

```
    Sort the names of the children of v;
    Set temp to the concatenation of v's sorted children's names;
    Give v the tuple name "1temp0";
  end
end;
```

In [1, p. 83] it is proved that if $A_1, A_2, \ldots, A_m$ are *strings* of 0's and 1's and l_i denotes the *length* of the ith string, then a lexicographical sort can be done in time proportional to Σl_i, that is, proportionate to the combined lengths of all the strings. Thus summing the sort time at each vertex we see that the sort time is proportional to the length of the sum of the vertex names throughout the tree.

Peter. That is going to produce a very crude estimate. Since there are n nodes in a tree, each with length at most $2n$, the total time is bounded by a multiple of n^2.

Jill. Unfortunately, it is *not* a crude estimate. Consider the time it takes to assign the name to the root of a tree of n vertices in one long strand. The names are 10, 1100, 111000, ... with the name of the root consisting of n 1's followed by n 0's. The time to process the name on the i-th level from the bottom is proportional to i. Therefore to compute the *name* of the root, takes time proportional to

$$1 + 2 + \cdots + i + \cdots + n,$$

which is $O(n^2)$.

Peter. Let me make two observations.

Observation 5. *Induction on the level number proves that a vertex's canonical name is a tree isomorphism invariant.*

Observation 6. *Two trees are isomorphic if and only if their roots have identical canonical names.*

In summary, the Post_Order_Version_Two algorithm is intuitively sound, and for infinitely many cases takes $O(n^2)$ time.

Post_Order_Version_Two tests for isomorphism by computing the canonical name of each root node, that is, the canonical name of the 0-th level. Let's modify the algorithm so that it can detect failure early. We introduce the *canonical level name* (rather than the canonical name of a single vertex).

Let me first prove that it is straightforward and takes time proportionate to $\|V\|$ to find all vertices of level i for all values of i. Indeed, it takes $O(\|V\|)$ time to give each vertex its ordinary tuple name with Post_Order_Version_One. While traversing the tree, simply keep track of the current distance to the root (the level number). Therefore, letting $LL[i]$, $i = 0, \ldots, \|V\|$, denote the *LevelList* of all vertices on level i, we can simply add the vertex to its appropriate level list at the same time it receives its ordinary name.

The *canonical level name* is formed by first arranging in order the canonical names of the vertices of that level and then concatenating those names.

Jill. Two observations.

Observation 7. *For all levels i, the canonical name of level i is a tree isomorphism invariant.*

Observation 8. *Two trees T_1 and T_2 are isomorphic if and only if for all levels i, the canonical level names of T_1 and T_2 are identical.*

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

These observations let us test for early failure. Instead of assigning names by Post_Order_Version_Two, we invest a pre-processing time of $O(\|V\|)$ and create the LevelLists $LL[i]$ of all vertices on level i. We then assign canonical names by level, sort by level, and check by level that the *canonical level names agree*. If at any level, the canonical level names disagree, then we stop since the trees are not isomorphic. Although this catches early failures, it still requires sorting and, if the failure does not occur till the last level, it may still take $O(\|V\|^2)$ time. Nevertheless, it does not have to process all the way to the root to detect failure.

Peter. Let me suggest another modification. The long canonical vertex names built by this algorithm contain unnecessary information, namely a recursive encryption of the complete genealogy of all of the descendants of the vertex. This complete encryption of information is propagated all the way to the root. Reading these increasingly lengthy genealogies creates the $\|V\|^2$ time bound.

I claim that keeping this complete *history* serves no useful purpose for the question of whether two trees are isomorphic. Consider the trees in FIGURE 8 that have billions of vertices. Suppose after computing the first 500 million vertices we find the trees have identical canonical names at level j. The canonical names of a and A are hundreds of millions of characters long and *identically* summarize all of the previous genealogy. But for what purpose is this information kept! Why *keep* this genealogy encrypted in a name hundreds of millions of characters long? All that the algorithm needs to know is the fact that the names are the same—the actual names aren't needed! It suffices to give a and A new, short names that indicate that the canonical names (whatever they actually are) match up to this point!

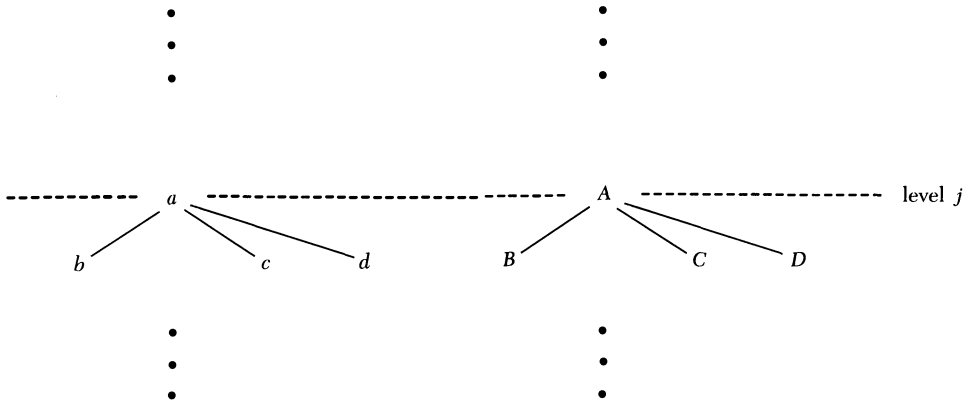

FIGURE 8

Jill. Sure. We could just call them both 1 and refer to them as *condensed* canonical names. Say, this is getting to look like step 6 of the AHU algorithm.

Peter. So how long will it take to determine isomorphism?

Jill. It depends. If you assume that the number of vertices in the tree is small enough to fit in a fixed computer word of k bits (i.e., on a 32-bit machine, trees have no more than 2^{32} nodes), then AHU prove that the bound is $O(\|V\|)$.

On the other hand, if you want a bound that works even when the number of nodes in the tree is so large that numbers must be treated as *strings* of 0's and 1's, then

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

there are trees for which it takes time $O(\|V\| \log_2 \|V\|)$ to calculate the *condensed* canonical name of the root.

Consider the tree of FIGURE 9 with $N = n^2 + n(n+1)/2$ nodes. The tree in FIGURE 9 consists of strands $s_i, i = 1, \ldots, n$, where the strand s_i has $n + i$ vertices. At level $2n$, the condensed canonical name is 1. At level $2n - 1$, the two condensed canonical names are 1 and 2. At level n, the condensed canonical names $1, \ldots, n$ are required. From level $n - 1$ to level 1, no additional canonical names are introduced or deleted. Therefore, the vertices on level 1 to level n have names whose *lengths* go from $\log_2 1$ to $\log_2 n$. Lexicographically sorting this list at level i, $1 \le i \le n$, takes time proportional to the sum of the lengths,

$$\sum_{i=1}^{n} \log_2 i,$$

which is $O(n \log_2 n)$. Since this sorting is repeated n times, once for each i, $1 \le i \le n$, the time taken is at least $O(n^2 \log_2 n) = O(N \log_2 N)$ as claimed.

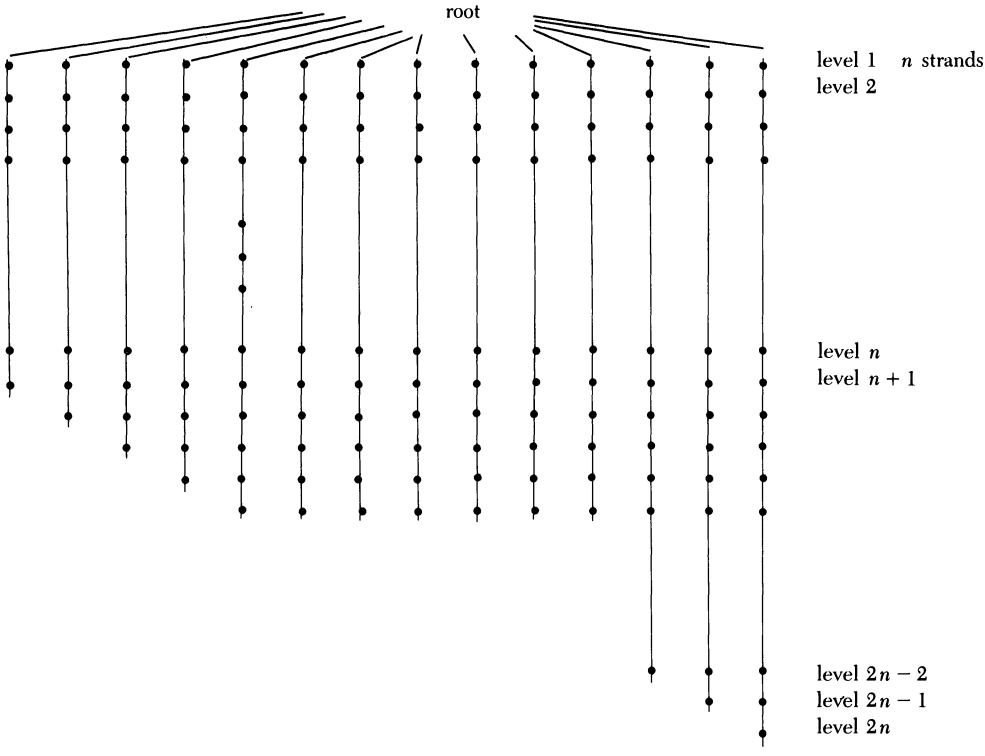

FIGURE 9

Peter. FIGURE 10 summarizes a $O(\|V\| \log_2 \|V\|)$ tree isomorphism algorithm, incorporating provisions for detecting failure early, and the use of condensed canonical names.

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions

```
Tree Isomorphism (T1, T2: trees);
Begin
    Assign all vertices of T1 and T2 to level numbers lists and let hi be the largest level number in Ti;

    If h1 <> h2 then
      write ('trees are not isomorphic'); Halt;
    else
      set h to h1; { h1 = h2 }

    { process from bottom to top level }
    for i:= h downto 0 do
    begin
      { assign vertices their string name }
      For all vertices v of level i do
        If v is a leaf then
          assign v the string 10
        Else
          assign v the tuple 1 i1 i2 ... ik 0, where i1, i2, ..., ik are the strings associated with the
          children of v, in non-decreasing order;
        { assign vertices to temporary sorting lists }
        For all vertices v of level i do
          If v belongs to Tj then
            add v's string to Tj(i);
        Sort T1(i) and T2(i) lexicographically;
        If T1(i) <> T2(i) then
            write ('trees are not isomorphic at level', i); Halt;

        { assign condensed canonical names }
        For all vertices v of level i do
            If v is the k-th element in Tj(i) then
              assign v the binary string for the integer k;

    end;
    write ('the trees are isomorphic')
  end.
```

FIGURE 10

REFERENCES

1. A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Co., Reading, MA, 1974, pp. 84–85.
2. Douglas M. Campbell, The computation of Catalan numbers, this MAGAZINE 57 (1984), 195–208.
3. Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, 1985.
4. Donald Knuth, *The Art of Computer Programming, Fundamental Algorithms*, Vol. 1, Addison-Wesley Publishing Co., Reading, MA, 1973, p. 334.

This content downloaded from 129.96.252.188 on Tue, 15 Dec 2015 00:01:46 UTC
All use subject to JSTOR Terms and Conditions