Computer Enumeration and Generation of Trees and Rooted Trees

J. V. KNOP, W. R. MÜLLER, Ž. JERIČEVIĆ, and N. TRINAJSTIĆ*

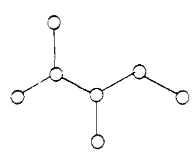
Computer Centre, The University of Düsseldorf, 4000 Düsseldorf, Federal Republic of Germany

Received June 20, 1980

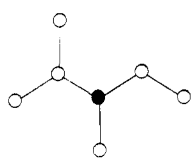
A computer-adopted method for enumerating and plotting trees (alkanes) and rooted trees (substituted alkanes) with N vertices (carbon atoms) is described. Results are compared to some previous work in this area.

INTRODUCTION

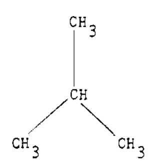
This paper presents a computer-oriented method for enumerating and plotting trees (alkanes) and rooted trees (substituted alkanes). A *tree* is a connected graph with no cycles.[1] A *rooted tree* is a tree in which one vertex has been distinguished from others.[2] This vertex is usually called the *root*. *Alkanes* C_NH_{2N+2} are represented by trees in which the maximal vertex degree is *four*. Alkane trees used here depict only carbon skeletons of alkane hydrocarbons. Substituted alkanes $C_NH_{2N+1}X$ are represented by hydrogen-suppressed rooted trees in which the maximal vertex degree is also four.



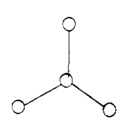
tree



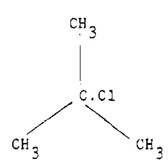
rooted tree



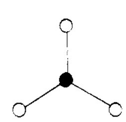
2-methyl-propane



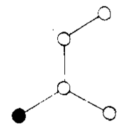
2-methyl-propane graph
(tree)



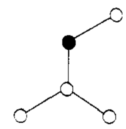
2-chlor-2-methyl-propane



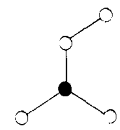
2-chlor-2-methyl-propane graph
(rooted tree)



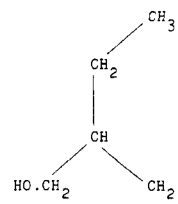
primary root



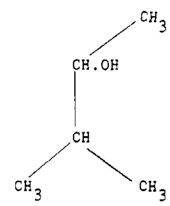
secondary root



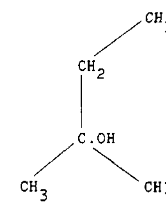
tertiary root



1-hydroxy-2-methyl-butane (primary alcohol)



2-hydroxy-3-methyl-butane (secondary alcohol)



2-hydroxy-2-methyl-butane (tertiary alcohol)

In the case of rooted trees, representing substituted alkanes, we differ the *primary* root (a rooted vertex with degree one), the *secondary* root (a rooted vertex with degree two), the *tertiary* root (a rooted vertex with degree three), and the *quaternary* root (a rooted vertex with degree four). A substituent is, of course, never attached to the quaternary atom. If, for example, alcohols $C_NH_{2N+1}OH$ are considered, the rooted tree(s) with the primary (secondary, tertiary) root would represent the primary (secondary, tertiary) alcohol.

The mathematical theory of trees was developed in the middle of the last century.[3–5] However, Cayley[6,7] was the first who realized the potential of this theory for the enumeration of (hydrocarbon) isomers. He enumerated alkane isomers up to $N = 13$, but the numbers of isomers obtained for C_{12} and C_{13} alkanes (357 and 799) were incorrect.[7] They were corrected 5 years later by Herrmann[8] (355 and 802). It is interesting to note that Losanitsch[9] was arguing that the number of isomers for C_{12} alkane is neither 357 nor 355 but 354. There was quite a discussion between Herrmann[10,12] and Losanitsch[11] at the end of the last century about whose number is correct. Herrmann, of course, produced the correct value.[13]

Since the work of Cayley, the mathematical theory of isomers was continuously developed in two directions. One direction was a development of mathematically well-founded isomer enumeration methods,[13–17] while the other was a development of practical schemes for enumeration of the particular kind of structural isomers.[8,9,18–25] In the early thirties, Blair and Henze were especially active in this area at the University of Texas at Austin. While Henze and Blair[20] enumerated correctly the number of primary, secondary, and tertiary alcohols for a given number of carbon atoms, there is an error in their work on the number of isomeric alkanes. The Henze–Blair number of isomers for C_{19} alkane (147 284) should be corrected (148 284).[22] The Henze–Blair approach was much later a basis for a computer program which allows the calculation of alkane isomers.[26] This program handles alkanes up to 57 carbon atoms in double precision. However, it is not only important *to enumerate* the isomers but also *to display* them all. This became possible with the advancement of computer technology and with the development of techniques for the production of graphs by computer.[27–29] The present work represents an effort in this direction.

METHOD

First we introduce a special representation for trees and rooted trees with N vertices by N tuples of nonnegative integers smaller than N. It allows, as this paper shows, a very efficient and easy way to produce just these tuples and to obtain graphic output from them [or other forms of representation for the underlying (rooted) tree].

This method allows us to mark identity trees and homeomorphically irreducible trees while generating all trees and also to restrict the generation to special types, as, e.g., alkane

0095-2338/81/1621-0091$01.25/0

(rooted) trees. We make extensive use of the notion of lexicographic order, so let us begin with a definition: A K tuple $(a_1, a_2, \ldots, a_K)$ of integers is defined lexicographically smaller than an L tuple $(b_1, b_2, \ldots, b_L)$, if there exists an index j with $1 \leq j \leq L$ so that $a_i = b_i$ for $1 \leq i \leq j$ and either $j = K + 1$ or $a_j < b_j$. Now we map the nonempty rooted trees into the N tuples of nonnegative integers by induction: The trivial (rooted) tree with one vertex is represented by the 1 tuple (0). A given rooted tree with $N > 1$ vertices and M edges incident to the root vertex gives rise to M rooted subtrees by removing the root vertex and all of its edges. These rooted subtrees (taking as the root in the subtree the neighbor of the removed vertex) with $L_1, L_2, \ldots, L_M$ vertices (where the sum $L_1 + L_2 + \ldots + L_M$ is $N - 1$) are by induction equipped with L_i subtuples. We sort these subtuples into the reverse lexicographic order, concatenate the 1 subtuple (M) and these subtuples, and get a tuple of $1 + L_1 + \cdots + L_M = N$ components which we define to be the representative for the rooted tree. Given a tree, we inspect all rooted trees above it (i.e., we select the vertices one after the other as the root for a rooted tree) and assign to the tree the lexicographically largest one of the N tuples obtained.

The same tuples can be constructed by inspecting all Ariadne threads for a given (rooted) tree, i.e., all closed sequences of edges which consider all the edges exactly twice and every vertex at least once (for a rooted tree only those starting at the root vertex). Every such sequence numbers, by the order of the first consideration, the vertices from 1 to N, and we may assign an N tuple to it by setting the ith component to the number of yet unconsidered neighboring vertices when the ith vertex is considered for the first time (i.e., the number of attached edges for the first vertex and one less for the others). The lexicographically largest tuple is then just the tuple defined for the (rooted) tree above. Some useful properties of this representation by N tuples are as follows.

(i) Given an N tuple representing a (rooted) tree, the sum of the first K components is greater than or equal to K for $K < N$ and $N - 1$ for $K = N$. This is useful when computing the extent of a subtree beginning at a given position. It further implies that tuples of different lengths must necessarily have unequal components at a position common to both tuples. When comparing subtrees, this may serve to save one of the two tests for the end of the subtuple.

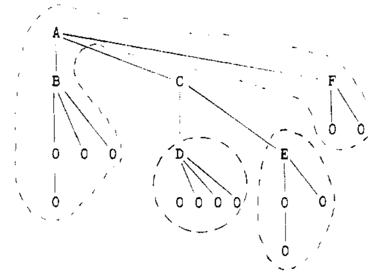
(ii) Given the N tuple for a rooted tree, the N tuple for a rooted tree above the same underlying tree with a neighbor of the old root as the new root vertex may be found by the following simple rearrangement of the given tuple: The given tuple consists of a first component $M > 0$ (the number of branches or neighbors) and subtuples of M, $\{M, [A_{1,1} \ldots, A_{1,L_1}], [A_{2,1}, \ldots, A_{M-1,L_{M-1}}], [A_{M,1}, \ldots, A_{ML_M}]\}$, in lexicographically descendant order, one of which, say $[A_{X+1,1}, \ldots, A_{X+1,L_{X+1}}]$, begins with the new root and consists of a first component $J = A_{X+1,1}$ and subtuples of J, $\{J, [B_{1,1}, \ldots, B_{1,K_1}], [B_{2,1}, \ldots, B_{J-1,K_{J-1}}], B_{J,1}, \ldots, B_{J,K_J}]\}$, again in lexicographically descendent order. Now the new root has $J + 1$ subtrees represented by the subtuples which are old subsubtuples of J becoming the subtuples of the new root $[B_{1,1}, \ldots, B_{1,K_1}]$, $[B_{2,1}, \ldots, B_{J-1,K_{J-1}}]$, $[B_{J,1}, \ldots, B_{J,KJ}]$ and the extuple (without the subtuple representing the new root), $[M - 1, A_{1,1}, \ldots, A_{X,L_X}, A_{X+2,1}, \ldots, A_{M,L_M}]$ becoming the subtuple and properly inserted in a tuple of the new root according to the reverse lexicographic order, say not greater than $[B_{Y,1}, \ldots, B_{Y,K_Y}]$ and not less than $[B_{Y+1,1}, \ldots, B_{Y+1,K_{Y+1}}]$. This leads to the new N-tuple $\{J + 1, [B_{1,1}, \ldots, B_{Y,K_Y}], [M-1, A_{1,1}, \ldots, A_{XL_X}, A_{X+2,1}, \ldots, A_{ML_M}], [B_{Y+1,1}, \ldots, B_{J,K_J}]\}$. This manipulation, which is useful when testing whether a given tuple for a rooted tree represents also the underlying tree, is best illustrated by an example (Chart I).

Chart I

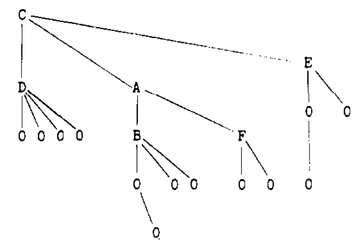
old root new root

V V

(3,3,1,Ø,Ø,Ø,2,4,Ø,Ø,Ø,Ø,2,1,Ø,Ø,2,Ø,Ø)



branches with respect to new root



The old root with its remaining branches forms one branch of the new root, which must be inserted at the right place to achieve reverse lexicographic ordering.

(3,4,Ø,Ø,Ø,Ø,2,3,1,Ø,Ø,Ø,2,Ø,Ø,2,1,Ø,Ø)

(iii) A tree represented by an N tuple $(A_1, \ldots, A_N)$ is homeomorphically irreducible (i.e., no vertex has exactly two adjacent edges) if, and only if, neither the first component, A_1, is 2 nor any other component, A_i, is 1.

(iv) A tree represented by an N tuple is an identity tree if, and only if, there exists neither a second vertex which taken as the root delivers the same N tuple nor any vertex with two identical subtuples. This can be seen by a simple case study.

COMPUTER PROGRAM

Now we describe the computer program for enumerating and plotting (rooted) trees by applying the method which we developed above.

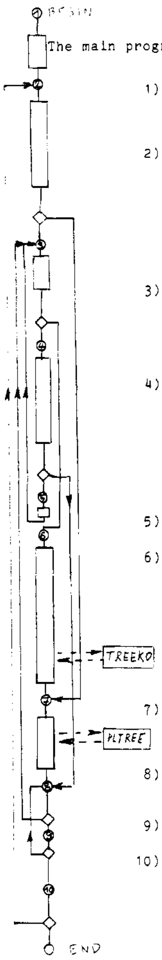
The main program generates the N tuples representing rooted trees in reverse lexicographic order, calls a logical function TREEKO to test for tree property, and, if wanted for this type of (rooted) tree, calls a subroutine PLTREE to produce graphic output.

The main program (Figure 1) nests N loops and selects bounds for inner loops dependant on actual values of outer controlled variables. Although the number of nested loops is variable, we do not need any recursion, since we formulate the N "virtual" loops explicitly by using a vector to store the loop variables and so to contain automatically the generated N tuple.

Therefore the main program may confine itself to three visible nested loops: the outer one controls the number of vertices, the second counts the (rooted) trees for a given number of vertices, and at the inner level there is one loop to find the innermost nonexhausted "virtual" loop and then one to initiate the inner "virtual" loops from there.

The upper bound and first value for a "virtual" loop originates from the fact that the sum of all components for an N tuple must be $N - 1$ and the request that adjacent branches (or subtuples) of the same vertex must be in reverse lexicographic order, i.e., they must be equal, or at the first differing position the second must have a smaller value than the first.

Now many subtuples themselves consist of further subtuples, which consist in their turn of still more subtuples, and so one could expect that a lot of conditions would have to be fulfilled and monitored at the same time. Fortunately, we must pay



The main program operates as follows:

1) Read in parameters.
Initiate some variables,
especially set N, the actual number of vertices,
to the smallest wanted value.

2) Set first lower bound to 1.
Set first component of the N-tuple to N-1 (to
obtain the lexicographically largest possible
N-tuple first).
Reset enumeration counters.
Set an index to the first position.
Set last valid (Nth) component to Ø.
If N < 3 this completes the only possible N-tuple,
so continue at 7).

3) Test indexed position, whether it makes the developing N-tuple lose its homeomorphical irreducibility
(here could also be inserted a test for alkane-property).
Increase the index. If beyond the N-tuple, then
continue at 6).

4) Compute upper bound and lower bound for this position (using RLOC and LBC as described above,
but also taking into account that the sum of all
components must not exceed N-1).
If the old RLOC is satisfied by exhaustion this
implies a non-identical automorphism.
If the upper bound is lower than the lower bound,
we have the remaining failure case mentioned above,
so continue at 8) for the next N-tuple.

5) Set the indexed component to the upper bound.
Repeat from 3).

6) A valid rooted tree has been obtained.
If the first component is not greater than any
other one, this is no tree-representative (by an
input-switch the program can be made to omit the
generation of these cases).
If the first component is by 2 greater than any
other one, this is a tree-representative.
The remaining cases must be studied further by
the logical function TREEKO to decide about tree-property.

7) Count cases.
If the actual N-tuple belongs to the wanted cases,
let subroutine PLTREE produce graphic output.
Set the index to the last valid (Nth) position.

8) If the indexed component is greater than the corresponding lower bound, decrease the component and
repeat from 3).

9) If the index is not at the first position, decrease
the index and repeat from 8).

10) All N-tuples generated.
Write out enumeration counters.
Increase N.
If N-tuples wanted, repeat from 2).

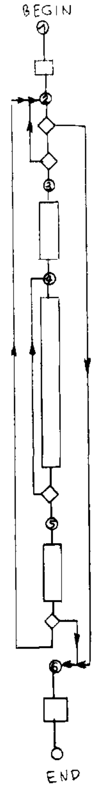
Figure 1. Description of the main program operation.

attention only to the oldest pending structure (i.e., neither exhausted nor fulfilled by a significantly smaller value), because any newer structures are automatically satisfied with the fulfillment of the oldest structure by the latest substructure. This is so since the comparative subtuple of the oldest structure has been built properly and by the pending condition transfers its own reverse lexicographic ordering of subtuples to the compared subtuple. Using this, a simple variable suffices to point to the actual comparative component. A smaller chosen value naturally fulfills the actual reverse lexicographic order condition (abbreviated "RLOC" in the sequel), and this is easily detected. However, we must also provide for the case that RLOC is satisfied by exhaustion of the subtuples, and to this end we keep a pointer to the very beginning of the branch which is subjected to the current RLOC.

Two types of failure are yet possible when generating a "last" branch, i.e., a subtuple which must extend to the end of the N tuple as it is the only remaining branch. The first case is when the starting subtuples of this last branch are chosen so small that the remaining subtuples forced by RLOC cannot fill the rest of the N tuple. This enforces a lower bound condition (abbreviated LBC in the sequel) which is not very difficult to control. The only proper subtuples not allowing a longer and at the same time lexicographically smaller tuple are of the form (1,1, . . ., 1,0) since any value of 2 or larger anywhere in a well-formed tuple allows smaller ones of arbitrary length, namely (1,1, . . ., 1,0).

So the LBC requires that for the subtuples of a last branch, the kth last subtuple must either contain a value greater than 1 or must consume at least one kth of the remaining room in the N tuple. If a subtuple of a last branch is begun, the necessary number of 1's is computed into a counter, which is decremented for every generated 1 and reset to 0 immediately

TREEKO operates as follows:



1) Set an index to 1.

2) Increase the index.
If beyond the N-tuple, the original N-tuple was
the tree-representative, so continue at 6).
If the indexed component is less than the first
component minus 1, repeat from 2).

3) Search towards the original root to find a vertex
for which the N-tuple with respect to this vertex
as the root has already been constructed (may be
the original root).
Name this vertex the father.

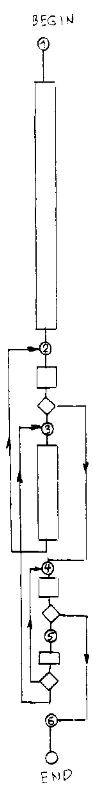
4) Name the next vertex from the father towards the
indexed vertex the son.
Use the permutation trace to find the position
of the son in the N-tuple of the father.
Use property (i) to determine the extent of the
subtuple beginning at the son.
Compare the remainder of the father's N-tuple to
the subtuples of the son to determine the right
place for insertion.
Execute the permutation according to property (ii).
Collect trace-information of applied permutations.
If the son is not the indexed vertex, now name
the former son the father and repeat from 4).

5) Compare the new N-tuple (of the indexed vertex)
to the original N-tuple.
Equality implies a non-identical automorphism.
If the new N-tuple is lexicographically larger,
the orignal N-tuple was no tree-representative,
so then continue at 6) for return.
Otherwise repeat from 2).

6) Return the value .FALSE. if the given N-tuple was
the tree-reprensentative, and return .TRUE. if
the candidate was knocked out.

Figure 2. Description of the operation of the logical function TREEKO.

PLTREE operates as follows:



1) For every component count the number of zeroes
in the subtuple beginning at this component (i.e.
the number of final vertices reachable from the
root through the considered vertex). This number
for the first component is the total number of
final vertices (if the first component is 1 the
root itself is final and a 1 has to be added to
the number).
The edges to the final vertices are spread equally
to all directions (divide PI by the number of final
vertices to obtain the angle increment).
Plot first vertex.
Set an index to 1.
Initialize a base angle.

2) Copy indexed component to a counter.
If indexed component is Ø, the corresponding vertex
is final, so increase base angle by 2 increments,
increase index and continue at 4).

3) Push plot-position and counter onto a stack.
Increase index.
Plot an edge and a vertex (the now indexed one)
using base-angle + K * increment where K is the
number of final vertices reachable through the
new indexed vertex.
Repeat from 2).

4) (Trace back along the Ariadne-thread.)
Pop plot-position and counter off the stack. If
nothing was on the stack, we are ready, so continue
at 6).

5) Draw back to popped plot-position.
Decrement the counter.
If the counter becomes zero, repeat from 4).
Otherwise repeat from 3).

6) Advance to the next frame.

Figure 3. Description of the operation of the subroutine PLTREE.

for any value greater than 1 generated, thus indicating by a countervalue greater than 0 that the lower bound for the actual component is 1 and not 0.

The other case happens when the last branch must be longer than the comparative branch of the RLOC. Because of the above provisions the wanted branch always exists, but often

```
The program listing
                      Main Program
        IMPLICIT INTEGER (A-Z)
        LOGICAL TREEKO
        LOGICAL NOROOT,NOTREE,NOTEST,NOPLOT,SPPLOT,ALKANE
        LOGICAL NOTID,NOHIR,NOTIDT,NOTHIR,MAXTEX,RERUN
        DIMENSION TREE(30),BRA(30),FATHER(30)
        DIMENSION MINIM(30),USED(30)
        DIMENSION NOTID(30),NOHIR(30)
C                 TREE(I) contains the controlled variable of the I-th
C                 nested "virtual" loop and therefore within the N-th
C                 (innermost) loop TREE contains the N-tuple defined
C                 above for the actual (rooted) tree
C                 FATHER(I) contains the index of that vertex from which
C                 vertex I was reached first (tree-theoretic father of I)
C                 i.e. I is one of the vertices counted by TREE(FATHER(I))
C                 BRA(I) contains the number of remaining branches of
C                 vertex FATHER(I) including that beginning at I
C                 USED(I) denotes the number of edges (plus 1) used up to
C                 but not including I,i.e. the sum TREE(1)+...+TREE(I-1)
C                 NOTID(I) when .TRUE. says that TREE(1),...,TREE(I) do
C                 not represent the beginning of an identity tree
C                 NOHIR(I) when .TRUE. says that no tree having the same
C                 first I components as TREE can be homeomorphically
C                 irreducible
C                 MINIM(I) describes the lower bound condition (in the
C                 sequel abbreviated LBC),i.e. MINIM(I) > 0 says that
C                 the lowest allowable value for TREE(I) is 1
C                 TI,PATER,BRAI,USI,NOTIDT,NOTHIR,MINI serve to optimize
C                 access to the current component of the resp vector
C                               TREE,FATHER,BRA,USED,NOTID,NOHIR,MINIM
C                 REF and ORG describe the actual reverse lexicographic
C                 order condition (RLOC in the sequel)
C                 REF contains the index of the vertex whose number of
C                 branches must be >= TREE(I) to achieve RLO of branches
C                 ORG contains the index of the vertex which was first
C                 subjected to the present RLOC
C                 OLDREF and OLDORG contain REF and ORG from the previous
C                 position
C                 RERUN is .TRUE. during the first pass through the loop
C                 if restart information was delivered, during such a
C                 first pass no N-tuple is generated but the other vectors
C                 are reconstructed
C                 HIRS,IDTS,TREES,ROOTS,TESTS,MAROTS are respectively
C                 homeomorphically irreducible trees,identity trees,
C                 trees,rooted trees,calls to TREEKO,rooted trees with
C                 maximal first component
C                 MAXIM contains an additional upper bound for components,
C                 if NOROOT is .FALSE. then MAXIM ist set to N and has no
C                 effect,if NOROOT is .TRUE. then MAXIM is set to the
C                 first component TREE(1) minus 1 thereby avoiding the
C                 generation of those rooted trees which obviously are not
C                 the trees
C                 MAXTEX is only used to remember the result of a test
C                 within a DO-loop,it is set to .TRUE. originally and
C                 reset to .FALSE. if the examined rooted tree must be
C                 tested by TREEKO to decide about tree-property
C                 ****  i n p u t   d a t a  ****
C                 NOROOT when .TRUE. says that the program shall as far
C                 as possible avoid generating rooted trees which will
C                 obviously be no trees (otherwise those rooted trees are
C                 generated in order to be counted)
C                 NOTREE when .TRUE. says that the program shall plot all
C                 rooted trees and may therefore omit any tests for
C                 tree-property
C                 NOTEST when .TRUE. says that the program shall not use
C                 the function TREEKO (which implies then that some trees
C                 may appear more than once). This was built in as a test
C                 facility to measure the time spent in TREEKO,as this
C                 is the only part for which the expense could not be
C                 securely estimated to be below or equal to the same order
C                 as the number of vertices in the generated trees.
C                 NOPLOT when .TRUE. says that no graphic output shall be
C                 produced
C                 SPPLOT when .TRUE. says that only identity trees and
C                 homeomorphically irreducible trees shall be plotted
C                 N denotes the actual number of vertices for one tree
C                 ALKANE when .TRUE. says that only (rooted) trees with
C                 atmost four edges at any vertex shall be generated
C                 NMIN and NMAX are lower and upper bounds for N
C                 (if on input NMIN <= N <= NMAX then the program expects
C                 as further input an N-tuple and generates new N-tuples
C                 beginning just below the given one, the only necessary
C                 checkpoint/restart information to resume the production
C                 of trees is the last valid (rooted) tree generated)
        READ(5,1000) NMIN,NMAX,N,NOROOT,NOTREE,NOTEST,NOPLOT,SPPLOT,ALKANE
 1000   FORMAT(3I2,10L1)
        IF(N.LT.NMIN) GOTO 20
        READ(5,1001) (TREE(I),I=1,N)
 1001   FORMAT(30I2)
        RERUN=.TRUE.
        GOTO 30
   20   N=NMIN
        RERUN=.FALSE.
   30   MINROT=0
C                 if rooted trees are of no interest,production of
C                 N-tuples with a first component 1 is omitted for N>2
        IF(NOROOT) MINROT=1
C                 ******* installation dependant *******
C                 installation dependent plotter-initialization
        IF(.NOT.NOPLOT) CALL PSTART(40000.,0,999)
C                 initialize counters
   50   HIRS=0
        IDTS=0
        TREES=0
        ROOTS=0
        TESTS=0
        MAROTS=0
C                 Last vertex is always final (no further edges)
        TREE(N)=0
C                 begin with trivial lexicographically largest N-tuple
        IF(.NOT.RERUN) TREE(1)=N-1
C                 valid settings for N<3
        NOTHIR=.FALSE.
        NOTIDT=N.EQ.2
        IF(N.LT.3) GOTO 500
C                 first vertex has all edges at hand
   90   USED(1)=1
        KK=N-1
        TI=TREE(1)
        I=1
C                 generate all rooted trees
        MAXIM=N
C                 useful initializations
        BRA(1)=1
        MINIM(1)=0
        FATHER(1)=1
        ORG=0
C                 the first component never settles the question for
C                 a non-identical automorphism
        NOTID(1)=.FALSE.
C                 if the following line is reached from above,the first
C                 component is concerned,which never has any RLOC;
C                 if reached from below,there exists no RLOC,because the
C                 component concerned has just been decremented from an
C                 allowed value,which definitely fulfills any old RLOC
  100   OLDREF=0
        IF(I.GT.1) GOTO 105
C                 if no rooted trees are wanted,avoid the generation of
C                 any N-tuples for which the first component is not the
C                 strongly largest
        IF(NOROOT) MAXIM=TI-1
C                 apply (PROP3)
        NOTHIR=TI.EQ.2
C                 switch ALKANE allows no more than 4 edges at a vertex
        IF(.NOT.ALKANE) GOTO 110
        IF(TI.GT.4) TI=4
        TREE(1)=TI
        IF(MAXIM.GT.3) MAXIM=3
        GOTO 110
C                 apply (PROP3)
  105   NOTHIR=NOHIR(I-1)
        IF(TI.EQ.1) NOTHIR=.TRUE.
  110   K=I+1
        NOHIR(I)=NOTHIR
        NOTIDT=NOTID(1)
        DO 490 I=K,N
        IF(TI.GT.0) GOTO 150
C                 preceding vertex was final
        J=I-1
C                 is J brother of I ?
  120   IF(BRA(J).GT.1) GOTO 130
C                 no,go back one generation
        J=FATHER(J)
C                 happens infrequently but overall only once for each I-value
        GOTO 120
C                 J is brother of I
  130   REF=J
C                 prepare for new RLOC
        ORG=I
C                 same father naturally
        PATER=FATHER(J)
C                 I is next branch of father
        BRAI=BRA(J)-1
        GOTO 180
C                 preceding vertex is father,no elder
C                 brother to be respected
  150   REF=0
        PATER=I-1
C                 first brother
        BRAI=TI
C                 no old RLOC ?
  180   IF(OLDREF.LE.0) GOTO 200
C                 old RLOC fulfilled ?
        IF(TI.LT.TREE(OLDREF)) GOTO 200
C                 old RLOC not yet exhausted ?
        IF(OLDORG.LE.PATER) GOTO 190
C                 the RLOC was fulfilled by exhaustion of the subtuples,
C                 i.e. they are equal,i.e. we have a non-identical
C                 automorphism
        NOTIDT=.TRUE.
        GOTO 200
C                 the previous RLOC is inherited
  190   REF=OLDREF+1
        ORG=OLDORG
C                 vertex I-1 has used TI edges
  200   USI=USED(I-1)+TI
C                 evaluate LBC
  300   MINI=MINIM(I-1)-1
C                 is previous condition exhausted ?
        IF(MINI.LE.0) GOTO 310
C                 is previous condition still pending ?
        IF(TI.EQ.1) GOTO 400
C                 no previous condition
  310   MINI=0
C                 if the branch beginning at FATHER(I) extends to the end
C                 of the N-tuple,this establishes a new LBC
        IF(USED(PATER).EQ.PATER) MINI=(KK-USI+BRAI)/BRAI
C                 assign all remaining edges to current vertex
  400   TI=N-USI
C                 is reconstruction of vectors from restart data
C                 yet in progress ?
        IF(RERUN) TI=TREE(I)
        IF(REF.LE.0) GOTO 420
C                 the RLOC of subtuples demands a smaller value
        TREF=TREE(REF)
        IF(TI.GT.TREF) TI=TREF
C                 avoid rooted trees if not wanted
  420   IF(TI.GT.MAXIM) TI=MAXIM
C                 store temporary variables
  460   TREE(I)=TI
        FATHER(I)=PATER
        BRA(I)=BRAI
        MINIM(I)=MINI
        USED(I)=USI
        OLDREF=REF
        OLDORG=ORG
C                 test for homeomorphical irreducibility (PROP3)
        IF(TI.EQ.1) NOTHIR=.TRUE.
        NOTID(I)=NOTIDT
        NOHIR(I)=NOTHIR
C                 test if current loop already exhausted
        IF(TI.GT.0) GOTO 490
C                 if the following branch is taken,we have the only
C                 remaining failure case mentioned above
        IF(MINI.GT.0) GOTO 920
  490   CONTINUE
        IF(.NOT.RERUN) GOTO 495
C                 other vectors have been initialized according to the
C                 read values for TREE
        RERUN=.FALSE.
        GOTO 910
  495   IF(REF.LE.0) GOTO 500
C                 if the last RLOC was fulfilled by exhaustion,we have
C                 two identical subtuples and therefore a non-identical
C                 automorphism
        IF(TI.GE.TREF) NOTIDT=.TRUE.
C                 count rooted trees
  500   ROOTS=ROOTS+1
C                 simply all rooted trees wanted ?
        IF(NOTREE) GOTO 800
C                 classify rooted tree (test whether the given N-tuple
C                 is the largest obtainable above the corresponding tree)
        MAXTEX=.TRUE.
        IF(N.LT.3) GOTO 530
        TI=TREE(1)-1
        DO 520 I=2,KK
C                 900:if another component is as large as the first,it
C                 will produce a lexicographically larger N-tuple when
C                 used as the root,therefore this is definitely no the
C                 tree-representative (NOROOT=.TRUE. suppresses this case)
C                 510:if the first component is not by 2 greater than any
C                 other one,this other vertex must be tested by TREEKO
C                 whether it as the root delivers a larger N-tuple
```

```
      IF(TREE(I)-TI) 520,510,900
  510 MAXTEX=.FALSE.
  520 CONTINUE
  530 MAROTS=MAROTS+1
C             can tree-property be seen without a call to TREEKO ?
      IF(MAXTEX) GOTO 800
C             count
      TESTS=TESTS+1
C             no call to TREEKO wanted ?
      IF(NOTEST) GOTO 800
C             test for tree-property
      IF(TREEKO(N,TREE,NOTIDT)) GOTO 900
C             count cases
  800 TREES=TREES+1
      IF(.NOT.NOTIDT) IDTS=IDTS+1
      IF(.NOT.NOTHIR) HIRS=HIRS+1
C             graphics for this case wanted ?
      IF(SPPLOT.AND.NOTHIR.AND.NOTIDT) GOTO 900
C             graphics wanted ?
      IF(NOPLOT) GOTO 899
C             produce graphics
      CALL PLTREE(N,TREES,TREE)
  899 CONTINUE
  900 CONTINUE
C             ******* possibly installation dependant *******
C             this would be the right place for a periodical
C             storing of checkpoint information (only vector TREE
C             and (if needed) the enumeration counters HIRS,IDTS,
C             TREES,ROOTS,TESTS,and MAROTS must be saved)
C
C             for N < 3 there is only one rooted tree
  910 IF(N.LT.3) GOTO 960
C             find innermost non-exhausted "virtual" loop
      I=N-2
  920 TI=TREE(I)-1
      IF(TI) 940,930,950
  930 IF(MINIM(I).LE.0) GOTO 950
  940 I=I-1
      GOTO 920
C             advance this loop
  950 TREE(I)=TI
C             outermost "virtual" loop exhausted ?
      IF(TREE(1).GT.MINROT) GOTO 100
C             ******* 3 lines installation dependant *******
C             installation dependent routine to get CPU-time used
  960 CPU=TIME(I)
C             write statistical information
      WRITE(4,1002) N,HIRS,IDTS,TREES,ROOTS,TESTS,MAROTS,CPU
 1002 FORMAT(8I10)
      N=N+1
C             next number of vertices wanted ?
      IF(N.LE.NMAX) GOTO 50
C             installation dependent plotter-termination
C             ******* installation dependant *******
      IF(.NOT.NOPLOT) CALL PEND
      STOP
      END
      LOGICAL FUNCTION TREEKO(N,TREE,NOTIDT)
C             *******************************
C             * t e s t   f o r   t r e e *
C             *******************************
      IMPLICIT INTEGER (A-Z)
      DIMENSION TREE(30),VALID(30)
      DIMENSION PERM(30,30),TRACE(30,30)
      DIMENSION PERMUT(30),POS(30),BRANCH(30)
      LOGICAL VALID,UNDEC,VALDAT,NOTREE,NOTIDT
C             TREE is described in the main program
C             NOTIDT becomes .TRUE.,when the tree under test is
C             found to be no identity tree
C             POS and BRANCH are stacks used,when scanning the Ariadne-
C             thread defined by the N-tuple,to store index and
C             remaining number of sons for actual fathers
C             VALID(I) says whether PERM(I,*) and TRACE(I,*) have been
C             defined
C             PERMUT stores the last permutation
C             PERM(I,*) contains the N-tuple for I as the root
C             TRACE(I,J) contains the position of vertex J from the
C             original N-tuple in the N-tuple for root I
C             UNDEC says whether the case is yet undecided
C             VALDAT says whether PERM and TRACE have already been
C             initialized
C             NOTREE is a temporary container for the function result
      VALDAT=.FALSE.
C             rooted tree not yet found to be not the tree
      NOTREE=.FALSE.
      T1=TREE(1)
      T0=T1-1
C             initiate stack pointer and first stack elements
      SP=1
      POS(1)=1
      BRANCH(1)=T1
C             last vertex is always final and therefore cannot have
C             a larger N-tuple
      KK=N-1
C             inspect other vertices
      DO 790 L=2,KK
C             number of branches
      TL=TREE(L)
      IF(TL.LE.0) GOTO 610
C             stack non-final vertex
      SP=SP+1
      POS(SP)=L
      BRANCH(SP)=TL
      GOTO 620
C             branch used
  610 BRASP=BRANCH(SP)-1
      BRANCH(SP)=BRASP
      IF(BRASP.GT.0) GOTO 620
C             if no more branches unstack
      SP=SP-1
C             if TREE has been built according to definition,the
C             following GOTO will always be taken
C             happens infrequently but overall only once for each L-value
      IF(SP.GE.1) GOTO 610
C             maximal vertex ?
  620 IF(TL.LT.T0) GOTO 790
C             vertex has at least as many edges as the root and must
C             be examined
      IF(VALDAT) GOTO 639
C             initialize locals only if necessary
      VALDAT=.TRUE.
      DO 630 I=1,N
C             first N-tuple is the given original
      PERM(1,I)=TREE(I)
C             first N-tuple was not rearranged
      TRACE(1,I)=I
C             all other N-tuples are yet undefined
  630 VALID(I)=.FALSE.
C             more useful when rearranging
      PERM(1,1)=T0
C             first N-tuple is defined
      VALID(1)=.TRUE.
C             search father,grandfather,... for valid N-tuple
  639 JJ=SP
  640 POSJJ=POS(JJ)
      IF(VALID(POSJJ)) GOTO 650
      JJ=JJ-1
C             happens infrequently but overall only once for each L-value
      GOTO 640
C             go and compare when N-tuple for vertex concerned has
C             been developed
  650 IF(JJ.GE.SP) GOTO 770
C             N-tuple defined
      FATHER=POS(JJ)
      JJ=JJ+1
C             N-tuple to be defined
      SON=POS(JJ)
C             position of the son in the N-tuple of the father
      V=TRACE(FATHER,SON)
      VALID(SON)=.TRUE.
      I=1
C             beginning of first old branch of new root
      J=V+1
C             search the extent of the subtree beginning at V,by
C             virtue of (PROP1)
      SUM=PERM(FATHER,V)-1
      FINV=V
  660 IF(SUM.LT.0) GOTO 670
C             sum is not negative,therefore the next vertex belongs
C             to the subtree as well
      FINV=FINV+1
      SUM=SUM+PERM(FATHER,FINV)-1
      GOTO 660
  670 SUM=0
C             find position for new branch of new root (old root and
C             remaining branches of old root)
      UNDEC=.TRUE.
      JK=J
      DO 690 K=JK,FINV
      HELP=PERM(FATHER,I)-PERM(FATHER,K)
C             leave loop if first difference between old and new
C             branch is positive
      IF(HELP.GT.0.AND.UNDEC) GOTO 700
      IF(HELP.GE.0) GOTO 675
C             first difference is negative,so this old branch is
C             larger than the new one
      UNDEC=.FALSE.
      GOTO 680
  675 I=I+1
C             new branch of new root is built by concatenating
C             remaining branches of old root
      IF(I.EQ.V) I=FINV+1
C             test end of old branch of new root
  680 SUM=SUM+PERM(FATHER,K)-1
      IF(SUM.GE.0) GOTO 690
C             old branch of new root exhausted,leave the loop if it
C             was not larger than new branch (new one may be put in
C             front of this old branch)
      IF(UNDEC) GOTO 700
C             old branch was larger than new,setup to compare new
C             to next old branch
      SUM=0
      J=K+1
      I=1
      UNDEC=.TRUE.
  690 CONTINUE
  700 CONTINUE
C             perform rearrangement
      DO 750 I=1,N
      IF(I.GE.V) GOTO 710
C             larger branches of old root
      K=I+J-V
      GOTO 740
  710 IF(I.GE.J) GOTO 720
C             larger branches of new root
      K=I-V+1
      GOTO 740
  720 IF(I.GT.FINV) GOTO 730
C             smaller branches of new root
      K=I+N-FINV
      GOTO 740
C             smaller branches of old root
  730 K=I-FINV+J-1
  740 PERM(SON,K)=PERM(FATHER,I)
C             I=old,K=new position
      PERMUT(I)=K
  750 CONTINUE
C             collect trace information of applied permutations
      DO 760 I=1,N
      TFI=TRACE(FATHER,I)
  760 TRACE(SON,I)=PERMUT(TFI)
C             happens infrequently but overall only once for each L-value
      GOTO 650
C             compare N-tuples
  770 CONTINUE
      DO 780 I=1,N
      IF(PERM(L,I)-PERM(1,I)) 790,780,775
  775 NOTREE=.TRUE.
      GOTO 795
  780 CONTINUE
C             tuples are equal,i.e. TRACE(L,*) describes a
C             non-identical automorphism
      NOTIDT=.TRUE.
  790 CONTINUE
  795 CONTINUE
      TREEKO=NOTREE
      RETURN
      END
      SUBROUTINE PLTREE(N,SERNUM,TREE)
C             *********************************
C             * g r a p h i c   o u t p u t *
C             *********************************
      IMPLICIT INTEGER (A-Z)
      REAL X,Y,XI,YI,XJ,YJ,X0,Y0,XD,YD,SIZE,WIDE,LENG
      LOGICAL VALID
      DIMENSION TREE(30),VALID(30)
      DIMENSION FATHER(31),BRANCH(31),FINALS(31)
      DIMENSION X(60,30),Y(60,30),XI(30),YI(30)
C             TREE is described in the main program
C             BRANCH(I) contains at any moment,when scanning the
C             Ariadne-thread,the number of yet unused edges starting
C             at vertex I
C             FATHER(I) contains the index of that vertex from which
C             vertex I was reached first (tree-theoretic father of I),
C             i.e. I is one of the vertices counted by TREE(FATHER(I))
C             FINALS(I) contains the number of final vertices which
C             can be reached from vertex 1 through this vertex I
C             XI(I),YI(I) contain the plot-coordinates of vertex I
C             XJ,YJ contain the plot-coordinates of the current vertex
C             X0,Y0 contain the plot-coordinates of vertex 1
C             X(K,J),Y(K,J) contain (only when VALID(J).EQ..TRUE.) the
C             plot-coordinate-differences for all needed directions of
C             edges (to save SIN and COS evaluations)
C             XD,YD contain plot-coordinate-differences for vertex 1
C             of different trees
```

```
C              SIZE contains the plotting size of the vertex-symbol
C              (centered symbol no.1,octagon)
C              LENG contains the length of one edge
C              WIDE contains half the plot paper width
C              PATER points to the youngest vertex with remaining
C              branches
      DATA VALID/30*.FALSE./
      DATA NOLD/0/,X0/0./
C              ******* possibly installation dependant *******
      DATA XD/3.6/,YD2.4/,SIZE/.04/,WIDE/40./,LENG/.3/
      IF(NOLD.EQ.N.AND.SERNUM.GT.1) GOTO 810
C              begin new column for new number of vertices
      NOLD=N
      YD=ABS(YD)
      Y0=YD
      X0=X0+XD
  810 CONTINUE
C              useful initializations
      PATER=N+1
      BRANCH(PATER)=PATER
      FINALS(PATER)=0
C              count final vertices
      DO 830 I=1,N
C              branches starting at I
      TI=TREE(I)
C              father is the youngest vertex with remaining
C              branches
      FATHER(I)=PATER
C              initially all branches are unused
      BRANCH(I)=TI
C              is vertex I final ?
      IF(TI.GT.0) GOTO 825
C              I is final
      FINP=1
      FINALS(I)=FINP
C              back to father
  820 FINP=FINP+FINALS(PATER)
C              increase father's final count by that of son
      FINALS(PATER)=FINP
C              this branch has been used
      BRAP=BRANCH(PATER)-1
      BRANCH(PATER)=BRAP
C              branches left ?
      IF(BRAP.GT.0) GOTO 830
C              no more branches,so back to father
      PATER=FATHER(PATER)
C              happens infrequently but overall only once for each I-value
      GOTO 820
C              vertex I is not final and therefore has sons (branches)
  825 PATER=I
      FINALS(I)=0
  830 CONTINUE
C              number of final vertices is number of final vertices
C              reachable through vertex 1
      FINMAX=FINALS(1)
C              (+1 if vertex 1 is itself final)
      IF(TREE(1).LE.1) FINMAX=FINMAX+1
C              compute plot-coordinate differences
      FIN2=FINMAX+FINMAX
C              (if not yet available)
      IF(VALID(FINMAX)) GOTO 850
      VALID(FINMAX)=.TRUE.
C              final edges spread equally to all directions
      XJ=3.1416/FLOAT(FINMAX)
C              always one inbetween for mean angles
      DO 840 I=1,FIN2
      X(I,FINMAX)=LENG*COS(XJ*FLOAT(I-1))
  840 Y(I,FINMAX)=LENG*SIN(XJ*FLOAT(I-1))
  850 BRANCH(1)=TREE(1)
C              start coordinates
      XJ=X0
      YJ=Y0
C              first vertex
      XI(1)=XJ
      YI(1)=YJ
C              move and mark
C              ******* possibly installation dependant *******
      CALL SYMBOL(XJ,YJ,SIZE,1,0.,-1)
      IF(N.LT.2) GOTO 880
C              initiate edge direction (first edge a little lower
C              than horizontal)
      LOW=1-FINAL(2)
      DO 870 I=2,N
C              angle of any edge is the mean value between the first
C              (LOW+1) and last values (LOW+2*FINALS(I)-1) final edge
C              reachable through this edge
      INDEX=LOW+FINALS(I)
C              angle modulo 2*pi
      IF(INDEX.LE.0) INDEX=INDEX+FIN2
      XJ=XJ+X(INDEX,FINMAX)
      YJ=YJ+Y(INDEX,FINMAX)
C              coordinates vertex I
      XI(I)=XJ
      YI(I)=YJ
C              draw and mark
C              ******* possibly installation dependant *******
      CALL SYMBOL(XJ,YJ,SIZE,1,0.,-2)
      TI=TREE(I)
C              available branches
      BRANCH(I)=TI
      IF(TI.GT.0) GOTO 870
C              final branch,increase lower bound of mean value
      LOW=LOW+2
      PATER=FATHER(I)
C              draw back to father
  860 XJ=XI(PATER)
      YJ=YI(PATER)
C              ******* possibly installation dependant *******
      CALL PLOT(XJ,YJ,2)
C              branch used
      BRAP=BRANCH(PATER)-1
      BRANCH(PATER)=BRAP
      IF(BRAP.GT.0) GOTO 870
C              no more branches,so back to father (if existent)
      IF(PATER.EQ.1) GOTO 870
      PATER=FATHER(PATER)
C              happens infrequently but overall only once for each I-value
      GOTO 860
  870 CONTINUE
C              next start coordinates
  880 Y0=Y0+YD
      IF(ABS(Y0-WIDE).LE.WIDE-ABS(YD)+.1) GOTO 890
      YD=-YD
      Y0=Y0+YD
      X0=X0+XD
  890 RETURN
      END
```

Figure 4. Computer program.

Table I. Number of Trees and Rooted Trees with N Vertices and the CPU Time Needed for the Calculation

no. of vertices (N)	no. of trees[a]	no. of rooted trees[a]	CPU time,[b] s
1	1	1	
2	1	1	
3	1	2	
4	2	4	
5	3	9	
6	6	20	
7	11	48	
8	23	115	
9	47	286	
10	106	719	
11	235	1842	
12	551	4766	
13	1301	12486	1
14	3159	32973	2
15	7741	87811	4
16	19320	235381	12
17	48629	634847	32
18	123867	1721159	90
19	317955	4688676	248
20	823065	12826228	689

[a] The corresponding diagrams of the (rooted) tree graphs may be obtained from the author on request. [b] CDC-CYBER 76.

the program will run into a contradiction first between RLOC and LBC. This could be excluded by additional tests for all components of all N tuples, but it proved less expensive to let the program generate all tuples and then ignore the contradicted ones. Even if many failures of this type occur before generating the next valid N tuple, the expense remains of the order N, because for every such failure the number of components generated and then canceled by searching for the innermost nonexhausted loop is less than or equal to the length of comparative branch of the current RLOC. Now this RLOC is no longer valid for the next trial (it has been satisfied by a smaller value) so that any new failure must originate from an RLOC whose comparative branch must be part of the rest of the newly generated N tuple. But this implies that the total number of backsteps cannot exceeed N. Therefore special provisions for this remaining failure case have been omitted.

The logical function TREEKO (Figure 2) tests whether a given N tuple representing a rooted tree is the representative of the tree or the representative of the underlying tree. This is the case if no other vertex chosen as the root delivers a lexicographically larger N tuple. The comparison is only carried out for the nontrival cases where the new root has as many adjacent edges as the original root. To get the new N tuple, TREEKO makes extensive use of property ii to transfer the root-property edge by edge on the way from the old to the new root.

The subroutine PLTREE (Figure 3) transforms the representation of a (rooted) tree as an N tuple to a graphic representation. It draws the Ariadne thread described by the N tuple and spreads final edges (i.e., edges which lead to vertices with no other edges) equally to all directions. Other edges take the mean value of the largest and smallest angles of final edges reachable through the edge concerned. The vertices were represented by a small octagon (SYMBOL, special symbol no. 1).

The computer program is listed in Figure 4. At the end of this section we want to say a few words about the efficiency of the program. The maximal expense to find out the next N tuple representing a rooted tree is of order N, because there are no nested DO loops and backward jumps occur perhaps infrequently but overall at most once for each step of the DO loop. The expense in PLTREE to produce graphic output from an N tuple is almost proportional to N for the same reason, and the same argument bounds the CPU time for a call to

Table II. Number of Identity Trees and Homeographically Irreducible Trees with N Vertices

no. of vertices (N)	no. of identity trees[a]	no. of homeographically irreducible trees[a]
1	1	1
2	0	1
3	0	0
4	0	1
5	0	1
6	0	2
7	1	2
8	1	4
9	3	5
10	6	10
11	15	14
12	29	26
13	67	42
14	139	78
15	310	132
16	667	249
17	1480	445
18	3244	842
19	7241	1561
20	16104	2988

[a] The corresponding diagrams may be obtained on request.

TREEKO to $N \times N$. Therefore the average expense to find an N tuple representing a tree is of an order less than or equal to $N \times N \times N$ because there are at most N rooted trees above, each of which is produced in order N, tested by the main program in order N and by TREEKO within order $N \times N$ (only if necessary).

Actual runs of the program have shown that these extensive tests are seldom enough to allow estimation of the expense for the generation of trees proportional to the total number of vertices in all generated trees (physical plotter actions as well need real time proportional to the total number of vertices). The efficiency of the method is best illustrated by test runs with dummy PLOT and SYMBOL which save about 90% of CPU time.

RESULTS

In Table I we give the number of all trees and rooted trees for $N = 1,2,\ldots,20$ vertices and the CPU time (in s) needed for the calculation on the computing machine CDC-CYBER 76. These values were compared with the numbers given by Harary in his classical book "Graph Theory".[30] In a separate table (Table II) we give the number of identity trees and homeographically irreducible trees with N vertices. Values for trees with $N = 1,2,\ldots,12$ vertices in Table II were checked against the numbers given by Harary and Prins.[31] The diagrams of the trees, rooted trees, identity trees, and homeographically irreducible trees may be obtained on request from the Computer Centre of the University of Düsseldorf.

If we wish to enumerate only trees corresponding to the carbon skeletons of alkanes (i.e., alkane tree graphs) or the rooted trees corresponding to the structures which could be generated from alkanes on substitution (i.e., alkane rooted tree graphs), the condition of the maximal valency of vertices must be set to four. In Table III we give the number of alkanes C_NH_{2N+2}, the number of alkane rooted trees with the primary, secondary, tertiary, and quaternary roots, respectively, the number of substituted alkanes, $C_NH_{2N+1}X$, and the total number of "rooted" alkanes, respectively.

These results were checked against the numbers given by Read.[13] The alkane diagrams may be also obtained on request from the Computer Centre of the University of Düsseldorf. However, as an example we present in Table IV the output listing containing graphs of all alkane trees with 11 vertices.

We are quite aware that it is very difficult to augment the already rich literature on the enumeration of trees and alkanes.[1,27,32] However, we have presented here, and well documented, a method with several excellent features.

(a) It enumerates trees and alkanes with high accuracy as any existing, cumbersome or elegant, method in literature.

(b) It is easily adopted for the computer calculations.

(c) It possesses an important advantage in comparison with other methods: it produces directly graphs of trees and alkanes. To our knowledge this is a unique method in this respect.

(d) It could be easily adopted for the classroom demonstration of the chemical enumeration problem on the computer.

Therefore, we believe that the presented method should be of general interest because it offers a very efficient approach for handling a class of important chemical (and graph-theoretical) structures by computer.

ACKNOWLEDGMENT

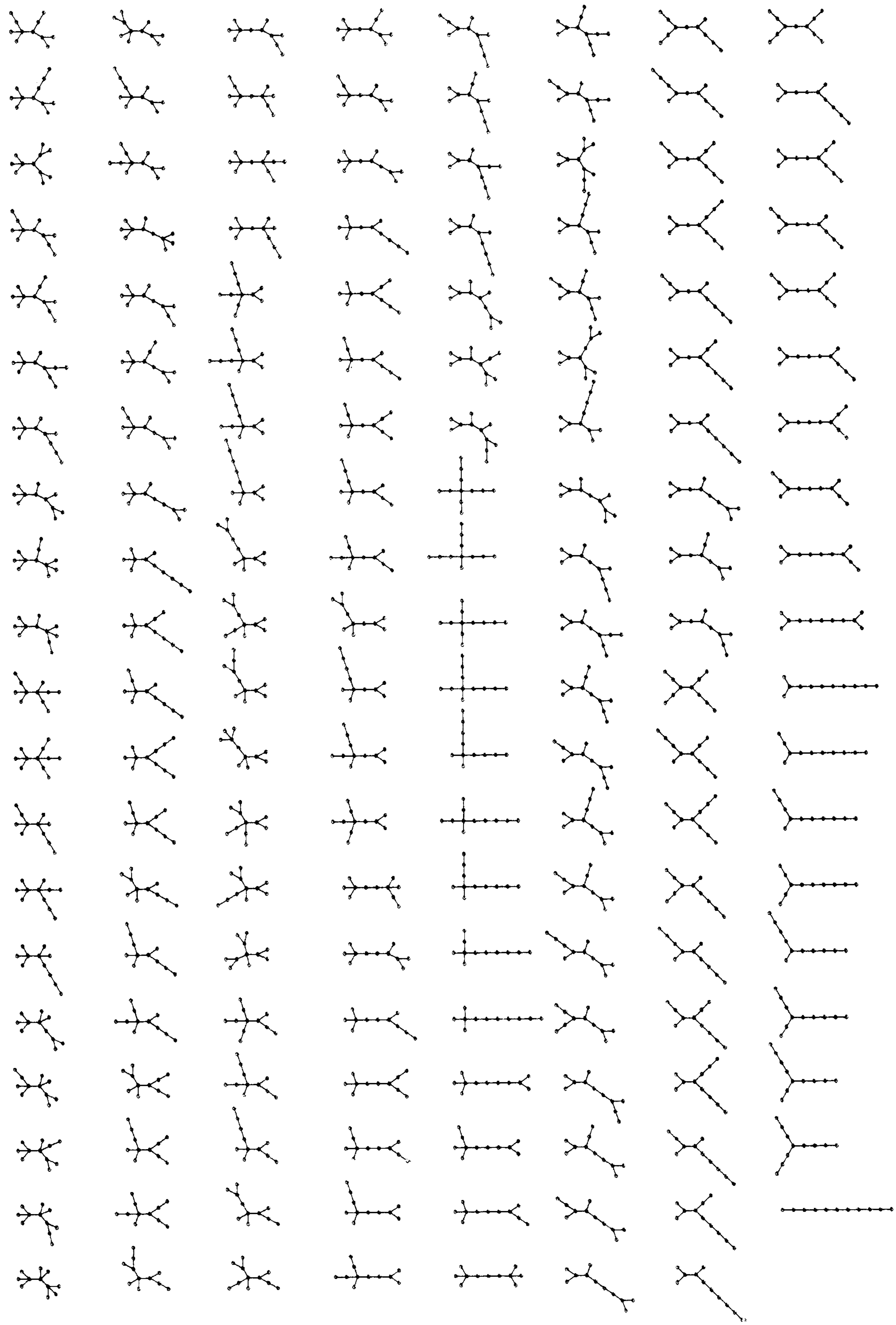
Discussions with Dr. A. J. Harget (Birmingham) and Dr. T. Živković (Zagreb) are gratefully acknowledged. We thank

Table III. Number of Alkanes and Substituted Alkanes with N Atoms

no. of atoms (N)	no. of alkanes[a]	no. of alkanes with a primary root	no. of alkanes with a secondary root	no. of alkanes with a tertiary root	no. of alkanes with quaternary root	no. of substituted alkanes	no. of "rooted" alkane trees
1	1	1	0	0	0	1	1
2	1	1	0	0	0	1	1
3	1	1	1	0	0	2	2
4	2	2	1	1	0	4	4
5	3	4	3	1	1	8	9
6	5	8	6	3	1	17	18
7	9	17	15	7	3	39	42
8	18	39	33	17	7	89	96
9	35	89	82	40	18	211	229
10	75	211	194	102	42	507	549
11	159	507	482	249	109	1238	1347
12	355	1238	1188	631	269	3057	3326
13	802	3057	2988	1594	691	7639	8330
14	1858	7639	7528	4074	1759	19241	21000
15	4347	19241	19181	10443	4542	48865	53407
16	10359	48865	49060	26981	11733	124906	136639
17	24894	124906	126369	69923	30559	321198	351757
18	60523	321198	326863	182158	79743	830219	909962
19	148284	830219	849650	476141	209136	2156010	2365146
20	366319	2156010	2216862	1249237	549959	5622109	6172068

[a] The corresponding diagrams may be obtained from the author on request.

Table IV. Alkane Graphs with 11 Vertices (Copy of the Computer Output)



the reviewers for their helpful comments which led to improvement of the paper.

REFERENCES AND NOTES

(1) F. Harary, "Graph Theory", Addison-Wesley, Reading, MA, 1972, p 32.
(2) Reference 1, p 187.
(3) G. Kirchhoff, *Ann. Phys. Chem.*, **72**, 497 (1847).
(4) A. Cayley, *Philos. Mag.*, **13**, 1–2 (1857).
(5) C. Jordan, *J. Reine Angew Math.*, **70**, 185 (1869).
(6) A. Cayley, *Philos. Mag.*, **67**, 444 (1874).
(7) A. Cayley, *Ber.*, **8**, 1056 (1875).
(8) F. Herrmann, *Ber.*, **13**, 792 (1880).
(9) S. M. Losanitsch, *Ber.*, **30**, 1917 (1898).
(10) F. Herrmann, *Ber.*, **30**, 2423 (1898).
(11) S. M. Losanitsch, *Ber.*, **30**, 3059 (1898).
(12) F. Herrmann, *Ber.*, **31**, 91 (1898).
(13) R. C. Read, in "Chemical Applications of Graph Theory", A. T. Balaban, Ed., Academic, London, 1916, p 25.
(14) G. Pôlya, *Acta Math.*, **68**, 145 (1937).
(15) R. Otter, *Ann. Math.*, **49**, 583 (1948).
(16) L. Weinberg, *Proc. IRE*, **46**, 1954 (1958).
(17) R. C. Read, in "Graph Theory and Applications", Y. Alavi, D. R. Lick, and A. T. White, Eds., "Lecture Notes in Mathematics", No. 303, Springer-Verlag, Berlin, 1972, p 243.
(18) H. Schiff, *Ber.*, **8**, 1542 (1815).
(19) F. Tiemann, *Ber.*, **26**, 1595 (1893).
(20) H. R. Henze and C. M. Blair, *J. Am. Chem. Soc.*, **53**, 3042, 3077 (1932).
(21) C. M. Blair and H. R. Henze, *J. Am. Chem. Soc.*, **54**, 1098, 1538 (1932).
(22) D. Perry, *J. Am. Chem. Soc.*, **54**, 2918 (1932).
(23) D. D. Coffman and C. M. Blair, *J. Am. Chem. Soc.*, **55**, 252 (1933).
(24) H. R. Henze and C. M. Blair, *J. Am. Chem. Soc.*, **55**, 680 (1933).
(25) H. R. Henze and C. M. Blair, *J. Am. Chem. Soc.*, **56**, 157 (1934).
(26) C. C. Davis, K. Cross, and M. Ebel, *J. Chem. Educ.*, **48**, 675 (1971).
(27) See, for example, various articles in "Graph Theory and Computing", R. C. Read, Ed., Academic, New York, 1972.
(28) L. M. Masinter, N. S. Sridharan, J. Lederberg, and D. H. Smith, *J. Am. Chem. Soc.*, **96**, 7702 (1974).
(29) J. B. Hendrikson, *J. Am. Chem. Soc.*, **93**, 6847 (1971); **93**, 6854 (1971); **97**, 5763 (1975); **97**, 5784 (1975).
(30) Reference 1, p 232.
(31) F. Harary and G. Prins, *Acta Math.*, **101**, 141 (1959).
(32) D. H. Rouvray, *Chem. Soc. Rev.*, **3**, 355 (1974).

ACS Committee on Nomenclature: Annual Report for 1980

KURT L. LOENING

Chemical Abstracts Service, Columbus, Ohio 43210

Received February 17, 1981

Nomenclature committees, both national and international, were very active in 1980, resulting in substantial progress in many different fields. A summary of the more important meetings and accomplishments follows.

The *ACS Committee on Nomenclature* held its annual meeting at CAS in November.† Progress of the work of the divisional committees and international commissions was reviewed. In addition, ways of working more closely with ACS Divisions, journal editors and authors as well as general means of promoting good nomenclature were explored. The chairman of the committee addressed the editors of ACS journals on the subject of chemical nomenclature at their conference in Columbus; this should lead to closer cooperation between the two groups. The feasibility study on compiling an authoritative chemical dictionary was extended, while the project on visual aids for chemical nomenclature was dropped. Contact was established between the committee and its recently established British equivalent. The subcommittee on chemical pronunciation continues to be active.

The *IUPAC Interdivisional Committee on Nomenclature and Symbols* (*IDCNS*) functioned effectively this year. It held its annual meeting in Cambridge in September. In addition to the IUPAC publications listed in the Appendix, specific documents in process and thus not yet recorded in this Appendix deal with the following topics: straightforward transformations, transport phenomena, biochemical equilibrium data, chemical kinetics, physicochemical quantities and units in clinical chemistry, calorimetric measurements on cellular systems, and various classes of carbohydrates.

The *IUPAC Inorganic Nomenclature Commission* met in September in Cambridge. Topics under discussion included neutral molecules and compounds, ions and radicals, rings and chains, polyhedral clusters, isopoly- and heteropolyanions, oxo acids, inorganic polymers, and stereochemical nomenclature. These topics were discussed in the context of providing a revision of the 1970 edition of the Red Book. Revised recommendations on the nomenclature of nitrogen hydrides and isotopically modified compounds are expected to be issued next year.

The *IUPAC Organic Nomenclature Commission* met in September in Cambridge. The commission continued its study of the reorganization and revision of the present rules according to a more logical arrangement (Section R) and of a more drastic long-range approach (Section G). In connection with Section G, several specific projects are under way: nodal nomenclature, radial nomenclature, "inorganic" ring nomenclature, nomenclature for delocalized ions and radicals, nomenclature of oxo acids, and general priority rules for numbering. The following topics are so well advanced that publications should be forthcoming within a year or two: lambda convention, classical ions and radicals, cyclophanes, and a revision of the Section E rules on stereochemistry. The 1979 provisional recommendations for the revision of the Hantzsch–Widman nomenclature system for naming heteromonocycles have generated so many diverse opinions and comments that the commission requires additional time and study before issuing definitive recommendations.

The *IUPAC Macromolecular Nomenclature Commission* met in September in Naples. The commission is continuing its work on (a) nomenclature and symbolism of copolymers, (b) subsidiary definitions of terms relating to polymers, (c) definitions for physical properties of polymers, (d) substitutive nomenclature for reacted polymers, (e) nomenclature of inorganic polymers, (f) classification and family names of polymers, and (g) interpenetrating polymer networks. Of these items (a), (e), and (f) are at the most advanced stage with recommendations expected to be issued in 1981 or 1982. A definitive version of the recommendations dealing with stereochemical definitions and notations relating to polymers will be issued in 1981.

† Abbreviations used, not identified in the text, are ACS, American Chemical Society; CAS, Chemical Abstracts Service; IUPAC, International Union of Pure and Applied Chemistry; JCBN, Joint Commission on Biochemical Nomenclature; NC-IUB, Nomenclature Committee of International Union of Biochemistry.

0095-2338/81/1621-0099$01.25/0