Annals of Discrete Mathematics 2 (1978) 107–120.

EVERY ONE A WINNER or HOW TO AVOID ISOMORPHISM SEARCH WHEN CATALOGUING COMBINATORIAL CONFIGURATIONS*

Ronald C. READ
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada

1. Introduction

For many reasons it is often useful to have available lists, or catalogues of all the graphs or other combinatorial configurations of a certain type. Thus, for example, a list of all the graphs on 9 nodes might be used to search for a counter-example to some conjecture, or it might happen that examination of the list may suggest conjectures or indicate possible lines of investigation. The production of such lists is therefore a matter of some interest. Many such lists are in existence, and a historical summary of some of the more important ones for graphs is given in Table 1.

Table 1

Type of graph	Parameters	Number	Catalogued by	Year	References
Graphs	6 nodes	156	I. Kagno	1946	[5]
Graphs	7 nodes	1044	B.R. Heap	1965	Unpublished
Graphs	8 nodes	12,346	B.R. Heap	1969	See [4]
Graphs	9 nodes	274,668	A.K. Dewdney et al.	1974	[1]
Digraphs	4 nodes	218	R.C. Read	1966	[8]
Digraphs	5 nodes	9608	R.C. Read	1966	[8]
Trees	$\leqslant$ 13 nodes	1301 for $p = 13$	P.A. Morris	1972	[7]
	$\leqslant$ 18 nodes	123,867 for $p = 18$	R.J. Frazer	1973	[2]
Tournaments	$\leqslant$ 7 nodes	456 for $p = 7$	P. McWha	1973	[6]

* This research was supported by the National Research Council, grant A8142.

Naturally, such lists are restricted to fairly small graphs since the numbers grow quite rapidly with the size of the graph. As an indication of the magnitude of the problems that have not yet been tackled, note that the number of graphs on 10 nodes is 12005168; the number of digraphs on 6 nodes is 1540744; the number of trees of 19 nodes is 317955; and the number of tournaments on 8 nodes is 6880.

Despite many differences, the above lists were all obtained by what is basically the same method — what we can call the "classical method". This is a recursive procedure which produces larger graphs from an existing list of smaller graphs. Thus for graphs on a given number of nodes we produce the graphs having $q+1$ edges from a list of graphs having q edges, and so on. To be more specific, let us take as a prototype problem that of constructing all digraphs on 5 nodes. Suppose we have a list $\mathscr{L}_q$ of all such digraphs having q arcs, and to each digraph in this list we add an extra arc in all possible ways. Clearly we shall obtain every digraph having $q+1$ arcs in this way, and equally clearly there will be many duplicates in the collection of digraphs thus obtained. It is therefore necessary to eliminate any duplicates from the new list, $\mathscr{L}_{q+1}$, that is being prepared, and the best time to do this is concurrently with the preparation of the list. In other words, as each digraph on $q+1$ edges is produced it is checked against the digraphs that are already in the list $\mathscr{L}_{q+1}$, and is rejected if it (or, more correctly, an isomorph of it) is already in the list, and added to the list if not.

It is therefore necessary to have a method for detecting the isomorphism or otherwise of two configurations of the kind being listed. For graphs or digraphs in general, this is known to be a difficult problem (see, for example, [10]), but since these catalogues are necessarily restricted to fairly small graphs, the detection of isomorphs is here not such a great hurdle. For example, in cataloguing digraphs on 5 nodes the brute force procedure of running through all the 120 permutations of the nodes of a digraph is not too time-consuming, and is certainly easier to programme than a more sophisticated technique. The main hurdle that the classical method has to overcome, and which chiefly limits the size of the configurations that can be listed, is the necessity of looking through the whole of the existing list $\mathscr{L}_{q+1}$ whenever a digraph on $q+1$ arcs is produced. If we recall that for the larger digraphs the list $\mathscr{L}_{q+1}$ will generally be too long to be stored in the immediate access memory of a computer, and will therefore have to be kept on magnetic tape or disc; and that this list has to be searched every time that a potential candidate for the list is produced (even when, as happens very frequently, it turns out to be a "looser", i.e., it does not get on to the list because an isomorph is already there) we can readily appreciate why it is that lists of graphs on 10 nodes or of digraphs on 6 nodes (each of which would be very useful) do not at present exist.

In this paper I shall demonstrate an algorithm for producing lists of the type mentioned above, without having to look back over the list being produced in order to eliminate duplicates. That is to say, when a candidate for the new list is constructed we can tell, simply by examining the configuration itself, whether it is a "winner" or not.

2. The general problem

Let us first describe a general cataloguing problem related to a set S of combinatorial configurations of a given type. We shall suppose that the elements of S are classified according to some parameteı which we shall usually call q. Thus we have a sequence of sub-sets $S_0, S_1, S_2, \ldots$, where S_q is the set of configurations for which the value of the parameter is q. We also have a relation of "isomorphism", which is an equivalence relation over each S_q. The problem is to prepare, for each value of q, a list $\mathscr{L}_q$ containing exactly one configuration from each isomorphism class. We shall suppose that we have a "canonicity definition" which enables us to pick out from any isomorphism class one particular configuration, which we shall call the "canonical configuration". The canonical configuration will be the one which represents, in $\mathscr{L}_q$, the isomorphism class to which it belongs. In addition we define an order relation (the "list order", denoted by "$<$") over the elements of each S_q. If $A < B$ we shall say, for convenience, that "A comes before B" or "A precedes B" or "B comes after A", and so on. If we want to allow the possibility that A and B are the same configuration we shall write, for example, $A \leqslant B$. Finally we shall need what we can call an "augmenting operation" whereby from one element of S_q we can produce an ordered sequence of elements of S_{q+1}. It will appear shortly that we shall require the order in which these elements of S_{q+1} are produced from a single element of S_q to be the list order, so we do not need to have a separate symbol for this order.

We shall now define a type of algorithm which we shall call an "orderly algorithm" for producing the list $\mathscr{L}_{q+1}$ from a list $\mathscr{L}_q$.

An orderly algorithm is one of the following form.

(1) Start with the list $\mathscr{L}_q$ of canonical configurations arranged according to the list order "$<$". The list $\mathscr{L}_{q+1}$ is initially empty.

(2) Take, in order, each configuration of $\mathscr{L}_q$, and act on it with the augmenting operation to get a sequence of elements of S_{q+1}. As each of these configurations is produced, apply to it the following rule:

Rule: If the configuration is canonical, and if it comes after the last element at present in $\mathscr{L}_{q+1}$, add it to $\mathscr{L}_{q+1}$; otherwise ignore it.

(3) When every element of $\mathscr{L}_q$ has been treated in this way, the list $\mathscr{L}_{q+1}$ is complete.

Another way of expressing the basic idea behind this type of algorithm is that

(i) each list consists of the canonical configurations only, arranged in the list order;

(ii) if a configuration produced by the augmenting operation can be added to the list $\mathscr{L}_{q+1}$ in conformity with (i), then it is added; otherwise it is ignored.

There will be many different choices for the definitions of canonicity and augmenting operation, and for the order in which to arrange the lists. It cannot be expected (and in fact is not true) that the above algorithm will be effective no matter what choices are made. However, we shall see that for a wide variety of combinatorial configurations these choices *can* be made in such a way that there is an effective orderly algorithm for producing the configurations. We therefore need to find out how to make these choices so that an orderly algorithm will exist.

There are two fairly obvious necessary conditions for the existence of an orderly algorithm.

Condition 1. *Each canonical configuration of S_{q+1} can be produced by the augmenting operation from at least one canonical configuration in S_q.*

The proof is immediate; any canonical configuration that could not be produced in this way would never get on to the list $\mathscr{L}_{q+1}$, since it would never be produced from any element of $\mathscr{L}_q$.

In general it will happen that several elements of $\mathscr{L}_q$ will produce the same canonical configuration X. Let us denote by $f(X)$ the first element of $\mathscr{L}_q$ which produces X. Then we have a mapping $f : \mathscr{L}_{q+1} \rightarrow \mathscr{L}_q$, and this mapping must obey the following necessary condition:

Condition 2. *The mapping f is weakly monotonic, i.e., if $X, Y \in \mathscr{L}_{q+1}$ and $X < Y$, then $f(X) \leqslant f(Y)$.*

Proof. Suppose that f is not weakly monotonic. Then there exist elements X, Y in $\mathscr{L}_{q+1}$ such that $X < Y$ but $f(Y) < f(X)$. Now it is clear that if X is not added to the list $\mathscr{L}_{q+1}$ when it is produced for the first time from $f(X)$, then it cannot be added at any later stage of the algorithm. Consider the occasion when $f(Y)$ is being considered in Step 2 of the algorithm. The canonical configuration Y is among those produced from $f(Y)$, and will be added to $\mathscr{L}_{q+1}$. Hence when the algorithm has finished with $f(Y)$, Y is certainly present in $\mathscr{L}_{q+1}$. But this means that when X is produced from $f(X)$ (later in the algorithm, since $f(Y) < f(X)$) it cannot be added to the list. The presence of Y "blocks" the addition of X to the list $\mathscr{L}_{q+1}$. Hence if this condition is not satisfied the algorithm will fail.

Conditions 1 and 2, taken together, do not form a sufficient condition for the algorithm to be effective, since no account has been taken so far of what happens when $f(X) = f(Y)$. If, for example, X and Y in S_{q+1} are produced for the first time from the same element of $\mathscr{L}_q$, and if $X < Y$, then it is necessary that X be produced before Y; otherwise Y will be already in $\mathscr{L}_{q+1}$ when X is produced and will block the addition of X to the list. This situation is avoided if the following condition is satisfied.

Condition 3. *From any given configuration in $\mathscr{L}_q$ the configurations of S_{q+1} produced by the augmenting operation are produced in the list order.*

Condition 3, as stated, is not a necessary one. It applies to *all* the configurations produced by the augmenting operation whereas we need it only for those that are new (configurations that are not added to $\mathscr{L}_{q+1}$ can be produced in any order whatever without affecting the operation of the algorithm). However, it is not asking very much to require that these latter configurations (whose order does not matter anyway) conform to the list order, and in practice the "natural" choices for the augmenting operations are systematic procedures which produce the larger configurations in order anyway. Thus we have the following theorem.

Theorem. *A sufficient condition for the orderly algorithm to be effective is that the definitions of canonicity, the list order, and the augmenting operation satisfy Conditions* 1, 2, *and* 3.

Proof. Condition 1 ensures that every canonical configuration X in S_{q+1} is produced at least once. Condition 2 ensures that when X is produced for the first time from $f(X)$ there cannot be in $\mathscr{L}_{q+1}$ an entry Y produced from $f(Y) \neq f(X)$, which follows X in $\mathscr{L}_{q+1}$ and whose presence will therefore block the addition of X to the list. Condition 3 ensures the same thing when $f(X) = f(Y)$.

3. An example

To illustrate the concepts introduced in Section 2 we shall refer briefly to the prototype problem of listing the digraphs on 5 nodes. For this the set S is the set of *labelled* digraphs of 5 nodes, or, equivalently, of the 0-1 adjacency matrices that correspond to them. S_q will be the set of those digraphs having q arcs, corresponding to adjacency matrices having exactly q 1's. Isomorphism is the usual isomorphism between digraphs; adjacency matrices of two isomorphic digraphs will be convertible, one to the other, by simultaneous permutation of the rows and columns by some permutation.

To construct an algorithm we need definitions of canonicity, list ordering and the augmenting operation. If we write the non-diagonal elements of an adjacency matrix row by row we get a string of 20 bits, which we can interpret as a binary integer. Define a canonical digraph to be one having the largest such integer of all the digraphs in its isomorphism class. This integer is called its "code". The list order will be defined as decreasing order of code. The augmenting operation will be taken to be the systematic changing of a 0 to a 1, in the order in which the elements occur in the 20-bit string. (This operation can, in fact, be improved on, as we shall see later).

With these choices of definitions the orderly algorithm will work. (The proof, for a more general problem, comes in the next section). We note here the advantages over the classical method. The production of candidates for $\mathscr{L}_{q+1}$ is easy; we change an existing 0 to a 1 in an adjacency matrix. We need to determine whether the new matrix is canonical, and this might take some time; but the classical method requires at least this amount of time anyway, merely to find the code of the new matrix in order to search the list $\mathscr{L}_{q+1}$. The new method will often require less time, since the discovery of any permutation of rows and columns giving a larger 20-bit code is enough to show non-canonicity.

Where the new algorithm really scores is that if a matrix has been shown to be canonical, and if its code is less than the last code in $\mathscr{L}_{q+1}$, it can be directly added to $\mathscr{L}_{q+1}$, which simply means writing it on an output file, after which it is not required again during the algorithm. The list $\mathscr{L}_{q+1}$ is never searched; only the last code in the current $\mathscr{L}_{q+1}$ needs to be kept and updated by the generating programme.

For the problem of cataloguing all digraphs on six nodes, the longest list to be handled is that containing the digraphs with 15 arcs, which are 220922 in number. Digraphs with more arcs can be obtained immediately from those with fewer arcs by complementation. There is no problem in writing 220922 codes on a magnetic tape, and hence the cataloguing of these digraphs is a feasible operation — one which we are currently undertaking along with other cataloguing projects at the University of Waterloo.[1]

4. A general problem

We shall now consider a problem of some generality for which an orderly algorithm exists. Let S be the set of vectors of dimension n whose elements are taken from the set $\{0,1,2,\ldots,k-1\}$. Let S_q be the set of those for which the sum of the elements is q. Let G be a given permutation group of degree n, and let us say that two vectors v_1, v_2 in S are isomorphic if there exists a permutation in G which, acting on the subscripts of v_1, converts it into v_2. We shall find an orderly algorithm for producing the lists $\mathscr{L}_q$, where $\mathscr{L}_q$ contains one representative from each equivalence class of elements of S_q.

Define the code of a vector v, denoted by code (v), to be the integer in the scale of k, formed by its elements. Thus if $k = 4$, $n = 7$ and $v = (1,0,3,1,2,0,0)$, then

$$\text{code}(v) = 1031200_4 = 4960_{10}.$$

The canonical vector of an equivalence class is defined to be the one which has the largest code.

For the list order we shall take the descending order of codes. The augmenting algorithm will be as follows. Given a vector v in $\mathscr{L}_q$ we scan it from right to left and find the rightmost non-zero element. If this element is less than $k-1$, we first form a vector by increasing this element by 1. The remaining vectors are obtained by changing in turn each of the trailing 0's of v to 1, proceeding from left to right. Thus from $(1,0,3,1,2,0,0)$ we obtain $(1,0,3,1,3,0,0)$, $(1,0,3,1,2,1,0)$ and $(1,0,3,1,2,0,1)$. Since the scanning is from left to right, these vectors are produced in descending sequence of their codes, and hence Condition 3 is verified.

To show that Condition 1 holds we prove the following theorem.

Theorem. *If w^* is a canonical vector in S_{q+1}, and if the last non-zero element of w^* is decreased by 1 to give a vector $f(w^*)$, then $f(w^*)$ is canonical.*

[1] This catalogue has now been completed (August 1976). Details can be obtained from the author.

Proof. There will be canonical vectors in $\mathscr{L}_q$ which differ, by 1, in one element only, from some isomorph w of w^*; they are the canonical isomorphs of the vectors obtained by decreasing by 1 some element of w^*. Of these vectors let v^* denote the one which occurs first in $\mathscr{L}_q$, i.e., which has the largest code. Thus code $(v^*) <$ code $(w) \leq$ code (w^*). Compare v^* and w^*, and let i and j be respectively the least and greatest values of α for which $w^*_\alpha > v^*_\alpha$, i.e.,

$$v^* = (\cdots\cdots\cdots\, v^*_i \cdots\cdots\cdots\, v^*_j \cdots\cdots\cdots)$$

$$w^* = (\underbrace{\cdots\cdots\cdots}_{\text{identical}}\, \underset{\substack{\uparrow \\ v^*_i < w^*_i}}{w^*_i} \cdots\cdots\cdots\, \underset{\substack{\uparrow \\ v^*_j < w^*_j}}{w^*_j}, \underbrace{\cdots\cdots\cdots}_{w^*_\alpha \leq v^*_\alpha}).$$

If $w^*_i > v^*_i + 1$, denote by x the vector obtained from w^* by decreasing w^*_i by 1. Let x^* be its canonical isomorph. Then

$$\text{code}\,(v^*) < \text{code}\,(x) \leq \text{code}\,(x^*). \tag{4.1}$$

But x differs in one place only from w^* hence x^* differs in one place only from some isomorph of w^*. But code $(x^*) >$ code (v^*), which contradicts the definition of v^*.

Hence $w^*_i = v^*_i + 1$. Suppose $j > i$, and this time let x denote the vector obtained from w^* by decreasing w^*_j by 1. Again (4.1) holds and we get the same contradiction.

Hence we must have $j = i$. This means that v^* and w^* differ only in the ith place; for otherwise the sum of the elements of v^* would exceed q.

Suppose that in w^* this element (in the ith place) is not the last non-zero element. Then we have

$$v^* = (\cdots\cdots\cdots, a, \cdots\cdots\cdots, b, 0\,0 \cdots 0)$$

$$w^* = (\underbrace{\cdots\cdots\cdots}_{\text{identical}}, \underset{\substack{\uparrow \\ i^{\text{th}}\text{ place}}}{a+1}, \underbrace{\cdots\cdots\cdots}_{\text{identical}}, \underset{\substack{\uparrow \\ \text{last non-zero}}}{b}, 0\,0 \cdots 0).$$

Define

$$x = (\cdots\cdots\cdots, a+1, \cdots\cdots\cdots, b-1, 0\,0 \cdots 0).$$

Then x is a vector (not necessarily canonical) which differs by 1 from w^* in one place only. Once more the inequalities (4.1) are seen to hold, and we get a contradiction.

Hence v^* is the vector obtained from w^* by decreasing its *last* non-zero element, i.e., it is the vector $f(w^*)$. Since v^* was canonical the theorem is proved.

From this theorem it follows immediately that w^* will be produced by the augmenting operation from the canonical vector v^*, and that $v^* = f(w^*)$, with f defined as in Section 2.

To show that Condition 2 holds, consider two vectors y and z of $\mathcal{L}_{q+1}$, with y occurring before z. Then $\operatorname{code}(y) > \operatorname{code}(z)$. Compare the elements of y and z, from left to right, and let i be the smallest integer such that $y_i > z_i$. Thus the first $i-1$ elements of y and z will be the same.

Since y and z have the same element sum, $q+1$, it is clear that z_i cannot be the last non-zero in z. If y_i is not the last non-zero in y, then changing the last non-zeros, to get the vectors $u = f(y)$ and $v = f(z)$, cannot affect the order of the codes. Hence $\operatorname{code}(u) > \operatorname{code}(v)$, and Condition 2 holds.

If y_i is the last non-zero in y, and if $y_i > z_i + 1$, then subtracting one from y_i still leaves it larger than z_i, and hence again we have $\operatorname{code}(u) > \operatorname{code}(v)$. On the other hand, if $y_i = z_i + 1$, then (from the element-sum argument) z has just one non-zero element — a "1" — somewhere to the right of z_i. Thus when the final non-zeros are decreased by 1, y and z will yield the same vector. Hence Condition 2, which requires only weak monotonicity, still holds.

Hence all three conditions hold, and it follows that, with the given definitions, the orderly algorithm will work.

As special cases of this general problem we have the following:

(A) The cataloguing of digraphs (our prototype problem).

For this the vectors are the 20-bit strings; the group is the group of permutations of the elements of a string induced by the permutations of the nodes, and $k = 2$. Note that the augmenting operation has been improved; to produce new digraphs we need only change *trailing* 0's to 1's.

(B) The cataloguing of graphs.

The vectors are formed from the upper-triangular portion only of the adjacency matrix. Otherwise the problem is as above in (A). If $k > 2$ we allow multiple edges up to $(k-1)$-fold edges. If the diagonal elements of the adjacency matrix are included in the vector, then graphs with loops (possibly multiple) are allowed.

(C) Other variations.

Many variations are possible, and one that was used to test the algorithm (before I could prove it!) was the production of graphs with a hamiltonian circuit whose edges are distinguished from the other edges of the graph. The presence of this distinguished hamiltonian circuit forces the group G to be isomorphic to a dihedral group, thus making it easy to test for canonicity (for graphs and digraphs the group G is isomorphic to a symmetric group).

Those familiar with enumerational methods will recognize that the configurations in the general problem are those that are enumerated by an application of Pólya's theorem (see, for example [3]), yielding the configuration counting series

$$Z(G; 1 + x + x^2 + \cdots + x^{k-1}).$$

(*Note*: We may well speculate whether there is an orderly algorithm which would be analogous, in a similar way, to Pólya's theorem for a general figure counting series (as opposed to $1 + x + x^2 + \cdots + x^{k-1}$ for which there is only one figure with each content). I have such an algorithm for the case of a finite total number of figures, but it is inelegant. An elegant solution of the completely general problem has so far eluded me).

5. Rooted trees

To produce lists of rooted (unlabelled) trees by means of an orderly algorithm we need a suitable augmenting operation and a suitable ordering for the lists. Let us look first at the matter of the list order.

We shall order each list by defining a code for each rooted tree, and listing the trees by order of their codes. There are many ways of coding rooted trees (see [9] for a survey), but it turns out that if we are to satisfy Conditions 1, 2 and 3, we must use a code somewhat different in detail from any of those mentioned in that reference. However, the code we shall use is of a common type — namely it is a sequence of $2q$ 0's and 1's forming a parenthetical string, i.e., if we interpret 0 to mean "(" and 1 to mean ")" the code becomes a properly nested set of q pairs of parentheses.

If we remove the root from a rooted tree T we obtain a collection of rooted trees — the "principal subtrees" of T. The code of T is then defined recursively as follows:

(a) The code of the trivial tree consisting of one node and no edges is the empty string.

(b) To obtain the code of a general rooted tree T, write down the codes of its principal subtrees, each of these codes being preceded by a "0" and followed by a "1". The concatenation of these several strings is the code of T.

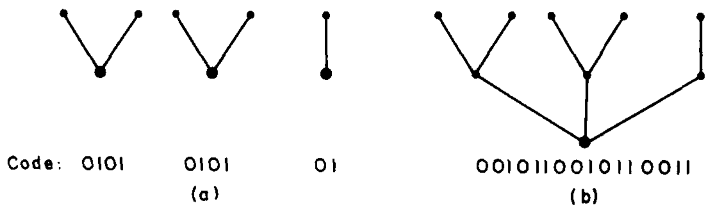


Fig. 1.

This is illustrated in Fig. 1. Such a code is strictly speaking the code of a *planted tree*, since it ascribes a particular order in which the principal suotrees occur. Hence for this problem the set S_q will be the set of planted trees having q edges. If we ignore the order of the principal subtrees (and hence of *their* principal subtrees, and so on) these planted trees fall into equivalence classes which are the rooted trees that we wish to catalogue. Hence we must define a canonical form for each rooted tree. A rooted tree will be said to be canonical if

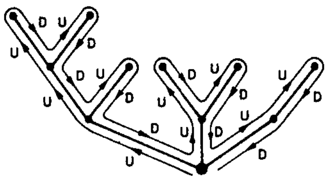
(a) each principal subtree is canonical and

(b) the order of the principal subtrees is according to the following two rules:

(i) Subtrees with more edges precede subtrees with fewer edges.

(ii) For subtrees having the same number of edges, those with larger codes (when the codes are regarded as binary integers) precede those with smaller codes. Clearly to every canonical rooted tree there is a corresponding canonical code, and also a canonical drawing of the tree in the plane, namely that in which the principal subtrees are drawn above the root and in order from left to right.

An easy way to draw a rooted tree directly from its code is to interpret "0" to mean "up" and "1" to mean "down". Starting at the root and moving up or down according to the code, we obtain a drawing of the tree in a manner which will be clear from Fig. 2.



Code: 0001011011001011 0011

Fig. 2.

Let us now consider the augmenting operation by which trees on $q + 1$ edges are produced from a tree on q edges. An operation that would certainly work would be to add to the tree on q edges, in all possible ways, an extra edge, one end of which is at an existing node of the tree and the other end of which is a new node (which is therefore an "end node" or "leaf"). It turns out however that we can manage with much less than that. If we look at a drawing of a rooted tree (such as Fig. 3) it is clear intuitively what is meant by the "right-hand side" of the tree — in Fig. 3 it

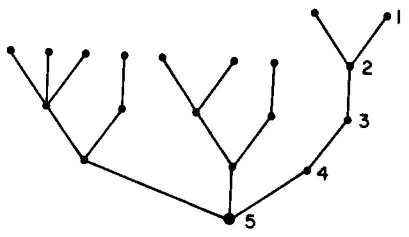


Fig. 3.

consists of the nodes numbered 1, 2, 3, 4, 5, the last of these being the root. Our augmenting operation will consist of adding an edge to just these right-hand nodes, in the order indicated, finishing with the root, in such a way that the new node thus introduced becomes the rightmost node of all. To make this more precise let us interpret it in terms of the tree code, which always ends with a string of 1's; the augmenting operation produces trees with an extra edge by inserting "01" in front of each of these 1's in turn, proceeding from left to right. Last of all, it inserts "01" at the very end of the code (this giving the tree obtained by adding an extra edge in the root). Thus from

0010110010110011

we obtain

0010110010110000111,

0010110010110010111,

0010110010110011101.

It is clear that the codes produced differ only in the location of the extra zero, which moves to successively lower order positions. Hence the codes are produced in increasing order of the binary integers that they represent. Consequently, if we decree that our lists of rooted trees shall be ordered in increasing order of codes, Condition 3 will automatically be satisfied.

We must now show that Conditions 1 and 2 hold. The proofs will be by mathematical induction, and we easily verify that the conditions hold for small values of q. Suppose that Condition 1 is satisfied for trees with no more than q edges, and consider any canonical tree with $q+1$ edges. Denote by P its "right-most" end node. In the "up-down" description of the tree this is the node from which the final descent to the root begins. It is node 1 in Fig. 3 and it corresponds to the final zero to occur in the code of the tree. Remove this node together with its incident edge, so as to get a rooted tree T' on q edges. We shall show that T' is canonical.

If P is adjacent to the root of T, then the trivial tree consisting of P alone is a principal subtree of T. As such it must come last in the order of the subtrees, and hence its removal will leave the other subtrees in the correct order. Hence T' has all its principal subtrees canonical (they were canonical in T) and in the correct order. Hence T' is canonical. In the contrary case the node P belongs to the last principal subtree of T and by the induction argument its removal from that subtree yields a canonical subtree. Furthermore since this subtree was the last one in T, and since it now has one fewer edge than before, it is *a fortiori* the last when the edge has been removed. Hence T' has its principal subtrees canonical and in the correct order, and hence is itself canonical. This completes the proof that Condition 1 is satisfied.

We now prove that Condition 2 holds. Suppose that it holds for trees with up to q edges and let T_1 and T_2 be canonical trees with $q+1$ edges. For $i = 1, 2$ let $T'_i = f(T_i)$ be the tree on q edges obtained by removing the right-most end-node of T_i, as described above. We need to show that if $T_1 < T_2$, i.e., code$(T_1) <$ code(T_2), then code$(T'_1) \leq$ code(T'_2). Compare the principal subtrees of T_1 and T_2, and consider the first subtree (reading from left to right) where they differ. If this subtree is not the last, then the modifications made to the last subtree by the removal of the edge will not have any effect on the relative order of the trees, since the difference of their codes in a higher order position is not altered. On the other hand, if it is only in the last subtree that they differ, then the code of the last subtree of T_1 is smaller than the code of the last subtree of T_2, and, by the induction argument, the same order persists (in the weak sense) after the removal of the right-most edges. This completes the proof that Condition 2 holds.

6. Unrooted trees

It is possible that there is an orderly algorithm which produces only unrooted trees, but I do not know of one. But then I have not looked for one, the reason for that being that since rooted trees are of interest in themselves one would like to have lists of them anyway. That being the case, one might as well produce the unrooted trees concurrently with the production of their rooted brethren, and this is easily done. To get a canonical version of an unrooted tree we simply root it at its centre (if it has one) or at one of its bicentral nodes (see [9] for details). Having produced a rooted tree we can easily check whether it is the canonical version of an unrooted tree and, if it is, write it on a separate magnetic tape, in addition to writing it on the tape in which all rooted trees are being stored.

7. Tournaments

A tournament is a complete oriented graph, i.e., a graph in which each pair of nodes forms an arc oriented in one direction or the other (but not both). Hence a tournament on p nodes contains $p(p-1)/2$ oriented edges. The adjacency matrix of a tournament has the following properties

(a) $a_{ii} = 0$, for every i,

(b) if $i \neq j$, then either $a_{ij} = 0$ and $a_{ji} = 1$, or $a_{ij} = 1$ and $a_{ji} = 0$.

It follows then that a tournament is specified entirely by the upper triangular portion of its adjacency matrix.

Let us define the code of a tournament to be the sequence of 0's and 1's obtained by reading off the upper triangular portion of the adjacency matrix column by column. Thus the code of

$$\begin{bmatrix} 01101 \\ 00010 \\ 01001 \\ 10101 \\ 01000 \end{bmatrix}$$

is 1100101011.

Two tournaments are isomorphic if one can be obtained from the other by permuting the nodes. A tournament will be said to be canonical if it has the largest code of all the tournaments in its isomorphism class.

To obtain an orderly algorithm for listing tournaments we shall make the following choices. Each list will consist of canonical tournaments listed in decreasing order of their codes. The augmenting operation, converting a tournament T on p nodes into a collection of tournaments on $p+1$ nodes will be the addition to the adjacency matrix of T of an extra column. The code of a new tournament will therefore consist of the code of T followed by p new digits. For these new digits we shall take all the possible sequences of p 0's and 1's, taken in descending order of the corresponding binary integers, from $2^p - 1$ down to 0. With this definition of the augmenting operation Condition 3 is automatically satisfied.

Given a canonical tournament on $p+1$ nodes, with (canonical) adjacency matrix A, let A' be the adjacency matrix obtained by deleting the last row and column of A. Then A' is the adjacency matrix of a tournament T' on p nodes. Suppose that T' is not canonical. Then some permutation of the nodes of T' would give a larger code for T'. But the code of T' forms the leading portion of the code of T, and hence this same permutation, applied to the first p nodes of T would give a larger code for T, which is not possible. Hence T' is canonical. Thus every canonical tournament on $p+1$ nodes can be obtained by the augmenting operation from the canonical tournament on p nodes, and Condition 1 is satisfied. Moreover it is clear that T' is the only canonical tournament on p nodes which can give rise to the canonical tournament T. Hence $f(T) = T'$, where f is the mapping defined in Section 2.

Condition 2 now follows almost immediately. For if $T_1, T_2 \in \mathscr{L}_{p+1}$ and code$(T_1) >$ code(T_2), then either the adjacency matrices of T_1 and T_2 differ only in the last column, in which case code$(T_1') =$ code(T_2') and this is covered by Condition 3; or else they differ elsewhere, in which case it is this difference that determines the order of T_1' and T_2', and we have code$(T_1') >$ code(T_2').

Thus an orderly algorithm exists for cataloguing tournaments. It appears to be less useful than the other algorithms described here since we lack, at present, an easy method of testing for canonicity. The brute force method is good for 5 or even 6 nodes, but something better is needed for the larger problems.

8. A note on a depth-first version of the algorithm

Consider a cataloguing problem for which the total number of configurations is finite — the problem of 5-node digraphs is one such. The orderly algorithm gives a sequence of lists $\mathscr{L}_0, \mathscr{L}_1, \ldots, \mathscr{L}_{10}$, and each element x of a list $\mathscr{L}_{q+1}$ is derived from an element $f(x)$ of $\mathscr{L}_q$. The graph whose nodes are the configurations, and whose edges are the pairs $(x, f(x))$ is a tree, having the single element of $\mathscr{L}_0$ as its root, and with each $\mathscr{L}_q$ forming one level of the tree. The operation of the orderly algorithm is then equivalent to a breadth-first search of this tree.

Instead of this we can programme a depth-first, or backtrack, search of the tree. At a general stage of this search the programme is looking at the last configuration in some list, and produces, by the augmenting operation, a new configuration which becomes the last configuration in the next list. Backtracking to the last element of the previous list occurs when the last list ($\mathscr{L}_{10}$ in our example) is reached, or when no further new configurations can be produced from the configuration currently being looked at.

Such a programme would generate *de novo* all the canonical configurations of the set S, without the need for any external storage devices. Internally the programme would need to store only the last element of each $\mathscr{L}_q$. For configurations that can easily be tested for canonicity, this might be an attractive alternative to using a previously prepared list of configurations.

References

[1] H.H. Baker, A.K. Dewdney, A.L. Szilard, Generating the nine-point graphs, Research Report # 11, Department of Computer Science, University of Western Ontario.
[2] R.J. Frazer, Graduate course project, Department of Combinatorics and Optimization, University of Waterloo, unpublished (May 1973).
[3] F. Harary, E.M. Palmer, Graphical Enumeration (Academic Press, New York and London, 1973).
[4] B.R. Heap, The production of graphs by computer, in: R.C. Read, ed., Graph Theory and Computing (Academic Press, New York and London, 1972) 47–62.
[5] I. Kagno, Linear graphs of degree less than 7 and their graphs, Am. J. Math. 68 (1946) 505–529.
[6] P. McWha, Graduate course project, Department of Combinatorics and Optimization, University of Waterloo, unpublished (May 1973).
[7] P.A. Morris, A catalogue of trees on n nodes, $n < 14$, Mathematical observations, research and other notes. Paper No. 1 StA. (Mimeographed). Publication of the Department of Mathematics, University of the West Indies.
[8] R.C. Read, The production of a catalogue of digraphs on 5 nodes, Report UWI/CC1, Computing Centre, University of the West Indies.
[9] R.C. Read, The coding of various kinds of unlabelled trees, in: R.C. Read, ed., Graph Theory and Computing (Academic Press, New York and London, 1972) 153–182.
[10] R.C. Read, D.G. Corneil, The isomorphism disease, J. Graph Theory (to appear).