ELSEVIER

Theoretical Computer Science 178 (1997) 103–118

Theoretical Computer Science

Permutations generated by token passing in graphs

M.D. Atkinson*, M.J. Livesey, D. Tulley

School of Mathematical and Computational Sciences, North Haugh, St. Andrews KY16 9SS, UK

Received May 1995; revised February 1996
Communicated by M.S. Paterson

Abstract

A transportation graph is a directed graph with a designated input node and a designated output node. Initially, the input node contains an ordered set of tokens $1, 2, 3, \ldots$. The tokens are removed from the input node in this order and transferred through the graph to the output node in a series of moves; each move transfers a token from a node to an adjacent node. Two or more tokens cannot reside on an internal node simultaneously. When the tokens arrive at the output node they will appear in a permutation of their original order. The main result is a description of the possible arrival permutations in terms of regular sets. This description allows the number of arrival permutations of each length to be computed. The theory is then applied to packet-switching networks and has implications for the resequencing problem. It is also applied to some complex data structures and extends previously known results to the case that the data structures are of bounded capacity. A by-product of this investigation is a new proof that permutations which avoid the pattern 321 are in one to one correspondence with those that avoid 312.

1. Introduction

Many problems in Computer Science involving the transfer of data items between various locations can be modelled using directed graphs. The nodes of the graph represent the locations where data can be stored and the edges represent the possibility that data might be transferred from one location to another. Examples include:

1. *Distributed computing networks*: The nodes are computers, the edges are high-speed data communication channels, and the data items are files or fragments of files.
2. *Parallel computers*: The nodes are processing elements, the edges are the data highways represented by the architecture (hypercube, mesh, butterfly, etc.), and the data items are the bits, bytes and words which flow along the highways.

* Corresponding author.

PII S0304-3975(96)00057-6

3. *Models of transportation systems*: The nodes are cities, the edges are roads, and the data items are commodities being shipped from one city to another.

4. *Data structures*: We shall say more about this example later but for now it will be enough to consider Binary Search Trees. The nodes are the memory cells containing a key and associated values. The edges connect each node to its left and right children, and the data items are the (key, value) pairs which move around the tree as it is updated.

This paper addresses some features common to all these situations. In the next section we pose an abstract problem concerned with the movement of "tokens" around the nodes of a graph and the way in which they may be permuted. Then, in Section 3, we solve the problem in terms of regular sets (see [8]). The solution is then applied to some of the above areas. Section 4 discusses the resequencing problem in packet-switching networks. Finally, Section 5 applies the techniques to abstract data types extending the work of [11, 17, 13], gives a new solution to a combinatorial problem solved in [14, 16], and demonstrates how our general approach can be used efficiently.

2. Problem formulation

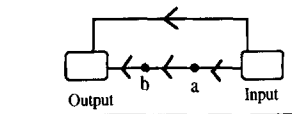
Let N be a finite directed graph with a designated *input* node and a designated *output* node. The input node has no incoming edges and the output node has no outgoing edges. The remaining nodes are called *internal* nodes. Such a graph will be called a *transportation graph*.

Each internal node is allowed to contain 0 or 1 *tokens*. In applications of this problem the tokens are items of data but in our abstract formulation we shall assume only that tokens are denoted by positive integers and therefore are distinguishable from each other.

The input node generates an ordered sequence $1, 2, 3, \ldots$ of tokens. How this is done is unimportant at present; the tokens could be generated by local computation at the input node or may have been received from an external source. The tokens are then moved from one node to another along the edges of the network until they arrive in some order at the output node. Tokens cannot be stored on the edges of the network. A token t on a node x can only move to a node y if the following conditions hold:

1. There is an edge from x to y.
2. x is the input node and tokens $1, 2, \ldots, t-1$ have already been moved from the input node or x is an internal node.
3. y is an internal node and y does not currently contain any token, or y is the output node.

Condition 1 implies that tokens move from the input node to the output node along a path in the graph, condition 2 implies that tokens leave the input node in the natural order $1, 2, 3, \ldots$, and condition 3 ensures that every internal node contains at most one token at any time.



Time Step	output	b	a	input
0				123
1			1	23
2		1		23
3		1	2	3
4	3	1	2	
5	31		2	
6	31	2		
7	312			

Fig. 1.

Example. Fig. 1 shows a simple graph with 2 internal nodes and a possible movement of 3 tokens through it. The tokens arrive at the output node in the order 3 1 2. It is not difficult to see that different choices of moves could result in any arrival order other than 3 2 1. From the theory that we develop in this paper results for n tokens can be derived. For example, if x_n is the number of possible arrival orders of n tokens then $x_0 = 1$, $x_1 = 1$, and $x_n = 3x_{n-1} - x_{n-2}$ for $n \geqslant 2$.

We shall consider general transportation graphs and the movement of tokens through them. As described above the tokens leave the input node in the order $1, 2, 3, \ldots$. If the input stream is finite and the tokens all arrive at the output node, the output node will then contain some permutation σ of the input stream; we say that σ is *generated* by the transportation graph. The set of all permutations generated by a transportation graph N will be denoted by $\mathrm{Out}(N)$. We let $\mathrm{Out}_n(N)$ denote the set of permutations of $1, 2, \ldots, n$ in $\mathrm{Out}(N)$.

The main aim of this paper is to give a description of $\mathrm{Out}(N)$. This description allows the membership problem (given a permutation π determine whether $\pi \in \mathrm{Out}(N)$) to be answered in time proportional to the length of π. It also permits $|\mathrm{Out}_n(N)|$ to be calculated.

3. General transportation graphs

Throughout this section N will denote a transportation graph. We shall develop a method for characterising $\mathrm{Out}(N)$ and counting the number of permutations in $\mathrm{Out}_n(N)$ for each n.

Our method depends on an encoding of the permutations in Out(N). For every permutation $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ we define its encoded form $\chi(\sigma) = c_1c_2 \ldots c_n$ by defining c_i to be the rank of σ_i in $\sigma_i\sigma_{i+1} \ldots \sigma_n$. For example the encoded form of 2761453 is 2651221. It is easy to see that every code sequence $c_1c_2 \ldots c_n$ satisfies $c_{n-i} \leqslant i+1$ and that every sequence of positive integers of length n which fulfils this condition is the encoding of some permutation. We let $C(N)$ and $C_n(N)$ denote the set of strings which are the coded forms of the elements of Out(N) and Out$_n(N)$ respectively. Our main theorem is

Theorem 1. *$C(N)$ is a regular set.*

The proof of Theorem 1 is constructive. We shall show how a regular grammar can be defined which generates $C(N)$. We shall then use some standard constructions for regular languages to obtain a method for computing $|C_n(N)| = |\mathrm{Out}_n(N)|$.

We label the internal nodes of N in an arbitrary order as $1, 2, \ldots, k$. As a sequence $1, 2, \ldots, n$ of tokens is transferred, step by step, through the transportation graph to the output node the graph passes through a series of *configurations*. A configuration is a disposition of tokens among the internal nodes Such a disposition may be described by an *occupancy function* $f: \{1, 2, \ldots, k\} \to \{0, 1, 2, \ldots\}$; $f(i)$ is the token residing on node i except in the case $f(i) = 0$ which represents that node i is empty. The configuration where $f(i) = 0$ for all i is denoted by ε; it is both the initial and the final configuration of the transportation graph.

Two configurations with occupancy functions f, g are defined to be equivalent if

(a) $f(i) = 0$ if and only if $g(i) = 0$,

(b) $f(i) < f(j)$ if and only if $g(i) < g(j)$.

A *state* of the transportation graph is defined to be an equivalence class of configurations. The state corresponding to the equivalence class $\{\varepsilon\}$ is denoted by q_0.

Lemma A. *The number of states is finite.*

Proof. Every configuration is equivalent to one in which the tokens on the internal nodes are $1, 2, \ldots, h$ with $h \leqslant k$. The occupancy function then takes values in $\{0, 1, 2, \ldots, k\}$ and there are at most k^{k+1} such functions. □

Let C denote the set of configurations, and E the set of edges of N. We shall define a partial function $v: C \times E \to C \times \{1, 2, \ldots, k, \lambda\}$. Let $c \in C$ and $e = (i, j) \in E$. Informally, $v(c, e)$ specifies the configuration that arises from configuration c if a token is transferred from node i to node j, and the new symbol (if any) that is placed at the output node. More precisely, $v(c, e)$ is defined whenever, in configuration c, node i contains a token t (if node i is the input node then t is the next input token to be transferred from the input node) and either node j is empty or node j is the output node. Under these conditions $v(c, e)$ is defined as (d, x), where

(i) d is the configuration that would arise from configuration c if t was transferred from i to j, and

(ii) x is the empty symbol λ unless j is the output node in which case x is the rank of t among all the tokens residing on internal nodes in configuration c.

Lemma B. *The permutation $\sigma \in \mathrm{Out}(N)$ if and only if there is a sequence of edges $e_1, e_2, \ldots, e_p$ and configurations $\varepsilon = a_0, a_1, \ldots, a_p = \varepsilon$ such that $v(a_{i-1}, e_i) = (a_i, x_i)$, $i = 1, 2, \ldots, p$, and $x_1 x_2 \ldots x_p = \chi(\sigma)$.*

Proof. Suppose $\sigma_1 \sigma_2 \ldots \sigma_n = \sigma \in \mathrm{Out}(N)$. Then there is a sequence of transfers along edges $e_1, e_2, \ldots, e_p$ which causes tokens $1, 2, \ldots, n$ on the input node to move through the transportation graph until they have arrived on the output node in the order $\sigma_1, \sigma_2, \ldots, \sigma_n$. Let $\varepsilon = a_0, a_1, \ldots, a_p = \varepsilon$ be the sequence of configurations induced in the transportation graph by these transfers and let t_i be the token taking part in the ith transfer. Then, certainly, $v(a_{i-1}, e_i) = (a_i, x_i)$ where

(i) if t_i is not being transferred to the output, $x_i = \lambda$ and

(ii) if t_i is being transferred to the output, x_i is the rank of t_i among all the tokens on internal nodes in configuration a_{i-1}; and since the tokens remaining in the input node are greater than all these tokens x_i is actually the rank of t_i among all tokens not yet transferred to the output.

Thus $x_1 x_2 \ldots x_p$ has the property that the jth (non-empty) symbol is the rank of σ_j among $\sigma_j, \sigma_{j+1}, \ldots, \sigma_n$; in other words, $x_1 x_2 \ldots x_p = \chi(\sigma)$.

Conversely suppose, for some permutation $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$, there exist edges $e_1, e_2, \ldots, e_p$ and configurations $\varepsilon = a_0, a_1, \ldots, a_p = \varepsilon$ such that $v(a_{i-1}, e_i) = (a_i, x_i)$, $i = 1, 2, \ldots, p$, and $x_1 x_2 \ldots x_p = \chi(\sigma)$. Then, by definition of v, it is possible to move tokens $1, 2, \ldots, n$ from the input node in p transfers using edges $e_1, e_2, \ldots, e_p$ in turn and passing through configurations $a_0, a_1, \ldots, a_p$. At a step which transfers a token t to the output (corresponding to the equation $v(a_{i-1}, e_i) = (a_i, x_i)$ say) the token has rank x_i among all tokens not yet placed in the output. Hence the permutation τ defined by the sequence of transfers satisfies $\chi(\tau) = x_1 x_2 \ldots x_p = \chi(\sigma)$ and therefore $\sigma = \tau \in \mathrm{Out}(N)$ as required. □

Lemma C. *Let a, b be equivalent configurations and let e be an edge of N. Suppose that $v(a, e)$ is defined and equal to (c, x). Then $v(b, e)$ is defined and equal to (d, y) where d is equivalent to c and $y = x$.*

Proof. Let e be the edge (r, s). We give the proof in the case that r is not the input node; the proof when r is the input node is almost the same. Let f be the occupancy function of a.

Suppose that s is not the output node. Then $x = \lambda$, $f(r) = t \neq 0$, $f(s) = 0$ and the occupancy function f' of c differs from f only in that $f'(r) = 0$, $f'(s) = t$. From the equivalence of a and b the occupancy function g of b satisfies $g(r) = u \neq 0$, $g(s) = 0$. It follows that $v(b, e)$ is defined and has occupancy function g' differing from g only in that $g'(r) = 0$, $g'(s) = u$. Therefore, for all i, j, we have $f'(i) = 0$ if and only if $g'(i) = 0$ and $f'(i) < f'(j)$ if and only if $g'(i) < g'(j)$. Thus c and d are equivalent. Also $y = \lambda$.

If s is the output node the equivalence of c and d follows in the same way. From the equivalence of a and b the token that is being transferred from node i to the output has the same rank among the tokens of the internal nodes of a as it does among those of b; that is, $x = y$. □

Let Q be the set of states. We define a partial function $\mu: Q \times E \to Q \times \{1, 2, \ldots, k, \lambda\}$. Let $q \in Q$ and $e \in E$. Thus q is an equivalence class of configurations. Let a be any configuration in this equivalence class. If $v(a, e)$ is defined and equal to (d, x) then we define $\mu(q, e)$ to be (r, x) where r is the equivalence class of configurations that contains d. Lemma C guarantees that μ is well-defined (independently of the choice of a in the equivalence class q). The next lemma follows immediately from these definitions.

Lemma D. *Let $e_1, e_2, \ldots, e_p$ be any sequence of edges. Then there exists a sequence of configurations $\varepsilon = a_0, a_1, \ldots, a_p = \varepsilon$ such that $v(a_{i-1}, e_i) = (a_i, x_i)$, $i = 1, 2, \ldots, p$ if and only if there exists a sequence of states $q_0, q_1, \ldots, q_p = q_0$ such that $\mu(q_{i-1}, e_i) = (q_i, x_i)$, $i = 1, 2, \ldots, p$.*

Lemmas B and D have the following consequence:

Lemma E. *The permutation $\sigma \in \mathrm{Out}(N)$ if and only if there is a sequence of edges $e_1, e_2, \ldots, e_p$ and states $q_0, q_1, \ldots, q_p = q_0$ such that $\mu(q_{i-1}, e_i) = (q_i, x_i)$, $i = 1, 2, \ldots, p$, and $x_1 x_2 \ldots x_p = \chi(\sigma)$.*

Proof of Theorem 1. We define a context-free grammar (N, T, P, S) as follows:

(i) The set N of non-terminals is (indexed by) the set of states.

(ii) The goal symbol S corresponds to the state q_0.

(iii) The set T of terminal symbols is $\{1, 2, \ldots, k\}$.

(iv) The set of productions P contains productions of the form $q \to xr$ for every defined value $\mu(q, e) = (r, x)$ of the function μ together with a production $q_0 \to \lambda$.

Derivations in this grammar have the form $q_0 \to x_1 q_1 \to x_1 x_2 q_2 \to \cdots \to x_1 x_2 \ldots x_p q_p \to x_1 x_2 \ldots x_p$ and such a derivation exists if and only if there is a sequence of edges $e_1, e_2, \ldots, e_p$ for which $\mu(q_{i-1}, e_i) = (q_i, x_i)$, $i = 1, 2, \ldots, p$. Therefore, by Lemma E, the strings $x_1 x_2 \ldots x_p$ generated by this grammar are precisely those strings of the form $\chi(\sigma)$ for $\sigma \in \mathrm{Out}(N)$, that is, they are the strings of $C(N)$.

The grammar (N, T, P, S) is not itself regular since among its productions $q \to xr$ there may be some of the form $q \to r$ (that is, $x = \lambda$). However, standard results in language theory (see [8, pp. 26, 218]) show that there is, nevertheless, a regular grammar which defines the same language; thus $C(N)$, being generated by a regular grammar, is a regular set. □

We now turn to the question of computing the numbers $|\mathrm{Out}_n(N)|$ for all values of n. The method for calculating these numbers is a special case of a more general treatment valid for any context-free grammar [4]. However, we reproduce that part of

this general theory here for convenience and to establish the basis for the case studies of the next section. There is a well-known equivalence [8, p. 218] between regular grammars and finite state automata. In this equivalence transitions from a state p to a state q occasioned by a symbol x correspond either to productions $p \to xq$ (if q is not a final state) or productions $p \to x$ (if q is a final state). There is also a well-known construction [8, p. 22] that produces an equivalent deterministic finite state automaton from a non-deterministic one. The regular grammar that is associated with a deterministic finite state automaton has the property that the productions on each fixed non-terminal have distinct initial terminals beginning their right-hand sides. The consequence of this is that $C(N)$ can be generated by a regular grammar with non-terminal symbols $S = S_1, S_2, \ldots, S_h$ and productions of the form

$$S_i \to t_1 U_1 \mid t_2 U_2 \mid \ldots \mid t_w U_w,$$

where $t_1, t_2, \ldots, t_w$ are distinct together with a production $S \to \lambda$.

Let $s_n^{(i)}$ be the number of terminal strings of length n that can be derived from S_i. Note that $s_0^{(i)} = 1$ if $i = 1$ (since $S \to \lambda$) and $s_0^{(i)} = 0$ otherwise. Let z_{ij} be the number of productions of the form $S_i \to tS_j$ (i.e. there are z_{ij} terminals t that can appear in this form of production). Since the derivations that begin with the production $S_i \to tS_j$ produce a set of terminal strings that is disjoint from the set of terminal strings that are derived beginning from any other production on S_i (because none of those terminal strings will begin with t) we have the recurrence

$$s_n^{(i)} = \sum_{j=1}^{h} z_{ij} s_{n-1}^{(j)}.$$

This gives us a set of h linear recurrences which, together with the h boundary conditions, completely define each $s_n^{(i)}$. In particular $s_n^{(1)} = |\mathrm{Out}_n(N)|$ can be found in this way.

There are standard techniques for solving sets of recurrence equations. These methods tell us that the general solution depends on the eigenvalues of the matrix $[z_{ij}]$. The most straightforward case is when the eigenvalues $\mu_1, \mu_2, \ldots, \mu_h$ are distinct and in that case $s_n^{(i)}$ has the form $\sum_{j=1}^{h} k_{ij} \mu_j^n$ where the constants k_{ij} are determined by the initial values. When there are repeated eigenvalues there is a similar expression for $s_n^{(i)}$ in which the k_{ij} are replaced by polynomials in n of degrees depending on the multiplicities of the eigenvalues.

The outcome of this is that $|\mathrm{Out}_n(N)|$ is given by a formula of the form $p_1(n)\mu_1^n + p_2(n)\mu_2^n + p_3(n)\mu_3^n + \cdots$ where $\mu_1, \mu_2, \ldots$ are certain distinct constants and $p_1(n), p_2(n), \ldots$ are certain polynomials, all of which can be calculated. We shall choose the notation so that $|\mu_1| \geqslant |\mu_2| \geqslant \cdots$ so that $|\mathrm{Out}_n(N)|$ is asymptotic to $p_1(n)|\mu_1|^n$ as $n \to \infty$. We can therefore conclude the following

Theorem 2. *There exists a constant* $|\mu|$ *depending only on* N *such that*

$$(1/n)\log(|\mathrm{Out}_n(N)|) \to \log|\mu| \text{ as } n \to \infty.$$

Corollary. $|\mathrm{Out}_n(N)|/|\mathrm{Out}_{n-1}(N)| \to |\mu_1|$ *as* $n \to \infty$.

There is an information-theoretic interpretation of this result. We might consider $|\mathrm{Out}_n(N)|$ as a measure of how much choice exists in moving n tokens from the input node to the output node. The larger $|\mathrm{Out}_n(N)|$ is, the more uncertainty there is in the result. In information theory uncertainty is measured in bits and we can define the entropy of the arrival permutation as $\log_2|\mathrm{Out}_n(N)|$ which is asymptotic to $n\log_2|\mu_1|$. To put this in another way the entropy per output token is $\log_2|\mu_1|$. By analogy with the results of [2] we therefore define the entropy of the entire transportation graph to be $\log_2|\mu_1|$. The entropy is a rough measure of how much permutational capability is possessed by a transportation graph.

It should be noted that the calculation of $|\mathrm{Out}_n(N)|$ has linear time complexity in n. However this understates the difficulty of carrying out the calculation since there is, potentially, a doubly exponential dependence on the size of N. In Section 5 we shall see that this combinatorial explosion can sometimes be contained.

4 Packet-switching networks

Packet-switching networks are computer communication systems used for transmitting streams of data in the form of standardised packets. Typically a message originates at one of the nodes in the network and is partitioned into packets. The packets are then sent independently through the network and eventually all arrive at their destination node. Because packets travel by different routes and may experience delay because of congestion at intermediate nodes, the packets will, in general, not arrive in their proper order. The destination node must therefore rebuild the correct order of packets and so it may need to wait for any packets which have been delayed during transmission through the network.

The analysis of the packet delays has been conducted by many authors [10, 3, 18, 15, 9, 7]. This work has concentrated on stochastic properties of the delay times. For example the authors of [10] made the assumption that packet delays came from an exponential distribution and were stored after their delay until they could be properly ordered; they analysed the distribution of the number of stored packets and calculated the average delay due to resequencing. This analysis was extended in [7] to the case that the packet delays came from a general distribution and carried forward in [3]. For networks where the delay is caused by a multi-server exponential queue (with different service rates) the delay distribution was found in [18].

None of this work addresses the question of which permutations of the packets might arise through transmission delays. The results of the previous section allow this question to be attacked. To do this we must show how a transportation graph may be used to model a packet-switching network and for that we need to make some plausible assumptions about how packets are stored and forwarded at a network node. We shall assume that each node collects packets in a finite packet buffer that is subject to a queuing discipline. A node can only receive a packet if its packet buffer is not filled to capacity. If a packet is received it must wait in the packet buffer until it comes to the head of the queue, at which point it may be forwarded to another node with spare capacity in its packet buffer. We shall consider 3 possible queuing disciplines:

(i) FIFO: the packets pass through the packet buffer in "first in, first out" fashion;

(ii) LIFO: the packet buffer behaves like a stack and always outputs the most recently received packet;

(iii) SIRO ("serial in, random out"): the packet buffer may output any of the packets it is currently storing.

If the queuing disciplines at the nodes of the network are of the above types we can construct a transportation graph N from the packet-switching network P as follows. Every node of P is replaced by a collection of nodes and edges in a new graph N in such a way as to simulate the queuing discipline as shown in Fig. 2 (in the case of packet buffers of capacity 5). The input node of N is defined to be the origin of the message represented by the packets and the output node is the destination of the packets. It is assumed that the packet routing never returns a packet to its origin and that once it reaches the destination it is not further forwarded. It is obvious that the set $\mathrm{Out}_n(N)$ is the set of possible packet arrival orders of packets $1, 2, \ldots, n$.

Networks where the nodes are subject to the queuing disciplines FIFO and LIFO, but with no bounds on the capacities of the nodes, have been considered by several authors [1, 6, 13, 17]; the focus in these papers was on the set of output permutations. Thus our packet-switching application may be considered to be an extension of this work to the case where the queues are of bounded capacity.

These notions also allow the main result of Section 3 to be slightly generalised to the case where the nodes of the transportation graph are allowed to contain a bounded number of tokens greater than 1. A node of capacity k behaves like a packet buffer with SIRO queuing and so could be replaced by a group of nodes as in the third diagram of Fig. 3.

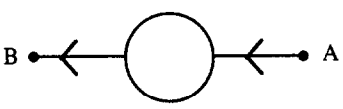


Fig. 2. A packet buffer of capacity 5 with a typical input channel and typical output channel.

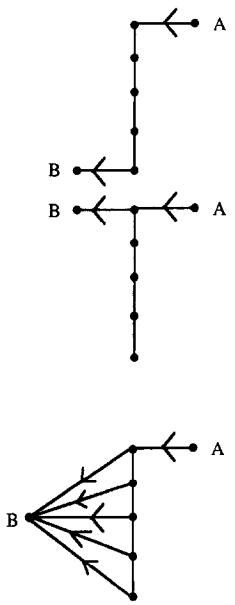


Fig. 3. The simulation of a packet buffer of size 5 according to whether the queue is FIFO, LIFO, or SIRO.

5. Data structures

In this section we apply the techniques in Section 2 to some common data structures. The permutations that can be generated by unbounded stacks and deques have been studied in the previous work of [17; 13; 11, 2.2.1]. We show how to extend these results to the bounded case. As a consequence we are able to give a new proof of a combinatorial result that appears in [12, 14, 16] and which has figured in some recent work on trees.

There is a further motive for this section. The construction of the regular grammar given in Section 3 is usually not an easy one to carry out since, as noted previously, it may involve two exponential explosions. The first arises since the number of states may be exponential in the transportation graph size and the second is due to replacing a non-deterministic finite automaton by a deterministic one. The work of this section shows that this combinatorial explosion can sometimes be contained.

The bounded stack data structure can be modelled by the transportation graph shown in Fig. 4.

In this transportation graph all edges between the k internal nodes are bidirectional. It is easy to see that the tokens on internal nodes in any configuration occur in increasing order of value (read from node 1 to node k). However, there are many ways in which a given set of tokens can be disposed on the internal nodes. We obtain a significant simplification by observing that two configurations whose internal nodes

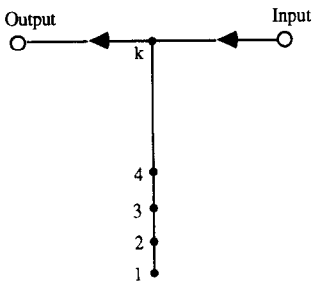


Fig. 4.

contain the same set of tokens need not be distinguished; for each such configuration is reachable from the other by transfers which affect neither the tokens on the input node or the output node. Thus we can carry out the computations using a set of $k+1$ states $q_0, q_1, \ldots, q_k$ where q_i is represented by a configuration with occupancy function f defined by $f(j) = j$, $j = 1,2, \ldots, i$, $f(j) = 0$, $j = i+1, \ldots, k$. The possible transitions out of this state correspond to placing a new input token into node $i+1$ or transferring token i to the output node. The associated grammar is therefore

$$q_0 \to \lambda \mid q_1$$

$$q_1 \to 1q_0 \mid q_2$$

$$q_2 \to 2q_1 \mid q_3$$

$$\ldots$$

$$q_{k-1} \to k-1\, q_{k-2} \mid q_k$$

$$q_k \to k\, q_{k-1}$$

Rewriting it to eliminate all productions whose right-hand side consists of a single non-terminal we obtain

$$q_0 \to \lambda \mid 1q_0 \mid 2q_1 \mid 3q_2 \mid \ldots \mid k\, q_{k-1}$$

$$q_1 \to \quad 1q_0 \mid 2q_1 \mid 3q_2 \mid \ldots \mid k\, q_{k-1}$$

$$q_2 \to \qquad 2q_1 \mid 3q_2 \mid \ldots \mid k\, q_{k-1}$$

$$\ldots$$

$$q_{k-1} \to \qquad k-1\, q_{k-2} \mid k\, q_{k-1}$$

If we let $Q_n^{(i)}$ denote the number of terminal strings of length n that can be derived from q_i we obtain the recurrence equations

$$
\begin{aligned}
Q_n^{(0)} &= Q_{n-1}^{(0)} + Q_{n-1}^{(1)} + Q_{n-1}^{(2)} + \cdots + Q_{n-1}^{(k-1)} \\
Q_n^{(1)} &= Q_{n-1}^{(0)} + Q_{n-1}^{(1)} + Q_{n-1}^{(2)} + \cdots + Q_{n-1}^{(k-1)} \\
Q_n^{(2)} &= \qquad Q_{n-1}^{(1)} + Q_{n-1}^{(2)} + \cdots + Q_{n-1}^{(k-1)} \\
Q_n^{(3)} &= \qquad\qquad Q_{n-1}^{(2)} + \cdots + Q_{n-1}^{(k-1)} \\
&\cdots \\
Q_n^{(k-1)} &= \qquad\qquad Q_{n-1}^{(k-2)} \qquad + Q_{n-1}^{(k-1)}
\end{aligned}
$$

with initial conditions $Q_0^{(0)} = 0$, $Q_i^{(0)} = 1$ for $i \geqslant 1$.

Since the transportation graph models a stack of bounded size k, the outputs generated by it are *stack computable* in the sense of [13] (where the stacks are unrestricted in size). The well-known correspondence between stack computations and ordered trees in which a sequence of pushes and pops corresponds to a traversal in which each edge is visited once in the down direction and once in the up direction (see Fig. 5) extends to correspondence between bounded stack computations and bounded height trees. Thus, the recurrences can be used to generate the numbers of ordered trees on n elements of height bounded by k.

Another treatment of the number of trees of bounded height is given in [5].

Our second application is to a queue which allows first-come-first-served queuing but sometimes permits newly arrived customers to be served immediately rather than standing in line. It may be modelled by the transportation graph in Fig. 6.

Ostensibly, this example is slightly more complicated than the bounded stack example. However we can make a similar simplification that makes it possible to handle the example with about the same degree of difficulty. As in the bounded stack example we need not distinguish two configurations which differ only in the disposition of the same set of tokens on nodes $1, 2, \ldots, k-1$. Therefore, we can take as representatives of the set of states q_i $(i = 0, 1, \ldots, k-1)$ the configurations whose occupancy functions f_i $(i = 0, 1, \ldots, k-1)$ are given by

$$f_i(j) = j, \quad j = 1, 2, \ldots, i; \qquad f_i(j) = 0, \quad j = i+1, \ldots, k-1$$

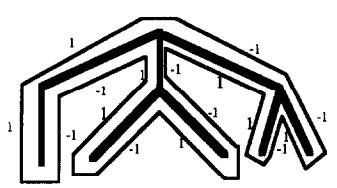


Fig. 5. Push–pop sequence: $1, 1, -1, -1, 1, 1, -1, 1, -1, -1, 1, 1, -1, 1, -1, -1$ (1 = push, -1 = pop).

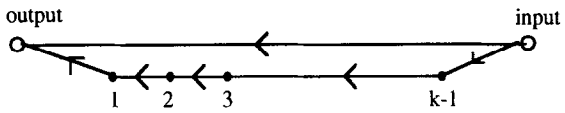


Fig. 6.

and whose transfers are defined by the grammar

$$q_0 \to \lambda \mid 1q_0 \mid q_1$$

$$q_1 \to 1q_0 \mid 2q_1 \mid q_2$$

$$q_2 \to 1q_1 \mid 2q_2 \mid q_3$$

$$\ldots$$

$$q_{k-2} \to 1q_{k-3} \mid k-1\ q_{k-2} \mid q_{k-1}$$

$$q_{k-1} \to 1q_{k-2} \mid k-1\ q_{k-1}$$

If we now eliminate all the productions whose right-hand side consists of a single non-terminal we obtain

$$q_0 \to \lambda \mid 1q_0 \mid 2q_1 \mid 3q_2 \mid \ldots \mid k\,q_{k-1}$$

$$q_1 \to \quad 1q_0 \mid 2q_1 \mid 3q_2 \mid \ldots \mid k\,q_{k-1}$$

$$q_2 \to \qquad 1q_1 \mid 3q_2 \mid \ldots \mid k\,q_{k-1}$$

$$\ldots$$

$$q_{k-2} \to 1q_{k-3} \mid k-1q_{k-2} \mid k\,q_{k-1}$$

$$q_{k-1} \to \qquad\qquad 1q_{k-2} \mid kq_{k-1}$$

This grammar is fairly similar to the grammar of the first example. It does not generate the same words but it is clear from the structural similarity that the recurrence equations for the numbers of words of length n generated by each non-terminal are identical to those of the bounded stack example. This observation leads to a proof of a result concerned with permutations that avoid certain patterns. If $\pi = \pi_1\pi_2 \ldots \pi_m$, $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ are permutations and there is a subsequence of σ whose elements, when read from left to right, are ordered in the same way as the elements of π then we say that σ *contains* the pattern π. The relation of containment is reflexive and transitive and defines a partial order on the set of all permutations. We say that σ *avoids* the pattern π if σ does not contain π. Permutations which avoid the pattern 312 are precisely those which can be generated by a stack. Permutations which avoid the pattern 321 are precisely those which may be represented as the merge of two increasing sequences. We give a proof of the following theorem (see also [14; 12, 5.1.4; 16]).

Theorem 3. *The number of permutations of length n which avoid* 312 *is the same as the number of permutations of length n which avoid* 321.

Proof. We let n be fixed and consider the transportation graphs in Figs. 4 and 6 above for any value of $k \geqslant n$. The first of these transportation graphs generates the set of permutations which avoid 312 and so we merely have to check that the second generates the set which avoids 321. However, this may be proved by noting that any permutation generated by the second transportation graph produces each symbol either by moving it along the top or bottom paths. The permutation is therefore a merging of the two subsequences which arrive in this way and these two subsequences are increasing. Conversely, any sequence which is the union of two increasing subsequences may be generated by channelling the elements of one subsequence along the bottom path and the elements of the other along the top path. □

Other data types which can be analysed in this way include bounded deques with or without restricted input using the two transportation graphs shown in Fig. 7.

It might be hoped from the examples in Figs. 4 and 6 that, for every transportation graph N, there was a finite set S of permutations such that Out(N) was precisely the set of permutations that avoided each of the permutations of S. In other words, one might hope that the complement of Out(N) had but a finite number of minimal elements under the containment partial order. Unfortunately this is false. A counter-example is the transportation graph M in Fig. 8.

Using arguments similar to those given in [17] it may be shown that the infinite set of permutations of the form $[4, 1, 6, 3, 8, 5, 10, 7, \ldots, 4n, 4n-3, 2, 4n-1]$ cannot be generated by the transportation graph M, yet if any symbol is deleted from one of these permutations the resulting permutation can be sorted; thus the set is an infinite set of minimal elements of the complement of Out(M).

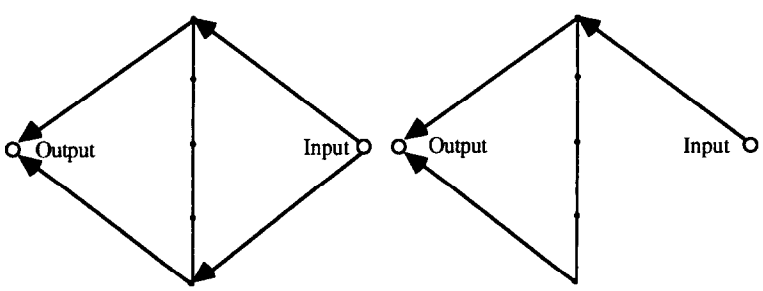


Fig. 7.

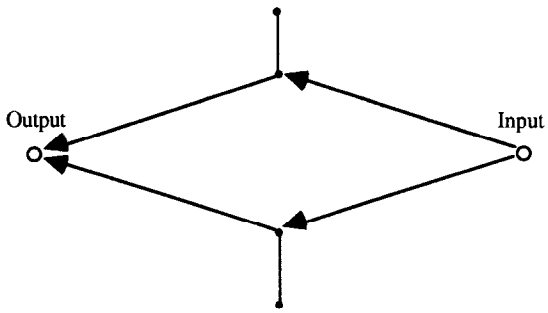


Fig. 8. The network M.

Acknowledgements

We thank Gerald Ostheimer, Michael Weatherill, and Peter Burgess for insightful comments in the early stages of this work.

References

[1] M.D. Atkinson, Sorting permutations with networks of stacks, Technical Report TR-210, School of Computer Science, Carleton University, Ottawa, Canada, August 1992.
[2] M.D. Atkinson and D. Tulley, The combinatorics of abstract data types, *Proc. IMA Conf. on Applications of Combinatorics*, to appear.
[3] F. Baccelli, E. Gelenbe and B. Plateau, An end-to-end approach to the resequencing problem, *J. ACM* **31** (1984) 474–485.
[4] N. Chomsky and M.P. Schutzenberger, The algebraic theory of context-free languages, in: *Computer Programming and Formal Systems* (North-Holland, Amsterdam, 1963) 118–161.
[5] N.G. de Bruijn, D.E. Knuth and S.O. Rice, The average height of planted plane trees, in: *Graph Theory and Computing* (Academic Press, New York, 1972) 15–22.
[6] S. Even and A. Itai, Queues, stacks and graphs, in: Z. Kohavi and A. Paz, eds., *Theory of Machines and Computations, Proc. Internat. Symp. on the Theory of Machines and Computations*, Technion – Israel Inst. of Technol., Haifa, Israel, August 1971 (Academic Press, New York, 1971) 71–86.
[7] G. Harris and B. Plateau, Queuing analysis of a re-ordering issue, *IEEE Trans. Software Engrg.* **8** (1982) 113–123.
[8] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
[9] I. Iliadis and L.Y.-C. Lien, Resequencing delay for a queuing system with two heterogeneous servers under a threshold-type scheduling, *IEEE Trans. Commun.* **36** (1988) 692–702.
[10] L. Kleinrock, F. Camoun and R. Muntz, Queuing analysis of the re-ordering issue in a distributed database concurrency control mechanism, in: *Proc. 2nd. Internat. Conf. Distrib. Database Syst.*, Versailles, France, April 1981.
[11] D.E. Knuth, *Fundamental Algorithms, The Art of Computer Programming*, Vol. 1 (Addison-Wesley, Reading, MA, 2nd ed., 1973).
[12] D.E. Knuth, *Sorting and Searching, The Art of Computer Programming*, Vol. 3 (Addison-Wesley, Reading, MA, 1973).

[13] V.R. Pratt, Computing permutations with double-ended queues, parallel stacks and parallel queues, *Proc. ACM Symp. Theory of Computing* **5** (1973) 268–277.
[14] D. Rotem, On a correspondence between binary trees and a certain type of permutation, *Inform. Process. Lett.* **4** (1975) 58–61.
[15] N. Shacham and D. Towsley, Resequencing delay and buffer occupancy in selective repeat ARQ with multiple receivers, *IEEE Trans. Commun.* **39** (1991) 928–937.
[16] R. Simion and F.W. Schmidt, Restricted permutations, *European J. Combin.* **6** (1985) 383–406.
[17] R.E. Tarjan, Sorting using networks of queues and stacks, *J. ACM* **19** (1972) 341–346.
[18] T.-S.P. Yum, T.-Y. Ngai, Resequencing of messages in communication networks, *IEEE Trans. Commun.* **34** (1986) 143–149.