ELSEVIER

Theoretical Computer Science 182 (1997) 145–157

Theoretical Computer Science

Bounded capacity priority queues

M.D. Atkinson*, D. Tulley

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

Received October 1994; revised May 1996
Communicated by M.S. Paterson

Abstract

A k-bounded priority queue transforms an input sequence σ into an output sequence τ which is a re-ordering of the sequence σ while never storing more than k elements during the transformation. We consider the set of all such related pairs (σ, τ) in both the case that σ is a binary sequence and the case that σ is a permutation of $1, 2, \ldots, n$. We derive properties of this relation and use it to describe systems of priority queues in series. In the binary case we give an efficient algorithm for computing the number of outputs achievable from a given input and the number of inputs which can give rise to a given output. Finally, we give a one-to-one correspondence between related binary input–output pairs and ordered forests of restricted height.

1. Introduction

Abstract data types are an important design tool in modern software systems. Although there is an infinity of possible data types only a small number of them occur repeatedly in algorithm design (stacks, arrays, queues, dictionaries, etc.), suggesting that some data types are "more fundamental" than others. Many of these fundamental data types are *container* data types: they are holders for collections of data items and support an *Insert* operation and a *Delete* operation (often restricted in some way).

In practice, a container data type is used as follows. It is initialised as empty. Then it is subjected to a sequence of Insert and Delete operations and, normally, on termination it will be empty. The insertions and deletions may be interleaved (in any way, except that a Delete operation is not permitted if the data type is empty). The result of this is that some input sequence of data items has been transformed into an output sequence (generated, one item at a time, by Delete operations). In effect, a container data type is just a mechanism for transforming an input sequence into an output sequence. Its functional behaviour is essentially characterised by the relationship between the input sequences and the output sequences. An understanding of this relationship allows us to

* Corresponding author. E-mail: mda@dcs.st-and.ac.uk.

PII S0304-3975(96)00149-1

judge the capabilities of a data type and to assess its potential applications. In general, if σ is an input sequence that gives rise to an output sequence τ then we shall say that the pair (σ, τ) is *allowable*. If a pair is allowable then there is at least one sequence of Insert and Delete operations which transform σ into τ. Any such sequence is called a *computation*. Numerical invariants of the allowability relation allow us to measure the transformational capability of the data type.

Natural questions to ask about the allowability relation of a container data type include: how many allowable pairs of each length are there, is there a useful characterisation of them, how can one test whether a pair is allowable, how could one calculate the number of allowable pairs with a given first component, does the relation have any interesting symmetries? In the case of a queue (where the Delete operation always removes the element which has been in the queue the longest) the allowability relation consists only of pairs (σ, σ) since the input sequence is merely copied to the output without re-ordering. Hence, for queues, all the questions above are trivial.

In the case of stacks (where the Delete operation always removes the element which was placed in the stack most recently) the allowability relation is more complicated but answers to all the questions above are known [8, 2.2.1]. For example, there are c_n (the nth Catalan number) allowable pairs whose first component is some fixed permutation of n elements; hence the number of allowable pairs of permutations of length n is $n!c_n$.

For priority queues (where the Delete operation always removes the smallest item) many combinatorial results are known [5, 2, 1, 7]. For example, the number of allowable pairs of permutations (σ, τ) of length n is $(n+1)^{n-1}$.

For dictionaries (which have an unrestricted Delete operation) the output can be any permutation of the input. This is therefore the opposite extreme to that of queues but the main questions are just as trivial.

In all these cases it is assumed that there is no restriction on how many items the container data type can hold. Of course, this is an unrealistic assumption since, in practice, there is only a finite amount of memory available and therefore there is an implicit restriction on the number of items that can be placed into the data type at any one time. This paper addresses the bounded case: we shall suppose that the container data type can hold no more than k data items simultaneously.

In the case of queues the allowability relation is unchanged. However, for stacks the allowability relation depends on k. In the case that the input sequence consists of distinct elements the possible outputs are well understood since it is not difficult to show that they are in one-to-one correspondence with ordered trees of height at most $k+1$. The results of [6] then allow very detailed information about the bounded stack allowability relation to be derived. When the input sequence to a stack is allowed to contain repetitions even the unbounded case remains unsolved.

The case of bounded dictionaries was solved as a special case of the more general problem considered in [4].

Our main focus is on priority queues of bounded size. If a priority queue is not permitted to have more than k elements in it we shall call it a *k-bounded* priority

queue. We shall say that a pair (σ,τ) that is allowable under the stricter discipline of a k-bounded priority queue is *k-allowable*.

To study the relation of k-allowability we shall use the following notation. We let $S_k(\tau) = \{\sigma \mid (\sigma,\tau) \text{ is } k\text{-allowable}\}$, the set of inputs that, using a k-bounded priority queue, can result in the output τ. We also define $T_k(\sigma) = \{\tau \mid (\sigma,\tau) \text{ is } k\text{-allowable}\}$ which is the set of possible outputs that can be generated by a k-bounded priority queue given the input sequence σ. Finally, we let $s_k(\tau) = |S_k(\tau)|$ and $t_k(\sigma) = |T_k(\sigma)|$.

Guided by work on the unbounded case we divide our results into two sections. In the next section we assume that the items to be placed in the priority queue are all distinct (the *permutation* case) and we shall often take these to be the integers $1,2,\ldots,n$ in some order. We consider especially the case that $k = 2$. This case is the first non-trivial one and is interesting in that the transitive closure of the 2-allowability relation contains every k-allowability relation for priority queues [2, Theorem 3.5]. By using generating function techniques we are able to derive a recurrence for the number of 2-allowable pairs of length n and a rate of growth result for this number. We also show how to compute $t_2(\sigma)$ and $s_2(\tau)$ efficiently. Finally, we consider systems of priority queues formed by serial composition and union.

In Section 3 we turn to the opposite extreme: where 0, 1 are the only possible priorities (the *binary* case). Here our results are more extensive. We give a necessary and sufficient condition for a pair of binary sequences to be k-allowable and use it in deriving algorithms to calculate $s_k(\tau)$ and $t_k(\sigma)$ by way of partially ordered sets. We then use the condition again to describe the effect of several priority queues combined in series. Finally, we give a correspondence between k-allowable pairs and forests of height at most $k+1$ and this enables us to enumerate the binary k-allowable pairs of each length. Although several of the results in Section 2 have counterparts in Section 3 the techniques are usually very different.

2. Priority queues with permutation inputs

2.1. The 2-allowability relation

Lemma 2.1. *If σ and τ are permutations of $1,2,\ldots,n$ then (σ,τ) is 2-allowable if and only if there exist decompositions of σ and τ into substrings such that*

$$\sigma = \alpha n \beta \gamma,$$
$$\tau = \delta \beta n \varepsilon,$$

where each of (α,δ) and (γ,ε) are also 2-allowable.

Proof. Suppose first that there is a sequence of Insert and Delete-Minimum operations that, using a 2-bounded priority queue, transforms σ into τ. At the point that the symbol n is inserted in the priority queue all the symbols of σ which precede n (a segment α say) will have been inserted and all except possibly one will have been output; the remaining symbol of α, if any, will then be output immediately since it is smaller than

n and no Insert operation is possible if the priority queue contains two items. Thus τ will have an initial segment δ with (α, δ) 2-allowable.

After δ has been generated the priority queue will contain only the symbol n. Since the priority queue is 2-bounded there must then be a number (possibly zero) of Insert, Delete-Minimum pairs of operations followed by a Delete-Minimum which copies a segment β of σ into the output τ and then outputs n. The priority queue will now be empty and the remaining segment γ of σ will be transformed into a final segment ε of τ so that (γ, ε) will be 2-allowable. This proves one implication. The other is similar and rather easier. □

An immediate consequence is

Corollary 2.1. *If (σ, τ) is a 2-allowable pair of permutations then the pair (τ^R, σ^R) is also 2-allowable (where α^R denotes the reverse of a sequence α).*

Corollary 2.2. *If σ, τ are permutations of $1, 2, \ldots, n$, where*

$$\sigma = \alpha n \phi,$$

$$\phi = f_1 f_2 \cdots f_r,$$

then

$$T_2(\alpha n \phi) = T_2(\alpha) T_2(n\phi), \tag{1}$$

$$T_2(n\phi) = \{n\} T_2(\phi) \cup \{f_1\} T_2(n f_2 \cdots f_r), \tag{2}$$

where juxtaposition XY of sets denotes the set of concatenations $xy, x \in X, y \in Y$.

Proof. (1) Using the juxtaposition notation and Lemma 2.1 we have

$$T_2(\alpha n \phi) = T_2(\alpha)\{\beta n T_2(\gamma) \mid \phi = \beta\gamma\} = T_2(\alpha) T_2(n\phi).$$

(2) $T_2(n\phi)$ consists of those outputs arising from inserting and immediately deleting n from the priority queue together with those that are obtained by inserting n, inserting f_1, and then deleting f_1. The first of these sets is $\{n\} T_2(\phi)$ and the second is $\{f_1\} T_2(n f_2 \cdots f_r)$. □

Taking set cardinalities we now obtain

Corollary 2.3.

$$t_2(\alpha n \phi) = t_2(\alpha) t_2(n\phi),$$

$$t_2(n\phi) = t_2(\phi) + t_2(n f_2 \cdots f_r) = t_2(n) + \sum_{i=0}^{r-1} t_2(f_{i+1} \cdots f_r).$$

This corollary gives a recursive method for computing $t_2(\sigma)$. If applied directly the execution time of the resulting method would be exponential in n. However by using

the same dynamic programming technique used in [2] for the unbounded case the calculation can be carried out in $O(n^4)$ steps and since $s_2(\tau) = t_2(\tau^R)$ by Corollary 2.1 we obtain

Corollary 2.4. *If σ, τ are permutations of length n then both $t_2(\sigma)$ and $s_2(\tau)$ can be computed in time $O(n^4)$.*

The total number of 2-allowable pairs of permutations of length n is, of course,

$$x_n = \sum t_2(\sigma), \tag{3}$$

where the summation is over all $n!$ permutations of $1,2,\ldots,n$. Although it seems to be difficult to find a closed form for x_n we can at least determine the exponential generating function $U(t) = \sum x_n t^n/(n!)$ of the sequence (x_n).

Theorem 2.1.

$$U(t) = \frac{1}{1+\log(1-t)}.$$

Proof. Let (σ,τ) be a 2-allowable pair of permutations of length n with $\sigma = s_1 s_2 \ldots s_n$. Suppose s_1 appears as the ith symbol of τ. Then, since the priority queue has capacity 2,

$$\sigma = s_1 s_2 \ldots s_i \alpha,$$
$$\tau = s_2 s_3 \ldots s_i s_1 \beta,$$

where all of $s_2, s_3, \ldots, s_i$ are smaller than s_1 and (α,β) is a 2-allowable pair of permutations on a set Δ of size $n-i$. The set Δ may be chosen in $\binom{n}{n-i}$ ways and, once chosen, there are x_{n-i} choices for (α,β) and $(i-1)!$ choices for $s_1 s_2 \ldots s_i$. Since (σ,τ) determines the value of i uniquely we have

$$x_n = \sum_{i=1}^{n} (i-1)! \binom{n}{n-i} x_{n-i},$$

which may be rewritten as

$$\frac{x_n}{n!} = \sum_{i=1}^{n} \frac{x_{n-i}}{(n-i)!} \frac{1}{i}.$$

Expressing this in terms of $U(t)$ we obtain

$$\begin{aligned} U(t) &= 1 + U(t)\left(t + \frac{t^2}{2} + \frac{t^3}{3} + \cdots\right) \\ &= 1 - U(t)\log(1-t), \end{aligned}$$

giving the result. □

We are indebted to M.S. Paterson for the above proof which is substantially shorter than our original argument.

Corollary 2.5.

$$x_n = \mathrm{O}\left(n!\left(\frac{e}{e-1}\right)^n\right)$$

Proof. The only singularity of the analytic function $U(t)$ is where $1 + \log(1-t) = 0$, which is when $t = (e-1)/e$. Hence the radius of convergence of $\sum x_n t^n/n!$ is $(e-1)/e$. Since the series converges absolutely within the circle of convergence we have $x_n/n! = \mathrm{O}((e/(e-1))^n)$ giving the result. □

2.2. *Serial composition of bounded priority queues*

We define the *allowability relation* A_k of a priority queue P_k to be $A_k = \{(\sigma,\tau) \mid (\sigma,\tau)$ *is k-allowable*$\}$. Then the allowability relation of systems of priority queues formed by serial composition can be represented by the composition of the relations for each of the queues within the system. One of the simplest non-trivial cases would be a priority queue of size 2 followed by a priority queue of size 3 which we shall denote P_2P_3 and which has allowability relation A_2A_3. We shall assume that the priority queues always have capacity greater than 1 since priority queues of size 1 cannot permute the input and are therefore of little interest to us.

Lemma 2.2.

$$i < j \Rightarrow A_i \subset A_j, \tag{4}$$

$$k < l \Rightarrow A_i^k \subset A_i^l, \tag{5}$$

$$A_{r+s-1} \subset A_r A_s. \tag{6}$$

Proof. The inclusions of (4) and (5) are straightforward. They are strict inclusions since it is easily verified that

$$((j, j-1, \ldots, 1),\ (1, 2, \ldots, j)) \in A_j \backslash A_i$$

and

$$((l, l-1, \ldots, 1),\ (1, l, l-1, \ldots, 2)) \in A_i^l \backslash A_i^k.$$

To prove the inclusion of (6) we show that every computation with P_{r+s-1} can be simulated by P_rP_s. Consider an arbitrary allowable pair of P_{r+s-1} and the computation which transforms the input into the output. Whenever this computation inserts an element into the priority queue we shall insert an element into P_r. This is always possible although we may first have to move an element from P_r to P_s. The total capacity of the priority queue P_{r+s-1} is $r+s-1$ and so there is always at least one free location in P_rP_s, and by only moving elements into P_s when there is no free location in P_r we can guarantee that there is always a free location in P_s. When an element is removed from P_{r+s-1} we know that the same element is in P_rP_s somewhere and there

is no smaller element in the system. If the element is in P_s we can simply delete it and continue with the simulation. If it is in P_r we can move it into the free location in P_s and then delete it. So we see that P_rP_s can simulate P_{r+s-1}. Finally, assuming $r,s > 1$, the pair ((3,2,1), (1,3,2)) is an allowable pair of P_rP_s but not of P_{r+s-1} and so containment is strict. □

Lemma 2.3. *$A_{i_1}A_{i_2}\cdots A_{i_k} = A_{j_1}A_{j_2}\cdots A_{j_l}$ if and only if $k = l$ and $i_a = j_a$ for $a = 1,2,\ldots,k$.*

Proof. Suppose $A_{i_1}A_{i_2}\cdots A_{i_k} = A_{j_1}A_{j_2}\cdots A_{j_l}$. Then $l \leqslant k$ otherwise $((l, l-1,\ldots,1), (1,l,l-1,\ldots,2))$ would be an allowable pair of the right-hand system but not the left. Similarly $k \leqslant l$; thus $k = l$.

If, for some a with $1 \leqslant a \leqslant k$, we have $i_a \neq j_a$ then we may suppose that $i_a > j_a$ and put $s = k - 1 + i_a$, $p = k - a - 1$. Then it is readily checked that $((s,s-1,\ldots,1),(1,s, s-1,\ldots,s-p,s-p-r+1,s-p-r+2,\ldots,s-p-1,s-p-r,s-p-r-1,\ldots,2))$ is an allowable pair of the left-hand system but not of the right. The other implication is trivial and the proof is complete. □

3. Priority queues with binary inputs

3.1. The number of inputs and outputs

We begin, as in Section 2, by giving some criteria for a pair of sequences to be k-allowable.

Lemma 3.1. *Let σ and τ be binary sequences expressed as*

$$\sigma = 1^{s_0}01^{s_1}\cdots 1^{s_r}$$

and

$$\tau = 1^{t_0}01^{t_1}\cdots 1^{t_r}.$$

Then (σ,τ) is k-allowable if and only if for $j = 0,1,\ldots,r$

$$0 \leqslant \sum_{i=0}^{j} s_i - t_i \leqslant k-1$$

with equality on the left when $j = r$.

Proof. For convenient reference in the proof and subsequently we call this set of inequalities the *bounded partial sum criterion.*

We begin by showing the necessity of the bounded partial sum criterion. Consider a computation which transforms σ into τ with a priority queue of capacity k. Let j be any integer in $0,1,\ldots,r$ and let $p = \sum_{i=0}^{j} t_i$. At the point that the pth 1 is output exactly j 0's have been output and the priority queue cannot contain the symbol 0. Thus, the

$(j+1)$th 0 of τ has not yet been input and so at most $\sum_{i=0}^{j} s_i$ 1's have been input. But, since at least p 1's must have been input, $\sum_{i=0}^{j} s_i \geqslant \sum_{i=0}^{j} t_i$. To confirm the other inequality consider the state of the computation just before the $(j+1)$th 0 is output. At this point at least $j+1$ 0's have previously been inserted into the priority queue and so at least $\sum_{i=0}^{j} s_i$ 1's have been inserted. Moreover, exactly $\sum_{i=0}^{j} t_i$ of these 1's have been output so there remain at least $\sum_{i=0}^{j} s_i - \sum_{i=0}^{j} t_i$ 1's in the priority queue, together with at least one 0; thus $1 + \sum_{i=0}^{j} s_i - \sum_{i=0}^{j} t_i \leqslant k$.

To prove the sufficiency of the bounded partial sum criterion we construct a computation that, when the condition holds, transforms σ into τ. The computation begins by inserting s_0 1's and deleting t_0 of them. This is clearly possible within a priority queue of capacity k since $0 \leqslant s_0 - t_0 \leqslant k-1$. The computation then proceeds in r stages, the jth of which has the form:

(a) insert 0 and delete 0,

(b) insert and delete 1's, alternating as far as possible, until s_j further 1's have been input and t_j have been output.

An inductive argument establishes that the priority queue contains $\sum_{i=0}^{j-1} s_i - t_i$ 1's just before the jth stage begins $(1 \leqslant j \leqslant r)$. It is clear that, since $\sum_{i=0}^{j} s_i - t_i \geqslant 0$, there remain always sufficient 1's in the priority queue to carry out the jth stage. Moreover, in carrying it out, the capacity of the priority queue is only required to be $\max\{1 + \sum_{i=0}^{j-1} s_i - t_i, \sum_{i=0}^{j} s_i - t_i\} \leqslant k$. □

Corollary 3.1. *Let σ and τ be binary sequences of length n and let $a = (a_1, a_2, \ldots, a_r)$ and $b = (b_1, b_2, \ldots, b_r)$ denote, respectively, the positions within σ and τ where each of the* 0's *occur. Then (σ, τ) is k-allowable if and only if*

$$0 \leqslant a_i - b_i \leqslant k-1 \quad \textit{for each } i = 1, 2, \ldots, r.$$

Proof. The given conditions are a re-statement of the bounded partial sum criterion since $a_i = i + \sum_{j=0}^{i-1} s_i$ and $b_i = i + \sum_{j=0}^{i-1} t_i$. □

Corollary 3.2. *If (σ, τ) is a k-allowable pair of binary sequences then (τ^R, σ^R) is also k-allowable.*

Proof. Using the notation of the previous corollary the positions of the zeros in τ^R and σ^R are given by the vectors $(a'_1, a'_2, \ldots, a'_r)$ and $(b'_1, b'_2, \ldots, b'_r)$ where $a'_i = n+1-b_i$ and $b'_i = n+1-a_i$. Since $a'_i - b'_i = a_i - b_i$ the result follows. □

The characterisation of binary k-allowable pairs given in Corollary 3.1 leads to a useful description of $S_k(\tau)$ for any given binary sequence τ. Every vector $b = (b_1, \ldots, b_r)$ with $1 \leqslant b_i < b_{i+1} \leqslant n$ defines a binary sequence of length n whose 0's occur at positions $b_1, \ldots, b_r$ and, correspondingly, every sequence defines such a vector. Therefore, if b corresponds to a binary sequence τ, $S_k(\tau)$ can be described as the set

of vectors a such that

$$1 \leqslant a_i < a_{i+1} \leqslant n$$

and

$$b_i \leqslant a_i \leqslant b_i + k - 1.$$

For each such vector a it is convenient to define $\tilde{a} = \tilde{a}_1, \ldots, \tilde{a}_r$ where $\tilde{a}_i = a_i - i + 1$; then the above conditions become

$$1 \leqslant \tilde{a}_i \leqslant \tilde{a}_{i+1} \leqslant n - i \tag{7}$$

and

$$b_i - i < \tilde{a}_i \leqslant b_i + k - i. \tag{8}$$

To count the vectors $\tilde{a}$ satisfying the conditions we use the language of partially ordered sets (posets). Let P be any poset of width 2 having a bottom and top element. We may take P to be the disjoint union of two chains $X = x_1, \ldots, x_p$ and $Y = y_1, \ldots, y_q$ where y_1 and y_q are the bottom and top elements of P. In addition to the constraints defining the chains X, Y all the other constraints can be derived as consequences of covering relations of the form $y_{c(i)} < x_i < y_{d(i)}$. Every linear extension of P can be described by stipulating, for each x_i, the least element $y_{e(i)}$ of Y which succeeds x_i in the linear extension. Clearly,

$$1 \leqslant e(1) \leqslant \cdots \leqslant e(p) \leqslant q \tag{9}$$

and

$$c(i) < e(i) \leqslant d(i).$$

Moreover, any sequence $e(1), \ldots, e(p)$ satisfying these conditions defines a linear extension of P. If we now let $\tilde{a}_i$ correspond to $e(i), b_i - i$ correspond to $c(i)$, $b_i + k - i$ correspond to $d(i)$ and $q = n - r$ then the conditions in (7) and the conditions in (9) are equivalent. Therefore, the vectors $\tilde{a}$ satisfying (7) are in one-to-one correspondence with the linear extensions of a certain poset of width 2. In particular, $s_k(\tau)$ is equal to the number of linear extensions of this poset. In [3] an algorithm is presented for calculating the number of such linear extensions in $O(n^2)$ time. In conjunction with Corollary 3.2 we therefore have the following:

Lemma 3.2. *For a binary bounded priority queue $s_k(\tau)$ and $t_k(\sigma)$ can be calculated in $O(n^2)$ time.*

3.2. Serial composition of bounded priority queues

Let B_k denote the relation of k-allowability on binary sequences (that is, B_k is the set of all k-allowable pairs of binary sequences). Then, as we saw in Section 2, $B_k B_j$

(composition of relations) is the set of allowable pairs for the data type formed by connecting a k-bounded priority queue P_k with a j-bounded priority queue P_j in series (so that the output of P_k is the input of P_j). However, compositions of allowability relations are simpler than in the permutation case for we have:

Lemma 3.3. $B_k B_j = B_{k+j-1}$.

Proof. Suppose that $(\sigma,\tau) \in B_k B_j$. Then, by definition, there exists a binary sequence ρ such that $(\sigma,\rho) \in B_k$ and $(\rho,\tau) \in B_j$. Let a and b be the vectors defined in Corollary 3.1 which give the positions of the 0's in σ and τ and let c be the corresponding vector for ρ. Then, by Corollary 3.1, we have, for each $i = 1,2,\ldots,r$,

$$0 \leqslant a_i - c_i \leqslant k-1 \tag{10}$$

and

$$0 \leqslant c_i - b_i \leqslant j-1. \tag{11}$$

Hence, by addition,

$$0 \leqslant a_i - b_i \leqslant (k+j-1)-1 \tag{12}$$

and, by Corollary 3.1 again, $(\sigma,\tau) \in B_{k+j-1}$.

Conversely, if $(\sigma,\tau) \in B_{k+j-1}$ there are two vectors a and b of 0 positions such that (12) holds. We now define a vector c by the rule

$$c_i = \max\{a_i - (k-1), b_i\}$$

and it is easily verified that (10) and (11) hold. Moreover, since each of a and b have strictly increasing components the same is true of c. Thus c defines a binary vector of length n with r 0's at positions $c_1, c_2, \ldots, c_r$. Conditions (10) and (11) and Corollary 3.1 show that $(\sigma,\rho) \in B_k$ and $(\rho,\tau) \in B_j$ from which we can conclude that $(\sigma,\tau) \in B_k B_j$. □

By repeated application of Lemma 3.3 we have:

Theorem 3.1.

$$B_{k_1} B_{k_2} \cdots B_{k_t} = B_m \quad \text{where } m = \sum_{i=1}^{t} k_i - (t-1).$$

3.3. Allowable pairs and ordered forests

Our next aim is to prove the following:

Theorem 3.2. *If $k \geqslant 1$, there is a one-to-one correspondence between the set of k-allowable pairs of binary sequences of length n and the set of ordered forests of height at most $k+1$ on $n+1$ nodes.*

The theorem is proved by studying in greater detail the process by which an input sequence σ is transformed into an output sequence τ. In general there is more than one such transformation and the central idea of the proof is to identify a canonical such transformation. However, in order to do this we need to introduce, in addition to the operations Insert and Delete-Minimum, an operation that we call *Transfer*. A Transfer operation can only be used when the priority queue is empty and the next input is a 1; it moves this next symbol directly from the input to the output. Clearly, permitting Transfer operations does not affect the definition of k-allowability (at least, if $k \geqslant 1$) since a Transfer operation can be simulated by an Insert and Delete-Minimum.

Not every sequence of Insert, Delete-Minimum and Transfer operations makes sense. As always, it is necessary that the ith Delete-Minimum operation is preceded by at least i Insert operations and there must be, in all, equal numbers of Delete-Minimum operations and Insert operations. In addition a Transfer operation is only permitted if there are equal numbers of Insert and Delete-Minimum operations preceding it. A sequence of Insert, Delete-Minimum and Transfer operations which satisfies these two conditions will be called an *extended computation*. An extended computation containing a total of n Insert and Transfer operations is said to have size n (on the grounds that it would be applied to an input sequence of length n).

An extended computation is said to be *standard for* (σ, τ) if it transforms σ into τ and never performs an Insert operation when it is possible to generate a further symbol of τ (by either a Delete-Minimum or Transfer operation). To clarify this definition consider how $\sigma = 100$ might be transformed into $\tau = 001$. An extended computation necessarily begins by inserting 1 and then inserting 0 into the priority queue. It could continue either with another Insert and then 3 Delete-Minimum operations or it could have a Delete-Minimum, an Insert and then 2 Delete-Minimum operations. Only the latter would be standard for $(100, 001)$ since it generates the output symbols as soon as possible.

Notice that if an extended computation is standard for the pair (σ, τ) and is applied to σ then there is never more than one 0 in the priority queue at a time and as soon as a 0 is inserted it must be removed. This is because once a 0 is inserted it is necessarily the next symbol to be output and therefore, by the definition of standard, must be output immediately.

For ease of exposition we shall express the fact that an extended computation C is standard for the pair (σ, τ) by writing $C \sim (\sigma, \tau)$. Then we have:

Lemma 3.4. *$\sim$ defines a one-to-one correspondence between the set of allowable pairs of binary sequences and the set of extended computations.*

Proof. First of all we show that for every binary allowable pair (σ, τ) there exists a unique extended computation C with $(\sigma, \tau) \sim C$.

Since (σ, τ) is allowable there exists some sequence D of Insert and Delete-Minimum operations that transforms σ into τ. D itself may not be standard for (σ, τ) because at

some point elements are inserted into the priority queue even though further elements of τ could have been generated by Delete-Minimum or Transfer operations instead. However we can change D into an extended computation, C, that is standard for (σ,τ) by systematically deferring Insert operations until Delete-Minimum operations have generated whatever further elements of τ are possible and by replacing every Insert, Delete-Minimum pair of operations which inserts and deletes a 1 from an empty priority queue by a Transfer operation. Since each operation performed by an extended computation that is standard for (σ,τ) is determined entirely by the next output symbol to be generated and the contents of the priority queue, C is the unique extended computation with $(\sigma,\tau)\sim C$.

Conversely, for every extended computation $C=X_1X_2\ldots$ there is a unique allowable pair (σ,τ) with $(\sigma,\tau)\sim C$. The existence and uniqueness of (σ,τ) follows from (i) to (v) below which are easy consequences of the definition of $\sim$ and allow (σ,τ) to be constructed from C.

Let i,d,t denote Insert, Delete-Mininum, Transfer. Let X_j be an arbitrary operation of C and let there be $r-1$ operations of the form i or t preceding X_j in C and $s-1$ operations of the form d or t preceding X_j.

(i) If $X_j=t$ then $\sigma_r=1,\tau_s=1$ (and $r=s$).
(ii) If $X_j=d$ and $X_{j-1}=d$ then $\tau_s=1$.
(iii) If $X_j=d$ and $X_{j-1}=i$ then $\tau_s=0$.
(iv) If $X_j=i$ and $X_{j+1}=d$ then $\sigma_r=0$.
(v) If $X_j=i$ and $X_{j+1}=i$ then $\sigma_r=1$. □

Proof of Theorem 3.2. According to Lemma 3.4, for every allowable pair (σ,τ) there is an extended computation C which may be written as $C_0tC_1t\ldots C_r$ where each X_j is a computation consisting of Insert operations and Delete-Minimum operations only, and t denotes Transfer operations. There is a well-known correspondence between computations X_j and rooted trees (in which Insert corresponds to traversing one branch down the tree and Delete-Minimum corresponds to traversing one branch up the tree, see [9]). Thus C corresponds to an ordered forest of $r+1$ such trees.

In this correspondence it is easily seen that, for $k\geqslant 1$, a pair (σ,τ) which is k-allowable corresponds to a forest of height $k+1$. This completes the proof. □

Finally, we note that ordered forests of height $k+1$ on $n+1$ nodes are in one-to-one correspondence with ordered trees of height $k+2$ on $n+2$ nodes and so we can appeal to the theory developed in [6]. There a generating function for the number of trees with n nodes and height h or less is given in several different forms and from this the following closed form for the number of such trees $A_{n,h}$ is derived:

$$A_{n,h}=\frac{1}{h+1}\sum_{1\leqslant j\leqslant h/2}4^n\sin^2(j\pi(h+1))\cos^{2n-2}(j\pi(h+1)),\quad n\geqslant 2.$$

Using Theorem 3.2 we can conclude that the number of k-allowable pairs of binary sequences of length n is $A_{n+2,k+2}$.

References

[1] M.D. Atkinson, Transforming binary sequences with priority queues, *Order* **10** (1993) 31–36.
[2] M.D. Atkinson and R. Beals, Priority queues and permutations, *SIAM J. Comput.* **23** (1994) 1225–1230.
[3] M.D. Atkinson and H.W. Chang, Computing the number of mergings with constraints, *Inform. Process. Lett.* **24** (1987) 289–292.
[4] M.D. Atkinson, M.J. Livesey and D. Tulley, Permutations generated by token passing in graphs, *Theoret. Comput. Sci.*, to appear.
[5] M.D. Atkinson and M. Thiyagarajah, The permutational power of a priority queue, *BIT* **33** (1993) 2–6.
[6] N.G. de Bruijn, D.E. Knuth and S.O. Rice, The average height of planted plane trees, in: R.C. Read, ed., *Graph Theory and Computing* (Academic Press, New York, 1972) 15–22.
[7] M. Golin and S. Zaks, Labelled trees and pairs of input–output permutations in priority queues, in: *Proc. 20th Internat. Conf. on Graph-Theoretic Concepts in Computer Science* (*WG*), Munich, Germany (June 1994).
[8] D.E. Knuth, *Fundamental Algorithms*, *The Art of Computer Programming* (Addison-Wesley, Reading, MA, 1973).
[9] J.H. van Lint and R.M. Wilson, *A Course in Combinatorics* (Cambridge University Press, Cambridge, 1992).