Priority Queues and Multisets

M.D. Atkinson
S.A. Linton
L.A. Walker

Department of Mathematical and Computational Sciences
University of St. Andrews
North Haugh,
St. Andrews KY16 9SS, Scotland
{sal,mda,louise}@dcs.st-and.ac.uk

Submitted: February 14 1995; Accepted November 10 1995.

Abstract

A priority queue, a container data structure equipped with the operations insert and delete-minimum, can re-order its input in various ways, depending both on the input and on the sequence of operations used. If a given input σ can produce a particular output τ then (σ, τ) is said to be an *allowable pair*. It is shown that allowable pairs on a fixed multiset are in one-to-one correspondence with certain k-way trees and, consequently, the allowable pairs can be enumerated. Algorithms are presented for determining the number of allowable pairs with a fixed input component, or with a fixed output component. Finally, generating functions are used to study the maximum number of output components with a fixed input component, and a symmetry result is derived.

Mathematical Reviews Subject Classifications: 68R05,05A15,68P05

1 Introduction

Abstract data types (ADTs) are a fundamental design tool in modern software systems. They are setting the direction of programming in the 1980's and 1990's as firmly as Structured Programming set it in the 1960's and 1970's. In principle there is an infinity of possible ADTs since an ADT is defined once its permitted set of operations has been specified, and there is no restriction on this set except for common sense. In practice, a small number of ADTs occur over and over again (stacks, arrays, queues, dictionaries etc.) suggesting that some ADTs are more "natural" than others.

Exactly the same phenomenon occurs in algebra and, significantly, the natural algebraic systems (groups, rings, modules etc.) have very rich theories. Many of the commonly occurring ADTs 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).

Normally, a container data type is used in the following way. First it is initialised as empty. Then some input sequence of data items is inserted into it one by one (the *Insert* operation) and these items are removed (the *Delete* operation) in some order and placed in an output sequence. The insertions and deletions may be interleaved in any way (provided that the *Delete* operation is never used on the empty container). Thus, a container data type is a mechanism for transforming an input sequence into an output sequence. The functional behaviour of a container data type is essentially characterised by the relationship between the input sequences and the output sequences. An understanding of this relationship allows us to 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 (σ, τ) is *allowable* (with respect to the data type). The statistics of the allowability relation associated with a given data type are measures of the transformational capability of that data type.

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 is trivial since the output sequence is always the same as the input sequence; in other words, the only allowable pairs are of the form (σ, σ). In the case of stacks (where the *Delete* operation always removes the element which was placed in the stack most recently) the relation is more complicated and there are significant connections with a number of other combinatorial objects such as trees [9] 2.3.4.4, ballot sequences [10] p531, Young tableaux [10] pp63,64, and triangulations of polygons [6] pp320–324. For dictionaries (which have an unrestricted *Delete* operation) the output can be any permutation of the input.

The allowability relation has also been considered for various types of deques. For a general deque (where insertions and deletions are permitted at both ends) Pratt [12] has characterised the allowable pairs in terms of forbidden subsequences. For input restricted deques (where insertion is permitted at one end only) and output restricted deques, Knuth [9] 2.2.1 has given generating functions and recurrence relations for computing the number of allowable pairs of a given length.

In this paper we shall consider the case of priority queues. Priority queues are characterised by two main operations, *Insert* and *Delete-Minimum*. Several efficient implementations of them are known in which $O(\log n)$ operations suffice for each of *Insert* and *Delete-Minimum*, where n is the number of items currently in the priority queue. A *priority queue algorithm* is defined to be a sequence of *Insert* and *Delete-Minimum* operations which is well-formed in the same sense as bracket sequences. In other words there are equal numbers of each operation and (in order that a *Delete-Minimum* is only executed when the priority queue is non-empty) the t^{th} *Delete-Minimum* operation must be preceded by at least t *Insert* operations. A priority queue algorithm with n *Insert* and n *Delete-Minimum* operations transforms an input sequence of length n into an output sequence of length n. In the terminology introduced above, the pair (σ, τ) is allowable if and only if there is a priority queue algorithm which transforms σ into τ.

Our focus here is on the allowability relation for priority queues. The case of input and output sequences which are permutations of $1, 2, \ldots, n$ was considered in [1, 2, 7] whilst binary sequences were considered in [3]. As was shown in [2, 7] allowable pairs of permutations are in one-to-one correspondence with labelled trees for which there is an extensive enumeration theory [11] beginning with Cayley's famous enumeration formula. Labelled trees are in one-to-one correspondence with other combinatorial objects; for example, with the number of ways of expressing an n-cycle as a product of $n-1$ transpositions [5] p86. We shall consider the general case in which the input and output sequences are elements of a finite multiset S. From now on S will denote a fixed multiset of size n whose elements are $1, 2, \ldots, k$ with element i having multiplicity a_i. Our results unify those in [1, 2, 3, 7] and lead to some combinatorial results which are peculiar to the general case.

In the next section we give a one-to-one correspondence between the set of allowable pairs and a set of k-way trees. This correspondence permits us to enumerate the set of allowable pairs. In section 3 we give polynomial time algorithms for computing the number of outputs that can arise from a fixed input, and for the number of inputs that can give rise to a fixed output. Finally, in section 4, we consider the maximal number of outputs that a given input can generate as a function of $a_1, \ldots, a_k$. We derive recurrences and a multi-variate generating function, and conclude with an unexpected symmetry result.

2 Allowable pairs

The main theorem in this section concerns the number of allowable pairs of sequences in which the elements of the input (and output) sequences are permutations of the fixed multiset S. Although in reality equal symbols of a multiset cannot be distinguished we shall find it convenient, as an expositional device only, to pretend that each symbol has a unique identity. The symbols all equal to some fixed i will be distinguished from each other by their order of occurrence in the input sequence σ. We further suppose, again as an expositional device only, that occurrences of multiple copies of the same symbol are deleted from the priority queue in the same order as they are inserted; so the order of occurrence of equal symbols in the output sequence τ is the same as their order in σ.

We occasionally make this convention explicit by subscripting each symbol of a given value with the order of its occurrence among the symbols of this value. For example, the allowable pair of Figure 1 below would be written:

$$(3_1 2_1 3_2 1_1 1_2 2_2 2_3 3_3 1_3 2_4 1_4,\ 1_1 2_1 3_1 1_2 2_2 3_2 2_3 1_3 3_3 1_4 2_4)$$

These two devices allow us to associate each input symbol with a unique output symbol. This allows us to track a symbol's progress as it goes into the priority queue and eventually emerges as a member of the output sequence. In practice, of course, when there are several equal minimal elements in the priority queue, any one of them might be deleted by a *Delete-Minimum* operation. But each of them will produce the same value in the output sequence, so it does no harm to suppose that it is the minimal value of longest residence in the priority queue that is deleted.

Theorem 1 *The number of allowable pairs (σ, τ) where σ (and τ) are permutations of the multiset S is*

$$\frac{1}{n+1}\prod_{i=1}^{k}\binom{n+1}{a_i}$$

Corollary 1 *The number of allowable pairs (σ, τ) where σ ranges over all the k^n sequences of length n with at most k distinct members is*

$$\frac{1}{n+1}\binom{kn+k}{n}$$

Proof of Corollary. The number in question is

$$\sum \frac{1}{n+1}\prod_{i=1}^{k}\binom{n+1}{a_i}$$

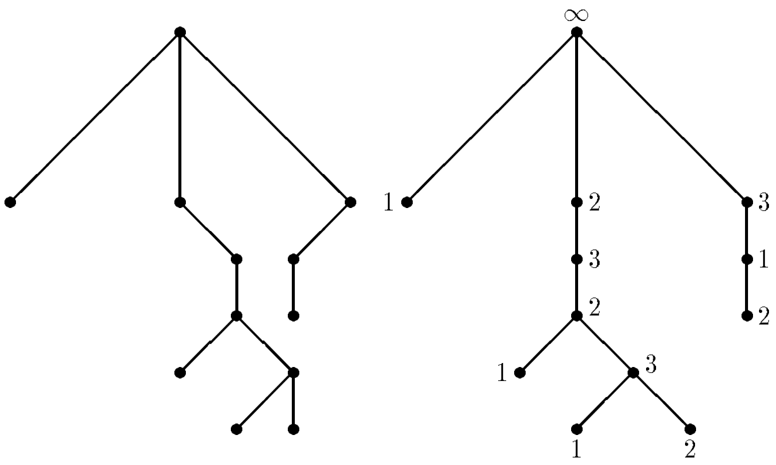
where the summation is over all integer vectors $(a_1, a_2, \ldots, a_k)$ with non-negative components such that $\sum a_i = n$. This is clearly the coefficient of x^n in

$$\frac{1}{n+1}(1+x)^{n+1}(1+x)^{n+1}\ldots(1+x)^{n+1} = \frac{1}{n+1}(1+x)^{kn+k}$$

from which the result follows.

In the case that all $a_i = 1$ and $k = n$, Theorem 1 corresponds to sequences which are permutations of $\{1, 2, \ldots, n\}$ and gives the main theorem of [1] while the case $k = 2$ of the Corollary corresponds to binary sequences and gives one of the results of [3].

Theorem 1 is proved by establishing a one-to-one correspondence between allowable pairs and certain trees. We recall the definition of a k-way tree: a *k-way tree* is either empty or it consists of a root node and a sequence of k k-way subtrees. It is common to represent a k-way tree by a diagram in which the root node is connected to its non-empty subtrees by edges which point in one of k fixed directions.



$$(3_1 2_1 3_2 1_1 1_2 2_2 2_3 3_2 1_3 2_4 1_4,\, 1_1 2_1 3_1 1_2 2_3 3_3 2_3 1_3 3_3 1_4 2_4)$$

Figure 1: a 3-way tree, its corresponding unambiguous tree and allowable pair

We also define an *unambiguous* tree. Such a tree is rooted, unordered, and labelled; the root is labelled ∞ and the non-root nodes are labelled with the elements of a multiset of integers subject to the condition that the children of every node have distinct labels.

There is an obvious one-to-one correspondence between k-way trees and unambiguous trees on multisets on $\{1, 2, \ldots, k\}$. Given the diagram of a k-way tree we can label the root node ∞, and label the lower node of any edge in the i^{th} direction with the label i; this gives an unambiguous tree. Conversely, from any unambiguous tree we can use the labels to direct the edges in the appropriate directions thereby obtaining a k-way tree. In this correspondence the number of labels equal to i in an unambiguous tree is equal to the number of edges pointing in the i^{th} direction in the associated k-way tree. Figure 1 depicts a 3-way tree and corresponding unambiguous tree. It also shows the allowable pair associated with the unambiguous tree by the one-to-one correspondence of Theorem 2.

Our first lemma gives a decomposition of an allowable pair which we use as an inductive tool throughout this section and the next. In this lemma we introduce the practice, continued in the proof of Theorem 2, of prefacing input sequences by the symbol ∞. There are two reasons for this. The first is that we often need to refer to the symbol that immediately precedes one of the symbols k in σ; and if k happens to be the first symbol of σ, we can, by use of the ∞ symbol, handle this in a uniform way. The second reason has to do with Theorem 2, where we associate unambiguous trees on $n + 1$ nodes with allowable pairs (σ, τ) of length n; we can then find a labeling of the tree nodes with the symbols in $\infty\sigma$.

Lemma 1 *Let σ, τ be sequences of length n with a_i occurrences of i (for $i = 1, 2, \ldots, k$). Suppose that τ is expressed as $\tau = \tau_0 k \tau_1 \ldots k \tau_{a_k}$ where none of the τ_i contain k. The pair (σ, τ) is allowable if and only if there exist subsequences $\sigma_0, \sigma_1, \ldots, \sigma_{a_k}$ of σ and elements $x_1, \ldots, x_{a_k}$ of $\infty\sigma$ such that*

(i) for each $0 \leq i \leq a_k$, (σ_i, τ_i) is allowable,

(ii) σ with all occurrences of k removed is the sequence $\sigma_0\sigma_1 \ldots \sigma_{a_k}$,

(iii) for each $1 \leq j \leq a_k$, x_j is immediately before the j^{th} k in $\infty\sigma$ and the position of x_j in $\infty\sigma$ is strictly to the left of the position of the j^{th} k in $\infty\tau$

Proof First assume that (σ, τ) is allowable and let $\mathcal{G}$ be a priority queue algorithm which transforms σ into τ. We recall our convention that occurrences of multiple copies of the same symbol are deleted from the priority queue by $\mathcal{G}$ in the same order as they are inserted.

For each $j = 1, 2, \ldots, a_k$, the j^{th} k must be deleted by $\mathcal{G}$ after all elements appearing before it in σ have been deleted (because those elements are all less than k or equal to and more senior than this k). Therefore, defining x_j to be the symbol of $\infty\sigma$ immediately before the j^{th} k, the position of x_j in $\infty\sigma$ is strictly to the left of the position of the j^{th} k in $\infty\tau$; this gives condition (iii).

Define, for every $i = 0, \ldots, a_k$, the sequence σ_i to be the subsequence of σ which is mapped to τ_i under the action of $\mathcal{G}$. Since (σ, τ) is allowable, (σ_i, τ_i) is also allowable for all $0 \leq i \leq a_k$ and so (i) holds.

To show (ii), it is enough to prove that for any $i = 0, \ldots, a_{k-1}$, if x appears in τ_i and y appears in τ_{i+1}, then x precedes y in σ. Assume otherwise and argue for a contradiction. Since y precedes x in σ and succeeds x in τ, it follows that $y > x$. However the i^{th} k in τ appears before y in τ and so this symbol k appears before y in σ. Therefore when x is deleted from the priority queue by $\mathcal{G}$, y and k must be present in the priority queue; it follows that y will be deleted before k since $y < k$. This is the required contradiction.

Conversely, assume there exist $\sigma_0, \sigma_1, \ldots, \sigma_{a_k}$ and $x_1, \ldots, x_{a_k}$ satisfying (i), (ii), and (iii). For each $i = 0, \ldots, a_k$ let $\mathcal{G}_i$ be a priority queue algorithm that transforms σ_i to τ_i and let $\mathcal{A}$ be the sequence of *Insert* and *Delete-Minimum* operations $\mathcal{G}_0\mathcal{G}_1 \ldots \mathcal{G}_{a_k}$. Form a new sequence obtained from $\mathcal{A}$ as follows: add an extra *Delete-Minimum* operation at the end of each subsequence $\mathcal{G}_i$ of $\mathcal{A}$ ($i = 0, 1, \ldots, a_{k-1}$); in the resulting sequence, if x_j ($1 \leq j \leq a_k$) is the $(r+1)^{\text{th}}$ element of $\infty\sigma$, add an extra *Insert* operation after the r^{th} *Insert* operation (or at the beginning of $\mathcal{A}$ if $r = 0$). Conditions (i) and (iii) ensure that the resulting sequence of *Insert* and *Delete-Minimum* operations is a well-formed priority queue algorithm $\mathcal{G}$. From condition (ii) it follows that $\mathcal{G}$ transforms σ into τ which completes the proof.

As an example, we consider

$$(\sigma, \tau) = (3_1 2_1 3_2 1_1 1_2 2_2 2_3 3_3 1_3 2_4 1_4, 1_1 2_1 3_1 1_2 2_2 3_2 2_3 1_3 3_3 1_4 2_4)$$

Dividing τ at each symbol 3, we obtain

i	σ_i	τ_i	x_i
0	21	12	–
1	12	12	∞
2	21	21	2_1
3	21	12	2_3

Theorem 2 *There is a one-to-one correspondence between unambiguous trees on $n+1$ nodes with, for $1 \le i \le k$, a_i nodes labelled i and allowable pairs (σ, τ) where σ and τ are sequences of length $n = \sum_{i=1}^{k} a_i$ with a_i occurrences of i $(1 \le i \le k)$.*

Proof The proof of the theorem is by construction. We shall show how to construct an unambiguous tree for each allowable pair and prove that the construction rule is a bijection between the set of allowable pairs and the set of unambiguous trees.

The tree Γ that is associated with an allowable pair (σ, τ) has nodes labelled by the elements of σ together with a label ∞, and the multisets of node labels and elements of $\infty\sigma$ are, of course, equal. It will convenient to have a specific correspondence between the set of nodes of Γ and the elements of $\infty\sigma$ such that each node is labelled by its corresponding element in $\infty\sigma$. Such a correspondence will be called a *matching*. Our construction of Γ produces a particular matching which is itself used, via induction, in defining the construction. The matching has an additional property that we call *conformal*: if x and y are nodes in Γ, x the parent of y, and the label of x is less than or equal to the label of y, then the element of $\infty\sigma$ which matches x occurs earlier in $\infty\sigma$ than the element which matches y.

The construction of Γ is inductive and the base case is when σ and τ are empty; then the corresponding tree has a single node labelled ∞ and, in the matching, this node matches the first (and only) element of $\infty\sigma$.

If σ and τ are non-empty we consider the decomposition in Lemma 1. By induction, we may assume that, for each (σ_i, τ_i), there exists an unambiguous tree Γ_i, and a conformal matching between its set of nodes and the elements of $\infty\sigma_i$.

We now form a new tree from $\Gamma_0, \Gamma_1, \ldots, \Gamma_{a_k}$. The first step is to replace each of the labels ∞ on the roots of $\Gamma_1, \ldots, \Gamma_{a_k}$ by a label k. We regard the label on the root of Γ_i as corresponding to the ith k, k_i of $\infty\sigma$; then we have a matching μ between the elements of $\infty\sigma$ and the nodes of the forest $\{\Gamma_0, \Gamma_1, \ldots, \Gamma_{a_k}\}$. The second step is to define new edges by the following rule:

(a) if k_j immediately follows a symbol (x_j) of $k_i\sigma_i$ in σ an edge is placed between Γ_i and Γ_j

(b) the edge connects the root of Γ_j to the node z_j of Γ_i that matches the symbol x_j

Condition (b) together with condition (iii) of Lemma 1 ensures that if two trees Γ_i and Γ_j are joined in this way then $i < j$; thus the resulting graph Γ is indeed a tree. Moreover, by the conformal property, the node z_j of Γ_i has no child node in Γ_i labelled k; thus Γ is unambiguous. Furthermore, μ is now a conformal matching between the nodes of Γ and the elements of $\infty\sigma$.

Notice that, in Γ, the set of subtrees Γ_i may be readily identified; they are the connected components that remain when all edges with a lower end point labelled k have been removed. Moreover, their order, $\Gamma_0, \Gamma_1, \ldots, \Gamma_{a_k}$ may be reconstructed also. Obviously Γ_0 is the subtree whose root is labelled ∞. The immediate subtrees of it will be $\Gamma_1, \Gamma_2, \ldots, \Gamma_c$ and their order is determined by the order of the nodes of Γ_0 to which they are attached. The immediate subtrees of Γ_1 will come next (ordered by the nodes of Γ_1 to which they are attached); then the subtrees of Γ_2, Γ_3 and so on.

These observations allow the construction to be reversed. Suppose that Γ is an unambiguous tree. We can produce a corresponding allowable pair (σ, τ) and a conformal mapping between the nodes of Γ and the elements of $\infty\sigma$ as follows. First we delete all edges whose lower end point in labelled k to obtain a forest of trees $\{\Gamma_0, \Gamma_1, \ldots, \Gamma_{a_k}\}$. We let Γ_0 be the unique tree of the forest whose root is labelled ∞; all the other subtrees have root labelled k.

We now replace all the labels k by the label ∞ to obtain a forest of unambiguous trees, and using induction, we let (σ_i, τ_i) be the allowable pair that corresponds to Γ_i. Again by induction there will exist conformal mappings between the nodes of each Γ_i and the elements of $\infty\sigma_i$. Now, by the remarks following the previous construction, we can determine an order on the trees $\Gamma_0, \Gamma_1, \ldots, \Gamma_{a_k}$ in the forest.

Finally we construct the desired pair (σ, τ) according to Lemma 1. In other words we set $\tau = \tau_0 k \tau_1 \ldots k \tau_{a_k}$ and define σ to be the result of inserting a_k symbols k in $\sigma_0\sigma_1 \ldots \sigma_{a_k}$; the jth k is inserted immediately after x_j, where x_j is the symbol that matches the parent of the root of Γ_j in Γ (notice that the unambiguous property of Γ is used here in ensuring that $x_1, x_2, \ldots, x_{a_k}$ are all different elements of σ).

We have now shown that the construction of an unambiguous tree from an allowable pair is a bijection and so the proof of Theorem 2 is complete.

As an example, we show the construction of the unambiguous tree associated with the allowable pattern

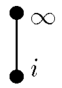
$$(\sigma, \tau) = (3_1 2_1 3_2 1_1 1_2 2_2 2_3 3_3 1_3 2_4 1_4, 1_1 2_1 3_1 1_2 2_2 3_2 2_3 1_3 3_3 1_4 2_4)$$

Decomposing this pattern in accordance with lemma 1, we obtain the table of σ_i, τ_i and x_i which was given just before theorem 2.

We first catalogue the unambiguous trees associated with all allowable pairs of length not more than 2.

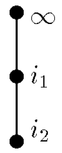
With the empty pair $(,)$ is associated a tree $\overset{\infty}{\bullet}$ consisting simply of a root.

With the pair (i, i) is associated the unambiguous tree



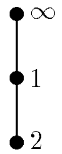
This is evidently the only possibility, but it is easy to check that it is produced by the construction.

For the pair (i_1i_2, i_1i_2), the only possible unambiguous tree with the proper labels is.

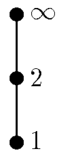


For the pair $(12, 12)$ we have to use the construction of Theorem 2. Here $k = 2$, $a_k = 1$ and $(\sigma_0, \tau_0) = (1, 1)$, $(\sigma_1, \tau_1) = (,)$ and $x_1 = 1$.

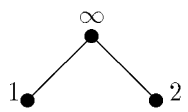
The corresponding tree is thus:



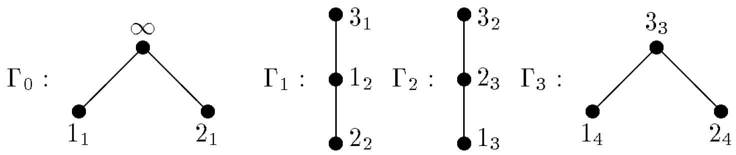
For the pair $(21, 21)$ we have $(\sigma_0, \tau_0) = (,)$, $(\sigma_1, \tau_1) = (1, 1)$ and $x_1 = \infty$. We obtain the tree:



The remaining allowable pair of length 2 is $(21, 12)$, for which we have $(\sigma_0, \tau_0) = (1, 1)$, $(\sigma_1, \tau_1) = (,)$ and $x_1 = \infty$. This gives rise to the tree:



This gives us the components $\Gamma_0, \Gamma_1, \Gamma_2$ and Γ_3 of the forest from which the unambiguous tree of figure 1 is constructed. Relabeling the roots of Γ_1, Γ_2 and Γ_3 and attaching subscripts to the labels, they are:



Finally, we add edges joining x_i to 3_i: between ∞ and 3_1; between 2_1 and 3_2 and between 2_3 and 3_3. We can then delete the subscripts (which represent the conformal map, used in the inductive step but not needed in the final tree) to give the tree of figure 1.

The reverse construction goes straight-forwardly. Decomposing Γ, we obtain the four subtrees $\Gamma_0, \ldots, \Gamma_3$, with their labels. By induction we can determine the sub-patterns σ_i and τ_i. Finally the x_i are determined by the positions of the edges we deleted in the decomposition. According to Lemma 1, σ and τ are determined by the σ_i, τ_i and x_i.

In view of Theorem 2 we must determine the number of unambiguous trees to obtain a proof of Theorem 1. Since unambiguous trees are in one-to-one correspondence with k-way trees, Theorem 1 follows from the main result of [4].

3 The number of inputs and outputs

In this section we study the two quantities

$t(\sigma) = |\{\tau : (\sigma, \tau) \text{ is allowable}\}|$, and

$s(\tau) = |\{\sigma : (\sigma, \tau) \text{ is allowable}\}|$

Clearly $t(\sigma)$ is the number of different outputs that may arise when a priority queue is presented with the input σ; and $s(\tau)$ is the number of inputs capable of generating the output τ.

It turns out that $t(\sigma)$ can be calculated by adapting the algorithm of [2] that was developed for permutations. Assuming that $a_k > 0$ (if not, we replace k by $k-1$) we may put $\sigma = \alpha k \beta$ where $\beta = b_1 b_2 \ldots b_r$ contains no element equal to k. Then we note that

$$t(\alpha k \beta) = \begin{cases} t(\alpha) & \text{if } r = 0 \\ t(\alpha)t(\beta) + t(\alpha b_1 k b_2 \ldots b_r) & \text{otherwise} \end{cases}$$

(the term $t(\alpha)t(\beta)$ enumerates those outputs which arise from deleting the element k from the priority queue before inserting the first symbol of β (at that point the priority queue will be empty) and the term $t(\alpha b_1 k b_2 \ldots b_r)$ enumerates the outputs which arise from inserting b_1 before the element k is deleted).

As in [2] these equations can be made the basis of a dynamic programming algorithm with execution time $O(n^4)$.

The quantity $s(\tau)$ cannot be calculated by generalising the permutation algorithm of [2] (that algorithm uses the distinctness of the elements in an essential way). We can, however, use Lemma 1. In the notation of that lemma we have

$$s(\tau) = P \prod_{i=1}^{k} s(\tau_i)$$

where P is the number of ways in which the a_k occurrences of k may be validly positioned within $\sigma_0 \sigma_1 \ldots \sigma_{a_k}$. A valid positioning is, by Lemma 1, one where each k occurs no later than the corresponding k in τ.

In order to calculate P, let $r = a_k$ and let $n_1 < n_2 < \ldots < n_r$ be the positions of the elements equal to k in τ. Then, clearly, P is the number of vectors $(m_1, m_2, \ldots, m_r)$ such that

1. $1 \leq m_1 < m_2 < \ldots < m_r \leq n$, and
2. $m_i \leq n_i$

since each vector gives a possible list of positions for the elements equal to k in σ. We now give a relatively simple algorithm for computing P.

Define the $(r+1)-$variable function

$$Q_{r+1}(m_0, n_1, n_2, \ldots, n_r) = \sum_{m_1=m_0+1}^{n_1} \sum_{m_2=m_1+1}^{n_2} \cdots \sum_{m_r=m_{r-1}+1}^{n_r} 1$$

Then $P = Q_{r+1}(0, n_1, n_2, \ldots, n_r)$ since each term in the summation corresponds to a vector satisfying the conditions above. Then the recurrences

$$\begin{aligned} Q_1(m_r) &= 1 \qquad\qquad \text{for all } m_r \\ Q_{r-k+1}&(m_k, n_{k+1}, \ldots, n_r) \\ &= \sum_{m_{k+1}=m_k+1}^{n_{k+1}} Q_{r-k}(m_{k+1}, n_{k+2}, \ldots, n_r) \\ &= Q_{r-k+1}(m_k+1, n_{k+1}, \ldots, n_r) + Q_{r-k}(m_k+1, n_{k+2}, \ldots, n_r) \end{aligned}$$

allow the values $Q_{r-k+1}(m_k, n_{k+1}, \ldots, n_r)$ for all values of m_k to be calculated in constant time per value. The quantity P can therefore be calculated in $O(na_k)$ steps.

An efficient algorithm for computing $s(\tau)$ is now evident. Recursively, we compute each $s(\tau_i)$, compute P, and then compute $s(\tau)$. The total work done is dominated by the time taken to compute P and all the analogous quantities in the recursive steps. One of these computations is done for every symbol of τ and so the total time is $O(n \sum_{i=1}^{k} a_i) = O(n^2)$

4 The maximum number of outputs

It appears to be impossible to find a closed formula for $t(\sigma)$ valid for arbitrary re-arrangements of the multiset S. Clearly, $t(\sigma)$ is minimal in the case $\sigma = 1^{a_1} 2^{a_2} \ldots k^{a_k}$ (here $t(\sigma) = 1$). The maximal value of $t(\sigma)$ occurs when $\sigma = k^{a_k} \ldots 2^{a_2} 1^{a_1}$ and, in this section, we shall investigate this maximal value. The analogous problem for $s(\tau)$ is more straightforward: its maximal value is $\binom{n}{a_1, a_2, \ldots, a_k}$ since every arrangement of S can generate the output $1^{a_1} 2^{a_2} \ldots k^{a_k}$.

Let $c(a_1, a_2, \ldots, a_k) = t(k^{a_k} \ldots 2^{a_2} 1^{a_1})$. We shall derive recurrences for this quantity and a closed expression for its multi-variate generating function. Remarkably, this generating function is a symmetric function in its k variables and therefore $c(a_1, a_2, \ldots, a_k)$ is symmetric in its arguments. We know of no simple reason for this fact.

Lemma 2 *Let $\sigma = k^{a_k} \ldots 2^{a_2} 1^{a_1}$. Then a sequence τ on the same multiset is one of the possible outputs arising from σ as input if and only if τ has no subsequence abc whose relative values are as* 132

Proof When a priority queue is given input in non-increasing order the current minimal element may be taken to be the last element inserted. We may therefore presume that elements are inserted and deleted according to a Last-In, First-Out discipline; in other words the priority queue behaves like a stack. Stack permutations can be characterised by the avoidance of a forbidden pattern [9] p.239 ((132) when the input is decreasing); this remains true even when the input has repeated elements.

Remarks

1. It is convenient to describe the conclusion of this lemma by saying that an output generated by σ has "no (132) pattern", or that (σ, τ) has "no $(321, 132)$ pattern"
2. A more general result than this can be proved: if σ and τ are any arrangements of S then (σ, τ) is allowable if and only if it has no $(12, 21)$ pattern nor $(321, 132)$ pattern.

Lemma 3 *$c(a_1) = 1$ and for $k \geq 2$,*

$$c(a_1, \ldots, a_k) =$$
$$\begin{cases} \displaystyle\sum_{r=1}^{a_1} c(r, a_2 - 1, \ldots, a_k) + c(a_1 + a_2, a_3, \ldots, a_k) & \text{for } a_2 \geq 1 \\ c(a_1, a_3, .., a_k) & \text{for } a_2 = 0 \end{cases}$$

Proof If $a_2 = 0$ the result follows by relabeling the elements of S. Assume $a_2 \geq 1$ and let R be the set of outputs that can be generated by the input $k^{a_k} \ldots 2^{a_2} 1^{a_1}$. Let $\sigma \in R$ be such that all the 2's in σ come before all the 1's, and let σ' be the sequence obtained from σ by replacing all the 1's by 2's. Then σ' is an output for the sequence $k^{a_k} \ldots 2^{a_2+a_1}$. Conversely, for any output σ' generated by the input $k^{a_k} \ldots 2^{a_2+a_1}$, we can construct a sequence σ in R by replacing the last a_1 2's in σ' by 1's. The new sequence is in R since this change cannot introduce any (132) patterns. Hence the total number of sequences in R with all the 2's before all the 1's is $c(a_1 + a_2, a_3, \ldots, a_k)$.

Next consider a sequence σ in R containing at least one occurrence of a 1 before a 2. By considering the first occurrence of 1 in σ and the first 2 succeeding this 1 we obtain a unique decomposition $\sigma = \sigma_1 1 \alpha 2 \sigma_2$ where σ_1 contains no 1's and α contains no 2's. But then, because σ contains no 132 pattern α has the form $\alpha = 1^q$ for some $q \geq 0$.

Consider the sequence $\hat{\sigma} = \sigma_1 1 \sigma_2$ obtained by removing the uniquely determined substring $1^q 2$ from σ. Clearly $\hat{\sigma}$ has no 132 pattern either and so is one of the $c(a_1 - q, a_2 - 1, a_3, \ldots, a_k)$ outputs obtainable from the input sequence $k^{a_k} \ldots 2^{a_2-1} 1^{a_1-q}$.

Conversely, from any one of these latter outputs $\hat{\sigma}$ we can obtain a sequence of R merely by inserting the substring $1^q 2$ immediately after the first 1 of $\hat{\sigma}$ (since this cannot introduce a 132 pattern). Therefore the total number of sequences in R which do not have all their 1's before all their 2's is

$$\sum_{r=1}^{a_1} c(r, a_2 - 1, \ldots, a_k)$$

which proves the result.

Using the previous lemma one can obtain the following results fairly easily.

$$\begin{aligned}
c(a_1) &= 1 \\
c(a_1, a_2) &= \binom{a_1 + a_2}{a_1} \\
c(a_1, a_2, a_3) &= 2^{a_1+a_2+a_3} - \sum_{r=0}^{a_1-1} \binom{a_1+a_2+a_3}{r} \\
&\quad - \sum_{r=0}^{a_2-1} \binom{a_1+a_2+a_3}{r} - \sum_{r=0}^{a_3-1} \binom{a_1+a_2+a_3}{r}
\end{aligned}$$

However, obtaining an expression for $c(a_1, a_2, \ldots, a_k)$ in general appears to present more difficulty. Further progress can be made by defining

$$G(x_1, \ldots, x_k) = \sum_{a_1=0} \cdots \sum_{a_k=0} c(a_1, \ldots, a_k) x_1^{a_1} \ldots x_k^{a_k}$$

as the generating function for $c(a_1, \ldots, a_k)$. The recurrence for $c(a_1, \ldots, a_k)$ above can be used to obtain a recursive formula for $G(x_1, \ldots, x_k)$.

Lemma 4

$$\begin{aligned}
G(x_1) &= \frac{1}{1 - x_1}, \\
G(x_1, \ldots, x_k) &= \\
&\frac{x_2(1-x_2)G(x_2, x_3, \ldots, x_k) - x_1(1-x_1)G(x_1, x_3, \ldots, x_k)}{(1 - x_1 - x_2)(x_2 - x_1)} \qquad \text{for } k \geq 2
\end{aligned}$$

Proof Clearly, $c(a_1) = 1$ for all $a_1 \geq 0$ and so $G(x_1) = 1/(1-x_1)$. Now assume $k \geq 2$.

$$\begin{aligned}
&G(x_1, \ldots, x_k) \\
&= \sum_{a_1=0} \cdots \sum_{a_k=0} c(a_1, \ldots, a_k) x_1^{a_1} \cdots x_k^{a_k} \\
&= \sum_{a_1=0} \sum_{a_2=1} \cdots \sum_{a_k=0} \left(\sum_{r=0}^{a_1} c(r, a_2 - 1, \ldots, a_k) \right.
\end{aligned}$$

$$
\begin{aligned}
& - c(0, a_2 - 1 \ldots, a_k) + c(a_1 + a_2, a_3, \ldots, a_k)\Big) x_1^{a_1} \cdots x_k^{a_k} \\
& + \sum_{a_1=0} \sum_{a_3=0} \cdots \sum_{a_k=0} c(a_1, a_3, \ldots, a_k) x_1^{a_1} x_3^{a_3} \cdots x_k^{a_k} \\
= \quad & x_2 \sum_{a_1=0} \cdots \sum_{a_k=0} \sum_{r=0}^{a_1} c(r, a_2, \ldots, a_k) x_1^{a_1} \cdots x_k^{a_k} \\
& + x_2 \sum_{a_1=0} \cdots \sum_{a_k=0} \left(c(1 + a_1 + a_2, a_3, \ldots, a_k) - c(a_2, a_3, \ldots, a_k)\right) x_1^{a_1} \ldots x_k^{a_k} \\
& + G(x_1, x_3, \ldots, x_k) \\
= \quad & x_2 \sum_{a_2=0} \cdots \sum_{a_k=0} \sum_{r=0} \sum_{a_1=r} c(r, a_2, \ldots, a_k) x_1^{a_1} \cdots x_k^{a_k} \\
& + x_2 \sum_{a_1=0} \cdots \sum_{a_k=0} c(1 + a_1 + a_2, a_3, \ldots, a_k) x_1^{a_1} \cdots x_k^{a_k} \\
& - \frac{x_2}{1 - x_1} G(x_2, x_3, \ldots, x_k) + G(x_1, x_3, \ldots, x_k) \\
= \quad & \frac{x_2}{1 - x_1} G(x_1, x_2, \ldots, x_k) - \frac{x_2}{1 - x_1} G(x_2, x_3, \ldots, x_k) \\
& + G(x_1, x_3, \ldots, x_k) + H
\end{aligned}
$$

where

$$
\begin{aligned}
H \quad = \quad & x_2 \sum_{a_1=0} \cdots \sum_{a_k=0} c(1 + a_1 + a_2, a_3, \ldots, a_k) x_1^{a_1} \cdots x_k^{a_k} \\
= \quad & \sum_{a_1=0} \left(\frac{x_1}{x_2}\right)^{a_1} \sum_{a_2=0} \cdots \sum_{a_k=0} c(1 + a_1 + a_2, a_3, \ldots, a_k) x_2^{1+a_1+a_2} \cdots x_k^{a_k} \\
= \quad & \sum_{a_1=0} \left(\frac{x_1}{x_2}\right)^{a_1} \left(\sum_{a_2=0} \cdots \sum_{a_k=0} c(a_2, a_3, \ldots, a_k) x_2^{a_2} \cdots x_k^{a_k} \right. \\
& \left. - \sum_{a_3=0} \cdots \sum_{a_k=0} \sum_{a_2=0}^{a_1} c(a_2, a_3, \ldots, a_k) x_2^{a_2} \cdots x_k^{a_k} \right) \\
= \quad & \sum_{a_1=0} \left(\frac{x_1}{x_2}\right)^{a_1} G(x_2, \ldots, x_k) \\
& - \sum_{a_1=0} \left(\frac{x_1}{x_2}\right)^{a_1} \sum_{a_2=0}^{a_1} \sum_{a_3=0} \cdots \sum_{\alpha_k=0} c(a_2, a_3, \ldots, a_k) x_2^{a_2} \cdots x_k^{a_k} \\
= \quad & \frac{x_2}{x_2 - x_1} G(x_2, x_3, \ldots, x_k) \\
& - \sum_{a_2=0} \left(\sum_{a_1=a_2} \left(\frac{x_1}{x_2}\right)^{a_1} \right) \sum_{a_3=0} \cdots \sum_{a_k=0} c(a_2, a_3, \ldots, a_k) x_2^{a_2} \cdots x_k^{a_k}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{x_2}{x_2 - x_1} G(x_2, x_3, \ldots, x_k) \\
&\quad - \sum_{a_2=0} \sum_{a_3=0} \cdots \sum_{a_k=0} \frac{x_1^{a_2}}{x_2^{a_2-1}(x_2 - x_1)} c(a_2, a_3, \ldots, a_k) x_2^{a_2} \cdots x_k^{a_k} \\
&= \frac{x_2}{x_2 - x_1} \left(G(x_2, x_3, \ldots, x_k) - G(x_1, x_3, \ldots, x_k) \right)
\end{aligned}
$$

It follows that

$$
G(x_1, \ldots, x_k) = \frac{x_2(1 - x_2) G(x_2, x_3, \ldots, x_k) - x_1(1 - x_1) G(x_1, x_3, \ldots, x_k)}{(1 - x_1 - x_2)(x_2 - x_1)}
$$

as required.

To solve this recurrence and obtain an expression for $G(x_1, \ldots, x_k)$ we use the notation

$$y_i = x_i(1 - x_i).$$

Then the recurrence becomes:

$$G(x_1) = x_1 / y_1$$

$$G(x_1, \ldots, x_k) = \frac{1}{(y_1 - y_2)} \left(y_1 G(x_1, x_3, \ldots, x_k) - y_2 G(x_2 \ldots, x_k) \right).$$

Now let

$$A(z_1, \ldots, z_n) = \prod_{1 \le i < j \le n} (z_i - z_j),$$

for $n \geq 2$, and let $A(z) = 1$ and $A() = 1$. The following result was initially conjectured as a result of computational exploration with the AXIOM system [8].

Theorem 3

$$G(x_1, \ldots, x_k) = -\frac{\sum_{i=1}^{k} (-1)^i x_i y_i^{k-2} A(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k)}{A(y_1, \ldots, y_k)}$$

Proof by induction on k. When $k = 1$, the result can be checked directly. Otherwise assume the result holds for $k - 1$ and substitute that twice into the recurrence above. We get

$$
\begin{aligned}
&G(x_1, \ldots, x_k) \\
&= \frac{-1}{y_1 - y_2} \\
&\quad \left(\frac{y_1 \left(\begin{array}{l} -x_1 y_1^{k-3} A(y_3, \ldots, y_k) \\ + \sum_{i=3}^{n} (-1)^{i-1} x_i y_i^{k-3} A(y_1, y_3, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k) \end{array} \right)}{A(y_1, y_3, \ldots, y_k)} \right.
\end{aligned}
$$

$$- y_2 \frac{\left(\sum_{i=2}^{n} x_i y_i^{k-3}(-1)^{i-1} A(y_2, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k)\right)}{A(y_2, \ldots, y_k)} \Bigg) .$$

Placing this over a common denominator of $A(y_1, \ldots, y_k)$ the numerator becomes:

$$\begin{aligned}
& x_1 y_1^{k-2} A(y_3, \ldots, y_k) \prod_{j=3}^{k} (y_2 - y_j) \\
& + \sum_{i=3}^{k} (-1)^i x_i y_i^{k-3} y_1 A(y_1, y_3, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k) \prod_{j=3}^{k} (y_2 - y_j) \\
& - \sum_{i=2}^{k} (-1)^i x_i y_i^{k-3} y_2 A(y_2, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k) \prod_{j=3}^{k} (y_1 - y_j)
\end{aligned}$$

Collecting terms in each x_i and combining the products with the $A(\ldots)$ expressions this becomes:

$$\begin{aligned}
& x_1 y_1^{k-2} A(y_2, \ldots, y_k) - x_2 y_2^{k-2} A(y_1, y_3, \ldots, y_k) \\
& + \sum_{i=3}^{k} (-1)^i x_i y_i^{k-3} A(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k) \left(\frac{y_1(y_2 - y_i) - y_2(y_1 - y_i)}{y_1 - y_2} \right)
\end{aligned}$$

Finally, we note that

$$\frac{y_1(y_2 - y_i) - y_2(y_1 - y_i)}{y_1 - y_2} = -y_i$$

which gives the desired result.

Corollary 2 *$c(a_1, \ldots, a_k)$ is symmetric in $a_1, \ldots, a_k$*

Proof The denominator of the generating function $G(x_1, \ldots, x_k)$ is clearly anti-symmetric in its variables. Furthermore, it is clear that exchanging two variables x_i and x_{i+1} $(1 \leq i < k)$ in the numerator negates it, which implies that the numerator is also anti-symmetric. Thus the generating function is symmetric and the result now follows.

Remark: H.S. Wilf [13] has observed that both the numerator and the denominator of the expression for G can be viewed as determinants, giving

$$G(x_1,\ldots,x_k)=\frac{\begin{vmatrix} y_1^0 & y_1^1 & \cdots & y_1^{k-2} & x_1y_1^{k-2} \\ y_2^0 & y_2^1 & \cdots & y_2^{k-2} & x_2y_2^{k-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ y_k^0 & y_k^1 & \cdots & y_k^{k-2} & x_ky_k^{k-2} \end{vmatrix}}{\begin{vmatrix} y_1^0 & y_1^1 & \cdots & y_1^{k-1} \\ y_2^0 & y_2^1 & \cdots & y_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_k^0 & y_k^1 & \cdots & y_k^{k-1} \end{vmatrix}}$$

This is exactly the form given by Cramer's rule for determining the last unknown u_k in the system of linear equations (indexed by i):

$$\sum_{j=1}^{k} u_j y_i^{j-1} = x_i y_i^{k-2}$$

Letting

$$P(t)=\sum_{j=0}^{k-1} u_{k-j}t^j$$

we see that u_k is in fact the constant term in the unique polynomial $P(t)$ of degree $k-1$ defined by

$$P(1/y_i)=x_i/y_i$$

It is not known whether the other coefficients of this polynomial or the polynomial itself have any combinatorial meaning.

Acknowledgment We thank the referee for a number of suggestions which greatly improved the exposition of this paper.

References

[1] Atkinson M D and Thiyagarajah M: The permutational power of a priority queue, BIT 33 (1993), 2-6.

[2] Atkinson M D and Beals R: Priority queues and permutations, SIAM J. Computing 23 (1994), 1225–1230.

[3] Atkinson M D: Transforming binary sequences with priority queues, ORDER 10 (1993), 31-36.

[4] Atkinson M D and Walker L A: Enumerating k-way trees, Information Processing Letters 48 (1993), 73-75.

[5] Cameron P J: Combinatorics: Topics, Techniques, Algorithms, Cambridge University Press, (Cambridge, New York, Melbourne) 1994.

[6] Cormen T H, Leiserson C E, Rivest R L: Introduction to Algorithms, McGraw-Hill (Cambridge, Mass.) 1992

[7] Golin M, Zaks S: Labelled trees and pairs of input-output permutations in priority queues, Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), München, Germany, June 1994

[8] Jenks R D, Sutor R S: Axiom, the Scientific Computation System, Springer-Verlag, (New York), 1992

[9] Knuth D E: Fundamental Algorithms, The Art of Computer Programming Vol. 1, Addison-Wesley, (Reading, Massachusetts), 1973

[10] Knuth D E: Sorting and Searching, The Art of Computer Programming Vol. 3, Addison-Wesley, (Reading, Massachusetts), 1973

[11] Moon J W: Counting Labelled Trees, Canad. Math. Monographs No. 1 (1970).

[12] Pratt V R: Computing permutations with double-ended queues, parallel stacks and parallel queues, Proc. ACM Symp. Theory of Computing 5 (1973), 268–277.

[13] Wilf H S: private communication, 1995.