Part I

COMPUTER SCIENCE

BIT 33 (1993), 2–6.

THE PERMUTATIONAL POWER OF A PRIORITY QUEUE

M. D. ATKINSON* and MURALI THIYAGARAJAH

Department of Mathematical and Computational Sciences, North Haugh, St. Andrews, Fife KY16 9SS, Scotland

School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

Abstract.

A priority queue transforms an input permutation σ of some set of size n into an output permutation τ. It is shown that the number of such pairs (σ, τ) is $(n+1)^{n-1}$. Some related enumerative and algorithmic questions are also considered.

Keywords: Priority queue, permutation, enumeration.

CR Categories: E.1, G.2.1.

Priority Queues are abstract data types which support the operations: Insert, Delete-Minimum. They have many applications and several efficient implementations of them are known. In this paper we shall be concerned with the effect of a priority queue on the order of the data items that pass through it. Suppose that $\sigma_1, \sigma_2, \ldots, \sigma_n$ is some stream of input to a priority queue. Each Insert operation places the next item of the stream in the priority queue and each Delete-Minimum operation removes the current smallest element and places it in an output stream. After n Insert operations and n Delete-Minimum operations the input stream will be exhausted, the priority queue will be empty, and the output stream will contain some permutation of the input stream. The only restriction on a valid sequence of Insert and Delete-Minimum operations is that Delete-Minimum must not be applied to the priority queue if it is empty and hence a sequence of Inserts and Delete-Minimums must be *well-formed* in the sense of bracket sequences (in any initial segment there must always be at least as many Inserts as Delete-Minimums). There are therefore c_n valid sequences of n Insert operations and n Delete-Minimum operations where

* Supported by the National Science and Engineering Research Council of Canada under Grant A4219.

Received April 1992. Revised September 1992.

$$c_n = \binom{2n}{n} \Big/ (n+1)$$

is the nth Catalan number.

We shall consider input streams of distinct elements drawn from some totally ordered set which, for convenience, we may take to be the set of positive integers. Let σ be some input (a sequencing of the elements in some set C) which gives rise to some output sequence τ; then we shall call (σ, τ) an *allowable pair* on C. Our main result is

Theorem 1. *The number of allowable pairs on a set of size n is* $(n+1)^{n-1}$.

It is interesting to compare this result with the well-known situation where a stack is used in place of a priority queue (see [1] §5.1.4 for a full discussion of the combinatorics of this problem). In this case every sequence of inserts and deletes (pushes and pops) gives rise to a different permutation of the input, and the number of outputs for a given input is therefore c_n which is of the order 4^n. For priority queues there is considerable variability. If the input is the sequence $1, 2, \ldots, n$ only one output is possible. On the other hand, if the input sequence is $n, n-1, \ldots, 1$ every legal sequence of Inserts and Delete-Minimums gives rise to a different output, just as for stacks. It follows from our main result that the average number of outputs for a given input of length n is, by Stirling's formula, of the order e^n. These considerations suggest the notion of "permutational power" which occurs in the title of this paper. Priority queues do not have the same capability as stacks to produce permutations of their input, and the functions e^n and 4^n are illuminating measures which quantify this difference. We shall discuss the number of outputs for a given input below.

However, the proof of the theorem requires an understanding of the complementary problem: how many inputs can give rise to a given output? We are able to characterize these sequences as the set of linear extensions of a certain poset defined from the output. If π is a sequence we shall let $T(\pi)$ denote the set of all τ such that (π, τ) is allowable. Thus $T(\pi)$ is the set of sequences which can be output by a priority queue if the input stream is π. Let $S(\pi)$ be the set of all σ such that (σ, π) is allowable. This set is the set of input streams capable of generating π on the output stream. Moreover, let $t(\pi) = |T(\pi)|$ and $s(\pi) = |S(\pi)|$.

LEMMA 2. *Let* σ *be some input stream expressed in the form* $\alpha m \beta$ *where* m *is the maximal symbol. Suppose* $\beta = b\gamma$. *Then* $t(\sigma) = t(\alpha)t(\beta) + t(\alpha b m \gamma)$.

PROOF. Clearly, in any sequence of the set $T(\sigma)$, m must come after the symbols of α. There are $t(\alpha)t(\beta)$ sequences in $T(\sigma)$ arising from outputting m as the next symbol after all the symbols in α. Those sequences in $T(\sigma)$ for which m does not immediately succeed the symbols of α are precisely the outputs that arise if $\alpha b m \gamma$ is processed by a priority queue.

COROLLARY 3. *If* $\beta = b_1 b_2 \ldots b_r$ *and* $\alpha_i = \alpha b_1 \ldots b_i$, $\beta_i = b_{i+1} \ldots b_r$ *then*

$$t(\sigma) = \sum_{i=0}^{r} t(\alpha_i) t(\beta_i).$$

Suppose that $\tau = t_1 t_2 \ldots t_n$ is any sequencing of a set C. We define a partially ordered set $P(\tau) = (C, \prec)$ by the following set of constraints:

1. if $i < j$ and $t_i > t_j$ then $t_i \prec t_j$;
2. if $k < i < j$ and $t_i > t_j$ then $t_k \prec t_j$.

The relation $\prec$ is obviously irreflexive and transitive.

LEMMA 4. *Let* τ *be any sequencing of the elements of a set* C. *Then* $S(\tau)$ *is the set of linear extensions of the poset* $P(\tau)$. *Moreover, if* τ *is expressed in the form* $\alpha m \beta$, *where* m *is the maximal element of* C, $|\alpha| = k$, *then* $s(\tau) = s(\alpha)s(\beta)(k+1)$.

PROOF. Let $\sigma \in S(\tau)$. We shall prove that σ is a linear extension of $P(\tau)$. Let $y \prec z$ be one of the constraints of $P(\tau)$. We must show that y precedes z in its occurrence in σ. There are two cases:

1. y precedes z in τ and $y > z$, or
2. there exists a symbol x such that y, x, z occur in this order in τ and $x > z$.

In either case the supposition that z precedes y in σ leads to a contradiction. For, in order for the two symbols y, z to be transposed when processed by a priority queue, z must not be output until y (and all intervening symbols) have been placed in the priority queue. But, in the first case, the priority queue would then output z before y (because it is smaller) and, in the second case, the priority queue would output z before x.

To prove the converse, that every linear extension of $P(\tau)$ belongs to $S(\tau)$, we proceed by induction on $n = |C|$. We express τ as $\tau = \alpha m \beta$ where m is the maximal element of C and let σ be a linear extension of $P(\tau)$. Then, in σ, by the two conditions which define $P(\tau)$, we can deduce that m precedes every symbol of β and that every symbol of α precedes every symbol of β. Moreover, the symbols of α will be arranged in σ as some linear extension α^* of $P(\alpha)$ and the symbols of β will be arranged in σ as some linear extension β^* of $P(\beta)$ and m will occur somewhere among the symbols of α^*. Thus σ has the form $\sigma = \alpha_1^* m \alpha_2^* \beta^*$ where $\alpha_1^* \alpha_2^* = \alpha^*$. The inductive hypothesis guarantees that there are sequences of Insert and Delete-Minimum operations which transform an input stream α^* into α and transform an input stream β^* into β. It is now easy to see that there is a sequence of Insert and Delete-Minimum operations which, with σ as input, produces τ as output.

Finally, the description just given of the linear extensions of the poset $P(\alpha m \beta)$ proves the final statement of the lemma.

This lemma suggests a recursive algorithm for computing $s(\tau)$. However, it is more

illuminating to consider an iterative version of it. Put $\tau_0 = \infty$ and, for each $1 \leq i \leq n$, define

$$b(i) = \max_j \{j < i, \tau_j > \tau_i\}.$$

LEMMA 5.

$$s(\tau) = \prod_{i=1}^{n} (i - b(i)).$$

PROOF. Let $\tau = \tau^* \tau_n$. By induction on n there are $\prod_{i=1}^{n-1} (i - b(i))$ linear extensions of $P(\tau^*)$ (the values $b(1), \ldots, b(n-1)$ defined for τ are clearly the same for τ^*). Let $p = b(n)$. It follows from the definition of $b(n)$ that $\tau_i \prec \tau_j$ for all $1 \leq i \leq p$ and $p + 1 \leq j \leq n$. Thus there are exactly $n - p$ positions in every linear extension of $P(\tau^*)$ in which τ_n may be inserted to make a linear extension of $P(\tau)$. Hence $|S(\tau)| = |S(\tau^*)|(n - b(n))$ and the result follows.

Now we define a labelled binary tree $B(\tau)$. The root of $B(\tau)$ is labelled with (p, m), where m is the maximal element of τ and p is its position in τ, and if $\tau = \alpha m \beta$, the left and right subtrees are $B(\alpha)$ and $B(\beta)$ respectively. It is clear, again by induction, that if some node is labelled with the pair (i, τ_i) then $i - b(i)$ is the cardinal of the set of nodes in the subtree consisting of (i, τ_i) and all nodes in its left subtree.

The tree $B(\tau)$ can be constructed as a binary search tree by inserting the (position, value) pairs into an initially empty tree. The pairs are inserted in decreasing order of value but are keyed by position.

LEMMA 6. *There is an algorithm to compute $s(\tau)$ which, for random τ, has expected execution time $O(n \log n)$.*

PROOF. The algorithm is the one suggested above. The pairs (i, τ_i) are first sorted by second component and then inserted into an initially empty binary search tree thereby creating $B(\tau)$. Then the sizes of all subtrees are found and the product in the previous lemma is computed. The only part of this procedure which is not of time complexity $O(n \log n)$ is the creation of $B(\tau)$. However, it is well known that, if all input orders are equally likely, the expected height of a binary search tree is $O(\log n)$. Hence the expected time for creating $B(\tau)$ is $O(n \log n)$.

PROOF OF THEOREM 1. For any set X let $Sym(X)$ denote the set of all permutations of X. In proving the theorem we may suppose with no loss in generality that the n-element set in question is $C = \{1, 2, \ldots, n\}$. The theorem is clearly true when $n = 0$ and so we now take $n > 0$ and, as an inductive hypothesis, assume that the theorem is true for sets of size less than n.

The number of allowable pairs is $\sum_{\tau \in Sym(C)} s(\tau)$. We shall express each permutation τ in the form $\tau = \alpha n \beta$ and let A, B denote the sets of symbols occurring in α,

β respectively. Our sum can be expressed as a sum over the different possible subsets A (grouped according to size) where, for each possible A we sum over all $\alpha \in Sym(A)$ and all $\beta \in C\backslash\{n\}\backslash A = B$. It then becomes

$$\sum_{k=0}^{n-1} \sum_{A,|A|=k} \sum_{\alpha \in Sym(A)} \sum_{\beta \in Sym(B)} s(\alpha n \beta)$$

$$= \sum_{k=0}^{n-1} \sum_{A,|A|=k} \sum_{\alpha \in Sym(A)} \sum_{\beta \in Sym(B)} s(\alpha)s(\beta)(k+1)$$

$$= \sum_{k=0}^{n-1} \sum_{A,|A|=k} \sum_{\alpha \in Sym(A)} s(\alpha) \sum_{\beta \in Sym(B)} s(\beta)(k+1)$$

$$= \sum_{k=0}^{n-1} (k+1)\binom{n-1}{k}(k+1)^{k-1}(n-k)^{n-k-2}$$

and, by one of Abel's identities (see [2] §1.5), this is $(n+1)^{n-1}$.

REFERENCES

1. D. E. Knuth: *Sorting and Searching, The Art of Computer Programming Vol.* 3, Addison-Wesley, (Reading, Massachusetts), 1973.
2. J. Riordan: *An Introduction to Combinatorial Analysis*, Wiley (New York) 1968.