SIAM J. COMPUT.
Vol. 23, No. 6, pp. 1225–1230, December 1994

PRIORITY QUEUES AND PERMUTATIONS[*]

M. D. ATKINSON[†] AND ROBERT BEALS[‡]

Abstract. A priority queue transforms an input permutation σ of some set of size n into an output permutation τ. The set R_n of such related pairs (σ, τ) is studied. Efficient algorithms for determining $s(\tau) = |\sigma : (\sigma, \tau) \in R_n|$ and $t(\sigma) = |\tau : (\sigma, \tau) \in R_n|$ are given, a new proof that $|R_n| = (n+1)^{n-1}$ is given, and the transitive closure of R_n is found.

Key words. priority queue, permutation, enumeration

AMS subject classifications. primary 68R05; secondary 68P05, 05A15

1. Introduction. The topic studied in this paper was suggested by the beautiful and combinatorially rich theory of permutations that can be generated by a stack. In this theory an input sequence of n distinct elements, $x_1, x_2, \ldots, x_n$, is presented to a stack and is subjected to a series of *push* and *pop* operations; each push operation pushes the next input element on the stack and each pop operation removes the top element of the stack and places it in an output stream. When the entire input has been consumed and the stack is empty, the input sequence has been converted into an output sequence $x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}$ that is a permutation of the input sequence. The permutation π that arises in this way depends only on the series of push and pop operations and, in particular, is independent of the input sequence. It is well known that there are c_n such *stack permutations,* one for each push–pop sequence, where

$$c_n = \frac{\binom{2n}{n}}{n+1}$$

is the nth Catalan number. There are several interesting correspondences between stack permutations and other combinatorial objects (for example, binary trees, triangulations of a polygon, well-formed bracket sequences; see [3] and the references cited therein).

We shall investigate the analogous theory where a priority queue replaces a stack. In other words the pop operation "delete the most recently inserted" is replaced by "delete the smallest." We use the terms *Insert* and *Delete-Minimum* rather than push and pop and we shall call a series of n Inserts and n Delete-Minimums a *priority queue computation.* There are c_n different priority queue computations, each of which converts an input sequence (of n distinct elements) into an output sequence. But, unlike the case of stacks, it is no longer true that the permutational effect is independent of the input sequence nor that all inputs can be permuted in the same number of ways.

To handle this greater complexity we define a relation R_n called *allowability* on the set of sequences of length n by the rule $(\sigma, \tau) \in R_n$ if there exists a priority queue computation which transforms the input sequence σ into the output sequence τ. The elements of R_n will be called *allowable pairs.* It was shown in [1] that there are $(n+1)^{n-1}$ allowable pairs of permutations on a fixed set of size n and this result is a strong hint that an interesting combinatorial theory awaits investigation.

It is simple to recognize whether a pair of sequences (σ, τ) is allowable. One just constructs a suitable priority queue computation. If such a computation exists, it may not be unique, but it is easy to see that there is always a "natural" computation in which elements

[*]Received by the editors November 30, 1992; accepted for publication July 20, 1993.

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

[‡]Department of Computer and Information Science, University of Oregon, Eugene, Oregon 97403.

are output as soon as possible (that is, if x is the next element to output, and is present in the priority queue, then it should be output before further input elements are inserted). This observation results in the following algorithm:

```
$\mathcal{Q} :=$ empty priority queue
$i := 1$; $j := 1$
while $j \le n$ do
        while $\tau_j \notin \mathcal{Q}$ do
                insert($\sigma_i$)
                $i := i + 1$
        endwhile
        if min($\mathcal{Q}$) $\ne \tau_j$ then return (false) else
                deleteMin; $j := j + 1$
        endif
endwhile
return(true).
```

The time complexity of this algorithm depends, of course, on the implementation of the priority queue operations. With a heap-based implementation it would be $O(n \log n)$, whereas if we regarded Insert and Delete-Minimum as atomic constant time operations it would be $O(n)$.

In the next section of the paper we take up rather more interesting algorithmic questions by giving efficient methods for the two complementary problems:

(1) Given an output τ, find the number of allowable pairs (σ, τ).

(2) Given an input σ, find the number of allowable pairs (σ, τ).

Then, in §3 we concentrate more on combinatorial questions. We give a one-to-one correspondence between allowable pairs and labeled trees from which we obtain another proof of the main result of [1]. Next we give a result about the average number of outputs for a random priority queue computation. Finally we characterize the transitive closure of the allowability relation and discuss the connection with serial networks of priority queues.

2. The number of inputs and outputs.

In this section we consider the computation of the following two quantities: the number $s(\tau) = |\sigma : (\sigma, \tau) \in R_n|$ of allowable pairs having a fixed output τ, and the number $t(\sigma) = |\tau : (\sigma, \tau) \in R_n|$ of allowable pairs having a fixed input σ.

On the surface $t(\sigma)$ seems to be the more natural quantity to calculate since it enumerates the different ways that a priority queue can process a particular input, while $s(\tau)$ enumerates the different starting points that can give rise to some fixed result. Despite this, it turns out that the numbers $s(\tau)$ are rather easier to compute and have more obvious properties (for example, they are always divisors of $n!$). Indeed, in [1], an algorithm of average time complexity $O(n \log n)$ was given for computing $s(\tau)$. Here we shall give an algorithm for this problem whose worst case complexity is $O(n)$. We turn then to the problem of computing $t(\sigma)$. Although we have been unable to find a comparably efficient algorithm, we are at least able to place the problem in the complexity class $\mathcal{P}$ by giving a dynamic programming algorithm of time complexity $O(n^4)$.

For a given output sequence $\tau = [\tau_1, \ldots, \tau_n]$ we let τ_0 be any element greater than those occurring in the remainder of τ and define

$$b(i) = \max\{j : 1 \le j < i, \tau_j > \tau_i\}.$$

The following lemma was proved in [1].

LEMMA 2.1.

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

Thus, once $b(1), b(2), \dots, b(n)$ have been calculated, $s(\tau)$ can be found in $O(n)$ steps. The obvious algorithm for computing $b(1), b(2), \dots, b(n)$ is as follows.

```
$i := 0$
*repeat*
    $i := i + 1;\ j := i - 1$
    *while* $\tau_j < \tau_i$ *do* $j := j - 1$
    $b(i) := j$
*until* $i = n$.
```

It requires quadratic time in the worst case, since to compute $b(i)$ as many as i values of τ may need to be examined. A simple observation improves the computation time. Suppose that when computing $b(i)$, we tested the element τ_j (with $j < i$) and found that $\tau_j < \tau_i$. Then none of $\tau_{j-1}, \tau_{j-2}, \dots, \tau_{b(j)+1}$ need be compared with τ_i and the next test can be whether $\tau_{b(j)} < \tau_i$. In other words the statement $j := j - 1$ may be replaced by $j := b(j)$.

The new algorithm runs in linear time: the statement $j := b(j)$ cannot be executed more than once with the same value of j. Consequently $s(\tau)$ can be found from τ in $O(n)$ steps.

We now consider the second computation: finding $t(\sigma)$. The following result was proved in [1].

LEMMA 2.2. *Let σ be some input sequence expressed in the form $\alpha m \beta$ where m is the maximal symbol. Suppose $\beta = b_1 b_2 \cdots b_r$ and $\alpha_i = \alpha b_1 \cdots b_i$, $\beta_i = b_{i+1} \cdots b_r$, then*

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

We make this lemma the basis of a dynamic programming algorithm. For convenience assume that σ is a permutation of $\{1, 2, \dots, n\}$. Let $\sigma^{(m)}$ denote the permutation obtained from σ by deleting all the symbols $m + 1, \dots, n$ and let $\Sigma^{(m)}$ denote the set of substrings of $\sigma^{(m)}$.

We can compute $t(\theta)$ for all $\theta \in \Sigma^{(m)}$ for the values $m = 0, 1, 2, \dots, n$ in turn. The empty string λ is the only member of $\Sigma^{(0)}$ and $t(\lambda) = 1$. Suppose that $m \geq 1$ and $t(\theta)$ is known for all $\theta \in \Sigma^{(m-1)}$. A string ϕ of $\Sigma^{(m)}$ is either in $\Sigma^{(m-1)}$ (in which case $t(\phi)$ will be known) or it has the form $\alpha m \beta$. In the latter case we may compute $t(\alpha m \beta)$ in linear time using the formula of the previous lemma since all the strings required in the calculation belong to $\Sigma^{(m-1)}$. The total time required for the whole computation is

$$O\left(n \times \sum_{m=1}^{n} |\Sigma^{(m)}|\right) = O(n^4).$$

3. Combinatorial results. The number of allowable pairs was proved in [1] to be $(n + 1)^{n-1}$ using partially ordered sets and one of Abel's summation formulae. Here we give another proof which depends on establishing a 1-1 correspondence between allowable pairs and labeled trees.

Let (σ, τ) be an allowable pair on n symbols and let m be the maximal symbol. Suppose $\tau = \alpha m \beta$. At the point that m is output, the priority queue is empty. Therefore m and all the symbols of α occur in σ earlier than all the symbols of β. Let γ,δ be the symbols of α,

β, respectively, in order of their occurrence within σ. Then σ has the form $\gamma_{(i,m)}\delta$ where $\gamma_{(i,m)}$ denotes the result of inserting m within γ after the symbol i (i is given the conventional name "root" if m is inserted at the beginning of γ). Clearly (γ, α) and (δ, β) are allowable pairs.

Thus (σ, τ) has given rise to allowable pairs (γ, α), (δ, β) and a symbol i (one of the symbols of α or "root"). Conversely, (γ, α), (δ, β), and i define a unique allowable pair (σ, τ) by reversal of this construction.

We can now associate, with any allowable pair (σ, τ), a tree on $n + 1$ symbols ("root" and the n symbols being permuted). The construction is inductive. If $n = 0$ the tree is a single node called "root." If $n > 0$ we find (γ, α), (δ, β), and i as above. The trees for (γ, α), (δ, β) exist by induction and have roots "root1" and "root2." The tree for (σ, τ) is obtained by letting "root1" be the new "root," "root2" be labeled as m, and joining nodes i and m.

The construction is obviously reversible since the parent of the maximal node in a tree defines i and removal of the branch between i and m defines the trees for the allowable pairs (γ, α), (δ, β).

THEOREM 3.1 (Atkinson–Thiyagarajah [1]). *The number of allowable pairs on a set of size n is $(n + 1)^{n-1}$.*

Proof. By Cayley's theorem the number of labeled unrooted trees on $n + 1$ nodes is $(n + 1)^{n-1}$. The trees defined above are rooted (which would increase their number by a factor $n + 1$) but the root is labeled with a fixed symbol "root" (which decreases their number by a factor $n + 1$). The result now follows from the correspondence above. □

We now consider the number of outputs that a fixed priority queue computation can produce as the input varies over all permutations of an n-element set. Clearly, there is considerable variation. If we let $\mathcal{I}$ and $\mathcal{D}$ denote the operations Insert and Delete-Minimum, respectively, then priority queue computations can be represented by "well-formed" words in these two symbols. For example, the computation $(\mathcal{I}\mathcal{D})^n$ simply copies the input to the output so all $n!$ outputs are achievable. On the other hand, $\mathcal{I}^n\mathcal{D}^n$ only produces that output whose elements are in ascending order. For notational simplicity we shall assume that all inputs and outputs are permutations of $\{1, 2, \dots, n\}$. Let $k(w)$ denote the number of outputs that can be generated by the well-formed word w.

LEMMA 3.2.

(1) *If u, v are well formed of length $2a$, $2b$, then*

$$k(uv) = k(u)k(v)\binom{a+b}{a}.$$

(2) *If u is well formed, then $k(\mathcal{I}u\mathcal{D}) = k(u)$.*

Proof. First, consider $k(uv)$. The priority queue in the computation uv is empty just before the $(a + 1)$th symbol of the input is read. So the first a symbols of the output are a permutation of the first a symbols of the input. The first a symbols may be chosen in $\binom{a+b}{a}$ ways and, once chosen, may be permuted by u in $k(u)$ ways, after which the remaining b symbols may be permuted in $k(v)$ ways. The first part now follows.

Now consider $k(\mathcal{I}u\mathcal{D})$. Note first that if τ is an output of the priority queue computation u (arising, say, from the input σ) then τn is an output from the computation $\mathcal{I}u\mathcal{D}$ (clearly, it is the result of processing the input $n\sigma$). On the other hand, any output from $\mathcal{I}u\mathcal{D}$ must have the form τn for some τ. This is because, for the computation $\mathcal{I}u\mathcal{D}$, the priority queue cannot become empty at any intermediate step and so n cannot be deleted from it as smallest element until the end of the computation. To complete the proof, it is only necessary to prove that τ is an output of the priority queue computation u. Let ρ be an input to the computation $\mathcal{I}u\mathcal{D}$ which gives the output τn. If n is not the first symbol of ρ then we can write $\rho = \alpha m \beta n \gamma$, where m is the maximum symbol that precedes n in ρ. When this input is processed by $\mathcal{I}u\mathcal{D}$, the symbol m will remain in the priority queue after it is inserted at least until n is inserted (otherwise the priority queue would become empty before the computation terminates). It therefore follows that the input $\alpha n \beta m \gamma$ will result in the same output as ρ. By repetition of this principle, we obtain an input of the form $n\sigma$ which generates the output τn. It is apparent that the priority queue computation u transforms σ into τ as required. □

COROLLARY 3.3. *For every priority queue computation u, $k(u)$ divides $n!$.*

We have already remarked that $k(u)$ varies considerably as u varies over the possible priority queue computations. Despite this, it is possible to compute its average value defined as

$$k_n = \sum_u k(u)/c_n,$$

where the summation is over all priority queue computations with n Inserts and n Delete-Minimums.

THEOREM 3.4. $k_n = (n+1)!/2^n$.

Proof. Let $l_n = \sum_u k(u)$ so that $k_n = l_n/c_n$. Every priority queue computation can be expressed in the form $u = \mathcal{I}v\mathcal{D}w$, where v, w are themselves priority queue computations and are uniquely determined by u. Since the length of v can be any even integer from 0 to $2n-2$ we have

$$l_n = \sum_u k(u) = \sum_{i=0}^{i=n-1} \sum_v \sum_w k(\mathcal{I}v\mathcal{D}w),$$

where the second and third summations are over all priority queue computations v, w which process inputs of length i, $n-i-1$, respectively. Thus, by the previous lemma,

$$l_n = \sum_{i=0}^{i=n-1} \sum_v \sum_w k(v)k(w)\binom{n}{i+1} = \sum_{i=0}^{i=n-1} l_i l_{n-i-1}\binom{n}{i+1}.$$

Put $h_n = l_n/n!$. Then $h_n = \sum_{i=0}^{i=n-1} h_i h_{n-i-1}/(i+1)$. This recurrence can be solved by introducing the generating function

$$H(x) = \sum_{r=0}^{r=\infty} h_r x^r$$

which is easily seen to satisfy the integral equation

$$H(x)\int H(x)dx = H(x) - 1,$$

from which it follows that $H(x) = 1/\sqrt{1-2x}$ and the result follows by expanding the generating function. □

Finally we consider the transitive closure of the allowability relation R_n. One motivation for this is Tarjan's paper [5] where the permutations obtainable from a series network of stacks are considered. In a related work, Pratt [4] considers permutations computable by double-ended queues, parallel stacks, and parallel queues. The corresponding scenario for us is a series network of priority queues $P_1, P_2, \ldots, P_k$ on which the following operations are allowed:

(1) Insert, which transfers the next element of the input sequence into P_1,

(2) Move(i) with $1 \le i < k$, which transfers the minimal element of P_i into P_{i+1},

(3) Delete-Minimum, which appends the minimal element of P_k onto the output sequence.

Let ${R_n}^k$ denote the k-fold composition of R_n with itself. Then it is easy to verify that ${R_n}^k$ is precisely the set of (input, output) pairs associated with a series network of k priority queues.

The *weak order* W_n on the set of all permutations on n symbols is defined by $(\sigma, \tau) \in W_n$ if and only if every pair of symbols of σ, which are in increasing order, are also in increasing order in τ. The weak order is an important tool for the study of geometric and combinatorial properties of the symmetric group; its properties are discussed at length in [2]. It is clear that $R_n \subseteq W_n$ and, since W_n is transitively closed, ${R_n}^k \subseteq W_n$ for all k.

THEOREM 3.5. ${R_n}^{n-2} \neq {R_n}^{n-1} = W_n$.

Proof. For $(\sigma, \tau) \in W_n$ let $\rho(\sigma, \tau)$ be the smallest integer k such that all but the leftmost k symbols of σ and τ agree (that is, the rightmost $n - k$ symbols agree). Observe that ρ is never equal to 1. In order to show that ${R_n}^{n-1} = W_n$ it suffices to show that for any $(\sigma, \tau) \in W_n$ with $\sigma \neq \tau$, there exists a σ' such that all of the following hold:

(1) $(\sigma, \sigma') \in R_n$,

(2) $(\sigma', \tau) \in W_n$,

(3) $\rho(\sigma', \tau) < \rho(\sigma, \tau)$.

To see this, write σ as $\alpha x \beta \gamma$ and τ as $\delta x \gamma$ such that $|\gamma| = n - \rho(\sigma, \delta)$. Since $(\sigma, \tau) \in W_n$, it follows that x must be larger than all symbols in β. Therefore, $\sigma' = \alpha\beta x\gamma$ satisfies condition (1) above (in fact a priority queue can input σ and output σ' by only placing two symbols, x and the current input symbol, in the queue at one time). Also, σ' satisfies condition (2) because any pair of symbols in σ' appear either in the same order as in σ, or the same order as in τ. Finally, since σ' and τ both end with $x\gamma$, condition (3) is satisfied. Since ρ cannot have the value 1, the sequence $\sigma', \sigma'', \ldots$ must reach τ in at most $n - 1$ steps, proving ${R_n}^{n-1} = W_n$.

To complete the proof, we prove that the pair

$$([n, n-1, \ldots, 2, 1], [1, n, n-1, \ldots, 3, 2]),$$

which plainly lies in W_n, does not belong to ${R_n}^{n-2}$. Suppose that it were possible to transform $[n, n-1, \ldots, 2, 1]$ into $[1, n, n-1, \ldots, 3, 2]$ by a series network of $n - 2$ priority queues. When the final element of the input (symbol 1) has been placed in priority queue P_1 the other $n - 1$ symbols must all be in the $n - 2$ priority queues of the network since none can be output yet. One of the priority queues must therefore contain at least two of the symbols in $\{2, 3, \ldots, n\}$. Clearly this is impossible, since these two symbols would then have to be output eventually in increasing order. □

Therefore, W_n describes the relation between input and output permutations for a series network of k priority queues for any $k \geq n - 1$.

Acknowledgment. We thank Katherine Anderson and Murali Thiyagarajah for several useful conversations during the course of this work, and Gerald Ostheimer for the insight leading to the $O(n)$ algorithm for computing $s(\tau)$.

REFERENCES

[1] M. D. ATKINSON AND M. THIYAGARAJAH, *The permutational power of a priority queue*, BIT, 33 (1993), pp. 2–6.

[2] A. BJÖRNER, *Orderings on coxeter groups*, in Proceedings of Conference on Combinatorics and Algebra, American Mathematical Society, Providence, RI, 1983.

[3] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 1992.

[4] V. PRATT, *Computing permutations with double-ended queues, parallel stacks, and parallel queues*, in Proceedings of the 5th ACM STOC, ACM Press, New York, 1973, pp. 268–277.

[5] R. E. TARJAN, *Sorting using networks of queues and stacks*, J. ACM, 19 (1972), pp. 341–346.