Ballot Sequences and Restricted Permutations

Dana Richards
University of Virginia

Computer Science Report No. TR-86-15
June 11, 1986

Ballot Sequences and Restricted Permutations

Dana Richards

University of Virginia
Charlottesville, VA 22903

1. Introduction

The distribution of the length of the longest ascending subsequence of a permutation of $\{1,2,\cdots,n\}$, $\pi=(\pi_0,\pi_1,\ldots,\pi_{n-1})$, has been much studied (e.g., [PILP86, SCHE61]). An ascending subsequence is $\pi_{i_1}<\pi_{i_2}<\cdots<\pi_{i_k}$, where $i_1<i_2<\cdots<i_k$, and the length of the subsequence is k. The principal result in this area is that the expected length of the longest subsequence is $2\sqrt{n}$, over all permutations [PILP86]. Another intriguing result concerns the number, $p(n,l)$, of permutations with no ascending subsequence of length greater than l. Let the set of all such permutations be $P(n,l)$. It is known [ROGE78] that

$$p(n,2)=\frac{1}{n+1}\binom{2n}{n}$$

which is a Catalan number. Of course $p(n,1)=1$. The appearance of the Catalan number reveals an association with a great number of other well known combinatorial problems in which the Catalan numbers play a role. Often these problems are related by explicit bijections between their domains. Rogers [ROGE78] states that a "direct proof would be welcome as it might suggest other ways of calculating" $p(n,l)$ and related quantities. In this note we give a direct proof. In the sequel if we refer to a permutation we assume it is in $P(n,2)$, unless otherwise specified.

Of all the combinatorial objects counted by the Catalan numbers perhaps the canonical example is the set of ballot sequences $B(n)$ (e.g., [YAGL64]). A ballot sequence $B=(B_0,B_1,\cdots,B_{2n-1})$ is a sequence of n 0's and n 1's such that, left to right, the 1's are never outnumbered by the 0's, that is $\sum_{i=0}^{k}B_i \geqslant \lceil k/2\rceil$. Many techniques are known for ranking and unranking ballot sequences (e.g., [ZAKS79]) and, given a bijection between $B(n)$ and $P(n,2)$, these provide a way to rank and unrank the permutations of $P(n,2)$. We now exhibit such a bijection.

2. Mapping $B(n)$ into $P(n,2)$

Consider a permutation π from $P(n,2)$ and suppose that $\pi_i = 1$. It is clear that $\pi_{i+1} > \pi_{i+2} > \cdots > \pi_{n-1}$. In particular if $\pi_j = 2$ and $j > i$ then $j = n-1$, if $j < i$ then there is no restriction.

This observation leads to a simple recursive procedure for generating all the permuations with $\pi_i = 1$. First produce all those π' in $P(n-1,2)$ such that $\pi'_j = 1$ and $j \leqslant i$. Increment each entry of π' by one and insert a 1 before the ith entry of π' or at the end if $i = n-1$. If π'_i was 1 it now is a 2 behind a 1; that 2 must be moved to the end to form a valid permutation. This provides the motivation for our algorithm.

The algorithm *B-to-P* in figure 1 accepts a valid ballot sequence B and produces a permutation π with an array $P[0 \,..\, n-1]$ such that $P_i = \pi_i$.

Example: Given $B = (1100110100)$ the algorithm begins with $P = [0,0,0,0,0]$. The cursor c moves two positions left from the right end since $B_0 = B_1 = 1$; $B_2 = 0$ indicates that we should stop and a number should be entered at position c. Now $B_3 = 0$ indicates no left movement should occur and we should find the rightmost empty position, using the r cursor, and enter a number there, and so on. Note that it is an invariant, after the initial steps, that $c < r$, as shown in the proof below. The final permutation is given by $P = [4,3,5,1,2]$. □

Proposition: Algorithm *B-to-P* is a one-to-one mapping from $B(n)$ to $P(n,2)$.

Proof: By inspection, the algorithm when given two different ballot sequences will produce different outputs as soon as the sequences differ. What needs to be shown is that it indeed produces a permutation and it is in $P(n,2)$.

```
$P \leftarrow [0,0,\cdots,0]$
$j \leftarrow 0$
$r \leftarrow c \leftarrow n$
for $i \leftarrow 1$ to $n$ do
    if $B_j = 1$ then
        repeat
            $c \leftarrow c-1$
            $j \leftarrow j+1$
        until $B_j = 0$
        $P_c \leftarrow i$
    else
        repeat
            $r \leftarrow r-1$
        until $P_r = 0$
        $P_r \leftarrow i$
    endif
    $j \leftarrow j+1$
endfor
```

Algorithm *B-to-P*

Figure 1

Informally, the algorithm has a cursor c which only moves right to left. It has moved k positions after k 1's have been processed from B, which is read left to right. Every time a 0 is encountered an entry is made in P. By definition, no more than k 0's have been seen. Therefore there is always a position in P "behind" the cursor c that is still 0 where the entry can be made. Hence P is a permutation.

Suppose there is an ascending subsequence $a < b < c$ in P. After a is placed in P the only place b can appear, after a has been placed, is in the rightmost available position; therefore c cannot later be placed to the right of b. $\square$

Algorithm *B-to-P* runs in $O(n)$ time, since every **repeat** loop iteration moves a cursor relentlessly to the left.

3. Mapping $P(n,2)$ into $B(n)$

Algorithm *P-to-B* in figure 2 accepts a valid permutation from $P(n,2)$ and produces a ballot sequence. Logically its operation is just the inverse of algorithm *B*-

to-P. It searches, right to left, for the position of the 1, that is i where $\pi_i = 1$. It remembers which elements it has scanned over with the array $seen[1..n]$. If at some point the next larger element of the permutation has been scanned then a 0 is added to the ballot sequence since in algorithm *B-to-P* that would be an "instruction" to go to the right of the cursor. Otherwise, if it has not been scanned, 1's are added to the sequence since algorithm *B-to-P* would need to find 1's in the ballot sequence to move it to the left.

Example: Given the permutation $\pi = (5,3,1,4,2)$ the algorithm begins B with 1110 after finding the 1 in π. Since the 2 has been seen it appends a 0. Scanning for the 3 adds a 10 and the 4 appends a 0 since it has been seen. Finally the 5 adds a 10 giving $B = (1110010010)$ in this case. □

Proposition: Algorithm *P-to-B* is a one-to-one mapping from $P(n,2)$ to $B(n)$.

Proof: First we show that the output is a ballot sequence. There are clearly n 0's in B. Further, since every number has to be "seen", it follows that n 1's are inserted into B. Note that a 0 is inserted only for a number that has been seen

```
seen ← [0,0, ··· ,0]
c ← n
j ← 0
for i ← 1 to n do
    if seen_i = 1 then
        B_j ← 0
    else
        repeat
            c ← c − 1
            B_j ← 1
            j ← j + 1
            seen_{π_c} ← 1
        until π_c = i
        B_j ← 0
    endif
    j ← j + 1
endfor
```

Algorithm *P-to-B*

Figure 2

and when it was seen a 1 was inserted, before the 0, in B. Therefore it follows that, left to right, the 0's never outnumber the 1's. (The above argument is true for any permutation at all, therefore many permutations can map to the same ballot sequence.)

We need to show that no two permutations from $P(n,2)$ map to the same sequence. Suppose π and π' do, and further i is the least element such that $\pi_j = i$, $\pi'_k = i$, and $j \neq k$. When the algorithm processed i clearly i had been "seen", otherwise a different number of 1's would have been inserted into B during the scan for i. Let $\pi_l = i-1$ and w.l.o.g. $j < k$. Note that $\pi_k > i$ since the kth position cannot contain a smaller number, otherwise π' would not have been able to put i there. Therefore $\pi_l < \pi_j < \pi_k$ which gives a contradiction. □

Algorithm *P-to-B* runs in $O(n)$ time. It is not clear how to extend these techniques to $P(n,3)$ and further.

4. References

[PILP86] S. Pilpel, Descending Subsequences of Random Permutations, Report RC 11634, IBM T. J. Watson Reseach Center, 1986.

[ROGE78] D. G. Rogers, Ascending Sequences in Permutations, *Discrete Math*, **22**, 1978, pp. 35-40.

[SCHE61] C. Schensted, Longest Increasing and Decreasing Subsequences, *Canadian J Math*, **13**, 1961, pp. 179-191.

[YAGL64] A. M. Yaglom and I. M. Yaglom, *Challenging Problems in Mathematics with Elementary Solutions*, Holden-Day, San Francisco, 1964.

[ZAKS79] S. Zaks and D. Richards, Generating Trees and Other Combinaorial Objects Lexicographically, *SIAM J Computing*, **8**, 1979, pp. 73-81.