University of Houston

Programming Assignment 1

COSC 3320
Algorithms and Data Structures

Gopal Pandurangan

Solution

1 Pseudocode and Explanation

Suppose we have the permutations as desired for some n, i.e., we have the permutations of $P_n = \{a_1, a_2, \dots a_n\}$ so that any consecutive permutations differ by a single swap (we will call this *the swap requirement*). Now, pick a single permutation, say $[a_1, a_2, \dots, a_n]$, and consider how we can create $n+1$ new permutations by inserting a new element (which we will denote by N):

$$\begin{gathered}[N, a_1, a_2, \dots, a_n] \\ [a_1, N, a_2, \dots, a_n] \\ [a_1, a_2, N, \dots, a_n] \\ \vdots \\ [a_1, a_2, \dots, N, a_n] \\ [a_1, a_2, \dots, a_n, N]\end{gathered}$$

Now, these permutations clearly satisfy the swap requirement. Consider the next permutation, σ, of P_n. By construction, it must satsify the swap requirement on $\{a_1, a_2, \dots a_n\}$. In particular, then, ignoring the last element N, it differs by the last permutation above by a single swap. Then the permutation $\sigma + [N]$, i.e., the permutation σ with N appended to the end, differs from the above permutation by a single swap. For example, supposing $\sigma = [a_1, a_2, \dots, a_n, a_{n-1}]$, the permutation $[a_1, a_2, \dots, a_n, a_{n-1}, N]$ differs by a single swap from $[a_1, a_2, \dots, a_n, N]$. We then repeat the above process in the opposite direction:

$$\begin{gathered}[a_1, a_2, \dots, a_n, a_{n-1}, N] \\ [a_1, a_2, \dots, a_n, N, a_{n-1}] \\ [a_1, a_2, \dots, N, a_n, a_{n-1}] \\ \vdots \\ [a_1, a_2, N, \dots, a_n, a_{n-1}] \\ [a_1, N, a_2, \dots, a_n, a_{n-1}] \\ [N, a_1, a_2, \dots, a_n, a_{n-1}]\end{gathered}$$

Simply repeat for all permutations $\sigma \in P_n$, changing directions between iterations. This admits the following pseudocode.

Algorithm 1 Permutations – Permutations of an Array

```
1:  def Permutations(A):
        Input  ▷ An array A of distinct elements.
        Output ▷ All n! permutations of A such that consecutive permutations differ by a single swap.
2:      n ← |A|
3:      if n = 1:                                                          ▷ Base Case
4:          return A
5:      else:                                                              ▷ Recursive Step
6:          perms ← []
7:          A' ← A[: n]                          ▷ A' is the subarray containing the first n − 1 elements
8:          a ← A[n − 1]
9:          StartFromLeft ← True
10:         for σ ∈ Permutations(A'):
11:             perms.Append(Insertions(a, σ, StartFromLeft))              ▷ add insertions into perms
12:             StartFromLeft ← ¬StartFromLeft
13:         return perms
```

Algorithm 2 Insertions – Function to insert element into array

```
1: def Insertions(a, A, StartFromLeft):
       Input  ▷ Value a to be inserted into array A and boolean whether to start from left or right
       Output ▷ List of arrays so that a is inserted
2:     n ← |A|
3:     output ← []
4:     if StartFromLeft:
5:         range ← [0, 1, ..., n]
6:     else:
7:         range ← [n, n − 1, ..., 0]
8:     for i ∈ range:
9:         output.Append(A[: i] + [a] + A[i :])
10:    return output
```

2 Analysis

To see that the above algorithm is correct, first note that two permutations, σ_1 and σ_2, differ by a single swap if and only if they are identical at all but 2 indices. We first prove the following lemma:

Lemma 1

Fix some permutation and consider the insertions, as in Insertions, of any element. This set of insertions satisfies the swap requirement.

Proof. We prove this for left insertions. Consider some element of the set of insertions,

$$A_i = [a_1, a_2, \ldots, a_{i-1}, x, a_i, \ldots, a_n]$$

where x is the inserted element. Then the next element is

$$A_{i+1} = [a_1, a_2, \ldots, a_{i-1}, a_i, x, a_{i+1}, \ldots, a_n]$$

by construction. A_i and A_{i+1} agree on the first $i-1$ values and the last $n-i$ values, with the differences being $A_i[i] = x = A_{i+1}[i+1]$ and $A_i[i+1] = a_i = A_{i+1}[i+2]$. Thus, they differ at exactly two indices, and must differ by a single swap.

The argument for right insertions is nearly identical and we leave it to the reader. □

Next, we prove that Permutations satsifies the swap requirement.

Theorem 1

Algorithm Permutations outputs permutations that satisfy the swap requirements.

Proof. Without loss of generality, suppose the input array of length k is $A_k = [1, 2, \ldots, k]$. Proceed by induction on the length of the input array. The base case is trivially true. Suppose the algorithm is correct on an array of $A_{n-1} = [1, 2, \ldots, n-1]$. Say $P = [\sigma_1, \sigma_2, \ldots, \sigma_{(n-1)!}]$ are our permutations on A_{n-1}. By our induction hypothesis, permutation σ_i and σ_{i+1} differ by a single swap.

Now, fix a permutation $\sigma_i = [a_1, a_2, \ldots, a_{n-1}]$ as in the algorithm. We append the insertions of n into σ_i to our output and, by Lemma 1, these permutations satisfy the swap requirement. In particular, *every* collection of these consecutive $n-1$ permutations (which we will refer to as a group) satisfy the swap requirement. Thus, we need only show that the last permutation of group i and the first permutation of group $i+1$ satisfy the swap requirement.

Suppose `StartFromLeft` is `True`. Then the last permutation of group i is of the form

$$A = [\sigma_i[0], \sigma_i[1], \ldots, \sigma_i[n-2], n]$$

and the first permutation of group $i+1$ is of the form

$$B = [\sigma_{i+1}[0], \sigma_{i+1}[1], \ldots, \sigma_{i+1}[n-2], n]$$

Clearly, these two permutations agree on the final element. Additionally, by our induction hypothesis, the subarrays σ_i and σ_{i+1} differ by exactly two elements, hence these two permutations A and B similarly differ by exactly two elements, and satisfy the swap requirement.

A similar argument applies when `StartFromLeft` is `False`. Thus, these permutations satisfy the swap requirement. □

However, we have not yet fully demonstrated correctness. The astute reader might ask if we have repeated any permutations. We will show that this is impossible:

Theorem 2

Algorithm Permutations outputs all $n!$ distinct permutations.

Proof. Proceed by induction. The base case is clear. Suppose that Permutations outputs all distinct $(n-1)!$ permutations on $A_{n-1} = [1, 2, \ldots, n-1]$. Clearly, our algorithm outputs n permutations for each permutation on A_{n-1}, for a total of $n!$. We will show that these are distinct.

Consider, two permutations σ_i and σ_j on $[1, 2, \ldots, n]$, such that $\sigma_i = \sigma_j$. We will show that this forces $i = j$, which completes the proof. Since $\sigma_i = \sigma_j$, these permutations must have n in the same location, i.e.,

$$\sigma_i = [a_1, a_2, \ldots, n, \ldots, a_{n-1}]$$
$$\sigma_j = [b_1, b_2, \ldots, n, \ldots, b_{n-1}]$$

Now, consider the permutations on A_{n-1}:

$$\sigma'_i = [a_1, a_2, \ldots, a_{n-1}]$$
$$\sigma'_j = [b_1, b_2, \ldots, b_{n-1}]$$

By our induction hypothesis, the permutations on A_{n-1} are distinct, hence $i' = j'$. Then σ_i and σ_j belong to the same group. However, the set of permutations in a group are clearly all distinct. This forces $i = j$. □

To determine the runtime of Permutations, we must first determine the runtime of Insertions. There are $n+1$ iterations of the process on line 9, each of which requires $\mathcal{O}(n)$ time (either you create the array each time and point to it, or you perform a single swap and copy the contents into `output`$[i]$, a linear time operation in either case). In total, this is $\mathcal{O}(n^2)$.

With this, we can analyze the runtime:

Theorem 3

The runtime of algorithm Permutations is given by

$$T(n) = nT(n-1) + \mathcal{O}(n^2)$$

which has a runtime of $\mathcal{O}((n+1)!)$.

Proof. Let $T(n)$ denote the runtime of Permutations. Since A' in line 7 has length $n-1$, the call to Permutations on line 10 has runtime $T(n-1)$. Additionally, for each call, we call Insertions on an array of length $n-1$, which has runtime $\mathcal{O}(n^2)$, hence our recurrence is given by

$$T(n) = nT(n-1) + \mathcal{O}(n^2)$$

We will show that this is $\mathcal{O}((n+1)!)$ by induction. The base case is clearly constant time. Suppose that $T(n-1) \leq cn!$. Then

$$\begin{aligned}
T(n) &= nT(n-1) + \mathcal{O}(n^2) \\
&\leq ncn! + \mathcal{O}(n^2) \text{by the induction hypothesis} \\
&\leq ncn! + \alpha n^2 \text{ for some } \alpha > 0 \text{ by definition of Big-}\mathcal{O} \\
&= c(n+1-1)n! + \alpha n^2 \\
&= c(n+1)! - cn! + \alpha n^2 \\
&\leq c(n+1)! \text{ for sufficiently large } c
\end{aligned}$$

□