Design and Analysis of Algorithms ASSIGNMENT – 1

Group – 7
Medha Balani (IIT2019021)
Saksham Aggarwal (IIT2019022)
Utkarsh Gangwar (IIT2019023)

Problem Statement :

Generate all sequences of length 1 to 11. Detect and print sorted sequences. Range of numbers in a sequence should be the length of sequence like for sequence of length 4 numbers should be 1, 2, 3, and 4.

Formally , we have to generate all permutations of integers from 1 to 11 and check the sorted permutations amongst them and then finally print those 11 sorted permutations.

What is a permutation ?

A permutation of length n is a sequence **containing all the integers from 1 to n exactly once** in no specific order.

For Example : [1,2,3] , [3,1,2] , [2,1,3] are permutations but [2,2,1] or [1,2,4] are not.

Permutations of length n :

Let us understand this through simple combinatorics. Let there be ‘n’ blank spaces to be occupied by n distinct integers. The first integer will have n choices so it will occupy any one leaving (n-1) choices for the next. This goes on (n-2) (n-3) ...2 1 .

So, total permutations = n*(n-1)*(n-2)....*2*1 = n! i.e factorial n.

Proposed Algorithm :

Step 1 : First we will randomly select the length of the permutation which has not been already selected. (Let us suppose length to be n)

Step 2 : We will iterate over all the possible permutations of the selected length which is n! , we will calculate n! Before iterating.

Step 3 : For each permutation , we will check if that sequence is sorted or not . If it is sorted , we will print that permutation. Else, keep iterating over the rest permutations.

Step 4 : Mark this selected length as selected and repeat from step 1.

Algorithm to generate all permutations of length n:

We start with a permutation say [1,2,3...n] and using this current permutation, we keep generating next permutation till n! .

Let s be the current permutation

Step 1 : Iterate from back and find highest index i such that s[i]<s[i+1].

Step 2 : Again iterate from back and find highest index j such that , s[j] > s[i].

Step 3 : Swap s[i] and s[j].

Step 4 : Reverse the sequence from (i+1)th index to last index.

Idea behind the next permutation algorithm :

Here, we find next permutation in a lexicographically increasing order. And any sequence in descending order does not have a next permutation so we just reverse it.

So, to do this, first we find the first element from the last which is not following the descending order and let it be x at index i, for example in 1 2 4 3 , it is 2. Then we find the smallest greater element to x found element from right side , and then we swap the two.

Continue...

After this, observe that, after i, everything is in descending order, so to find lexicographically smallest, we reverse the sequence after i.

Idea is to find smallest next permutation which is greater then the current permutation.

If no i can be found in the first step then we reverse it completely. Let us understand with an example:

Let us begin with permutation, $\boxed{\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \end{array}}$.

Here, $i = 2$ and $j = 3$ so we obtain

1 2 4 3, reversing won't bring any change

so, next permutation $\rightarrow$ $\boxed{1\ 2\ 4\ 3}$.

$$1\ \underset{i}{\textcircled{2}}\ 4\ \underset{j}{\textcircled{3}} \xrightarrow{\text{swap}} 1\ 3\ \utilde{4\ 2} \xrightarrow{\text{reverse}} \underset{\text{next permutat}^n.}{\boxed{1\ 3\ 2\ 4}}$$

$$1\ 3\ \underset{i}{\textcircled{2}}\ \underset{j}{\textcircled{4}} \xrightarrow{\text{swap}} 1\ 3\ 4\ \utilde{2} \xrightarrow{\text{reverse}} \underset{\text{next permutat}^n.}{\boxed{1\ 3\ 4\ 2}}$$

$$1\ \underset{i}{\textcircled{3}}\ \underset{j}{\textcircled{4}}\ 2 \xrightarrow{\text{swap}} 1\ 4\ \utilde{3\ 2} \xrightarrow{\text{reverse}} \underset{\text{next permutat}^n.}{\boxed{1\ 4\ 2\ 3}}$$

This goes on till we obtain 4 3 21, which is the last permutation.

Possible Optimization in next permutation algo :

In step 2, we are finding smallest greater element to the right of i and as it is sorted in descending order, we can apply binary search in place of linear search which takes O(n) in worst case whereas binary search would take O(log n).

Algorithm used to check if current permutation is sorted or not :

A single traversal from left to right in the sequence , if there exist no element such that **Sequence[i] > Sequence[i+1]** ,then the permutation is sorted , else not.

Time Complexity Analysis

Firstly, iterating over all the different lengths (here it is 11) -> **O(n)**

Then, we iterate over all permutations of a particular length which gives **O(n!) in worst case and Ω(1) in best case**, inside which we do two things, check if sorted and to find the next permutation.

Time complexity of sort check -> **O(n) (worst case), Ω(1) (best case)**

Time complexity of next permutation algorithm -> **O(n)**

Overall Time complexity :

In Worst Case : $O(n \,.\, n! \,.\, (n+n)) \simeq O(n^2 .n!)$; In Best Case : $\Omega(n \,.\, 1 \,.\, (n+1) \simeq O(n^2)$

Space Complexity :

$O(n+n) \simeq O(n)$

THANK YOU