Algorithms and Problem Solving

Applications of Algorithms

Search

Home Contact Us

Posted on **November 16, 2015**

← Previous Next →

K-th permutation sequence

> *Given n and k, return the k-th permutation sequence of permutations of numbers {1,2,..,n}.*

Note: Given n will be between 1 and 9 inclusive.

By listing and labeling all of the permutations in order, we get the following sequence (ie, for n = 3):
“123”
“132”
“213”
“231”
“312”
“321”

Then, k=5th permutation sequence will be 312.

I have discussed a similar problem of finding the next permutation sequence of a given permutation in a previous post. In order to find the kth permutation one of the trivial solution would to call next permutation k times starting with the lexicographically first permutation i.e 1234…n. But this involves lots of extra computation resulting a worst case time complexity of O(n*k) = O(n*n!). We can do better but let's first discuss how to find next permutation.

Next Permutation

Observe that if all the digits are in non-decreasing order from right to left then the input itself is the biggest permutation of its digits. So, if we can detect the position where the non-decreasing sequence in disrupted then we can simply work on the part of the digits. We can do it in O(n) using the following (Wiki page also described this simple and efficient algorithm).

```
1. Find the largest index k such that nums[k] < nums
2. If no such index exists, the permutation is sorte
3. Find the largest index l greater than k such that
4 Swap the value of nums[k] with that of nums[l].
5. Reverse the sequence from nums[k + 1] up to and ir
```

```
public static void nextPermutation(int[] nums) {
    int k = -1;
```

```
        for (int i = nums.length - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) {
                k = i;
                break;
            }
        }
        if (k == -1) {
            reverse(nums, 0, nums.length-1);
            return;
        }
        int l = -1;
        for (int i = nums.length - 1; i > k; i--) {
            if (nums[i] > nums[k]) {
                l = i;
                break;
            }
        }
        swap(nums, k, l);
        reverse(nums, k + 1, nums.length-1);
    }

    public static void reverse(int A[], int i, int j)
            while(i < j){
                    swap(A, i, j);
                    i++;
                    j--;
            }
    }
```

Kth Permutation

Before reading this article further I strongly suggest to read my previous post to find rank of a permeation sequence. Basically given a permutation we compute how many permutations is possible to fix a number in a particular position of the array. Then we sum total such permutations for all positions to get the rank of the permutation. Now, in this question we are asking for the reverse question i.e. given the rank, find the permutation.

For example, n=4 and k=8, in this case we need to find 8th permutation in the permutation sequence of {1,2,3,4}. Below is all of 4! = 24 permutations in order. 8th permutation is [2, 1, 4, 3].

```
1. [1, 2, 3, 4]
2. [1, 2, 4, 3]
3. [1, 3, 2, 4]
4. [1, 3, 4, 2]
5. [1, 4, 2, 3]
6. [1, 4, 3, 2]
7. [2, 1, 3, 4]
8. [2, 1, 4, 3]
9. [2, 3, 1, 4]
10. [2, 3, 4, 1]
11. [2, 4, 1, 3]
12. [2, 4, 3, 1]
13. [3, 1, 2, 4]
14. [3, 1, 4, 2]
```

```
15. [3, 2, 1, 4]
16. [3, 2, 4, 1]
17. [3, 4, 1, 2]
18. [3, 4, 2, 1]
19. [4, 1, 2, 3]
20. [4, 1, 3, 2]
21. [4, 2, 1, 3]
22. [4, 2, 3, 1]
23. [4, 3, 1, 2]
24. [4, 3, 2, 1]
```

A permutation stating with a number has (n-1) positions to permute the rest (n-1) numbers giving total (n-1)! permutations stating with each of the elements in lexicographic order. For example, n=4, We can see the first (4-1)! = 3! = 6 permutations fixed for permutations starting with 1. Next 6 position is fixed for permutations starting with 2 and so on. For each of 6 permutations starting with 1, the next position will be fixed with 2 for total (n-2)! = (4-2)! = 2! = 2 permutations.

That is, first (n-1)! permutations will start with 1, next (n-1)! permutations will start with 2 and so on. That is for a given k the permutation will start with the element at index k/(n-1)!. We need to have this element fixed at the first spot and shift the remaining numbers down to right of it.

For example, n=4 and {1,2,3,4}, for k=8th permutation will start with element at index 7/3! = 1 (for 0-based index we start with k=k-1). So, first element 8th permutation is 2 i.e kth = [2, x, x, x]. As we are skipping 3! permutations for the 1 element at position less than the index=1 and there could be 3! total such combination with the rest of 3 numbers in the rest of 3 positions, so we will decrement k by 3!x1 i.e. k' = 7-3!x1 = 1.

Once we know the first element now, we need to find what will be second element. Note that second element will be from rest of the (n-1) elements i.e. {1, 3, 4}.

Similarly, permutations with second position fixed with a number (along with first position already fixed) has (n-2) positions to permute the rest (n-2) numbers giving total (n-2)! permutations for each of the elements in the second position. That is for a given k the permutation will start with the element at index k'/(n-2)!.

For example, continuing our example with updated k'=1 , second element will be at index 1/(4-2)! = 1/2! = 0 of {1, 3, 4} which is 1 i.e kth = [2, 1, x, x]. As we are skipping permutations for 0 elements at position less than the index=0 with 2 elements to put in rest of 2 positions, so we will decrement k' by 2!x0 i.e. k'' = 1-2!x0 = 1.

Next, with updated k"=1 , third element will be at index 1/(4-3)! = 1/1 = 1 of {3, 4} which is 4 i.e kth = [2, 1, 4, x]. As we are skipping permutations for 1 elements at position less than the index=1 with 1 elements to put in remaining 1 position, so we will decrement k by 1!x1 i.e. k"' = 1-1!x1 = 0.

Next, with updated k"'=0 , fourth element will be at index 0/(4-4)! = 0/1 = 0 of {3} which is 3 i.e kth = [2, 1, 4, 3]. As we are skipping permutations for 0 elements at position less than the index=0 with 0 elements remaining, so we will decrement k by 0!x0 i.e. k"" = 0-0!x0 = 0.

Now, we have tried all position so, we stop here and thus the resulting kth permutation is [2, 1, 4, 3].

Below is the implementation of the above idea. In order to handle the shrinking input set we used simple array shift to move the required element in the desired position and shift the rest of the element to right to maintain the order. So, overall complexity is O(n^2).

```
public static int[] kthPermutation(int n, int k){
        final int[] nums = new int[n];
        final int[] factorial = new int[n+1];
```

```
        factorial[0] = 1;
        factorial[1] = 1;
        nums[0] = 1;

        for (int i = 2; i <= n; i++) {
                nums[i-1] = i;
                factorial[i] = i*factorial[i - 1]
        }

        if(k <= 1){
                return nums;
        }
        if(k >= factorial[n]){
                reverse(nums, 0, n-1);
                return nums;
        }

        k -= 1;//0-based
        for(int i = 0; i < n-1; i++){
                int fact = factorial[n-i-1];
                //index of the element in the res
                //to put at i position (note, inc
                int index = (k/fact);
                //put the element at index (offse
                //and shift the rest on the right
                shiftRight(nums, i, i+index);
                //decrement k by fact*index as we
                //permutations for each element a
                k = k - fact*index;
        }

        return nums;
}
```

```
private static void shiftRight(int[] a, int s, ir
        int temp = a[e];
        for(int i = e; i > s; i--){
                a[i] = a[i-1];
        }
        a[s] = temp;
}
```

This entry was posted in **Algorithms** and tagged **kth permutation**, **next permutation** by জাহিদ. Bookmark the **permalink**.

2 THOUGHTS ON "K-TH PERMUTATION SEQUENCE"

zelo on **October 16, 2016 at 7:28 pm** said:

i really like you code i'm kind of new for c++…but i have a question…what if i want string like {A…..n}..A[26]… n= index;
i use lexicographic order permutation for string

and for k Johnson-Trotter algorithm…
n = 3 {A,B,C}
so if n=4 and k = 3 is n= {A,B,C,D}
k = {A,C,B,D}
i really have hard time to implement this appreciate you help please Thanks!

Reply ↓

zelo on **October 16, 2016 at 7:28 pm** said:

i really like you code i'm kind of new for c++…but i have a question…what if i want string like {A…..n}..A[26]… n= index;
i use lexicographic order permutation for string

and for k Johnson-Trotter algorithm…
n = 3 {A,B,C}
so if n=4 and k = 3 is n= {A,B,C,D}
k = {A,C,B,D}
i really have hard time to implement this appreciate you help please Thanks!

Reply ↓

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

Proudly powered by WordPress