Permutation Sequence

The set `[1,2,3,…,n]` contains a total of $n!$ unique permutations.

By listing and labeling all of the permutations in order,
We get the following sequence (ie, for $n = 3$):

1. `"123"`
2. `"132"`
3. `"213"`
4. `"231"`
5. `"312"`
6. `"321"`

Given n and k, return the k^{th} permutation sequence.

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

Solution 1

I'm sure somewhere can be simplified so it'd be nice if anyone can let me know. The pattern was that:

say n = 4, you have {1, 2, 3, 4}

If you were to list out all the permutations you have

1 + (permutations of 2, 3, 4)
2 + (permutations of 1, 3, 4)
3 + (permutations of 1, 2, 4)
4 + (permutations of 1, 2, 3)

We know how to calculate the number of permutations of n numbers... n! So each of those with permutations of 3 numbers means there are 6 possible permutations. Meaning there would be a total of 16 permutations in this particular one. So if you were to look for the (k = 14) 14th permutation, it would be in the

3 + (permutations of 1, 2, 4) subset.

To programmatically get that, you take k = 13 (subtract 1 because of things always starting at 0) and divide that by the 6 we got from the factorial, which would give you the index of the number you want. In the array {1, 2, 3, 4}, k/(n-1)! = 13/(4-1)! = 13/3! = 13/6 = 2. The array {1, 2, 3, 4} has a value of 3 at index 2. So the first number is a 3.

Then the problem repeats with less numbers.

The permutations of {1, 2, 4} would be:

1 + (permutations of 2, 4)
2 + (permutations of 1, 4)
4 + (permutations of 1, 2)

But our k is no longer the 14th, because in the previous step, we've already eliminated the 12 4-number permutations starting with 1 and 2. So you subtract 12 from k.. which gives you 1. Programmatically that would be...

k = k - (index from previous) * (n-1)! = k - 2*(n-1)! = 13 - 2*(3)! = 1

In this second step, permutations of 2 numbers has only 2 possibilities, meaning each of the three permutations listed above a has two possibilities, giving a total of 6. We're looking for the first one, so that would be in the 1 + (permutations of 2, 4) subset.

Meaning: index to get number from is k / (n - 2)! = 1 / (4-2)! = 1 / 2! = 0.. from {1, 2, 4}, index 0 is 1

so the numbers we have so far is 3, 1... and then repeating without explanations.

{2, 4}

k = k - (index from pervious) * (n-2)! = k - 0 * (n - 2)! = 1 - 0 = 1;
third number's index = k / (n - 3)! = 1 / (4-3)! = 1/ 1! = 1... from {2, 4}, index 1 has 4
Third number is 4

{2}
k = k - (index from pervious) * (n - 3)! = k - 1 * (4 - 3)! = 1 - 1 = 0;
third number's index = k / (n - 4)! = 0 / (4-4)! = 0/ 1 = 0... from {2}, index 0 has 2
Fourth number is 2

Giving us 3142. If you manually list out the permutations using DFS method, it would be 3142. Done! It really was all about pattern finding.

```
public class Solution {
public String getPermutation(int n, int k) {
    int pos = 0;
    List<Integer> numbers = new ArrayList<>();
    int[] factorial = new int[n+1];
    StringBuilder sb = new StringBuilder();

    // create an array of factorial lookup
    int sum = 1;
    factorial[0] = 1;
    for(int i=1; i<=n; i++){
        sum *= i;
        factorial[i] = sum;
    }
    // factorial[] = {1, 1, 2, 6, 24, ... n!}

    // create a list of numbers to get indices
    for(int i=1; i<=n; i++){
        numbers.add(i);
    }
    // numbers = {1, 2, 3, 4}

    k--;

    for(int i = 1; i <= n; i++){
        int index = k/factorial[n-i];
        sb.append(String.valueOf(numbers.get(index)));
        numbers.remove(index);
        k-=index*factorial[n-i];
    }

    return String.valueOf(sb);
}
```

}

written by tso original link here

Solution 2

```
string getPermutation(int n, int k) {
    int i,j,f=1;
    // left part of s is partially formed permutation, right part is the leftover
chars.
    string s(n,'0');
    for(i=1;i<=n;i++){
        f*=i;
        s[i-1]+=i; // make s become 1234...n
    }
    for(i=0,k--;i<n;i++){
        f/=n-i;
        j=i+k/f; // calculate index of char to put at s[i]
        char c=s[j];
        // remove c by shifting to cover up (adjust the right part).
        for(;j>i;j--)
            s[j]=s[j-1];
        k%=f;
        s[i]=c;
    }
    return s;
}
```

written by lucastan original link here

Solution 3

Recursion will use more memory, while this problem can be solved by iteration. I solved this problem before, but I didn't realize that using k = k-1 would avoid dealing with case k%(n-1)!==0. Rewrote this code, should be pretty concise now.

Only thing is that I have to use a list to store the remaining numbers, neither linkedlist nor arraylist are very efficient, anyone has a better idea?

The logic is as follows: for n numbers the permutations can be divided to (n-1)! groups, thus k/(n-1)! indicates the index of current number, and k%(n-1)! denotes remaining sequence (to the right). We keep doing this until n reaches 0, then we get n numbers permutations that is kth.

```
public String getPermutation(int n, int k) {
        List<Integer> num = new LinkedList<Integer>();
        for (int i = 1; i <= n; i++) num.add(i);
        int[] fact = new int[n];  // factorial
        fact[0] = 1;
        for (int i = 1; i < n; i++) fact[i] = i*fact[i-1];
        k = k-1;
        StringBuilder sb = new StringBuilder();
        for (int i = n; i > 0; i--){
            int ind = k/fact[i-1];
            k = k%fact[i-1];
            sb.append(num.get(ind));
            num.remove(ind);
        }
        return sb.toString();
    }
```

written by Adeath original link here