"Explain-like-I'm-five" Java Solution in O(n)

198

tso
Reputation: ★ 198

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 24 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);
}
```

}

2

FanFeng

Reputation: ★ 2

for the 1st step, 1+(2,3,4), 2+(1,3,4), 3+(1,2,4), 4+(1,2,3) i think it would give us 24 permutations instead of 16 cuz just as you said, each 3 numbers give us 3! -> 6 permutations, we have 4 here, so it should be 4*6 = 24 permutations, right??

0

tso
Reputation: ★ 198

Yes, you're right. Sorry for the typo.

3

yellowstone
Reputation: ★ 270

A little trick.

we can use

```
sb.append(numbers.get(index));  //becase sb.append function will append the string representation
```

instead of

```
sb.append(String.valueOf(numbers.get(index)));
```

1

yellowstone
Reputation: ★ 270

My idea is the same as yours.

```
public String getPermutation(int n, int k){
        ArrayList<Integer> list = new ArrayList<Integer>();
        StringBuilder sb = new StringBuilder();
        for(int i=1; i<=n; i++) {
                list.add(i);
        }
```

```
        int nth = k-1;
        int divider = factorial(n-1);
        while(n>1) {
                int index = nth/divider;
                nth = nth%divider;
                sb.append(list.get(index));
                list.remove(index);
                divider /= n-1;
                n--;
        }
        sb.append(list.get(0));// append last digit

        return sb.toString();
}

public int factorial(int n) {  // use tail recursion
        return fact(n, 1);
}
private int fact(int n, int k) {
        if(n==0)
                return k;
        else {
                return fact(n-1, n*k);
        }
}
```

0

Alexpanda
Reputation: ★ 78

Very clear and detailed explanation! Thanks!

5

brianbclan
Reputation: ★ 7

If you use remove on the array, it is not O(n).

3

WillowLeaf
Reputation: ★ 4

The complexity is actually O(n^2), since remove in an arrayList will take O(n) complexity.

5

yangyimeng
Reputation: ★ 5

not O(n), it is O(n^2)

1

TWiStErRob
Reputation: ★ 183

Heh, and I though I reinvented the wheel in my question (different problem, but used the same logic as you in a loop).

0

monkeyGoCrazy
Reputation: ★ 274

You save my day man

15

EdickCoding
Reputation: ★ 167

Thanks for sharing. I rewrote it to make it shorter

```
public class Solution {
    public String getPermutation(int n, int k) {
        StringBuilder sb = new StringBuilder();
        ArrayList<Integer> num = new ArrayList<Integer>();
        int fact = 1;
        for (int i = 1; i <= n; i++) {
            fact *= i;
            num.add(i);
        }
        for (int i = 0, l = k - 1; i < n; i++) {
            fact /= (n - i);
            int index = (l / fact);
            sb.append(num.remove(index));
            l -= index * fact;
        }
        return sb.toString();
    }
}
```

0

jaqenhgar
Reputation: ★ 754

OP is a redditor!!

0

raymartial
Reputation: ★ 0

thank you for explanation details!

0

airjordan919
Reputation: ★ 67

I had the same idea, but some parts are not as clear as your code. I re-wrote mine based on your idea and made some minor changes. Thanks for sharing!

```
public String getPermutation(int n, int k) {

    if (n < 1 || n > 9 || k < 1) return "";

    int[] fac = new int[n + 1];
    fac[0] = 1;
    for (int i = 1; i < n; i++) fac[i] = i * fac[i - 1];

    List<Integer> numbers = new ArrayList<>();
    for (int i = 1; i <= n; i++) numbers.add(i);
    StringBuilder sb = new StringBuilder();

    k--;

    for (int i = n - 1; i >= 0; i--) {
        int index = k / fac[i];
```

```
            sb.append(numbers.remove(index));
            k %= fac[i];
        }

        return sb.toString();

    }
```

0

ruby.zhu
Reputation: ★ 4

I am such a fool that I use Math.ceil(k/factorial[n-i])!!!

1

willcomeback
Reputation: ★ 24

can anyone explain how to come up with the idea "k--", I stuck here for a long time thx

0

iaming
Reputation: ★ 178

why not linkedlist, theoretically it is faster for the remove, isn't it?

1

CandyRobbery
Reputation: ★ 53

@willcomeback Because k in the input is 1-indexed. In the solution it's used as the index to look up numbers in each iteration, which is 0-indexed.

0

dianhua1560
Reputation: ★ 42

Getting TLE...

1

lisen @dianhua1560
Reputation: ★ 24

@dianhua1560 Did you solve the TLE problem?

I got it for my code and I can't figure out the problem so I copy this most voted code and it's still TLE for test case 8 8590.

I test it on my computer and count the time, it's as fast as 0ms.

Follow up: There might be something wrong with the network or server. I got accept with the same code after several time submit.

33 POSTS

14590 VIEWS

Log in to reply