Given n and k, return the kth permutation sequence

asked 1 year ago

viewed 2799 times

active 1 year ago

10

5

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 kth permutation sequence.

For example, given n = 3, k = 4, ans = "231".

There are multiple solutions out there. But all of them uses either factorial or there complexity is larger than O(n) such as O(n!). If you use factorial and find the number at position by k/(n-1)!, then problem comes when n is large(n = 100). Here as n is large, (n-1)! overflows and becomes 0. In result, I am getting divide by zero error...any solution or algorithm for that?

Here is my code:

```
public class KthPermutation {
    public String getPermutation(int n, int k) {
        // initialize all numbers
```

```
        ArrayList<Integer> numberList = new ArrayList<Integer>();

        for (int i = 1; i <= n; i++) {
            numberList.add(i);
        }
        int fact = 1;    // set factorial of n-1

        for (int i = 1; i <= n-1; i++) {
            fact = fact * i;
        }

        if ((long) k > (long) fact * n) {
            k = (int) ((long) k - (long) (fact * n));
        }
        k--; // set k to base 0

        StringBuilder result = new StringBuilder();
        result = getP(result, numberList, n, k, fact);
        return result.toString();
    }
    public static StringBuilder getP(StringBuilder result,
                ArrayList<Integer> numberList, int n, int k, int fact) {
        if (numberList.size() == 1 || n == 1) {
            result.append(numberList.get(0));
            return result;  // return condition
        }
        int number = (k / fact) + 1 ;
        result.append(numberList.get(number - 1));
        numberList.remove(number - 1);
        k = k % fact;  // update k
        fact = fact / (n - 1);
        n--;
        return getP(result, numberList, n, k, fact);
    }
}
```

java algorithm data-structures permutation backtracking

share improve this question

edited Jul 4 '15 at 23:38

yanana
769 7 22

asked Jul 4 '15 at 1:49

Vishlesh Patel
69 1 9

2 You haven't shown any effort. Sorry. – sstan Jul 4 '15 at 1:50

Linked

84 Fast permutation -> number -> permutation mapping algorithms

4 Fast exact bigint factorial

0 javascript : create for loops into for loops

1 Optimal way to create a tree with the following conditions

Related

261 Generating all permutations of a given string

@sstan I have edited the question and added my code. What do you mean by effort? – Vishlesh Patel Jul 4 '15 at 1:56

1 This seems like it may be homework, and if that's the case then you should include the tag [Homework] before your title. People will still be helpful as long as you show effort, but they don't want to feel like they're being taken advantage of. Give background for the post if it is not homework, because it will make people more willing to answer. – Tresdon Jul 4 '15 at 2:05

2 @Tresdon Whether a question is homework or not isn't really relevant. There's a reason there's no `homework` tag anymore. If you don't want to help out on this question, it's not required of you. – Sneftel Jul 4 '15 at 2:07

@Tresdon this is not the homework. I am practicing interview questions on Interviewbit.com. And the code fails for the large testcase with n = 100, k =10000000 – Vishlesh Patel Jul 4 '15 at 2:08

show **11** more comments

2 Answers

active oldest votes

9

So if I'm reading the question correctly, you want to find the kth permutation, preferrably without using BigIntegers, provided k is not large enough to require a BigInteger.

If we look at the sequence

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

We can rewrite it so that the number in each position is an index into a list of the numbers that haven't appeared so far on the line:

```
0 0 0
0 1 0
1 0 0
1 1 0
2 0 0
2 1 0
```

So for example "2, 0, 0" means start with the list "1, 2, 3", then take the third (because we are indexing from zero), which is a 3, then take the first of the remaining digits "1, 2" which is a 1, then the first of

Generating all permutations of a given string

11
Tips implementing permutation algorithm in Java

0
Permutations with a given integer

11
“K-transformed” permutations

4
Generating a set of permutation given a set of numbers and some conditions on the relative positions of the elements

6
Finding permutation of a set of 0 and 1, given index with O(N)

2
How to get nth permutation when repetition is allowed?

4
Permutation: return the kth permutation

0
Permutations: Possibility of a permutation to be obtained from a given sequence

1
Permutation of string using backtracking algorithm

the remaining digit, which is "2". So it produces "3, 1, 2".

To generate these indices, go from right to left and divide k by 1! for the rightmost two places, then 2! then 3! then 4! etc, and then modulo the result with the number of possible indices in that position, which is 1 for the rightmost, 2 for the second-rightmost etc. You don't have to calculate the factorial each time because you can keep a running product.

You can break out of the loop as soon as k divided by the factorial is zero, so you only have to compute factorials up until roughly the size of k multiplied by the last place in which k divided by the factorial is non-zero. If k is too large, you need to switch to BigIntegers.

Once you have the indices it's pretty straightforward to use them to generate the permutation.

Code (k starts from 0, so to find the first pass 0, not 1):

```
static public void findPermutation(int n, int k)
{
    int[] numbers = new int[n];
    int[] indices = new int[n];

    // initialise the numbers 1, 2, 3...
    for (int i = 0; i < n; i++)
        numbers[i] = i + 1;

    int divisor = 1;
    for (int place = 1; place <= n; place++)
    {
        if((k / divisor) == 0)
            break;  // all the remaining indices will be zero

        // compute the index at that place:
        indices[n-place] = (k / divisor) % place;
        divisor *= place;
    }

    // print out the indices:
    // System.out.println(Arrays.toString(indices));

    // permute the numbers array according to the indices:
    for (int i = 0; i < n; i++)
    {
        int index = indices[i] + i;

        // take the element at index and place it at i, moving the rest up
        if(index != i)
```

```
        {
            int temp = numbers[index];
            for(int j = index; j > i; j--)
               numbers[j] = numbers[j-1];
            numbers[i] = temp;
        }
    }
```

Demo

output:

```
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```

10000000th permutation for n = 100:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 92, 98, 96, 90, 91, 100, 94, 97, 95, 99, 93]

share improve this answer

edited Jul 7 '15 at 13:15

answered Jul 4 '15 at 7:00

samgak

14.9k 3 17 52

Thanks @samgak....efficient algorithm with best explanation and simple code...Thanks a lot! – Vishlesh Patel Jul 4 '15 at 17:09

add a comment

listening

How to use the 'modulus' operator?

How can two laptops have the same resolution and screen size but different pixel densities?

Did Donald Trump say that "global warming was a hoax invented by the Chinese"?

How much effort (and why) should consumers put into protecting their credit card numbers?

0

Of coarse there is a need for `bigints` with such interface

- when you have `n=100` then you have `n!` permutations
- which means `k` is in range `k=<1,n!>`

```
100!=933262154439441526816992388562667004907159682643816214685929638952175999
```

- which does not fit into standard `unsigned int`
- `2^32=          4294967296`
- `2^64=18446744073709551616`
- see Fast exact bigint factorial

if you change the interface a bit you suddenly do not need any bigints anymore

- just change api so it sequentialy returns 1st,2nd,3th,...permutation
- without specifying `k`
- so you need something like: Generalized Permutation (without repetitions) in C++
- of coarse this is usable only if your usage of permutation is also sequential
- you can also make function `previous()` to handle algorithms which are almost sequential
- for random or non sequential access you need use `bigints`

share improve this answer

edited Jul 4 '15 at 6:03

answered Jul 4 '15 at 5:55

Spektre

18k 4 22 78

add a comment

Your Answer

B B B B B B B B B B B B

Sign up or log in

Sign up using Google

Sign up using Facebook

Sign up using Email and Password

Post as a guest

Name

Email

required, but never shown

Post Your Answer

By posting your answer, you agree to the privacy policy and terms of service.

Not the answer you're looking for? Browse other questions tagged java algorithm data-structures permutation backtracking or ask your own question.

question feed