60. Permutation Sequence

Jan. 5, 2020 | 16.2K views
Average Rating: 3.72 (29 votes)

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 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.
- Given k will be between 1 and $n!$ inclusive.

Example 1:

```
Input: n = 3, k = 3
Output: "213"
```

Example 2:

```
Input: n = 4, k = 9
Output: "2314"
```

Solution

Solution Pattern

There are three main types of interview questions about permutations:

- 1.Generate all permutations.
- 2.Generate next permutation.
- 3.Generate the permutation number k (current problem).

If the order of generated permutations is not important, one could use "swap" backtracking to solve the first problem and to generate all $N!$ permutations in $\mathcal{O}(N \times N!)$ time.

Although, it is better to generate permutations in lexicographically sorted order using D.E. Knuth algorithm. This algorithm generates new permutation from the previous one in $\mathcal{O}(N)$ time. The same algorithm could be used to solve the second problem above.

The problem number three is where the fun starts because the above two algorithms do not apply:

- You will be asked to fit into polynomial time complexity, *i.e.* no backtracking.
- The previous permutation is unknown, *i.e.* you cannot use D.E. Knuth algorithm.

To solve the problem, one could use a pretty elegant idea that is based on the mapping. It's much easier to generate numbers than combinations or permutations.

> So let us generate numbers, and then map them to combinations/subsets/permutations.

This sort of encoding is widely used in password-cracking algorithms.

For example, in a previous article we discussed how one could map a subset with a binary bitmask of length N, where *ith* `0` means "the element number i is absent" and *ith* `1` means "the element number i is present".

One could do the same for permutations, mapping permutation with the integer in Factorial Number System Representation.

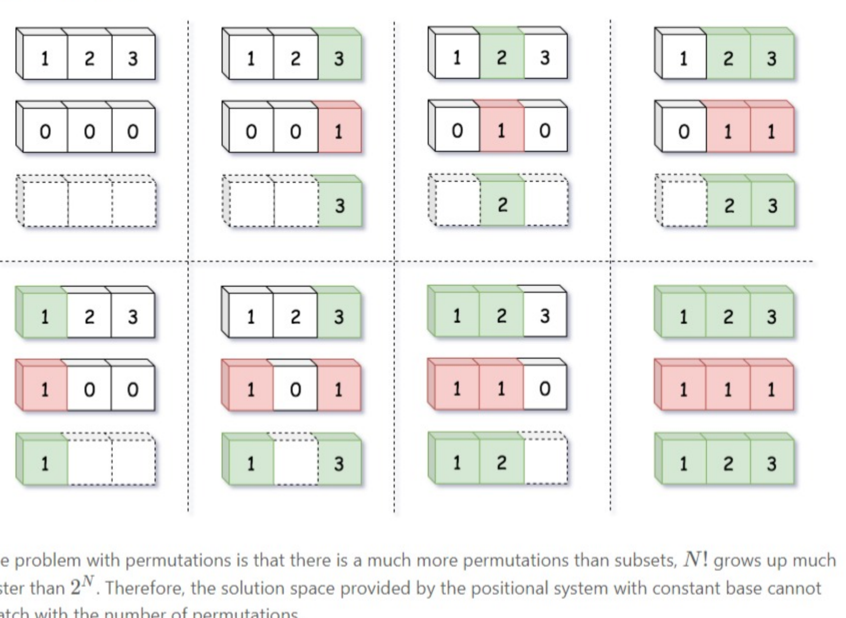
Approach 1: Factorial Number System

Why Do We Need Factorial Number System

Usually standard decimal or binary positional system could meet our needs. For example, each subset could be described by a number in binary representation

$$k = \sum_{m=0}^{N-1} k_m 2^m, \qquad 0 \le k_m \le 1$$

Here is how it works:



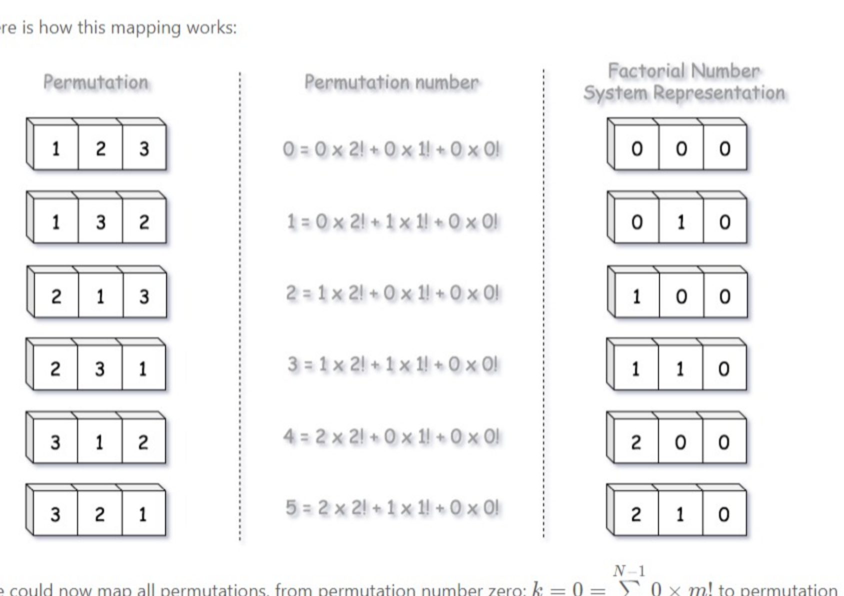
The problem with permutations is that there is a much more permutations than subsets, $N!$ grows up much faster than 2^N. Therefore, the solution space provided by the positional system with constant base cannot match with the number of permutations.

Here is where the factorial number system enters the scene. It's a positional system with *non-constant base* $m!$

$$k = \sum_{m=0}^{N-1} k_m m!, \qquad 0 \le k_m \le m$$

Note, that magnitude of weights is not constant as well and depends on base: $0 \le k_m \le m$ for the base $m!$, *i.e.* $k_0 = 0, 0 \le k_1 \le 1, 0 \le k_2 \le 2$, etc.

Here is how this mapping works:



We could now map all permutations, from permutation number zero: $k = 0 = \sum_{m=0}^{N-1} 0 \times m!$ to permutation number $N! - 1$: $k = N! - 1 = \sum_{m=0}^{N-1} m \times m!$.

> Hence we have a way to encode permutation number into factorial representation. Now let us use this factorial representation to construct the permutation itself.

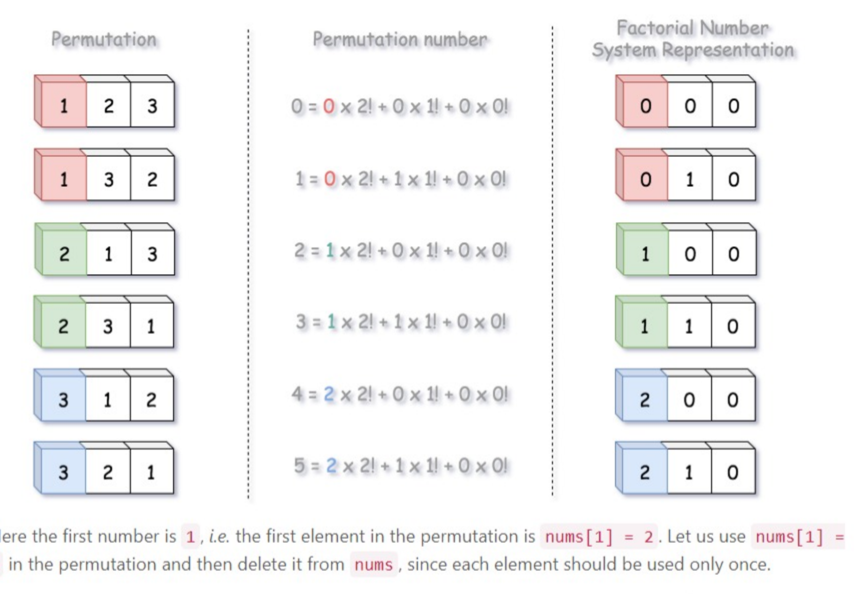
How to Construct the Permutation from its Factorial Representation

Let us pick up $N = 3$, which corresponds to the input array `nums = [1, 2, 3]`, and construct its permutation number $k = 3$. Since we number the permutations from 0 to $N! - 1$ (and *not* from 1 to $N!$ as in the problem description), for us that will be the permutation number $k = 2$.

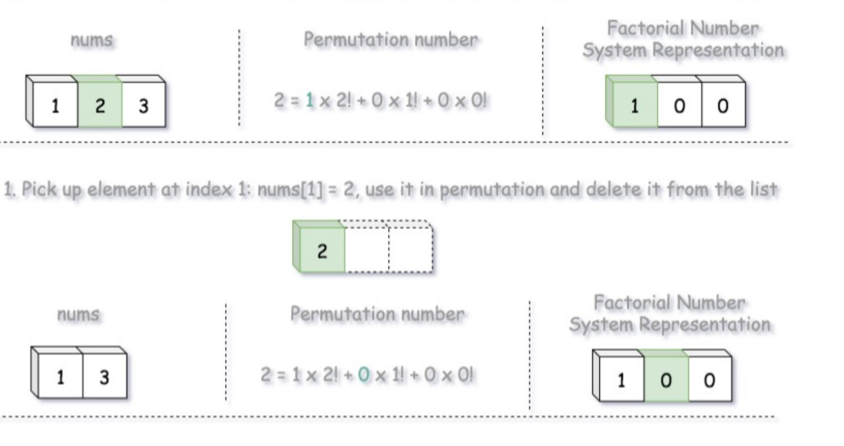
Let us first construct the factorial representation of $k = 2$:

$$k = 2 = 1 \times 2! + 0 \times 1! + 0 \times 0! = (1, 0, 0)$$

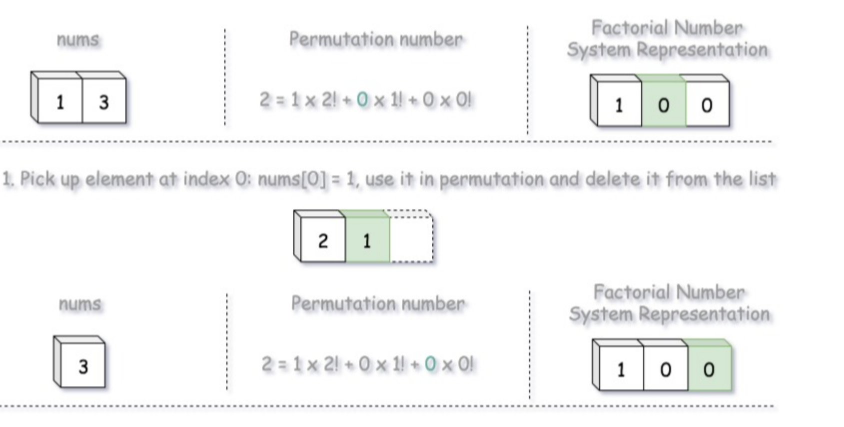
> The coefficients in factorial representation are indexes of elements in the input array. These are not direct indexes, but the indexes after the removal of already used elements. That's a consequence of the fact that each element should be used in permutation only once.



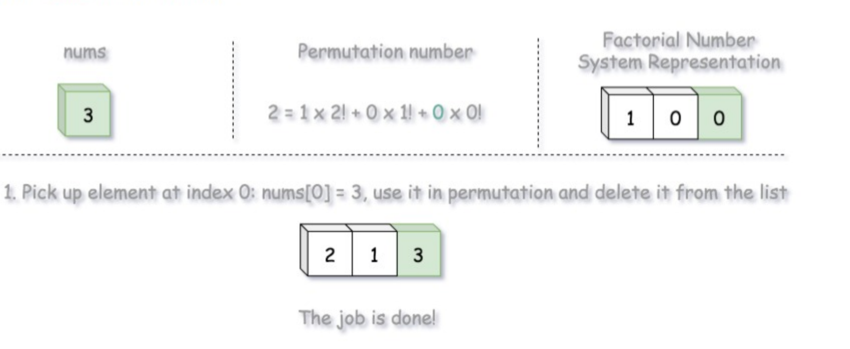
Here the first number is `1`, *i.e.* the first element in the permutation is `nums[1] = 2`. Let us use `nums[1] = 2` in the permutation and then delete it from `nums`, since each element should be used only once.



Next coefficient in factorial representation is `0`. Let's use `nums[0] = 1` in the permutation and then delete it from `nums`.



Next coefficient in factorial representation is `0`. Let's use `nums[0] = 3` in the permutation and then delete it from `nums`. The job is done.



Algorithm

- Generate input array `nums` of numbers from 1 to N.
- Compute all factorial bases from 0 to $(N-1)!$.
- Decrease k by 1 to make it fit into $(0, N! - 1)$ interval.
- Compute factorial representation of k. Use factorial coefficients to construct the permutation.
- Return the permutation string.

Implementation

Java Python Copy

```python
class Solution:
    def getPermutation(self, n: int, k: int) -> str:
        factorials, nums = [1], ['1']
        for i in range(1, n):
            # generate factorial system bases 0!, 1!, ..., (n - 1)!
            factorials.append(factorials[i - 1] * i)
            # generate nums 1, 2, ..., n
            nums.append(str(i + 1))

        # fit k in the interval 0 ... (n! - 1)
        k -= 1

        # compute factorial representation of k
        output = []
        for i in range(n - 1, -1, -1):
            idx = k // factorials[i]
            k -= idx * factorials[i]

            output.append(nums[idx])
            del nums[idx]

        return ''.join(output)
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N^2)$, because to delete elements from the list in a loop one has to perform $N + (N-1) + ... + 1 = N(N-1)/2$ operations.
- Space complexity: $\mathcal{O}(N)$.

Rate this article:

Previous Next

Comments: 12 Sort By

Type comment here... (Markdown is supported)

Preview Post

Dirk41 ★268 June 20, 2020 7:01 PM
I feel this should be a hard level question or maybe as a follow up for permutation
39 Share Reply
SHOW 1 REPLY

piofusco ★21 June 21, 2020 2:15 AM
Doesn't seem like a "beginner-friendly" problem at all.
20 Share Reply

new_01 ★23 June 21, 2020 6:08 AM
This June challenge is supposed to be beginner friendly.
10 Share Reply

sutirtho ★38 June 21, 2020 1:55 AM
A really tough question to answer in interview if you don't know about factorial number system! And factorial number system is not at all intuitive. The most amazing thing is indexes are considered after mutation of existing set of elements! (No one should ask this question in interview)
8 Share Reply
SHOW 3 REPLIES

thepatriot ★260 June 21, 2020 1:49 AM
very difficult question and while I appreciate the article, i question how useful such an "epic" is to learners and furthermore, how successful one would do when asked this in a live setting.
3 Share Reply

Haomin0817 ★30 February 2, 2020 10:13 AM
An O(n^2) solution

```
class Solution:
    def getPermutation(self, n: int, k: int) -> str:
        num_permu = 1
```

Read More
2 Share Reply
SHOW 2 REPLIES

heliumm ★3 January 11, 2020 7:20 AM Report
I think the time complexity of O(N^2) can be optimized to O(NlogN). If we utilize order-statistic tree which supports O(logN) deletion and O(logN) random access.

Correct me if I'm wrong.
2 Share Reply
SHOW 5 REPLIES

sush33 ★119 June 21, 2020 5:29 AM
While I enjoyed the question, don't think it's beginner friendly. Perhaps have two tracks the next month, one for beginners and one for advanced like this? That way, you can placate both groups.
1 Share Reply
Reputation

piofusco ★21 June 21, 2020 4:18 AM Report
Also, it would be great if the article could speak more to why `index = k / factorial[i]` and why it's necessary to do `k -= index * factorial[i]`. It does a good job explaining how to go from the factorial system to the permutation, however, the code doesn't reflect the same steps used in the explanation. Seems like a big jump from the algorithm to the code.
1 Share Reply
SHOW 1 REPLY

iamv ★1 June 24, 2020 1:24 AM
it is supposed to be a learning challenge for beginners from simple to less simple.
0 Share Reply

‹ 1 2 ›