19. K-th Permutation Sequence

60. Permutation Sequence

Hard 4229 402 Add to List Share

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.

Example 1:

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

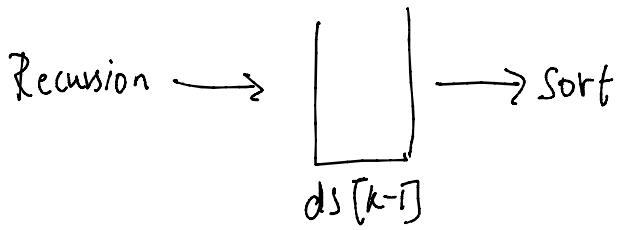
Example 2:

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

Example 3:

```
Input: n = 3, k = 1
Output: "123"
```

1) Brute force :



* Generate all the possible permutations of the given sequences.
* Using Recursion and every permutations generated is stored in some data structure (vector/array list).
* Sort the data structure in which we have stored all the sequences and return the k^th sequence from it.

T.C ⇒ O(N!N) + O(N! log N)

S.C ⇒ O(N)

S.C ⇒ O(N!)

2) Optimal Solution

n = 4 k = 17

Total = 24 permutation

n = 4
↓
(1, 2, 3, 4)

* I can start permutation from 1, 2, 3, 4.

1 + (2, 3, 4)] 6
2 + (1, 3, 4)] 6
3 + (1, 2, 4)] 6
4 + (1, 2, 3)] 6

24

k = 17

0 1 2 3
(1, 2, 3, 4)

0 based Indexing
1 2 3 4 → (0)

.
.
.
.

4 3 2 1 → 23th

(0) 1 + (2, 3, 4)] 6 [0 - 5)
(1) 2 + (1, 3, 4)] 6 (6 - 11)

4 3 2 1 → 23th

K = 17, so we look for 16th permut.

① 2 + (1, 3, 4)] 6 (6 – 11)

② 3 + (1, 2, 4)] 6 (12 – 17)

③ 4 + (1, 2, 3)] 6 (18 – 23)

24

4 seqs

3 _ _ _ _

16/6 = 2

16 % 6 = 4th seq

{1, 2, 4} K = 4/2 = 2

again same process

⓪ 1 + {2 4} 2 (0 – 1)

① 2 + {1 4} 2 (2 – 3)

② 4 + {1 2} 2 (4 – 5)

6

3 4 _ _

4 % 2 = 0

{1, 2} K = 0/1 = 0

again same process

⓪ 1 + {2} 1

① 2 + {1} 1

2

3 4 1 _

0 % 0 = 0 K = 0

{2} K = 0

2 + { }

3 4 1 2

↓ empty.

T.C = O(N) O(N) = O(N²)

S.C = O(N)

C++ • Autocomplete

```cpp
class Solution {
public:
    string getPermutation(int n, int k) {
        int fact = 1;
        vector<int> numbers;
        for(int i = 1;i < n;i++){
            fact = fact * i;
            numbers.push_back(i);
        }
        numbers.push_back(n);
        string ans = "";
        //reduce k value by 1, 0-based indexing
        k = k - 1;
        while(true){
            ans = ans + to_string(numbers[k/fact]);
            numbers.erase(numbers.begin() + k/fact);
            if(numbers.size() == 0){
                break;
            }
            k = k % fact;
            fact = fact / numbers.size();
        }
        return ans;
```

Java • Autocomplete

```java
class Solution {
    public String getPermutation(int n, int k){
        int fact = 1;
        List<Integer> numbers = new ArrayList<>();
        for(int i = 1;i < n;i++){
            fact = fact * i;
            numbers.add(i);
        }
        numbers.add(n);
        String ans = "";
        //reduce k value by 1, 0-based indexing
        k = k - 1;
        while(true){
            ans = ans + numbers.get(k/fact);
            numbers.remove(k/fact);
            if(numbers.size() == 0){
                break;
            }
            k = k % fact;
            fact = fact / numbers.size();
        }
        return ans;
```