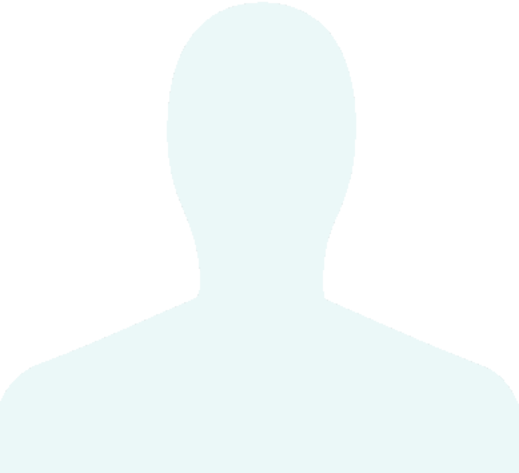
Coderust 2.0: Faster Coding Interview Preparation using Interactive Visualizations

Search

☰

LearnTeach



- My Profile

  - View
  - Edit
- Logout

Find kth permutation

Given a set of n elements find their kth permutation.

Description

Given a set of n elements find their k^{th} permutation. Consider the following set of elements:

1	2	3

All permutations of the above elements are (with ordering):

1st	123
2nd	132
3rd	213
4th	231
5th	312
6th	321

Here, we need to find the k^{th} permutation.

Hints

- Recursion
- Factorial

Solution

Runtime Complexity

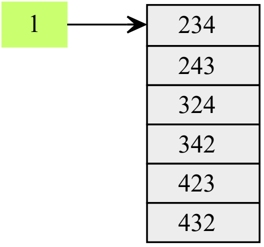
Linear, O(n).

Memory Complexity

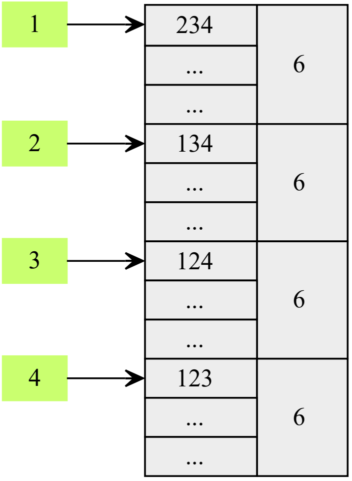
Linear, O(n).

Recursive solution will consume memory on the stack.

Let's discuss few basics first. We know that n! is the number of permutations of a set of size n. Another obvious and important concept is that if we choose an element for first position then the total permutations of remaining elements are (n-1)!. For example if we are given elements {1, 2, 3, 4} and we pick 1 as our first element then for remaining elements we have the following permutations:

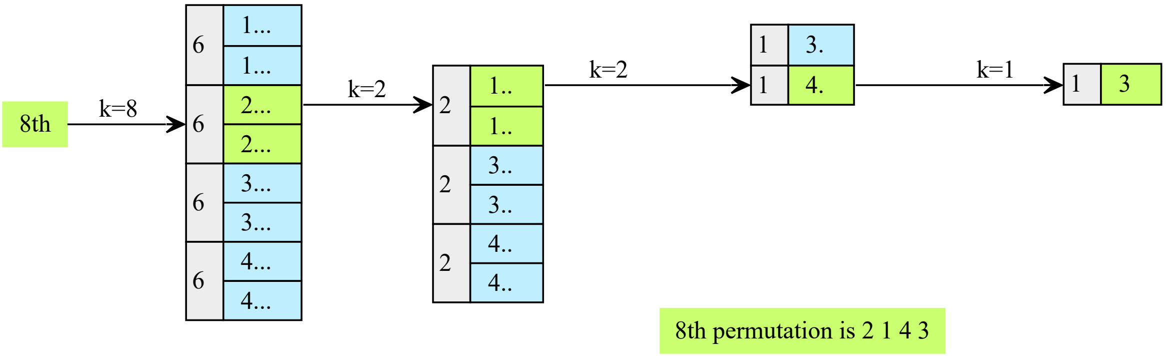


which is equal to 6 i.e. (n-1)!. This number is same if we pick another element for the first slot.



A naive way of finding k^{th} permutation will be to find all permutations and then return the k^{th} permutation or maintain a running count of permutations seen so far and return once k^{th} permutation is reached.

We can do better than this if we closely look at the diagram above. If we are given k and we somehow guess which block it's going to lie in that will help us find at least the first element. Similarly, within that block if we can identify a sub-block where k resides, it will help us find the second element. We can do this recursively until we run out of options. Here is a visual representation of this approach if k = 8:

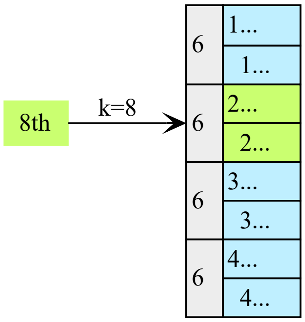


Here goes the algorithm we will follow:

```
9
1
If input vector is empty return result vector
2

3
block_size = (n-1)! ['n' is the size of vector]
4
Figure out which block k will lie in and select the first element of that block
5
(this can be done by doing (k-1)/block_size )
6

7
Append selected element to result vector and remove it from original input vector
8

9
Deduce from k the blocks that are skipped i.e k = k - selected*block_size and goto step 1
```

As you can notice the runtime complexity of this algorithm is linear (proportional to the size of the input vector) and memory is linear too because of the recursive calls. If we implement this algorithm non-recursively it will use constant memory.

Let's understand this example with k=8 step by step below.



block size = 3! = 6,
(k-1)/block size = 1
so 'k' lies in second
block,
k = k-6 = 2

result	2

input = {1, 2, 3, 4}, k = 8, n = 4

1 of 4

+

- C++
- Java
- Python
- JS
- Ruby

24

```
int factorial(int n) {
  if (n == 0 || n == 1) return 1;
  return n * factorial(n -1 );
}

void find_kth_permutation(
    vector<char>& v,
    int k,
    string& result) {
```

```
  if (v.empty()) {
    return;
  }

  int n = (int)(v.size());
  // count is number of permutations starting with each digit
  int count = factorial(n - 1);
  int selected = (k - 1) / count;

  result += v[selected];
  v.erase(v.begin() + selected);

  k = k - (count * selected);
  find_kth_permutation(v, k, result);
}
```

Mark as completed
← *Previous*Copy linked list with arbitrary pointer*Next* →Integer Division

Send feedback or ask a question

10 recommendations

- Home
- Featured
- Team
- Blog
- FAQ
- Terms of Service
- Contact Us