K-th Smallest in Lexicographical Order (/category/565/k-th-smallest-in-lexicographical-order) / Concise/Easy-to-understand Java 5ms solution with Explaination (/topic/64624.rss)

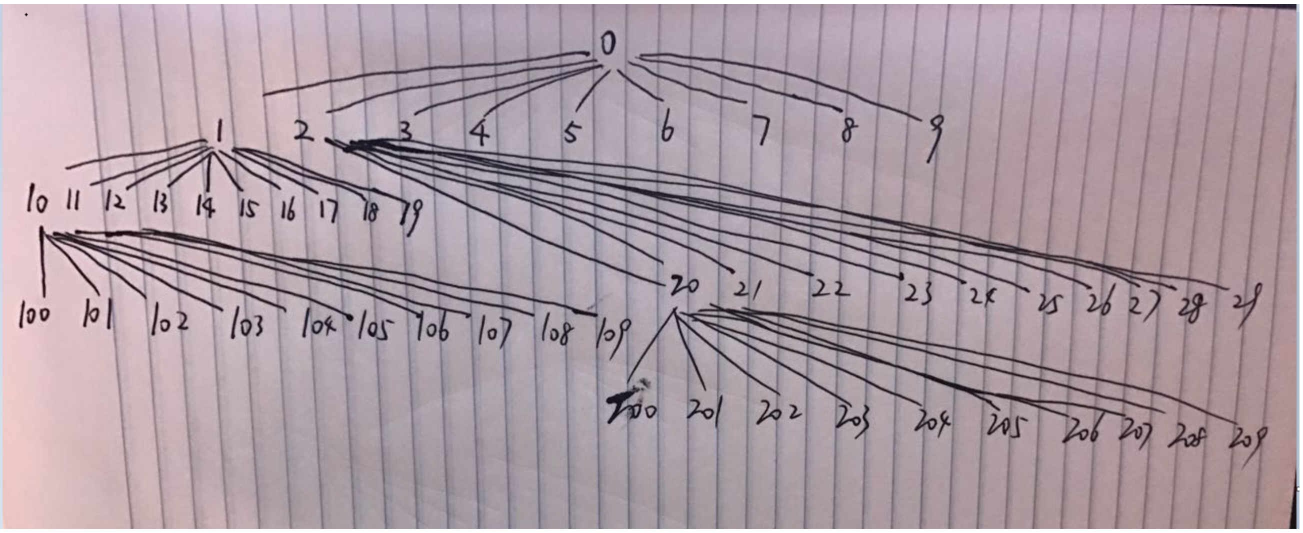
New users please **read the instructions** (https://discuss.leetcode.com/topic/22/welcome-new-users-please-read-this-before-posting) to *format your code* properly. Discuss is a place to **post interview questions** (/category/5/interview-questions) or share solutions / ask questions related to OJ problems (https://leetcode.com/problems).

5

(/user/nathanni) **NathanNi (/user/nathanni)**
Reputation: ★ 123

Original idea comes from http://bookshadow.com/weblog/2016/10/24/leetcode-k-th-smallest-in-lexicographical-order/ (http://bookshadow.com/weblog/2016/10/24/leetcode-k-th-smallest-in-lexicographical-order/)

Actually this is a denary tree (each node has 10 children). Find the kth element is to do a k steps preorder traverse of the tree.



(/uploads/files/1477293057263-upload-40379731-118a-4753-bed9-1cb372790d4b.png)

Initially, image you are at node 1 (variable: curr),
the goal is move (k - 1) steps to the target node x. (substract steps from k after moving)
when k is down to 0, curr will be finally at node x, there you get the result.

we don't really need to do a exact k steps preorder traverse of the denary tree, **the idea is to calculate the steps between curr and curr + 1 (neighbor nodes in same level), in order to skip some unnecessary moves.**

Main function
Firstly, calculate how many steps curr need to move to curr + 1.

1. if the steps <= k, we know we can move to curr + 1, and narrow down k to k - steps.
2. else if the steps > k, that means the curr + 1 is actually behind the target node x in the preorder path, we can't jump to curr + 1. What we have to do is to move forward only 1 step (curr * 10 is always next preorder node) and repeat the iteration.

calSteps function

1. how to calculate the steps between curr and curr + 1?
   Here we come up a idea to calculate by level.
   Let n1 = curr, n2 = curr + 1.
   n2 is always the next right node beside n1's right most node (who shares the same ancestor "curr")
   (refer to the pic, 2 is right next to 1, 20 is right next to 19, 200 is right next to 199).
2. so, if n2 <= n, what means n1's right most node exists, we can simply add the number of nodes from n1 to n2 to steps.
3. else if n2 > n, what means n (the biggest node) is on the path between n1 to n2, add (n + 1 - n1) to steps.
4. organize this flow to "steps += Math.min(n + 1, n2) - n1; n1 *= 10; n2 *= 10;"

1 out of 3

Here is the code snippet:

```
public int findKthNumber(int n, int k) {
    int curr = 1;
    k = k - 1;
    while (k > 0) {
        int steps = calSteps(n, curr, curr + 1);
        if (steps <= k) {
            curr += 1;
            k -= steps;
        } else {
            curr *= 10;
            k -= 1;
        }
    }
    return curr;
}
//use long in case of overflow
public int calSteps(int n, long n1, long n2) {
    int steps = 0;
    while (n1 <= n) {
        steps += Math.min(n + 1, n2) - n1;
        n1 *= 10;
        n2 *= 10;
    }
    return steps;
}
```

6 days ago (/post/141816)

Log in to reply (/login) (https://leetcode.com/problems/k-th-smallest-in-lexicographical-order)

▲
0
▼

A (/user/andrewma) ● **AndrewMa (/user/andrewma)**
Reputation: ★ 1

very good explain. Thank you

4 days ago (/post/142472)

▲
0
▼

(/user/undo) ● **Undo (/user/undo)**
Reputation: ★ 18

Very interesting solution!

4 days ago (/post/142578)

« ^ 1 out of 3 ˅ »

Log in to reply (/login) (https://leetcode.com/problems/k-th-smallest-in-lexicographical-order)