Lexicographical Numbers

Given an integer *n*, return 1 - *n* in lexicographical order.

For example, given 13, return: [1,10,11,12,13,2,3,4,5,6,7,8,9].

Please optimize your algorithm to use less time and space. The input size may be as large as 5,000,000.

Solution 1

```
public List<Integer> lexicalOrder(int n) {
        List<Integer> list = new ArrayList<>(n);
        int curr = 1;
        for (int i = 1; i <= n; i++) {
            list.add(curr);
            if (curr * 10 <= n) {
                curr *= 10;
            } else if (curr % 10 != 9 && curr + 1 <= n) {
                curr++;
            } else {
                while ((curr / 10) % 10 == 9) {
                    curr /= 10;
                }
                curr = curr / 10 + 1;
            }
        }
        return list;
    }
```

The basic idea is to find the next number to add.
Take 45 for example: if the current number is 45, the next one will be 450 (450 == 45 * 10)(if 450 <= n), or 46 (46 == 45 + 1) (if 46 <= n) or 5 (5 == 45 / 10 + 1)(5 is less than 45 so it is for sure less than n).
We should also consider n = 600, and the current number = 499, the next number is 5 because there are all "9"s after "4" in "499" so we should divide 499 by 10 until the last digit is not "9".
It is like a tree, and we are easy to get a sibling, a left most child and the parent of any node.

written by songzec original link here

Solution 2

```
The idea is pretty simple. If we look at the order we can find out we just keep ad
ding digit from 0 to 9 to every digit and make it a tree.
Then we visit every node in pre-order.
       1        2        3    ...
      /\       /\       /\
   10 ...19  20...29  30...39   ....
```

```
public class Solution {
    public List<Integer> lexicalOrder(int n) {
        List<Integer> res = new ArrayList<>();
        for(int i=1;i<10;++i){
          dfs(i, n, res);
        }
        return res;
    }

    public void dfs(int cur, int n, List<Integer> res){
        if(cur>n)
            return;
        else{
            res.add(cur);
            for(int i=0;i<10;++i){
                if(10*cur+i>n)
                    return;
                dfs(10*cur+i, n, res);
            }
        }
    }
}
```

written by xialanxuan1015 original link here

Solution 3

Three accepted solutions and me rambling on about failed attempts :-D

Solution 1 (accepted in 1792, 1747, 1700 ms)

I just sort the numbers 1 to n using my custom comparison. To compare two numbers, I "left-shift" them both before comparing them. For example if n = 49999, then I left-shift numbers so they're five digits. That is, 42 becomes 42000 and 123 becomes 12300. In case of ties, e.g., 420 also becoming 42000, the stability of `sorted` keeps them in order.

```
def lexicalOrder(self, n):
    top = 1
    while top * 10 <= n:
        top *= 10
    def mycmp(a, b, top=top):
        while a < top: a *= 10
        while b < top: b *= 10
        return -1 if a < b else b < a
    return sorted(xrange(1, n+1), mycmp)
```

Solution 2 (accepted in 1268, 1508, 1320, 1356, 1336 ms)

```
def lexicalOrder(self, n):
    withKeys = []
    for i in xrange(1, n+1):
        key = i
        while key < 1000000:
            key *= 10
        withKeys.append(key * 10000000 + i)
    withKeys.sort()
    return [ki % 10000000 for ki in withKeys]
```

Here I combine each number with a left-aligned version of it, for example:

```
    42  =>  42000000000042
  4200  =>  42000000004200
123456  =>  12345600123456
```

Then just sort these and then extract the lower parts.

Complexity

I think **Time** complexity and **space** complexity are both **O(n)** (at least if sort does what I think it does, I'll check some more), and the space complexity has a low

hidden factor.

The time and memory limits for Python for this problem are pretty low, requiring a fairly efficient solution. On LeetCode, Python ints are 64 bits, so embedding the left-aligned version of numbers in the numbers (solution 2) doesn't cost extra memory. Also, sorting simple ints is fast. Especially since the order from 1 to n is already largely sorted lexicographically, like the streak from 100 to 999 and the streak from 1000 to 9999. And Python's (Tim)sort can take advantage of those streaks and just merge them. If it merges "left to right" like I think it does, then it merges the small streaks first and only integrates the longest streaks last, which leads to overall O(n) time.

Optimizing Solution 2 (accepted in 980, 984, 980 ms)

Instead of assuming that we get numbers up to seven digits long and using constants, this uses the largest power of 10 up to n.

```
def lexicalOrder(self, n):
    highDigit = 1
    while highDigit * 10 <= n:
        highDigit *= 10
    higherDigit = highDigit * 10
    withKeys = []
    for i in xrange(1, n+1):
        key = i
        while key < highDigit:
            key *= 10
        withKeys.append(key * higherDigit + i)
    withKeys.sort()
    return [ki % higherDigit for ki in withKeys]
```

History...

Of course the first thing I had tried was this:

```
def lexicalOrder(self, n):
    return sorted(range(1, n+1), key=str)
```

Outrageously, this wasn't accepted! Got "Memory Limit Exceeded" at input n=49999! So next I tried the `cmp`-version of `sorted` instead of the `key`-version, and building strings only on the fly so it takes less memory:

```
def lexicalOrder(self, n):
    return sorted(range(1, n+1), lambda a, b: cmp(str(a), str(b)))
```

Horrendously, this wasn't accepted! Got "Time Limit Exceeded" at input n=49999! So next I tried converting to strings, sorting those, and converting back to ints. Uses

more memory, but less time:

```
def lexicalOrder(self, n):
    return map(int, sorted(map(str, xrange(1, n+1))))
```

Unfathomably, this wasn't accepted! Got "Memory Limit Exceeded" at input n=49999! The horror! Apparently LeetCode really didn't want me to get away with being lazy. So I tried it without sorting or strings and built the numbers in correct order:

```
def lexicalOrder(self, n):
    def dfs(i):
        if i <= n:
            result.append(i)
            for d in xrange(10):
                dfs(10 * i + d)
    result = []
    for i in range(1, 10):
        dfs(i)
    return result
```

Irritatingly, this wasn't accepted! Got "Time Limit Exceeded"! At input n=**14959**! So it was even **slower** than the above. Geez. And none of this was even remotely close to the "5,000,000" that the problem threatened me with. I gave up. And implemented that last solution in C++. It got accepted.

Later I found out that the "5,000,000" isn't even close to true, the largest actual test case is 49999. But even after lots of different attempts, I still can't get any simple stringify+sort solution accepted. The most efficient I came up with is this:

```
def lexicalOrder(self, n):
    return sorted(xrange(1, n+1), lambda a, b, s=str: 1 if s(b) < s(a) else -1)
```

That uses `xrange`, which is faster than `range`, uses the `cmp`-version of `sorted` because the `key` version gets memory limit exceeded, uses a fast local variable instead of the slower global `str`, and exploits that there are no duplicate numbers so I just have to distinguish two cases which I do with `<` instead of the `cmp` function. Still, after all of that optimization it's not fast enough. But based on comparing it with accepted solutions in custom testing, I think it's close. Maybe 10% too slow.

I did get one stringify+sort solution accepted, but it's less simple. I'll post that one later...

written by StefanPochmann original link here