Ugly Number II

Write a program to find the `n`-th ugly number.

Ugly numbers are positive numbers whose prime factors only include `2, 3, 5`. For example, `1, 2, 3, 4, 5, 6, 8, 9, 10, 12` is the sequence of the first `10` ugly numbers.

Note that `1` is typically treated as an ugly number.

1. The naive approach is to call `isUgly` for every number until you reach the n^{th} one. Most numbers are *not* ugly. Try to focus your effort on generating only the ugly ones.
2. An ugly number must be multiplied by either 2, 3, or 5 from a smaller ugly number.
3. The key is how to maintain the order of the ugly numbers. Try a similar approach of merging from three sorted lists: L_1, L_2, and L_3.
4. Assume you have U_k, the k^{th} ugly number. Then U_{k+1} must be Min(L_1 * 2, L_2 * 3, L_3 * 5).

Credits:
Special thanks to @jianchao.li.fighter for adding this problem and creating all test cases.

Solution 1

We have an array *k* of first n ugly number. We only know, at the beginning, the first one, which is 1. Then

k[1] = min(k[0]x2, k[0]x3, k[0]x5). The answer is k[0]x2. So we move 2's pointer to 1. Then we test:

k[2] = min(k[1]x2, k[0]x3, k[0]x5). And so on. Be careful about the cases such as 6, in which we need to forward both pointers of 2 and 3.

x here is multiplication.

```
class Solution {
public:
    int nthUglyNumber(int n) {
        if(n <= 0) return false; // get rid of corner cases
        if(n == 1) return true; // base case
        int t2 = 0, t3 = 0, t5 = 0; //pointers for 2, 3, 5
        vector<int> k(n);
        k[0] = 1;
        for(int i  = 1; i < n ; i ++)
        {
            k[i] = min(k[t2]*2,min(k[t3]*3,k[t5]*5));
            if(k[i] == k[t2]*2) t2++;
            if(k[i] == k[t3]*3) t3++;
            if(k[i] == k[t5]*5) t5++;
        }
        return k[n-1];
    }
};
```

written by jmty0083 original link here

Solution 2

```
struct Solution {
    int nthUglyNumber(int n) {
        vector <int> results (1,1);
        int i = 0, j = 0, k = 0;
        while (results.size() < n)
        {
            results.push_back(min(results[i] * 2, min(results[j] * 3, results[k]
* 5)));
            if (results.back() == results[i] * 2) ++i;
            if (results.back() == results[j] * 3) ++j;
            if (results.back() == results[k] * 5) ++k;
        }
        return results.back();
    }
};
```

Explanation:

The key is to realize each number can be and have to be generated by a former number multiplied by 2, 3 or 5 e.g. 1 2 3 4 5 6 8 9 10 12 15.. what is next? it must be x * 2 or y * 3 or z * 5, where x, y, z is an existing number.

How do we determine x, y, z then? apparently, you can just *traverse the sequence generated by far* from 1 ... 15, until you find such x, y, z that x * 2, y * 3, z * 5 is just bigger than 15. In this case x=8, y=6, z=4. Then you compare x * 2, y * 3, z * 5 so you know next number will be x * 2 = 8 * 2 = 16. k, now you have 1,2,3,4,....,15, 16,

Then what is next? You wanna do the same process again to find the new x, y, z, but you realize, wait, do I have to *traverse the sequence generated by far* again?

NO! since you know last time, x=8, y=6, z=4 and x=8 was used to generate 16, so this time, you can immediately know the newx *= 9 (the next number after 8 is 9 in the generated sequence), y=6, z=4. Then you need to compare newx* * 2, y * 3, z * 5. You know next number is 9 * 2 = 18;

And you also know, the next x will be 10 since new_x = 9 was used this time. But what is next y? apparently, if y=6, 6*3 = 18, which is already generated in this round. So you also need to update next y from 6 to 8.

Based on the idea above, you can actually generated x,y,z from very beginning, and update x, y, z accordingly. It ends up with a O(n) solution.

written by fentoyal original link here

Solution 3

The idea of this solution is from this page:http://www.geeksforgeeks.org/ugly-numbers/

The ugly-number sequence is 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, … because every number can only be divided by 2, 3, 5, one way to look at the sequence is to split the sequence to three groups as below:

```
(1) 1×2, 2×2, 3×2, 4×2, 5×2, …
(2) 1×3, 2×3, 3×3, 4×3, 5×3, …
(3) 1×5, 2×5, 3×5, 4×5, 5×5, …
```

We can find that every subsequence is the ugly-sequence itself (1, 2, 3, 4, 5, …) multiply 2, 3, 5.

Then we use similar merge method as merge sort, to get every ugly number from the three subsequence.

Every step we choose the smallest one, and move one step after,including nums with same value.

Thanks for this author about this brilliant idea. Here is my java solution

```
public class Solution {
    public int nthUglyNumber(int n) {
        int[] ugly = new int[n];
        ugly[0] = 1;
        int index2 = 0, index3 = 0, index5 = 0;
        int factor2 = 2, factor3 = 3, factor5 = 5;
        for(int i=1;i<n;i++){
            int min = Math.min(Math.min(factor2,factor3),factor5);
            ugly[i] = min;
            if(factor2 == min)
                factor2 = 2*ugly[++index2];
            if(factor3 == min)
                factor3 = 3*ugly[++index3];
            if(factor5 == min)
                factor5 = 5*ugly[++index5];
        }
        return ugly[n-1];
    }
}
```

written by syftalent original link here