625. Minimum Factorization

June 17, 2017 | 6.6K views

Average Rating: 5 (13 votes)

Given a positive integer `a`, find the smallest positive integer `b` whose multiplication of each digit equals to `a`.

If there is no answer or the answer is not fit in 32-bit signed integer, then return 0.

Example 1
Input:

```
48
```

Output:

```
68
```

Example 2
Input:

```
15
```

Output:

```
35
```

Solution

Approach #1 Brute Force [Time Limit Exceeded]

The simplest solution is to consider every possible 32-bit number starting from 1 which satisfies the given criteria. To check this, we obtain each individual digit of every such number and check if their product is equal to the given number a. As soon as such a number is found, we return the same. If no such 32-bit number is found, we return a 0 value.

Java

Copy

```
public class Solution {
    public int smallestFactorization(int a) {
        for (int i = 1; i < 999999999; i++) {
            long mul = 1, t = i;
            while (t != 0) {
                mul *= t % 10;
                t /= 10;
            }
            if (mul == a && mul <= Integer.MAX_VALUE)
                return i;
        }
        return 0;
    }
}
```

Complexity Analysis

- Time complexity : $O(9999999999)$. In case of prime numbers loop can go upto this large number.
- Space complexity : $O(1)$. Constant space is used.

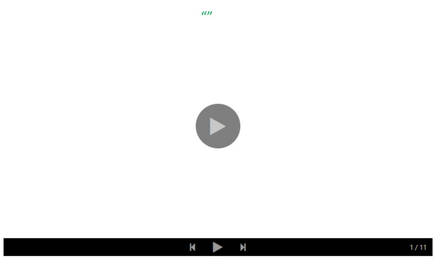
Approach #2 Better Brute Force [Time Limit Exceeded]

Algorithm

Instead of considering every possible number from the total search space, we can do the search in a smarter way. We can start putting the numbers from 9 to 2 at the ones position and keep on proceeding towards more significant places. For every number currently generated, we can check if the product of its digits exceeds the given number a. If so, there is no point in appending more digitas to this number. Thus, we can change the composition of the number generated till now and continue the checking process.

For doing this, we make use of a recursive function `search()`, which takes the number generated till now, res(as a string) as one of its arguments along with the number to be appended next as the res as a prefix as one of the other arguments. We can note that to obtain the smallest possible number, we need to try to put the largest number(which will be one of the factors for constituting the product a) at the least significant position and the smallest one at the most significant position. Thus, we start from the least significant position by trying to place a 9 at this position and then continue by trying to place smaller numbers at this position if the numbers generated by the previous arrangements fail. If some arrangement leads to a product of digits not larger than a, we continue with placing digits, equal to or smaller than the last digit placed, at the more significant positions.

The following animation illustrates the recursive process:



Java

Copy

```
public class Solution {
    long ans;
    public int smallestFactorization(int a) {
        if(a < 2)
            return a;
        int[] dig=new int[]{9, 8, 7, 6, 5, 4, 3, 2};
        if (search(dig, 0, a, 1, "") && ans <= Integer.MAX_VALUE)
            return (int)ans;
        return 0;
    }
    public boolean search(int[] dig, int i, int a, long mul, String res) {
        if (mul > a || i == dig.length )
            return false;
        if (mul == a) {
            ans = Long.parseLong(res);
            return true;
        }
        return search(dig, i, a, mul * dig[i], dig[i] + res) || search(dig, i + 1, a, mul, res);
    }
}
```

Complexity Analysis

- Time complexity : $O(l)$. Here l refers to total number of combinations.
- Space complexity : $O(log(a))$. In worst case, depth of recursion tree can go upto the $O(log(a))$.

Approach #3 Using Factorization[Accepted]

Algorithm

We know that the final number generated, res, should be such that its digits should have a product equal to the given number a. In other words, the digits of res will be the factors of the given number a. Thus, our problem reduces to finding the factors(not necessarily prime) of a and finding their smallest possible arrangement. Thus, we start with trying with the largest possible factor 9, obtain as many such counts of this factor as possible in res and place such factors obtianed at its least significant positions. Then, we go on decrementing the number currently considered as the possible factor and if it is a factor, we keep on placing it at relatively more significant positions in res. We go on getting such factors till we are done considering all the numbers from 9 to 2. At the end, res gives the required result.

Java

Copy

```
public class Solution {
    public int smallestFactorization(int a) {
        if (a < 2)
            return a;
        long res = 0, mul = 1;
        for (int i = 9; i >= 2; i--) {
            while (a % i == 0) {
                a /= i;
                res = mul * i + res;
                mul *= 10;
            }
        }
        return a < 2 && res <= Integer.MAX_VALUE ? (int)res : 0;
    }
}
```

Complexity Analysis

- Time complexity : $O(8loga)$. Outer loop will iterate only 8 times, while inner loop takes $O(logi)$ for particular i.
- Space complexity : $O(1)$. Constant space is used.

Analysis written by: @vinod23

Rate this article:

Previous Next

Comments: 1

Sort By

Type comment here... (Markdown is supported)

Preview Post

isucs ★0 August 30, 2017 9:24 PM Report

I think the last line: return a < 2 && res <= Integer.MAX_VALUE ? (int)res : 0;
use a < 10 is more understandable then a < 2, since a < 10 means the remaining a has to contains only one digit to meet the algorithm

0 Share Reply

SHOW 1 REPLY