Self Dividing Numbers

A *self-dividing number* is a number that is divisible by every digit it contains.

For example, 128 is a self-dividing number because `128 % 1 == 0`, `128 % 2 == 0`, and `128 % 8 == 0`.

Also, a self-dividing number is not allowed to contain the digit zero.

Given a lower and upper number bound, output a list of every possible self dividing number, including the bounds if possible.

Example 1:

```
Input:
left = 1, right = 22
Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]
```

Note:

- The boundaries of each input argument are `1 .`

Solution 1

Pretty self-explanatory.

- *Yangshun*

```
class Solution(object):
    def selfDividingNumbers(self, left, right):
        is_self_dividing = lambda num: '0' not in str(num) and all([num % int(digit)
== 0 for digit in str(num)])
        return filter(is_self_dividing, range(left, right + 1))
```

As pointed out by @ManuelP, `[num % int(digit) == 0 for digit in str(num)]` creates an entire list which is not necessary. By leaving out the `[` and `]`, we can make use of generators which are lazy and allows for short-circuit evaluation, i.e. `all` will terminate as soon as one of the digits fail the check.

The answer below improves the run time from 128 ms to 95 ms:

```
class Solution(object):
    def selfDividingNumbers(self, left, right):
        is_self_dividing = lambda num: '0' not in str(num) and all(num % int(digit)
== 0 for digit in str(num))
        return filter(is_self_dividing, range(left, right + 1))
```

written by yangshun original link here

Solution 2

The idea is to traverse each integer sequence from left to right incrementing by one (left, left + 1, left + 2,..., left + n, right). Each time check if the the current number i is self-divided.
Self-division check is done by using '%' operator(we check each digit of i moving from right to left)
E.g. if i = 128 number:

1. 128 % 10 = 8, check 8 != 0;
2. remove 8 from next step j = 128 / 10 = 12
3. repeat 1 and 2 until j == 0

Time complexity is O(nm), where n = right - left, and m is number of digits in iterated number

```
class Solution {
    public List<Integer> selfDividingNumbers(int left, int right) {
        List<Integer> list = new ArrayList<>();
        for (int i = left; i <= right; i++) {
            int j = i;
            for (; j > 0; j /= 10) {
                if ((j % 10 == 0) || (i % (j % 10) != 0)) break;
            }
            if (j == 0) list.add(i);
        }
        return list;
    }
}
```

written by anna.boltenko original link here

Solution 3

```
class Solution {
    public List<Integer> selfDividingNumbers(int left, int right) {

        List<Integer> ans = new ArrayList<Integer>();

        for(int i=left; i<=right ; i++){
            if(isSelfDividingNumber(i))
                ans.add(i);
        }
        return ans;
    }


    public boolean isSelfDividingNumber(int n){
        int original = n;
        while(n>0){
            int r = n%10;
            if(r == 0)             return false;
            if(original%r !=0)   return false;
            n /= 10;
        }
        return true;
    }

}
```

written by jassiuf93 original link here