Count Numbers with Unique Digits

Given a **non-negative** integer n, count all numbers with unique digits, x, where $0 \leq x < 10^n$.

Example:
Given n = 2, return 91. (The answer should be the total numbers in the range of $0 \leq x < 100$, excluding `[11,22,33,44,55,66,77,88,99]`)

1. A direct way is to use the backtracking approach.
2. Backtracking should contains three states which are (the current number, number of steps to get that number and a bitmask which represent which number is marked as visited so far in the current number). Start with state (0,0,0) and count all valid number till we reach number of steps equals to 10^n.
3. This problem can also be solved using a dynamic programming approach and some knowledge of combinatorics.
4. Let f(k) = count of numbers with unique digits with length equals k.
5. f(1) = 10, ..., f(k) = 9 * 9 * 8 * ... (9 - k + 2) [The first factor is 9 because a number cannot start with 0].

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

Solution 1

Following the hint. Let f(n) = count of number with unique digits of length n.

f(1) = 10. (0, 1, 2, 3, ..., 9)

f(2) = 9 * 9. Because for each number i from 1, ..., 9, we can pick j to form a 2-digit number ij and there are 9 numbers that are different from i for j to choose from.

f(3) = f(2) * 8 = 9 * 9 * 8. Because for each number with unique digits of length 2, say ij, we can pick k to form a 3 digit number ijk and there are 8 numbers that are different from i and j for k to choose from.

Similarly f(4) = f(3) * 7 = 9 * 9 * 8 * 7....

...

f(10) = 9 * 9 * 8 * 7 * 6 * ... * 1

f(11) = 0 = f(12) = f(13)....

any number with length > 10 couldn't be unique digits number.

The problem is asking for numbers from 0 to 10^n. Hence return f(1) + f(2) + .. + f(n)

As @4acreg suggests, There are only 11 different ans. You can create a lookup table for it. This problem is O(1) in essence.

```
public int countNumbersWithUniqueDigits(int n) {
      if (n == 0)     return 1;

      int res = 10;
      int uniqueDigits = 9;
      int availableNumber = 9;
      while (n-- > 1 && availableNumber > 0) {
          uniqueDigits = uniqueDigits * availableNumber;
          res += uniqueDigits;
          availableNumber--;
      }
      return res;
  }
```

written by stupidbird911 original link here

Solution 2

The idea is to append one digit at a time recursively (only append digits that has not been appended before). Number zero is a special case, because we don't want to deal with the leading zero, so it is counted separately at the beginning of the program. The running time for this program is O(10!) worst case, or O(n!) if n < 10.

The OJ gives wrong answer when n = 0 and n = 1. The correct answer should be:

> 0, 1
>
> 1, 10
>
> 2, 91
>
> 3, 739
>
> 4, 5275
>
> 5, 32491
>
> 6, 168571
>
> 7, 712891
>
> 8, 2345851
>
> 9, 5611771
>
> 10 and beyond, 8877691

```java
public class Solution {
    public static int countNumbersWithUniqueDigits(int n) {
        if (n > 10) {
            return countNumbersWithUniqueDigits(10);
        }
        int count = 1; // x == 0
        long max = (long) Math.pow(10, n);

        boolean[] used = new boolean[10];

        for (int i = 1; i < 10; i++) {
            used[i] = true;
            count += search(i, max, used);
            used[i] = false;
        }

        return count;
    }

    private static int search(long prev, long max, boolean[] used) {
        int count = 0;
        if (prev < max) {
            count += 1;
        } else {
            return count;
        }

        for (int i = 0; i < 10; i++) {
            if (!used[i]) {
                used[i] = true;
                long cur = 10 * prev + i;
                count += search(cur, max, used);
                used[i] = false;
            }
        }

        return count;
    }
}
```

written by lzb700m original link here

Solution 3

test case is wrong when n==1, it should be 11 instead of 10, from 0 to 10 inclusively

n=2 10+9 * 9

n=3 10+9 * 9+9 * 9 * 8

n=4 10+9 * 9+9 * 9 * 8+9 * 9 * 8 * 7

...

when n > 10, the total number won't increase any more, so we set n=10 in that case

```
public class Solution {
    public int countNumbersWithUniqueDigits(int n) {
        if (n == 0) {
            return 2;
        }
        if (n == 1) {
            return 10; // should be 11
        }
        n = Math.min(n, 10);
        int sum = 10;
        int tmp = 9;
        for (int i = 1; i < n; i++) {
            tmp *= 10 - i;
            sum += tmp;
        }
        return sum;
    }
}
```

written by xuyirui original link here