Daily Coding Problem #213

Problem

This problem was asked by Snapchat.

Given a string of digits, generate all possible valid IP address combinations.

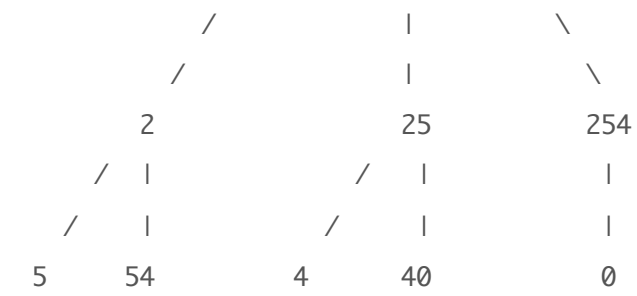
IP addresses must follow the format A.B.C.D, where A, B, C, and D are numbers between `0` and 255. Zero-prefixed numbers, such as `01` and `065`, are not allowed, except for `0` itself.

For example, given "2542540123", you should return ['254.25.40.123', '254.254.0.123'].

Solution

Each part of an IP address can be one, two, or three digits long. A valid part can be either a single digit, a two-digit number between `10` and 99, or a three-digit number between `100` and 255. For the input "2542540123", for example, we can either begin with "2", "25", or "254". Once we choose one of these, we can have up to three valid options for the second part. For example, if we started with "2", our next part could either be "5" or "54".

These choices can be represented in a tree, as follows:



```
  / | \  / | \    / | \  / | \      / | \
```

At each level of the tree, we find the valid parts that start with the first one, two, or three digits of our string. For each of these parts, we add it to our potential solution and recursively find the next valid parts in the remainder of the string. Once we reach the fourth level of the tree (meaning we have generated four valid parts to our IP address), and we have used all the characters in our input, we have found a solution. This kind of algorithm is called iterative deepening depth-first search (IDS).

```
def generate_IP_addresses(s, parts=[]):
    addresses = []

    if len(parts) > 4:
        return []

    if not s:
        if len(parts) == 4:
            return [".".join(parts)]
        else:
            return []

    addresses += generate_IP_addresses(s[1:], parts + [s[:1]])

    if len(s) > 1 and 10 <= int(s[:2]) <= 99:
        addresses += generate_IP_addresses(s[2:], parts + [s[:2]])

    if len(s) > 2 and 100 <= int(s[:3]) <= 255:
        addresses += generate_IP_addresses(s[3:], parts + [s[:3]])

    return addresses
```

The time complexity of IDS is $O(b^d)$, where b is the branching factor and d is the tree depth. Here $b = 3$ and $d = 4$.

Terms of Service
Press