Master algorithms together on Binary Search! Create a room, invite your friends, and race to finish the problems.

Daily Coding Problem #7

Problem

This problem was asked by Facebook.

Given the mapping a = 1, b = 2, ... z = 26, and an encoded message, count the number of ways it can be decoded.

For example, the message '111' would give 3, since it could be decoded as 'aaa', 'ka', and 'ak'.

You can assume that the messages are decodable. For example, '001' is not allowed.

Solution

This looks like a problem that is ripe for solving with recursion. First, let's try to think of a recurrence we can use for this problem. We can try some cases:

- "", the empty string and our base case, should return 1.
- "1" should return 1, since we can parse it as "a" + "".
- "11" should return 2, since we can parse it as "a" + "a" + "" and "k" + "".
- "111" should return 3, since we can parse it as:
    - "a" + "k" + ""
    - "k" + "a" + ""
    - "a" + "a" + "a" + "".
- "011" should return 0, since no letter starts with 0 in our mapping.
- "602" should also return 0 for similar reasons.

We have a good starting point. We can see that the recursive structure is as follows:

- If string starts with zero, then there's no valid encoding.
- If the string's length is less than or equal to 1, there is only 1 encoding.
- If the first two digits form a number k that is less than or equal to 26, we can recursively count the number of encodings assuming we pick k as a letter.
- We can also pick the first digit as a letter and count the number of encodings with this assumption.

```
def num_encodings(s):
    if s.startswith('0'):
        return 0
    elif len(s) <= 1: # This covers empty string
        return 1

    total = 0
```

```
    if int(s[:2]) <= 26:
        total += num_encodings(s[2:])

    total += num_encodings(s[1:])
    return total
```

However, this solution is not very efficient. Every branch calls itself recursively twice, so our runtime is $O(2^n)$. We can do better by using dynamic programming.

All the following code does is repeat the same computation as above except starting from the base case and building up the solution. Since each iteration takes O(1), the whole algorithm now takes O(n).

```
from collections import defaultdict

def num_encodings(s):
    # On lookup, this hashmap returns a default value of 0 if the key doesn't exist
    # cache[i] gives us # of ways to encode the substring s[i:]
    cache = defaultdict(int)
    cache[len(s)] = 1 # Empty string is 1 valid encoding

    for i in reversed(range(len(s))):
        if s[i].startswith('0'):
            cache[i] = 0
        elif i == len(s) - 1:
            cache[i] = 1
        else:
            if int(s[i:i + 2]) <= 26:
                cache[i] = cache[i + 2]
            cache[i] += cache[i + 1]
    return cache[0]
```

© Daily Coding Problem 2019 Privacy Policy Terms of Service Press