Daily Coding Problem #17

Problem

This problem was asked by Google.

Suppose we represent our file system by a string in the following manner:

The string `"dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext"` represents:

```
dir
    subdir1
    subdir2
        file.ext
```

The directory `dir` contains an empty sub-directory `subdir1` and a sub-directory `subdir2` containing a file `file.ext`.

The string `"dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext"` represents:

```
dir
    subdir1
        file1.ext
        subsubdir1
    subdir2
        subsubdir2
            file2.ext
```

The directory `dir` contains two sub-directories `subdir1` and `subdir2`. `subdir1` contains a file `file1.ext` and an empty second-level sub-directory `subsubdir1`. `subdir2` contains a second-level sub-directory `subsubdir2` containing a file `file2.ext`.

We are interested in finding the longest (number of characters) absolute path to a file within our file system. For example, in the second example above, the longest absolute path is `"dir/subdir2/subsubdir2/file2.ext"`, and its length is 32 (not including the double quotes).

Given a string representing the file system in the above format, return the length of the longest absolute path to a file in the abstracted file system. If there is no file in the system, return 0.

Note:

The name of a file contains at least a period and an extension.

The name of a directory or sub-directory will not contain a period.

Solution

There are two steps in solving this question: we must first parse the string representing the file system and then get the longest absolute path to a file.

Step 1: Parsing the file system

Ideally, we would initially parse the string given into a dictionary of some sort. That would mean a string like:

```
dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext
```

would become:

```
{
    "dir": {
        "subdir1": {
            "file1.ext": True,
            "subsubdir1": {}
        },
        "subdir2": {
            "subsubdir2": {
                "file2.ext": True
            }
        }
    }
}
```

where each key with a dictionary as its value represents a directory, and a key with `True` as its value represents an actual file.

To achieve this, we can first split the string by the newline character, meaning each item in our array represents a file or directory. Then, we create an empty dictionary to represent our parsed file system and traverse the file system on each entry. We keep track of the last path we've seen so far in `current_path` because we may need to return to some level in that path, depending on the number of tabs. Once we are at the correct place to put down the new directory or file, we check the name for a `.` and set the correct value to either `True` (if file) or `{}` (if directory).

```
def build_fs(input):
    fs = {}
    files = input.split('\n')

    current_path = []
    for f in files:
        indentation = 0
        while '\t' in f[:2]:
            indentation += 1
            f = f[1:]

        current_node = fs
        for subdir in current_path[:indentation]:
            current_node = current_node[subdir]

        if '.' in f:
            current_node[f] = True
        else:
            current_node[f] = {}

        current_path = current_path[:indentation]
        current_path.append(f)

    return fs
```

Step 2: Computing the longest path

After we've constructed a native representation of the file system, we can write a fairly straightforward recursive function that takes the current root, recursively calculates the `longest_path` of all the subdirectories and files under the root, and returns the longest one. Remember that since we specifically want the longest path to a file to discard any paths that do not have a `.` in them. And if there are no paths starting at this root, then we can simply return the empty string.

```
def longest_path(root):
    paths = []
    for key, node in root.items():
        if node == True:
            paths.append(key)
        else:
            paths.append(key + '/' + longest_path(node))
    # filter out unfinished paths
    paths = [path for path in paths if '.' in path]
    if paths:
        return max(paths, key=lambda path:len(path))
    else:
        return ''
```

Step 3: Putting it together

Now that the hard part is done, we just need to put the two together:

```
def longest_absolute_path(s):
    return len(longest_path(build_fs(s)))
```

This runs in O(n), since we iterate over the input string twice to build the file system, and then in the worst case we go through the string again to compute the longest path.