Longest Absolute File Path

Suppose we abstract 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 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 `.` and an extension.
- The name of a directory or sub-directory will not contain a `.`.

Time complexity required: `O(n)` where `n` is the size of the input string.

Notice that `a/aa/aaa/file1.txt` is not the longest file path, if there is another path `aaaaaaaaaaaaaaaaaaaaa/sth.png`.

Solution 1

```
public int lengthLongestPath(String input) {
        Deque<Integer> stack = new ArrayDeque<>();
        stack.push(0); // "dummy" length
        int maxLen = 0;
        for(String s:input.split("\n")){
            int lev = s.lastIndexOf("\t")+1; // number of "\t"
            while(lev+1<stack.size()) stack.pop(); // find parent
            int len = stack.peek()+s.length()-lev+1; // remove "/t", add"/"
            stack.push(len);
            // check if it is file
            if(s.contains(".")) maxLen = Math.max(maxLen, len-1);
        }
        return maxLen;
    }
```

An even shorter and faster solution using array instead of stack:

```
public int lengthLongestPath(String input) {
    String[] paths = input.split("\n");
    int[] stack = new int[paths.length+1];
    int maxLen = 0;
    for(String s:paths){
        int lev = s.lastIndexOf("\t")+1, curLen = stack[lev+1] = stack[lev]+s.len
gth()-lev+1;
        if(s.contains(".")) maxLen = Math.max(maxLen, curLen-1);
    }
    return maxLen;
}
```

written by sky-xu original link here

Solution 2

The number of tabs is my `depth` and for each depth I store the current path length.

```
def lengthLongestPath(self, input):
    maxlen = 0
    pathlen = {0: 0}
    for line in input.splitlines():
        name = line.lstrip('\t')
        depth = len(line) - len(name)
        if '.' in name:
            maxlen = max(maxlen, pathlen[depth] + len(name))
        else:
            pathlen[depth + 1] = pathlen[depth] + len(name) + 1
    return maxlen
```

written by StefanPochmann original link here

Solution 3

```
public:
    int lengthLongestPath(string input) {
        int maxi=0,count=0,ln=1;
        bool isFile=false;
        vector<int> level(200);
        level[0]=0;
        for(int i=0,fin=input.size();i<fin;++i){
            //find which level
            while(input[i]=='\t'){
                ++ln;++i;
            }
            //read file name
            while(input[i]!='\n'&&i<fin){
                if(input[i]=='.')isFile=true;
                ++count;++i;
            }
            //calculate
            if(isFile){
                maxi=max(maxi,level[ln-1]+count);
            }
            else{
                level[ln]=level[ln-1]+count+1;// 1 means '/'
            }
            //reset
            count=0;ln=1;isFile=false;
        }
        return maxi;
    }
};
```

written by XORNOTXOR original link here