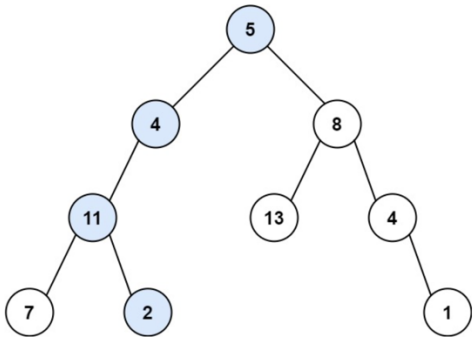
2023-10-07 - Handout – Graphs (DFS and BFS)

Q1. Path Sum

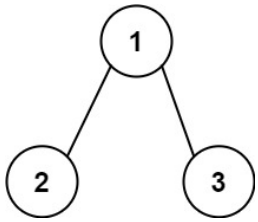
Link: https://leetcode.com/problems/path-sum/

Given the `root` of a binary tree and an integer `targetSum`, return `true` if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

A **leaf** is a node with no children.




```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
```



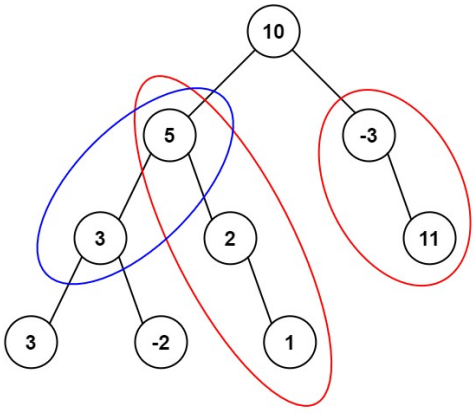

```
Input: root = [1,2,3], targetSum = 5
Output: false
```

Q2. Path Sum III

Link: https://leetcode.com/problems/path-sum-iii/

Given the `root` of a binary tree and an integer `targetSum`, return *the number of paths where the sum of the values along the path equals* `targetSum`.

The path does not need to start or end at the root or a leaf, but it must go downwards (i.e., traveling only from parent nodes to child nodes).




```
Input: root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8
Output: 3
```

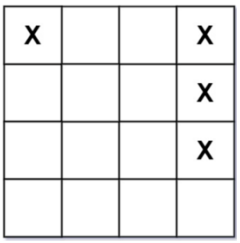
Q3. Battleships in a Board

Link: https://leetcode.com/problems/battleships-in-a-board/

Given an `m x n` matrix `board` where each cell is a battleship `'X'` or empty `'.'`, return *the number of the **battleships** on* `board`.

Battleships can only be placed horizontally or vertically on `board`. In other words, they can only be made of the shape `1 x k` (`1` row, `k` columns) or `k x 1` (`k` rows, `1` column), where `k` can be of any size. At least one horizontal or vertical cell separates between two battleships (i.e., there are no adjacent battleships).

Example:

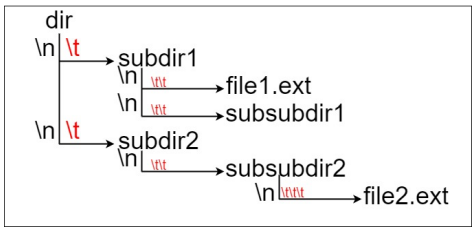



```
Input: board = [["X",".",".","X"],[".",".",".","X"],[".",".",".","X"]]
Output: 2
```

Q4. Longest Absolute File Path

Link: https://leetcode.com/problems/longest-absolute-file-path/

Suppose we have a file system that stores both files and directories. An example of one system is represented in the following picture:



Here, we have `dir` as the only directory in the root. `dir` contains two subdirectories, `subdir1` and `subdir2`. `subdir1` contains a file `file1.ext` and subdirectory `subsubdir1`. `subdir2` contains a subdirectory `subsubdir2`, which contains a file `file2.ext`.

In text form, it looks like this (with → representing the tab character):

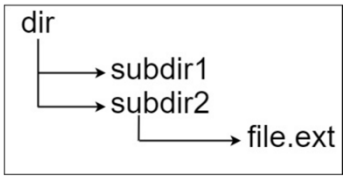
```
dir
→ subdir1
→ → file1.ext
→ → subsubdir1
→ subdir2
→ → subsubdir2
→ → → file2.ext
```

If we were to write this representation in code, it will look like this:
`"dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext"`.
Note that the `'\n'` and `'\t'` are the new-line and tab characters.

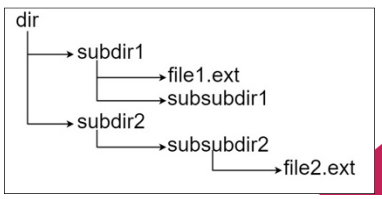
Every file and directory has a unique **absolute path** in the file system, which is the order of directories that must be opened to reach the file/directory itself, all concatenated by `'/'s`. Using the above example, the **absolute path** to `file2.ext` is `"dir/subdir2/subsubdir2/file2.ext"`. Each directory name consists of letters, digits, and/or spaces. Each file name is of the form `name.extension`, where `name` and `extension` consist of letters, digits, and/or spaces.

Given a string `input` representing the file system in the explained 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 that the testcases are generated such that the file system is valid and no file or directory name has length 0.




```
Input: input = "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext"
Output: 20
```




```
Input: input =
"dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\
t\tsubsubdir2\n\t\t\tfile2.ext"
Output: 32
```