**难度: 简单**

每当用户执行变更文件夹操作时，LeetCode 文件系统都会保存一条日志记录。

下面给出对变更操作的说明：

"../" ：移动到当前文件夹的父文件夹。如果已经在主文件夹下，则 继续停留在当前文件夹 。

"./" ：继续停留在当前文件夹。

"x/" ：移动到名为 x 的子文件夹中。题目数据 保证总是存在文件夹 x 。

给你一个字符串列表 logs ，其中 logs[i] 是用户在 ith 步执行的操作。

文件系统启动时位于主文件夹，然后执行 logs 中的操作。

执行完所有变更文件夹操作后，请你找出 返回主文件夹所需的最小步数 。

示例 1：

输入：logs = ["d1/","d2/","../","d21/","./"]

输出：2

解释：执行 "../" 操作变更文件夹 2 次，即可回到主文件夹

示例 2：

输入：logs = ["d1/","d2/","./","d3/","../","d31/"]

输出：3

示例 3：

输入：logs = ["d1/","../","../","../"]

输出：0

提示：

1 <= logs.length <= 103

2 <= logs[i].length <= 10

logs[i] 包含小写英文字母，数字，'.' 和 '/'

logs[i] 符合语句中描述的格式

文件夹名称由小写英文字母和数字组成

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/crawler-log-folder

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

**思路1(使用栈):**

代码有注释

|  |
| --- |
| public int minOperations(String[] logs) {  //创建一个栈  Deque<String> stack = new ArrayDeque<>();  for (String log : logs) {  //如果是./还是当前文件夹,于是什么都不动  if (log.equals("./")) {  continue;  }  //如果不是../则移动到子目录.于是入栈  if (!log.equals("../")) {  stack.push(log);  }else {  //说明是../,判断栈是否为空,不为空,就退栈(返回上级)  if (!stack.isEmpty()){  stack.pop();  }  }  }  //计数  int count = 0;  //循环退栈统计  while (!stack.isEmpty()) {  stack.pop();  count++;  }  return count;  } |

**思路2(不使用栈):**

代码有注释

|  |
| --- |
| public int minOperations2(String[] logs) {  //计数  int count = 0;  for (int i = 0; i < logs.length; i++) {  //如果是./还是当前文件夹,于是什么都不动  if (logs[i].equals("./")) {  continue;  }  //如果不是../则移动到子目录. 深度+1  if (!logs[i].equals("../")) {  count++;  }else {  //说明是../,判断当前深度是否是在当前文件夹,大于0表示有深度,于是返回上级  if (count > 0){  count--;  }  }  }  for (String log : logs) {  }  return count;  } |