

Leetcode Problem #1598 -Crawler Log Folder

The Leetcode file system keeps a log each time some user performs a *change folder* operation.

The operations are described below:

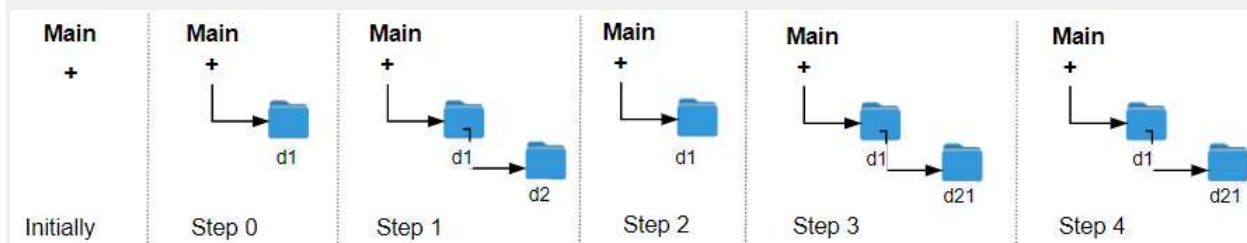
- `"../"` : Move to the parent folder of the current folder. (If you are already in the main folder, **remain in the same folder**).
- `"./"` : Remain in the same folder.
- `"x/"` : Move to the child folder named x (This folder is **guaranteed to always exist**).

You are given a list of strings logs where logs[i] is the operation performed by the user at the ith step.

The file system starts in the main folder, then the operations in logs are performed.

Return the *minimum number of operations needed to go back to the main folder after the change folder operations*.

Example 1:

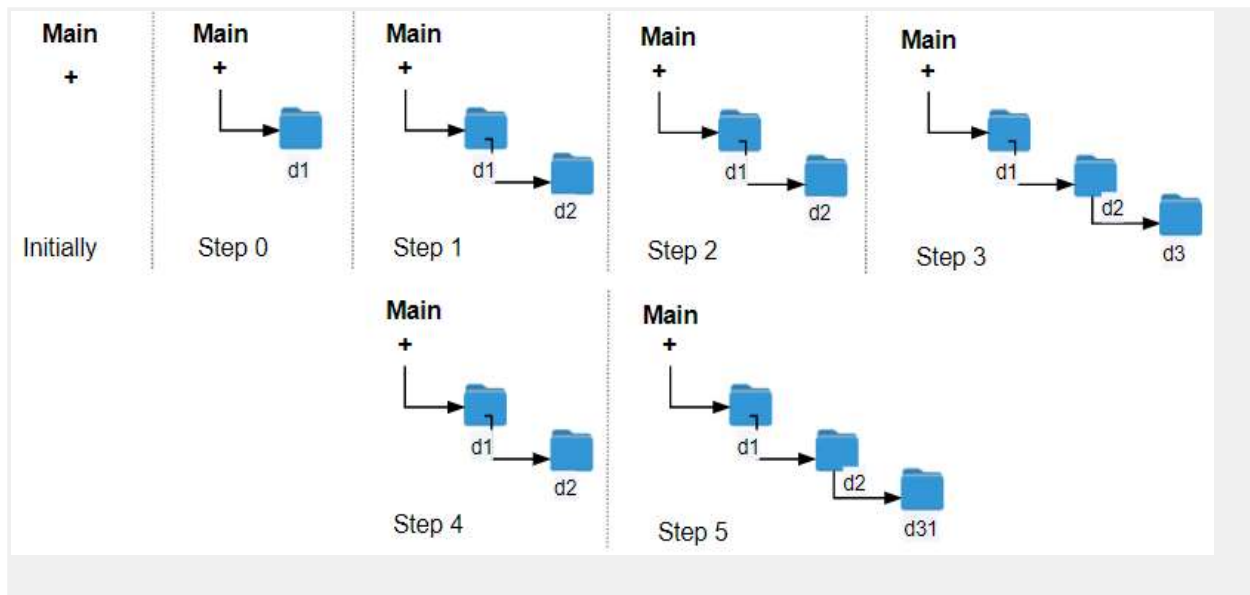


Input: logs = ["d1/", "d2/", "../", "d21/", "../"]

Output: 2

Explanation: Use this change folder operation `"../"` 2 times and go back to the main folder.

Example 2:



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

Output: 3

Example 3:

Input: logs = ["d1/", "../", "../", "../"]

Output: 0

Solution

```
class Solution {
public:
    int minOperations(vector<string>& logs) {
        stack<string>st;

        for(int i=0;i<logs.size();i++){
            string log=logs[i];

            if(log=="../"){
                if(!st.empty()){
                    st.pop();
                }
            }
        }
    }
};
```

```
        }else if(log=="./"){  
  
        }else{  
            st.push(log);  
        }  
    }  
    return st.size();  
  
    }  
};
```

Leetcode Problem #20 -Valid Parentheses

Given a string *s* containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: *s* = "()"

Output: true

Example 2:

Input: *s* = "()[]{}"

Output: true

Example 3:

Input: *s* = "([]"

Output: false

Example 4:

Input: s = "([])"

Output: true

Solution

```
class Solution {
public:
    bool isValid(string s) {
        stack<char>st;

        for(int i=0;i<s.length();i++){
            char ch=s[i];
            if(st.empty() || ch == '(' || ch == '{' || ch == '['){
                st.push(ch);
                continue;
            }

            if(ch==')' && st.top()=='('){
                st.pop();
            }else if(ch == '}' && st.top() =='{'){
                st.pop();
            }else if(ch == ']' && st.top() == '['){
                st.pop();
            }else{
                return false;
            }
        }
        return st.empty();
    }
};
```

Leetcode Problem #155 -Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function.

Example 1:

Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]  
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Explanation

```
MinStack minStack = new MinStack();  
minStack.push(-2);  
minStack.push(0);  
minStack.push(-3);  
minStack.getMin(); // return -3  
minStack.pop();  
minStack.top();    // return 0  
minStack.getMin(); // return -2
```

Solution

```
class MinStack {
private:
    stack<int> mainStack;
    stack<int> minStack;

public:
    MinStack() {
    }

    void push(int val) {
        mainStack.push(val);
        if (minStack.empty() || val <= minStack.top()) {
            minStack.push(val);
        }
    }

    void pop() {
        if (mainStack.top() == minStack.top()) {
            minStack.pop();
        }
        mainStack.pop();
    }

    int top() {
        return mainStack.top();
    }

    int getMin() {
        return minStack.top();
    }
};
```

Leetcode Problem #1475 - Final Prices with a Special Discount in a shop

You are given an integer array `prices` where `prices[i]` is the price of the i^{th} item in a shop.

There is a special discount for items in the shop. If you buy the i^{th} item, then you will receive a discount equivalent to `prices[j]` where j is the minimum index such that $j > i$ and `prices[j] <= prices[i]`. Otherwise, you will not receive any discount at all.

Return an integer array `answer` where `answer[i]` is the final price you will pay for the i^{th} item of the shop, considering the special discount.

Example 1:

Input: `prices = [8,4,6,2,3]`

Output: `[4,2,4,2,3]`

Explanation:

For item 0 with `price[0]=8` you will receive a discount equivalent to `prices[1]=4`, therefore, the final price you will pay is $8 - 4 = 4$.

For item 1 with `price[1]=4` you will receive a discount equivalent to `prices[3]=2`, therefore, the final price you will pay is $4 - 2 = 2$.

For item 2 with `price[2]=6` you will receive a discount equivalent to `prices[3]=2`, therefore, the final price you will pay is $6 - 2 = 4$.

For items 3 and 4 you will not receive any discount at all.

Example 2:

Input: `prices = [1,2,3,4,5]`

Output: `[1,2,3,4,5]`

Explanation: In this case, for all items, you will not receive any discount at all.

Example 3:

Input: `prices = [10,1,1,6]`

Output: `[9,0,1,6]`

Solution

```
class Solution {  
public:  
    vector<int> finalPrices(vector<int>& prices) {
```

```
int n = prices.size();
stack<int> st;
vector<int> result(prices);

for (int i = 0; i < n; i++) {
    while (!st.empty() && prices[i] <= prices[st.top()]) {
        int idx = st.top();
        st.pop();
        result[idx] = prices[idx] - prices[i];
    }
    st.push(i);
}

return result;
}
```

```
};
```



www.tpcglobal.in