

class Solution(object):

def simplifyPath(self, path):

"""

:type path: str

:rtype: str

"""

stack = []

parts = path.split('/')

for part in parts:

if part == '' or part == '.':

continue

elif part == '..':

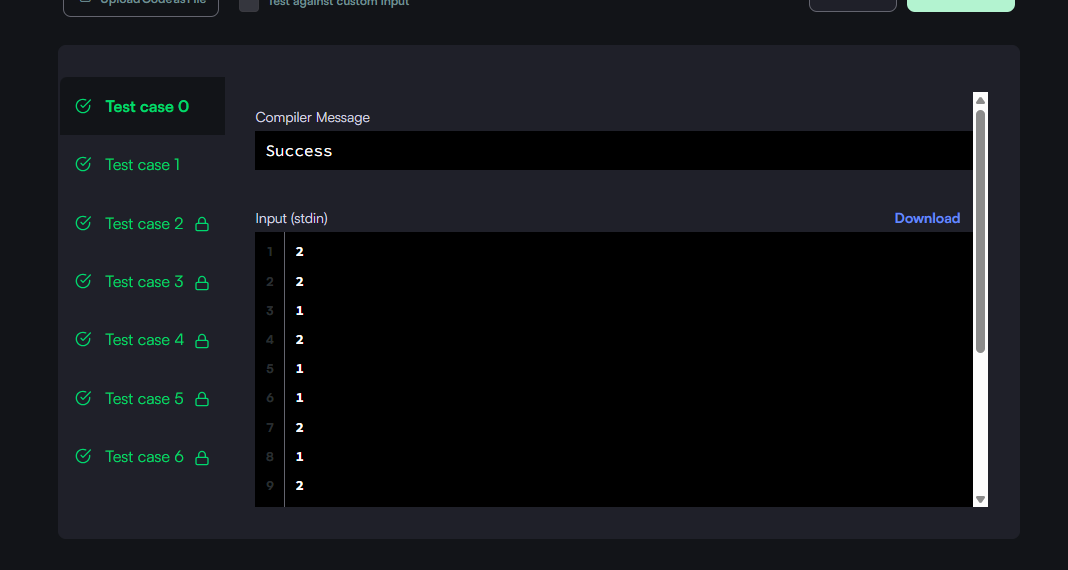
if stack:

stack.pop()

else:

stack.append(part)

return '/' + '/'.join(stack)



#!/bin/python3

import os

import sys

class SinglyLinkedListNode:

    def \_\_init\_\_(self, node\_data):

        self.data = node\_data

        self.next = None

class SinglyLinkedList:

    def \_\_init\_\_(self):

        self.head = None

        self.tail = None

    def insert\_node(self, node\_data):

        node = SinglyLinkedListNode(node\_data)

        if not self.head:

            self.head = node

        else:

            self.tail.next = node

        self.tail = node

def print\_singly\_linked\_list(node, sep, fptr):

    while node:

        fptr.write(str(node.data))

        node = node.next

        if node:

            fptr.write(sep)

# Complete the compare\_lists function below.

#

# For your reference:

#

# SinglyLinkedListNode:

#     int data

#     SinglyLinkedListNode next

#

#

def compare\_lists(llist1, llist2):

    while llist1 and llist2:

        if llist1.data != llist2.data:

            return 0

        llist1 = llist1.next

        llist2 = llist2.next

    # If both lists reached the end, they are equal

    if llist1 is None and llist2 is None:

        return 1

    else:

        return 0

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    tests = int(input())

    for tests\_itr in range(tests):

        llist1\_count = int(input())

        llist1 = SinglyLinkedList()

        for \_ in range(llist1\_count):

            llist1\_item = int(input())

            llist1.insert\_node(llist1\_item)

        llist2\_count = int(input())

        llist2 = SinglyLinkedList()

        for \_ in range(llist2\_count):

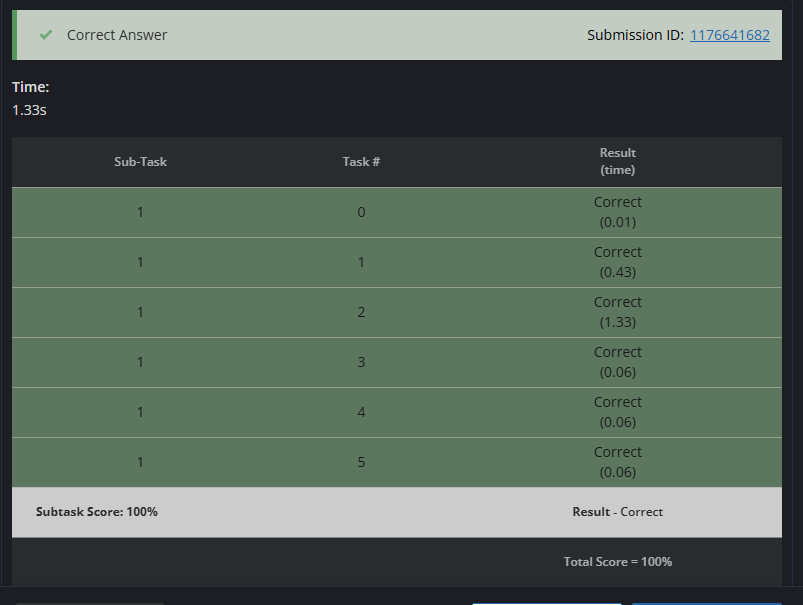
            llist2\_item = int(input())

            llist2.insert\_node(llist2\_item)

        result = compare\_lists(llist1.head, llist2.head)

        fptr.write(str(int(result)) + '\n')

    fptr.close()



# cook your dish here

def min\_operations\_to\_sort(S):

ops = 0

for i in range(len(S) - 1):

if S[i] == '1' and S[i + 1] == '0':

ops += 1

return ops

# Read input

T = int(input()) # Number of test cases

for \_ in range(T):

N = int(input()) # Length of binary string (not used directly)

S = input().strip() # Binary string

print(min\_operations\_to\_sort(S))