A string S consisting of N characters is considered to be *properly nested* if any of the following conditions is true:

* S is empty;
* S has the form "(U)" or "[U]" or "{U}" where U is a properly nested string;
* S has the form "VW" where V and W are properly nested strings.

For example, the string "{[()()]}" is properly nested but "([)()]" is not.

Write a function:

def solution(S)

that, given a string S consisting of N characters, returns 1 if S is properly nested and 0 otherwise.

For example, given S = "{[()()]}", the function should return 1 and given S = "([)()]", the function should return 0, as explained above.

Write an **efficient** algorithm for the following assumptions:

* N is an integer within the range [0..200,000];
* string S consists only of the following characters: "(", "{", "[", "]", "}" and/or ")".

def solution(S):

if len(S)== 0:

return 1

if len(S) % 2 != 0:

return 0

dic1 = { "[" : "]" , "(" : ")" , "{" : "}" }

lis1 = []

for i in S:

if i in "({[":

lis1.append(i)

elif lis1 and i == dic1[ lis1[-1] ]:

lis1.pop()

else:

return 0

return 1 if len(lis1) ==0 else 0

S = '{[()()]}'

car1 = solution(S)

print(car1)

or

def solution(S):

stack = []

for s in S:

if len(stack)==0:

stack.append(s)

elif s=='}' and stack[len(stack)-1]== '{':

stack.pop(len(stack)-1)

elif s == ')' and stack[len(stack) - 1] == '(' :

stack.pop(len(stack) - 1)

elif s == ']' and stack[len(stack) - 1] == '[':

stack.pop(len(stack) - 1)

else:

stack.append(s)

if len(stack)>0:

return 0

else:

return 1

P1 = "{[()()]}"

car1 = solution(P1)

print(car1)