'''

There is a String S given to a student of even length N whose indexes start from 0.

The number of open brackets '[' and close brackets ']' in the string S are equal.

Students have to balance the string S. The string is balanced if and only if

-It is the empty string, or

-It can be written as XY, where both X and Y are balanced strings, or

-It can be written as [Z], where Z is a balanced string.

-You may swap the brackets at any two indices any number of times.

Return the minimum number of swaps to make string balanced.

Input Format:

-------------

Line-1: A string S, contains open '[' and close ']' brackets

Output Format:

--------------

An integer result, number of swaps

Sample Input-1:

---------------

][

Sample Output-1:

----------------

1

Explanation:

------------

String is balanced by swapping index 0 with index 1.

The resultant string is "[]".

Sample Input-2:

---------------

[][][][][]

Sample Output-2:

----------------

0

Explanation:

------------

You can do the following steps to make the string balanced:

- Swap index 0 with index 4. s = "[]][][[]".

- Swap index 1 with index 5. s = "[[][]][]".

The resulting string is "[[][]][]".

Sample Input-3:

---------------

]]][[[[]

Sample Output-3:

----------------

2

Explanation:

------------

The string is already balanced

'''

s=[]

n=input()

count=0

for i in n:

if(i=='['):

s.append(i)

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

s.pop()

else:

count+=1

print((count+1)//2)

There is an integer array of N heights. You have to pick certain number of heights

from the array everytime. Find the maximum sum of all the heights picked

everytime that meet the following conditions:

- The length of the subarray is L, and

- All the heights of the subarray picked are distinct.

Return the maximum subarray sum of all the height subarrays that meet the

conditions. If no heights subarray meets the conditions, return 0.

- A subarray is a contiguous non-empty sequence of elements within an array.

Input Format:

-------------

Line-1: Two space separated integers, N and L

Line-2: N space separated integers, height[].

Output Format:

--------------

Print an integer result.

Sample Input-1:

---------------

7 3

7 7 7 1 5 4 2

Sample Output-1:

----------------

13

Explanation:

------------

The subarrays of length 3 of height array are:

- [7,7,7] this subarray does not meet the requirement because the 7 is repeated.

- [7,7,1] this subarray does not meet the requirement because the 7 is repeated.

- [7,1,5] this subarray meets the requirements and its sum is 13.

- [1,5,4] this subarray meets the requirements and its sum is 10.

- [5,4,2] this subarray meets the requirements and its sum is 11.

Return 13 because it is the maximum subarray sum of all the subarrays

that meet the conditions

Sample Input-2:

---------------

3 3

7 7 7

Sample Output-2:

----------------

0

Explanation:

------------

The subarrays of length 3 of height array are:

- [7,7,7] which does not meet the requirements because the element 7 is repeated.

Return 0 because no subarrays meet the conditions.

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int m=sc.nextInt();

int[] arr=new int[n];

for(int i=0;i<n;i++){

arr[i]=sc.nextInt();

}

int sum=0;

int max=0;

HashSet<Integer> hs=new HashSet<>();

for(int i=0;i<n-m+1;i++){

sum=0;

for(int j=i;j<i+m;j++){

if(!hs.contains(arr[j])){

hs.add(arr[j]);

sum+=arr[j];

}

}

if(hs.size()==m){

max=Math.max(max,sum);

}

hs.clear();

}

// int max=0;

// HashMap<Integer,Integer> hs=new HashMap<>();

// int sum=0;

// for(int i=0;i<m;i++){

// if(!hs.containsKey(arr[i])){

// hs.put(arr[i],1);

// }

// else{

// hs.put(arr[i],hs.get(arr[i])+1);

// }

// sum+=arr[i];

// }

// if(hs.size()==m){

// if(sum>max){

// max=sum;

// }

// }

// else{

// sum=0;

// }

// for(int i=1;i<n-m+1;i++){

// if(hs.get(arr[i-1])==1){

// hs.remove(arr[i-1]);

// }

// else{

// hs.put(arr[i-1],hs.get(arr[i-1])-1);

// }

// sum-=arr[i-1];

// sum+=arr[i+m-1];

// if(!hs.containsKey(arr[i+m-1])){

// hs.put(arr[i+m-1],1);

// if(sum>max){

// max=sum;

// }

// }

// else{

// hs.put(arr[i+m-1],hs.get(arr[i+m-1]));

// sum=0;

// }

// System.out.println(max);

// }

System.out.println(max);

}

}

You are playing a game. There are N boxes placed in a row (0-indexed),

every box has some points. You need to play the game with the following rules:

- Initially your score is '0' points.

- Suppose the box has P points in it.

if p>0 you will gain P points, if p<0 you will loose P points.

- You are allowed to choose any number of consecutive boxes, atleast 1 box.

You are given the points in the N boxes as points[].

Your target is to maximize your score, and return the highest score possible.

Input Format:

-------------

Line-1: An integer N.

Line-2: N space separated integers, points[].

Output Format:

--------------

An integer, print the highest score.

Sample Input-1:

---------------

9

-3 1 -2 4 -2 2 3 -5 4

Sample Output-1:

----------------

7

Explanation:

------------

By selecting consecutive boxes are [4,-2,2,3] has the highest score is 7

Sample Input-2:

---------------

2

1 -2

Sample Output-2:

----------------

1

Explanation:

------------

By picking the box is [1] has the highest score is 1.

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

for(int i=0;i<n;i++){

arr[i]=sc.nextInt();

}

int max=Integer.MIN\_VALUE;

for(int i=1;i<n;i++){

int k=i;

int sum=0;

for(int j=0;j<k;j++){

sum+=arr[j];

}

if(sum>max){

max=sum;

}

for(int j=1;j<n-k+1;j++){

sum-=arr[j-1];

sum+=arr[j+k-1];

if(sum>max){

max=sum;

}

}

}

System.out.println(max);

}

}

Sam and Lam want to distribute milk to n houses in their area. The houses are

arranged in a row with label from 0 to n - 1 from left to righ,

where the i-th house is located at x = i.

Each house needs a specific amount of milk. Sam and Lam have a their own milk

and initially full. They distribute the milk in the following way:

- Sam gives milk to the houses in order from left to right, starting

from the 0th house. Lam gives milk in order from right to left, starting

from (n-1)th house.

- They begin distributing the milk simultaneously. It takes the same amount

of time to give milk to each house regardless of how much milk it needs.

- Sam/Lam must distribute milk, if they have enough in their can to fully

give it. Otherwise, they first refill their can (instantaneously), then

give the milk to the house. In case both Sam and Lam reach the same house,

the one with more milk currently in their milk can should give milk to the

house. If they have the same amount of milk, then Sam should give the milk.

Given array of house needs starting from 0 to N-1 integers, where house[i]

is the amount of milk the i-th house needs, and two integers capA and capB

representing the capacities of Sam's and Lam's milk cans respectively. Return

the number of times they have to refill to distribute milk to all the houses.

Input Format:

-------------

Line-1: 3 space separarted integers, indiactes N, CapA and capB.

Line-2: N space separated integers, house[].

Output Format:

--------------

Print an integer result.

Sample Input:

-------------

4 5 5

5 3 3 4

Sample Output:

--------------

2

Explanation:

------------

- Initially, Sam and Lam have 5 units of milk each in their cans.

- Sam gives milk to house 0, Lam gives to house 3.

- Sam and Lam now have 0 units and 1 unit of milk respectively.

- Sam has to refil can for house 1, and Lam also does not have enough milk

for house 3, so he refills his can. So, the total number of times they have

to refill 0 + 0 + 1 + 1 = 2.

Sample Input:

-------------

3 9 7

4 5 2

Sample Output:

0

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int s=sc.nextInt();

int l=sc.nextInt();

int[] arr=new int[n];

for(int i=0;i<n;i++){

arr[i]=sc.nextInt();

}

int i=0;

int j=arr.length-1;

int s1=s;

int l1=l;

int count=0;

while(i<j){

if(s1>=arr[i]){

s1-=arr[i];

}

else{

count+=1;

s1=s;

s1-=arr[i];

}

if(l1>=arr[j]){

l1-=arr[j];

}

else{

count+=1;

l1=l;

l1-=arr[j];

}

i+=1;

j-=1;

}

if(i==j){

if(s1>l1){

if(s1<arr[i]){

count+=1;

}

}

else if(l1>s1){

if(l1<arr[i]){

count+=1;

}

}

else{

if(s1<arr[i]){

count+=1;

}

}

}

System.out.println(count);

}

}