LAB 7

Access code will be annouced at 18:40.

Problem 1

In an array `A` containing only 0s and 1s, a `K`*-bit flip* consists of choosing a (contiguous) subarray of length `K` and simultaneously changing every 0 in the subarray to 1, and every 1 in the subarray to 0.

Return the minimum number of `K`-bit flips required so that there is no 0 in the array. If it is not possible, return `-1`.

Example 1:

```
Input: A = [0,1,0], K = 1
Output: 2
Explanation: Flip A[0], then flip A[2].
```

Example 2:

```
Input: A = [1,1,0], K = 2
Output: -1
Explanation: No matter how we flip subarrays of size 2, we can't make the
array become [1,1,1].
```

Example 3:

```
Input: A = [0,0,0,1,0,1,1,0], K = 3
Output: 3
Explanation:
Flip A[0],A[1],A[2]: A becomes [1,1,1,1,0,1,1,0]
Flip A[4],A[5],A[6]: A becomes [1,1,1,1,1,0,0,0]
Flip A[5],A[6],A[7]: A becomes [1,1,1,1,1,1,1,1]
```

Problem 2

Given an array `A` of integers, return the **length** of the longest arithmetic subsequence in `A`.

Recall that a *subsequence* of `A` is a list `A[i_1], A[i_2], ..., A[i_k]` with `0 <= i_1 < i_2 < ... < i_k <= A.length - 1`, and that a sequence `B` is *arithmetic* if `B[i+1] - B[i]` are all the same value (for `0 <= i < B.length - 1`).

Example 1:

```
Input: A = [3,6,9,12]
Output: 4
Explanation:
The whole array is an arithmetic sequence with steps of length = 3.
```

Example 2:

```
Input: A = [9,4,7,2,10]
Output: 3
Explanation:
The longest arithmetic subsequence is [4,7,10].
```

Example 3:

```
Input: A = [20,1,15,3,10,5,8]
Output: 4
Explanation:
The longest arithmetic subsequence is [20,15,10,5].
```

Problem 3

Zuma Game: Suppose that you are given a string, each given character represents a color. Every time you need to take out the first ball from your package, either insert it into the original sequency or aboort it. Any 3 or more consequetive balls with the same color would be canceled out. Return the minimum number of balls you need to insert to cancel out all the balls in the original sequence.

Example 1:

W 0
R 1
B 2

```
Input: "WRRBBW", "RB"
Output: -1
Explanation:
WRRBBW -> WRR[R]BBW -> WBBW -> WBB[B]W -> WW
```

Example 2:

```
Input: "WWRRBBWW", "WRBRW"
Output: 2
Explanation:
WWRRBBWW -> WWRR[R]BBWW -> WWBBWW -> WWBB[B]WW -> WWWW -> empty
```

Example 3:

```
Input: "G", "GGGGG"
Output: 2
Explanation:
G -> G[G] -> GG[G] -> empty
```

Example 3:

```
Input: "RBYYBBRRB", "YRBGB"
Output: 3
Explanation:
RBYYBBRRB -> RBYY[Y]BBRRB -> RBBBRRB -> RRRB -> B -> B[B] -> BB[B] -> empty
```