<https://leetcode.com/problems/minimum-recolors-to-get-k-consecutive-black-blocks/description/>

You are given a 0-indexed string blocks of length n, where blocks[i] is either 'W' or 'B', representing the color of the ith block. The characters 'W' and 'B' denote the colors white and black, respectively.

You are also given an integer k, which is the desired number of consecutive black blocks.

In one operation, you can recolor a white block such that it becomes a black block.

Return the minimum number of operations needed such that there is at least one occurrence of k consecutive black blocks.

**Example 1:**

**Input:** blocks = "WBBWWBBWBW", k = 7

**Output:** 3

**Explanation:**

One way to achieve 7 consecutive black blocks is to recolor the 0th, 3rd, and 4th blocksso that blocks = "BBBBBBBWBW".

It can be shown that there is no way to achieve 7 consecutive black blocks in less than 3 operations.Therefore, we return 3.

**Example 2:**

**Input:** blocks = "WBWBBBW", k = 2

**Output:** 0

**Explanation:**

No changes need to be made, since 2 consecutive black blocks already exist.Therefore, we return 0.

**Constraints:**

n == blocks.length

1 <= n <= 100

blocks[i] is either 'W' or 'B'.

1 <= k <= n

**Attempt 1: 2024-12-25**

**Solution 1: Fixed length Sliding Window (10 min)**

**We can treat the k as fixed length sliding window size, and we try to find how many existing 'B' in each window during the move from left to right, then find out the maximum 'B' count in this process, the remain count as k - maxBCount requires update 'W' to 'B' operations**

class Solution {

    public int minimumRecolors(String blocks, int k) {

        char[] chars = blocks.toCharArray();

        int maxBCount = 0;

        int BCountInWindow = 0;

        // We can treat the k as fixed length sliding window size,

        // and we try to find how many existing 'B' in each window

        // during the move from left to right, then find out the

        // maximum 'B' count in this process, the remain count as

        // k - maxBCount requires update 'W' to 'B' operations

        for(int i = 0; i < chars.length; i++) {

            if(chars[i] == 'B') {

                BCountInWindow++;

            }

            // If directly get k 'B', no operation need

            if(i == k - 1 && BCountInWindow == k) {

                return 0;

            }

            // Truncate previous 'B' when Sliding Window shift

            if(i >= k) {

                if(chars[i - k] == 'B') {

                    BCountInWindow--;

                }

            }

            maxBCount = Math.max(maxBCount, BCountInWindow);

        }

        return k - maxBCount;

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to chatGPT**

You are given a string blocks consisting of characters 'W' and 'B' (white and black blocks, respectively). You are also given an integer k. The task is to find the minimum number of recolors needed to get at least one substring of length k consisting only of black blocks.

class Solution {

public int minimumRecolors(String blocks, int k) {

int n = blocks.length();

int minRecolors = k; // Maximum recolors needed would be k if all are 'W' in a substring of size k.

int whiteCount = 0;

// Initial count of white blocks in the first window of size k

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

if (blocks.charAt(i) == 'W') {

whiteCount++;

}

}

minRecolors = whiteCount;

// Slide the window over the rest of the string

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

if (blocks.charAt(i) == 'W') {

whiteCount++;

}

if (blocks.charAt(i - k) == 'W') {

whiteCount--;

}

minRecolors = Math.min(minRecolors, whiteCount);

}

return minRecolors;

}

}

**Explanation**

Sliding Window Approach:

Use a sliding window of size k to count the number of white blocks ('W') in each substring of size k.

At each step, maintain the count of white blocks within the current window.

Initialization:

Start by calculating the number of white blocks in the first window (first k characters).

Set minRecolors to this count.

Sliding the Window:

For each subsequent character, include the new character in the window and remove the leftmost character of the previous window.

If the new character is 'W', increment the whiteCount. If the outgoing character is 'W', decrement the whiteCount.

Update Minimum Recolors:

After processing each window, update minRecolors to be the minimum value between the current minRecolors and whiteCount.

Return the Result:

At the end of the loop, minRecolors holds the minimum number of recolors needed.

**Refer to**

<https://leetcode.com/problems/minimum-recolors-to-get-k-consecutive-black-blocks/solutions/2454159/c-using-sliding-window-very-simple-and-easy-to-understand-solution/>

Sliding window of size K and minimize the no. of 'W' in window

When found a 'W' in forward pointer, increament counter

when found 'W' in back pointer, decreament counter

Take the minimum of counter as the answer

int minimumRecolors(string blocks, int k) {

int back = 0, front = 0, count\_w = 0, ans = INT\_MAX;

while(front < blocks.size()){

if(blocks[front] == 'W'){ count\_w++; }

if(front - back + 1 == k){

ans = min(ans, count\_w);

if(blocks[back] == 'W') count\_w--;

back++;

}

front++;

}

return ans;

}