995. Minimum Number of K Consecutive Bit Flips

Leetcode Link

Problem Explanation

Given an array / list which contains only binary digits, (1s and 0s), we want to perform a K-bit flip operation in the list. A K-bit flip operation involves choosing a subarray or sublist of length K and simultaneously changing all 0s in the subarray to 1, and all the 1s in the subarray to 0.

The task is to find the minimum number of K-bit flips required to make all the numbers in the array / list as 1. If it is impossible to do so, we should return -1.

For example, if our array is [0,1,0] and K equals 1, the minimum number of 1-bit flips needed to make all the array elements 1 is 2 (flip 0 at index 0, and 0 at index 2).

Approach

This problem can be solved using a simple technique of flipping the subarrays. We traverse the array one by one and look ahead of the k-length subarray which needs to be flipped. We flip the 1s and 0s if found in the current k-length subarray and keep track of the total number of flips, to be returned as the result. One boundary condition to be kept in mind is whether the flip operation is feasible or not i.e. if the remaining length of the array is less than k, then flipping a k-length subarray is not possible, return -1.

Let's walk through an example. Take the array [0,1,0,1,0] and let K be 1. At index 0, we flip the bit from 0 to 1 and record the flip. We do the same at indices 2 and 4. At the end, our array is fully converted to 1s and we have done three 1-bit flips in total.

Python Solution

```python
class Solution:
    def minKBitFlips(self, A, K):
        n = len(A)
        flip = ans = 0
        is_flip = [0] * n
        for i in range(n):
            if i >= K and A[i - K] == 2:
                flip -= is_flip[i - K]
            if flip % 2 == A[i]:
                if i + K > n:
                    return -1
                A[i] = 2
                is_flip[i] = 1
                flip += 1
            ans += is_flip[i]
        return ans
```

Java Solution

```java
class Solution {
    public int minKBitFlips(int[] A, int K) {
        int n = A.length;
        int[] flip = new int[n];
        int ans = 0, revCnt = 0;
        for (int i = 0; i < n; ++i) {
            if (i >= K) {
                revCnt ^= flip[i - K];
            }
            if (revCnt % 2 == A[i]) {
                if (i + K > n) {
                    return -1;
                }
                flip[i] = 1;
                revCnt ^= 1;
                ++ans;
            }
        }
        return ans;
    }
}
```

JavaScript Solution

```javascript
var minKBitFlips = function(A, K) {
    let flip = 0, res = 0, arr = new Array(A.length).fill(0);
    for(let i=0;i<A.length;i++) {
        if(i>=K) flip^=arr[i-K];
        if(flip%2===A[i]) {
            if(i+K>A.length) return -1;
            flip^=1;
            arr[i]=1;
            res++;
        }
    }
    return res;
};
```

C++ Solution

```cpp
class Solution {
public:
    int minKBitFlips(vector<int>& A, int K) {
        vector<int> flip(A.size());
        int flipped = 0;
        int res = 0;
        for(int i = 0; i < A.size(); ++i){
            if(i >= K)
                flipped ^= flip[i-K];
            if(flipped%A[i] == 0){
                if(i+K > A.size())
                    return -1;
                flipped ^= 1;
                ++res;
                flip[i] = 1;
            }
        }
        return res;
    }
};
```

C# Solution

```csharp
public class Solution {
    public int MinKBitFlips(int[] A, int K) {
        int flips=0, result=0,n=A.Length;
        int[] isFlipped=new int[n];
        for(int i=0;i<n;i++){
            if(i>=K)
                flips^=isFlipped[i-K];
            if(flips%2==A[i]){
                if(i+K>n)
                    return -1;
                isFlipped[i]=1;
                flips^=1;
                result++;
            }
        }
        return result;
    }
}
```

Solution Explanation

Taking a look at the implementations in various languages, we can deduce that all solutions follow a similar approach.

Firstly, an auxiliary array (`flip[]` or `is_flip[]`) of the same length as the input array / list is initialized. This array is used to keep track of which indices of the input array / list have been flipped.

Next, the input array / list is iterated over. For each index, if it lies beyond the Kth position, it's checked whether the element at the current index minus K, was flipped or not. If it was, the flip count is decreased.

Then, the current number of flips performed is checked against the number at the current index. If they are equal, the solution checks whether there are enough elements beyond the current index to flip K bits. If not, -1 is returned indicating that no solution exists. Else, the current index is marked as flipped, the flip count is incremented and the number of flips performed is increased.

Finally, after the traversal is done, the total number of flips performed is returned.

In these solutions, the time complexity is O(N) where N is the length of the array / list A. This happens because each index of the array / list is explored once. The space complexity is also O(N) as an additional array / list of size N is used to keep track of flips.

This problem can serve as a good practice for understanding bit manipulation, array / list management and problem-solving using a greedy algorithm.