2516. Take K of Each Character From Left and Right

You are given a string `s` consisting of the characters `'a'`, `'b'`, and `'c'` and a non-negative integer `k`. Each minute, you may take either the **leftmost** character of `s`, or the **rightmost** character of `s`.

Return *the* ***minimum*** *number of minutes needed for you to take* ***at least*** `k` *of each character, or return* `-1` *if it is not possible to take* `k` *of each character.*

Example 1:

```
Input: s = "aabaaaacaabc", k = 2
Output: 8
Explanation:
Take three characters from the left of s. You now have two 'a' characters, and one 'b' character.
Take five characters from the right of s. You now have four 'a' characters, two 'b' characters, and two 'c' characters.
A total of 3 + 5 = 8 minutes is needed.
It can be proven that 8 is the minimum number of minutes needed.
```

Example 2:

```
Input: s = "a", k = 1
Output: -1
Explanation: It is not possible to take one 'b' or 'c' so return -1.
```

Constraints:

- `1 <= s.length <= 10^5`
- `s` consists of only the letters `'a'`, `'b'`, and `'c'`.
- `0 <= k <= s.length`

2516. Take K of Each Character From Left and Right

```
/*
   Naive
   Time complexity:  O(n^2)
   Space complexity: O(1)
*/
typedef std::pair<int,char> ic;
typedef std::vector<ic> vic;
typedef std::vector<int> vi;

class Solution {
   public:

      int takeCharacters(std::string s, int k){
         int n=s.size();

         if (k==0) return 0;

         long long a=0,b=0,c=0;
         for(auto& letter: s){
            a+=int(letter=='a');
            b+=int(letter=='b');
            c+=int(letter=='c');
         }

         if(a<k || b<k || c<k) return -1;

         int ans=INT_MAX;
         long long al=0,bl=0,cl=0;
```

```
        // For each substring from 0 to l
        for(int l=0;l<n;++l){
            // compute the count of 'a's , 'b's and 'c's in the substring s[0..l]
            al+=int(s[l]=='a'); // number of 'a's
            bl+=int(s[l]=='b'); // number of 'b's
            cl+=int(s[l]=='c'); // number of 'c's

            // Perform a linear search in the remaining subarray for the
            // remaining number of 'a's , 'b's and 'c's
            // Starting by the last index...
            int r=n-1;

            long long ar=0,br=0,cr=0;

            // ...extend the window to the left, until the count of 'a's , 'b's and 'c's
            // each one, became >=k
            // Take the farthest index from the right side
            // substring s[r..n-1] found
            while(r>l && (al+ar<k || bl+br<k || cl+cr<k)){
                ar+=int(s[r]=='a');
                br+=int(s[r]=='b');
                cr+=int(s[r]=='c');
                r--;
            }
            // Minimize the answer:
            // If count of 'a's , 'b's and 'c's in the substring s[0..l] are greater or
            // equal to k, just take the length of the substring s[0..l], which is l+1
            if(al>=k && bl>=k && cl>=k) ans=std::min(ans,l+1);

            // If count of 'a's , 'b's and 'c's in the substring s[r..n-1] are greater or
            // equal to k, just take the length of the substring s[r..n-1], which is n-r-1
            if(ar>=k && br>=k && cr>=k) ans=std::min(ans,n-r-1);

            // If count of 'a's , 'b's and 'c's in both substrings s[0..l] and s[r..n-1] are greater or
            // equal to k, just take the length both of them, which is (l+1)+(n-r-1)
            if(al+ar>=k && bl+br>=k && cl+cr>=k) ans=std::min(ans,(l+1)+(n-r-1));
        }
        return ans;
    }
};
```

2516. Take K of Each Character From Left and Right

```
/*
   Binary search
   Time complexity: O(n log n)
   Space complexity: O(6n)=o(n)
*/
typedef std::pair<int,char> ic;
typedef std::vector<ic> vic;
typedef std::vector<int> vi;

class Solution {
   public:
      // Precompute the numbers of a character `c` from right to left
      // in a string `s`
      vi preprocess(std::string& s, char c){
         int n=s.size();
         vi res(n);
         res[n-1]=int(s[n-1]==c);
         for(int i=n-2;i>=0;--i){
            res[i]=res[i+1]+int(s[i]==c);
         }
         return res;
      }
      // Search the lower bound of the integer `cnt` in the array `arr` of size n
      // from index `lo` to index `hi`
      int bs(vi& arr,int lo,int hi,int n,int cnt){
         if(cnt<=0) return hi;
         int i=std::lower_bound(arr.begin(),arr.begin()+(n-lo),cnt)-arr.begin();
         return i<n-lo?hi-i:n;
      }
```

```
int takeCharacters(std::string s, int k){
    int n=s.size();

    if (k==0) return 0;

    long long a=0,b=0,c=0;
    for(auto& letter: s){
        a+=int(letter=='a');
        b+=int(letter=='b');
        c+=int(letter=='c');
    }

    if(a<k || b<k || c<k) return -1;

    // Precompute the numbers of 'a's , 'b's and 'c's in s
    vi A=preprocess(s,'a');
    vi B=preprocess(s,'b');
    vi C=preprocess(s,'c');

    // Save precomputed arrays before reversing
    vi AA=A;
    vi BB=B;
    vi CC=C;

    // Reverse to be able to perfom a lower bound searching
    std::reverse(AA.begin(),AA.end());
    std::reverse(BB.begin(),BB.end());
    std::reverse(CC.begin(),CC.end());

    int ans=INT_MAX;
    long long al=0,bl=0,cl=0;
```

```
        // For each substring from 0 to l
        for(int l=0;l<n;++l){
            // compute the count of 'a's , 'b's and 'c's in the substring s[0..l]
            al+=int(s[l]=='a'); // number of 'a's
            bl+=int(s[l]=='b'); // number of 'b's
            cl+=int(s[l]=='c'); // number of 'c's

            // Perform a binary search in the remaining subarray for the
            // remaining number of 'a's , 'b's and 'c's
            int lo=l,hi=n-1;

            int ia=bs(AA,lo,hi,n,k-al); // index of the remaining numner 'a's (k-al)
            int ib=bs(BB,lo,hi,n,k-bl); // index of the remaining numner 'b's (k-bl)
            int ic=bs(CC,lo,hi,n,k-cl); // index of the remaining numner 'c's (k-cl)

            // Take the farthest index from the right side
            // substring s[r..n-1] found
            int r=std::min({ia,ib,ic});

            // compute the count of 'a's , 'b's and 'c's in the substring s[r..n-1]
            int ar=A[r]; // number of 'a's
            int br=B[r]; // number of 'b's
            int cr=C[r]; // number of 'c's

            // Minmize the answer:
            // If count of 'a's , 'b's and 'c's in the substring s[0..l] are greater or
            // equal to k, just take the length of the substring s[0..l], which is l+1
            if(al>=k && bl>=k && cl>=k) ans=std::min(ans,l+1);

            // If count of 'a's , 'b's and 'c's in the substring s[r..n-1] are greater or
            // equal to k, just take the length of the substring s[r..n-1], which is n-r
            if(ar>=k && br>=k && cr>=k) ans=std::min(ans,n-r);

            // If count of 'a's , 'b's and 'c's in both substrings s[0..l] and s[r..n-1] are greater or
            // equal to k, just take the length both of them, which is (l+1)+(n-r)
            if(al+ar>=k && bl+br>=k && cl+cr>=k) ans=std::min(ans,(l+1)+(n-r));
        }
        return ans;
    }
};
```

2516. Take K of Each Character From Left and Right

```
/*
   Sliding window
   Time complexity: O(n)
   space complexity: O(1)
*/
class Solution {
   public:
      int takeCharacters(std::string s, int k) {
         int n=s.size();

         if (k==0) return 0;

         // Count all 'a's , 'b' and 'c's
         long long a=0,b=0,c=0;
         for(auto& letter: s){
            a+=int(letter=='a');
            b+=int(letter=='b');
            c+=int(letter=='c');
         }

         if(a<k || b<k || c<k) return -1;

         int ans=INT_MAX;
```

```
        // Starting from the beginning, we'de like the window be
        // big as possible, such that the count of 'a's , 'b' and 'c's >= k
        // in the remaining characters.
        int l=0;
        for (int r=0;r<n;++r){
            //...and update the number of 'a's , 'b' and 'c's
            // in the remaining characters
            // The count of 'a's , 'b' and 'c's could be reduced
            // when we extend the window
            a-=int(s[r]=='a');
            b-=int(s[r]=='b');
            c-=int(s[r]=='c');

            // Count of 'a's , 'b' and 'c's not valid
            // in remaining characters
            // The count of 'a's , 'b' and 'c's could raise
            // when we shrink the window
            while(a<k || b<k || c<k){
                a+=int(s[l]=='a');
                b+=int(s[l]=='b');
                c+=int(s[l]=='c');
                l++;
            }

            // Minimize the number of charactrs in remaining portions
            // Which is the number of characters to remove such that
            // the count of 'a's , 'b' and 'c's >=k
            ans=std::min(ans,n-(r-l+1));
        }
        return ans;
    }
};
```