 Editorial by Errichto

Let's first think when Limak can choose some particular interval (substring). We should care about the remaining letters, both in the prefix and the suffix. If there are more than $n/4$ remaining letters of one type then we can't get a steady string. Otherwise, we know exactly how many letters of each type are missing and we can fill the removed interval with these exact letters. So, the interval can be chosen only if the remaining part doesn't contain more than $n/4$ letters of some type.

For each possible starting index of the interval let's find the nearest (leftmost) possible ending index. You can create four segment trees, one for each letter. Then, for fixed interval you can check in $O(1)$ whether there are at most $n/4$ remaining letters of each type. So, for each starting index you can binary search the earliest good ending index. Segment trees and binary search give us $O(n \log^2 n)$. Let's make it faster.

The first step is to get rid of segment trees. It's enough to store the prefix (and maybe the suffix) sum for each letter, so you don't need segment trees anymore. It's ok to get AC but you can change one more thing. As we move with the starting index to the right, the ending index also moves to the right (or it doesn't change). So, we can use the two pointers technique to get $O(n)$ solution. Check the code below for details.

 Set by Errichto

Problem Setter's code:

C++

```
// Bear and Steady Gene, by Errichto
#include<bits/stdc++.h>
using namespace std;
const int nax = 1e6 + 5;
char sl[nax];
const char W[5] = "ACTG";
int n, cnt[4];
bool ok() {
    for(int i = 0; i < 4; ++i)
        if(cnt[i] > n / 4)
            return false;
    return true;
}
int f(int i) {
```

```
    for(int j = 0; j < 4; ++j)
        if(sl[i] == W[j])
            return j;
    assert(false);
}
int main() {
    scanf("%d", &n);
    scanf("%s", sl);
    for(int i = 0; i < n; ++i)
        ++cnt[f(i)];
    if(ok()) { // already steady
        puts("0");
        return 0;
    }
    int ans = n;
    // two pointers i, j
    int j = 0;
    for(int i = 0; i < n; ++i) {
        while(j < n && !ok())
            --cnt[f(j++)];
        if(ok()) ans = min(ans, j - i);
        ++cnt[f(i)];
    }
    printf("%d\n", ans);
    return 0;
}
```

Tested by Stonefeang

Problem Tester's code:

C++

```
#include <bits/stdc++.h>
using namespace std;
int n;
char inp[500007];
int ok[1007];
int sum[500007][5];
int bsa, bsb, bss;
int p1;
inline int is_ok(int v, int u)
{
    for (int i=1; i<=4; i++)
    {
```

```
            if (sum[n][i]-sum[u][i]+sum[v-1][i]>n/4)
            {
                return 0;
            }
        }
        return 1;
    }
    inline int is_pos(int v)
    {
        for (int i=1; i+v-1<=n; i++)
        {
            if (is_ok(i, i+v-1))
            {
                return 1;
            }
        }
        return 0;
    }
    int main()
    {
        ok['A']=1;
        ok['C']=2;
        ok['G']=3;
        ok['T']=4;
        assert(scanf("%d", &n)==1);
        assert(n>=4 && n<=500000 && !(n&3));
        assert(scanf("%s", inp+1)==1);
        for (int i=1; i<=n; i++)
        {
            assert(ok[inp[i]]);
        }
        assert(!inp[n+1]);
        assert(scanf("%d", &p1)==EOF);
        for (int i=1; i<=n; i++)
        {
            for (int j=1; j<=4; j++)
            sum[i][j]=sum[i-1][j];
            sum[i][ok[inp[i]]]++;
        }
        bsa=0;
        bsb=n;
        while(bsa<bsb)
        {
            bss=(bsa+bsb)>>1;
            if (is_pos(bss))
            bsb=bss;
            else
            bsa=bss+1;
        }
```

```
    printf("%d\n", bsa);
    return 0;
}
```