Fraudulent Activity Notifications

locked

by Shafaet

Problem | Submissions | Leaderboard | Discussions | Editorial

Statistics

Difficulty: **Medium**
Time Complexity: O(n) with constant
Required Knowledge: Co
Sort
Publish Date: **Oct 05 201**

Editorial by Shafaet

In this problem, you need to find the running median. There can be several approaches to solving this. Notice that a client can spend at most $200 per day. We can take advantage of this small number.

Finding Median using Counting Sort

Let's see how can we find the median of an array using counting sort with an example. Suppose the array is $[2, 3, 2, 5, 7, 6, 6, 7, 4]$. The maximum number in the array is 7. If you write down the frequency of each numbers from 0 to 7, you will get a table like this:

- $freq[0] = 0$
- $freq[1] = 0$
- $freq[2] = 2$
- $freq[3] = 1$
- $freq[4] = 1$
- $freq[5] = 1$
- $freq[6] = 2$
- $freq[7] = 2$

There are 9 elements in the array, so the median is the 5^{th} number in the sorted array. You can loop over the frequency table to find the 5^{th} number.

Get back to the original problem

In the original problem, you need to maintain a frequency table for each window of size d in the array. You can do it by keeping track of the starting and ending point of the window. Let's suppose the start point of the current window is s and the end point is e and $e - s + 1 = d$. Also, assume you already have a frequency table for that window. When you go to next window $(s + 1, e + 1)$, you can update the frequency table by reducing the frequency of the element in index s and by increasing the frequency of the element in index $e + 1$. Using the frequency table you can find the median and your problem is solved.

The complexity is $O(n \times \textbf{maximum number in the array})$

Tricky Part

Note that the median can be a floating point value when the size of the array is even. For example, if the array is $[3, 1, 2, 4]$, the median is 2.5. You will not pass the 2nd sample I/O if you don't handle it properly.

Another Solution

This can be solved using two priority queues in $O(nlogn)$. This video explains it nicely.

Set by Shafaet

Problem Setter's code :

```
import sys
#sys.stdin = open("in", "r")
```

```
        freq = 0
        if i in dic:
            freq = dic[i]
        s = s + freq
        if s>=idx:
            return i

ans = 0
for i in xrange(0, n):
    val = arr[i]

    if i>=d:
        med=find(d/2 + d%2)

        if d%2==0:
            ret = find(d/2+1)
            if val >=med + ret:
                ans = ans+1
        else:
            if val>=med*2:
                ans = ans + 1

    if val not in dic: dic[val] = 0
    dic[val] = dic[val] + 1

    #print i,dic
    if i>=d:
        prev = arr[i-d]
        dic[prev] = dic[prev]-1

print ans
```

 **Tested by pkacprzak**

Problem Tester's code :

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <ctime>
#include <cassert>
using namespace std;
#define SZ(x) ((int)(x.size()))
#define FOR(i,n) for(int (i)=0;(i)<(n);++(i))
#define FOREACH(i,t) for (typeof(t.begin()) i=t.begin(); i!=t.end(); i++)
#define REP(i,a,b) for(int (i)=(a);(i)<=(b);++i)


typedef long long ll;
const int INF = 1e9;

const int N = 2e5;
const int V = 200;

int a[N];

int cnt[V+1];

int main()
{
    ios_base::sync_with_stdio(0);
    int n, d;
    cin >> n >> d;
    assert(n >= 1 && n <= N);
    assert(d >= 1 && d <= n);
    FOR(i, n) cin >> a[i];
    FOR(i, n) assert(a[i] >= 0 && a[i] <= V);
    int res = 0;

    FOR(i, d) cnt[a[i]]++;
    REP(i, d, n-1)
    {
        //SOLVE HERE
```

```
            }
            if(high_median == -1 && acc >= int(ceil((d+1)/2.0)))
            {
                high_median = v;
            }
        }
        assert(acc == d);
        int double_median = low_median + high_median;
        //cout << low_median << " " << high_median << " -> " << median << endl;
        if(a[i] >= double_median)
        {
            res++;
        }
        cnt[a[i-d]]--;
        cnt[a[i]]++;
    }
    cout << res << endl;
    return 0;
}
```