Unique Art

Crack a Hack

01FE15BEC216

Vaishnavi J Hurakadli

MO’s Algorithm:

MO’s algorithm is just an order in which we process the queries. Mo's algorithm is an efficient way to compute the answers to a bunch of range queries. M queries will be given, we will re-order the queries in a particular order and then process them. Clearly, this is an offline algorithm. Each query has L and R, we will call them opening and closing. Let us divide the given input array into Sqrt(N) blocks. Each block will be N / Sqrt(N) = Sqrt(N) size. Each opening has to fall in one of these blocks and each closing has to fall in one of these blocks.

For Mo's algorithm to apply, the following conditions must be met:

- The data should fall in a range (typically, this means you have an array of values).
- The queries should be known ahead of time (this is why it's called an offline algorithm).
- It should be able to incrementally build the answer for a particular query by including/excluding one value in the range at a time. For example, to find the sum of values in a range, one can maintain a running sum of values you've seen so far, and we can update that value as you encounter more values in the range. To remove values from the left side of the range, we can subtract the running sum of those values.

Algorithm:

- First find the block size which is square root of N
- Sort the queries with respect to Left pointer/block size. If this is same then sort by the right pointer.
- Change the left and right pointer according to the range and keep the count of frequency
    - If the left pointer is increasing w.r.t to cuurent pointer then the range is decreasing.
    - If the left pointer is decreasing w.r.t to cuurent pointer then the range is increasing.

- If the right pointer is increasing w.r.t to cuurent pointer then the range is increasing.
- If the right pointer is decreasing w.r.t to cuurent pointer then the range is decreasing.
- When the range is decreasing check the condition and decrease the frequency count and if the range is increasing then increase the frequency count

❖ As we are checking the unique elements the counter of unique elements increases if the frequency is 1 and decreases if the frequency is greater than 1 or equal to 0.

❖ Print the counter value as the answer.

Efficiency:

As we are just reordering the queries we get the complexity of O(N * Sqrt(N)).

Code:

```
#include <iostream>

#include <algorithm>

#include<math.h>

#include <bits/stdc++.h>

using namespace std;


#define N 1000000

#define A 1111111

#define BLOCK 1000 // ~sqrt(N) Block size
```

```
int cnt[A], a[N], ans[N],ncnt[A], answer = 0;
map<int, int> m;

struct node {
        int L, R, i; //Left pointer and right pointer
}q[N];

bool cmp(node x, node y) {
        if(x.L/BLOCK != y.L/BLOCK) {
                // different blocks, so sort by block.
                return x.L/BLOCK < y.L/BLOCK;
        }
        // same block, so sort by R value
   else
          return (x.R < y.R)^(x.L/BLOCK%2);
}

void add(int position) {

      cnt[a[position]]++;
      if(cnt[a[position]] == 1) { // Frequency is one then it is unique
         answer++;
      }
```

```
        if(cnt[a[position]]==2) //Frequency is more than one then it is not unique
        {
            answer--;
        }

}

void remove(int position) {

        cnt[a[position]]--;
        if(cnt[a[position]] == 0) {
            answer--;
        }
        if(cnt[a[position]] == 1) {
            answer++;
        }

}

int main() {
    ios::sync_with_stdio(0);  //flushing the memory
    cin.tie(0);
    cout.tie(0);
```

```
        int n;
        cin>>n;
        for(int i=0; i<n; i++)
{
                cin>>a[i];
    m[a[i]];
}
int ctr=1;
for(auto &it:m)
    it.second=ctr++;
 for(int i=0;i<n;i++)
{
    a[i]=m[a[i]];
}
        int t;
        cin>>t;
        for(int i=0; i<t; i++) {
                cin>>q[i].L>>q[i].R;
                q[i].L--; q[i].R--;
                q[i].i = i;
        }

        sort(q, q + t, cmp); // calling the sort function to sort queries
```

```
int currentL = 0, currentR = 0;
for(int i=0; i<t; i++) {
        int L = q[i].L, R = q[i].R;
        while(currentL < L) { // If Left pointer is increasing
                remove(currentL);
                currentL++;
        }
        while(currentL > L) { // If Left pointer is decreasing
                add(currentL-1);
                currentL--;
        }
        while(currentR <= R) { // If right pointer is increasing
                add(currentR);
                currentR++;
        }
        while(currentR > R+1) { // If right pointer is decreasing
                remove(currentR-1);
                currentR--;
        }
        ans[q[i].i] = answer; // Storing the answer in array
}
```

```
    for(int i=0; i<t; i++)
        cout<<ans[i]<<"\n";
}
```

References:

https://www.hackerearth.com/practice/notes/mos-algorithm/

https://blog.anudeep2011.com/mos-algorithm/

https://www.quora.com/How-exactly-is-the-square-root-decomposition-of-queries-also-sometimes-referred-to-as-Mos-Algorithm-used-for-offline-processing-of-queries