Understanding Mo's Algorithm

easy-medium   mos-algorithm

aryan12   Jun '20

Firstly, Mo's Algorithm uses offline queries to its advantage. If you don't know about offline queries, they are basically inputting the queries into a vector/list and then sorting them in your way, and finding the answers of each using the previously answered queries.

Prerequisites: None

I will explain the question with a problem.

Problem:
You are given an array A of N elements. You have to answer Q queries of type l, r. For each query you need to find the sum of elements from A_l to A_r and output it.

Naive Implementation:
What would be a naive way to do this question? Probably take each query, and iterate from l to r and print the sum. But, for large values of N and Q this will time out.

Its complexity is $O(N * Q)$, which is pretty bad because you can't work with values like $N = 10^5$ and $Q = 10^5$. Let us try an think of a solution which is faster than this.

Segment Tree?

Yes, this can work here, giving us the time complexity of $O(NlogN)$, but it is very complex and beginners can't do this question with that. Also, there are many blogs about that, so I am not going into this.

I need something which has the time complexity less than $O(N * Q)$ and does not involve and data structure except the basic ones.

This is where Mo's Algorithm comes to play.

Idea: We can use the answer for 1 query for the next.
Is this possible? Yes it is.

So, we have an array $A[1, 2, .., N]$ and a vector of pairs, $Queries[1, 2, 3, ..Q]$

Lets divide the array A into $sqrt(N)$ blocks.

Skip to main content

Thus, the size of each block will be $sqrt(N)$. It is easy to prove, because $sqrt(N) * sqrt(N)$ = N. Our main aim is to **answer queries in the first block first, then go to the second block** and so on. Thus, each l and r in the query has to fall in one of these blocks. So we sort all queries from $(1...sqrt(N) - 1)$, $(sqrt(N)...2 * sqrt(N) - 1)$ and so on.

So, we sort the queries on basis of the block where l lies, and then sort in increasing order of r.

Note: It might happen that queries are sorted in a way that l is not increasing, that is because we are sorting l by blocks only. Then it depends on the value of r.

Now, we maintain sum and two pointers $currL$ and $currR$.
sum stores the answer for segment $currL$ to $currR$, and $currL$ and $currR$ are the left and right index respectively.

When moving from the ith query to the $i + 1th$ query, we compare $Queries[i].left$ and $currL$. Then we increment or decrement $currL$ by one and change sum accordingly. Similarly we compare $Queries[i].right$ and $currR$ and change things accordingly.

Implementation:

```
//other stuff to be done here. This is just for understanding sorting
//and the implementation of Mo's Algorithm
long long block;

struct queries {
    long long l, r, idx, ans;
};

vector<queries> query;

bool cmp(queries a, queries b) {
    if((a.l / block) != (b.l / block))
        return ((a.l / block) < (b.l / block));
    return a.r < b.r;
}

int main() {
    //input to be taken
    block = (long long)(sqrt(n));
    sort(query.begin(), query.begin() + q, cmp); //query is the array for quer
    //q is the number of queries
    long long currL = 1, currR = n, sum = 0;
    for(long long i = 0; i < query.size(); i++) {
        long long l = query[i].l, r = query[i].r;
                  < l) {
```

```
            sum -= a[currL];
            currL++;
        }
        while(currL > l) {
            currL--;
            sum += a[currL];
        }
        while(currR < r) {
            currR++;
            sum += a[currR];
        }
        while(currR > r) {
            sum -= a[currR];
            currR--;
        }
        query[i].ans = sum;
    }
    //now we sort the queries by indexes to give the answer to the correspondi
    //output the answer to the corresponding query
}
```

Time Complexity:
$O(N * sqrt(N))$

How much currR is moved?
For each block, queries are sorted in increasing order of r. So, for a block, $currR$ moves in increasing order. In worst case, before beginning of every block, $currR$ at **extreme right** and current block moves it back the **extreme left**. This means that for every block, $currR$ moves at most $O(N)$. Since there are $O(sqrt(N))$ blocks, total movement of $currR$ is $O(N * sqrt(N))$.

How much currL is moved?
Since all queries are sorted in a way that L values are grouped by blocks, movement is $O(sqrt(N))$ when we move from one query to another query. For Q queries, movement is $O(Q * sqrt(N))$

Thus, time complexity is $O((N + Q) * sqrt(N))$ which is equivalent to $O(N * sqrt(N))$

Resources:
These are to help you where you get stuck, they are beginner-friendly.

1. [GeeksForGeeks]
2. [Video By Gaurav Sen]

P.S. Th[illegible]oblems which can be done with Segment Tree too, but some which h[illegible]'s Algorithm

Example Question: https://codeforces.com/contest/86/problem/D

Practice Questions:

1. https://www.spoj.com/problems/DQUERY/
2. https://www.codechef.com/MARCH14/problems/GERALD07
3. https://codeforces.com/problemset/problem/375/D
4. https://www.codechef.com/problems/IITI15

If you have any queries, please comment down, I will try to answer them asap!

---

ganeshkumarm1 Aug '20

Please do checkout my medium article on MO's algorithm
https://medium.com/javarevisited/mos-algorithm-range-queries-made-easy-6c35047369ca