Parallel Binary Search

Parallely we AC, Sequentially we TLE!

Pre Requisites

Binary Search - How it works and where can it be applied!

Motivation Problem

We aim to solve this problem : Meteors.

The question simply states :
There are N member states and M sectors. Each sector is owned by a member state. There are Q queries, each of which denote the amount of meteor shower in a $[L, R]$ range of sectors on that day. The i^{th} member state wants to collect $reqd[i]$ meteors over all its sectors. For every member state, what is the minimum number of days it would have to wait to collect atleast the required amount of meteors?

Solution

The naive solution here is to do a binary search for each of the N member states. We can update in a range using segment trees with lazy propagation for each query. The time complexity of such a solution would be $O(N * logQ * Q * logM)$. But this one will easily TLE.

Let's see if there's something we are overdoing. For every member state, the binary search applies all the queries until a point multiple times! For example, the first value of mid in the binary search is same for all member states, but we are unnecessarily applying this update everytime, instead of somehow caching it.

Let's do all of these N binary searches in a slightly different fashion. Suppose, in every step we group member states by the range of their binary search. In the first step, all member states lie in range $[1, Q]$. In the second step, some lie in range $[1, Q/2]$ while some lie in range $[Q/2, Q]$ depending on whether the binary search predicate is satisfied. In the third step, the ranges would be $[1, Q/4]$, $[Q/4, Q/2]$, $[Q/2, 3Q/4]$, $[3Q/4, Q]$. So after $logQ$ steps, every range is a single point, denoting the answer for that member state. Also, for each step running the simulation of all Q queries once is sufficient since it can cater to all the member states. This is pretty effective as we can get our answer in $Q*logQ$ simulations rather than $N*Q*logQ$ simulations. Since each simulation is effectively $O(logM)$, we can now solve this problem in $O(Q * logQ * logM)$.

Implementation

We would need the following data structures in our implementation :

1. linked list for every member state, denoting the sectors he owns.
2. arrays L and R denoting range of binary search for each member state.
3. range update and query structure for Q queries.
4. linked list *check* for each mid value of current ranges of binary search. For every mid value, store the member states that need to be checked.

The implementation is actually pretty straight forward once you get the idea. For every step in the simulation, we do the following :

1. Clear range tree, and update the linked lists for mid values.
2. Run every query sequentially and check if the linked list for this query is empty or not. If not, check for the member states in the linked list and update their binary search interval accordingly.

Pseudo Code

```
for all $logQ$ steps:
    clear range tree and linked list $check$
    for all member states $i$:
        $if\ L[i] \neq R[i]$ :
            $mid = (L[i] + R[i])/2$
            insert $i$ in $check[mid]$.
    for all queries $q$:
        apply($q$)
        for all member states $m$ in $check[q]$:
            $if\ m$ has requirements fulfilled:
                $R[m] = q$
            else:
                $L[m] = q + 1$
```

In this code, the *apply*() function applies the current update, i.e. , it executes the range update on segment tree. Also to check if the requirements are fulfilled, one needs to traverse over all the sectors owner by that member state and find out the sum. In case, you still have doubts go over to the next page and see my code for this problem.

Problems to try

1. Meteors - My AC solution
2. Make n00b land great again
3. Travel in HackerLand
4. SRM 675 Div1 500

Conclusion

I heard about this topic pretty recently, but was unable to find any good tutorial. I picked up this trick from some comments on codeforces blog on Meteors problem. As we can see, we can now solve a recent 500 pointer on topcoder using this trick! Feel free to point out errors, or make this tutorial better in any way! Alternate solutions to the mentioned problems are also welcome.

Happy Coding!