DATA STRUCTURE AND ALGORITHM THEORETICAL ASSIGNMENT 1

RANGE MINIMA PROBLEM

Problem: Given an array A storing n numbers and two integers i and j ,compute the minma in range i to j in array A.

Algorithm:

Let A be the input array. In this algorithm we are taking $n/log(n)$ tiny data structures each of which storing $O(log(n))$ values. Let that tiny data structure be B. Let k be the index of any element in A Each of this tiny data structure stores values corresponding to a particular k. This k are $0, log(n), 2log(n), 3log(n)$.... and so on upto a multiple of $log(n)$ just less than n. Each data structure is storing minima from k to $2^0 log(n), (2^1)log(n), (2^2)log(n)$.... and so on.

Now for a given i and j we need to find minima. Let k be the index in A which is just greater than i and multiple of log(n). Let l be the index in A which is just less than j and multiple of log(n). So, we first we find minima from range k to l. To achieve this we select an number x such that j lies between $k + 2^x(log(n))$ (say m) and $k + 2^{x+1}(log(n))$ for some x and an element q such that $q = l - 2^x(log(n))$.

So we have minima from k to m stored in our tiny data structure as $m - k$ and k is multiple of power-of-two multiplied by $log(n)$. We also have a minima from q to l as again $l - q$ is a multiple of $log(n)$ and so is the q. We now compare this two minima to get the minima1 in range k to l. Now we compare this minima1 with all the elements in range i to k and l to j to get the minima in the range i to j.

Algorithm 1 computes range minima in range i to j on the basis of above discussion.

Data: Input will be an array $A[0..n-1]$ and i and j. However, the algorithm will use additional two dimensional array $B[0..n/(log(n))][0..log(n)]$ and variables $minima$, k and l,m and q.
Result: The range minima in the range i to j.
$k \leftarrow$ Entry-Stored-in-B-Just-greater-than(i) $l \leftarrow$ Entry-stored-in-B-just-less-than(j) $m \leftarrow$ (Power-of-two)*log(n) which is just less-than(l) $q \leftarrow l$-((Power-of-two)*log(n) which is just less-than(l)) **if** $B[k][m] \leq B[q][l]$ **then**
  $minima \leftarrow B[k][m]$
else
  $minima \leftarrow B[q][l]$
end
for $index \leftarrow i$ **to** k **do**
  if $A[index] \leq minima$ **then**
    $minima \leftarrow A[index]$;
  end
end
for $index \leftarrow (l+1)$ **to** j **do**
  if $A[index] \leq minima$ **then**
    $minima \leftarrow A[index]$
  end
end
Output $minima$.

Algorithm 1: Algorithm for finding Range-Minima

Proving correctness of the algorithm:
We need to prove that Algorithm 1 is correct. It follows from the discussion that it suffices if we can show that lemma given below is true.

Lemma 1. *Range minima is indeed the minimum of elements* $A[itok], B[k][m], B[q][l], A[ltoj]$.

Proof. To prove the above lemma it is sufficient to prove that we are finding correct minima in the range k to l as rest of the part is brute force and involves simple comparisions. Since k is selected such that it is a multiple of $log(n)$ we have information k in our tiny data structure B. Similary we have information for l. m is defined as sum of a l and a multiple of $log(n)$ hence we have its information in B. For similar reasons information of q is also present in B. Now since q is selected such that it lies between k and m and we have minima stored in B for range k to m and also for q to l. Minimum of these two minima is the minima between k to l. So, it proves that we have correct minima for whole range i to j.

□

Analysis of Time and Space complexity the Algorithm

Algorithm 1 first calculates the k which is the entry just greater than i and stored in B this takes $O(log(n))$ time as maximum distance between i and k can be $log(n)$ due to structure of our tiny data structures. Next it calculates l which again requires $O(log(n))$ time as difference of j and l can be at most $log(n)$. Thereafter it spends a total time of $O(1)$ to calculate minima between k and l. After that it spends a total of $O(log(n))$ time in the two "for" loops for brute force part. So, the overall time complexity of the Algorithm is $O(log(n))$.

The algorithm uses an additional matrix B of size $(n/log(n))*(log(n))$ in addition to a few variables. Hence, the algorithm uses $O(n)$ extra space.