Sample Solution

Assignment 10

CSCI 3110 — Summer 2018

The key observation is the following: If we store the elements in S in sorted order, then the minimum difference between two elements in S is realized by a pair of consecutive elements in this sorted order. This suggests the following data structure:

We store the elements of S in an (a,b)-tree T. Every leaf storing an element x in S also stores the difference δ_x between x and the next-larger element y in S as well as the pair $p_x = (x, y)$. For the maximum element x in S, $\delta_x = \infty$. Every internal node v stores a value δ_v that is the minimum of all values δ_x associated with v's descendant leaves and the pair p_v of elements that realize this difference. In other words, if $\delta_v = \delta_x$, then $p_v = p_x$.

Closest pair query: Given that the root r of T stores the minimum difference δ_r between all pairs of consecutive elements in S and the corresponding pair p_r that realizes this difference, a closest pair query amounts to reporting p_r. Thus, it takes $O(1)$ time.

Insertion: To insert a new element x into S, we insert x into S as into a standard (a,b)-tree. The two leaves whose δ-values need to be recomputed are x and its predecessor. To do so, we need to find x's predecessor y and x's successor z. Given these two nodes, we have $p_y = (y,x)$, $\delta_y = x - y$, $p_x = (x,z)$ and $\delta_x = z - x$. Note that y or z may not exist. If y does not exist, then δ_y and p_y do not need to be updated. If z does not exist, then $\delta_x = \infty$. To find y, we follow the path from x to the root until we reach a node v that is not the leftmost child of its parent. We then locate v's left sibling u and follow the path from u to its rightmost descendant leaf, which is y. z can be found analogously.

After updating δ_y, δ_x, p_y, and p_x, the internal nodes whose δ and p-values may change are ancestors of x and y. Thus, we traverse the paths from x and y to the root and recompute the δ and p-values of all nodes on these paths bottom-up. Since δ_v and p_v can be computed in constant time from the δ and p-values of v's descendants, this takes constant time per node.

Overall, we spend $O(\lg n)$ time to insert x, $O(\lg n)$ time to locate y and z and update δ_y, δ_x, p_y, and p_x, and $O(\lg n)$ time to update the δ and p-values of all ancestors of x and y. The insertion may also trigger up to $O(\lg n)$ node splits to rebalance the tree. We argue below that each node split takes constant time. Thus, an insertion takes $O(\lg n)$ time.

Deletion: We delete the element x from T as from a standard (a,b)-tree. Before removing the leaf storing x, however, we locate x's predecessor y and successor z as we did for an insertion. If y does not exist, the deletion of x does not affect the δ-value of any leaf. If y exists but z does not exist, then $\delta_y = \infty$ and all other δ-values associated with leaves remain unchanged. If y and z both exist, then $\delta_y = z - y$ and $p_y = (y,z)$. Now, as after an insertion, the internal nodes whose δ and p-values may have to be updated are ancestors of x and y. Thus, as for an insertion, we traverse the paths from x and y to the root and recompute the δ and p-values of the nodes on these paths from the δ and p-values of their children. Excluding the cost of rebalancing, a deletion thus takes $O(\lg n)$ time. Since a deletion triggers at most one node split and up to $O(\lg n)$ node fusions, and we show below that each node split or node fusion takes constant time, rebalancing after a deletion also takes $O(\lg n)$ time. Thus, the total cost of a deletion is $O(\lg n)$.

Node split: After a node split, we need to compute the δ and p-values associated with the two nodes created by the split. Since these values can easily be computed in constant time from the δ and p-values associated with the nodes' children, this takes constant time.

Node fusion: After a node fusion, we need to compute the δ and p-values associated with the node created by the fusion. Since these values can easily be computed in constant time from the δ and p-values associated with the node's children, this takes constant time.