Sample Solutions

Assignment 9

CSCI 3110 — Summer 2014

1 The Data Structure

The base data structure is a standard \((a, b)\)-tree \(T\) over the set \(S\). Now, the key observation is that a min-pair is always formed by two consecutive elements in the sorted sequence of elements of \(S\), that is, by two neighbouring leaves in \(S\). For any node \(v\) of \(T\), let \(l_v\), \(r_v\), \(m_v\), and \(P_v\) be the minimum element in the subtree \(T_v\) with root \(v\), the maximum element in \(T_v\), the minimum difference between any pair of consecutive elements in \(T_v\), and a pair realizing this difference \(m_v\), respectively.

2 Query

By definition, if \(r\) denotes the root of \(T\), then \(P_r\) is a min-pair and \(m_r\) is the difference between the elements in \(P_r\). Thus, by inspecting \(P_r\) and \(m_r\), a min-pair can be reported in constant time.

3 Updates

As usual, we divide the discussion of updates into four parts:

- Insertions without rebalancing
- Deletions without rebalancing
- Node splits
- Node fusions

We show that the first two take \(O(\lg n)\) time and that the last two take \(O(1)\) time. Since a complete insertion or deletion performs up to \(O(\lg n)\) rebalancing operations, this shows that the cost per update is \(O(\lg n)\).
3.1 Recalculating Labels

Apart from the standard steps involved in insertions, deletions, node splits, and node fusions, the basic operation we use is to calculate \( l_v, r_v, m_v, \) and \( P_v \) for a node whose children already store these labels for their own subtrees. Let \( w_1, w_2, \ldots, w_k \) be \( v \)'s children. Then \( l_v = l_{w_1} \) and \( r_v = r_{w_k} \), so these values are easy to calculate in constant time for \( v \). As for \( P_v \), we observe that it can be of two types: Either it is formed by two elements that belong to the same subtree \( T_{w_i} \) or it is formed by the rightmost element in \( T_{w_i} \) and the leftmost element in \( T_{w_{i+1}} \), for some \( 1 \leq i < k \). Thus, we iterate over the children of \( v \) to calculate \( m_v \) as

\[
m_v = \min( \min_{1 \leq i \leq k} m_{w_i}, \min_{1 \leq i < k} (l_{w_{i+1}} - r_{w_i}) ).
\]

This clearly takes constant time because \( k \in O(1) \). To calculate \( P_v \) as a by-product, we only need to remember which term in the above expression for \( m_v \) is the minimum term. If it is some term \( m_{w_i} \), then \( P_v = P_{w_i} \). If it is a term \( (l_{w_{i+1}} - r_{w_i}) \), then \( P_v = (r_{w_i}, l_{w_{i+1}}) \).

3.2 Insertions

The only nodes whose labels \( l_v, r_v, m_v, \) and \( P_v \) may have to be updated are ancestors of the leaf \( x \) created by the insertion. We can do this by traversing the path from \( x \) to the root \( r \) of \( T \) and computing \( l_v, r_v, m_v, \) and \( P_v \) for each node we encounter. Since the children of \( v \) store the correct labels at the time we compute \( v \)'s labels, we can use the procedure from Section 3.1 to compute \( v \)'s labels in constant time. Since the path from \( x \) to the root of \( T \) has length \( O(\lg n) \), we thus spend \( O(\lg n) \) time to update the labels of all nodes whose labels may have been invalidated as a result of the insertion of \( x \).

3.3 Deletion

Similar to insertions, the only nodes whose labels may have to updated are the ancestors of the deleted leaf \( x \). Thus, we traverse the path from \( x \) to the root \( r \) of \( T \) and update the labels of all nodes on this path as for an insertion, which takes \( O(\lg n) \) time.

3.4 Node split

A node split creates two new nodes whose labels can be computed from the labels of their children in constant time. For all other nodes of \( T \), the set of descendant leaves does not change, so their labels remain correct. Therefore, a node split can be performed in constant time.
3.5 Node fusion

A node fusion creates one new node whose labels can be computed from the labels of its children in constant time. For all other nodes of $T$, the set of descendant leaves does not change, so their labels remain correct. Therefore, a node fusion can be performed in constant time.