6.854J / 18.415J Advanced Algorithms
Fall 2008

18.415/6.854 Advanced Algorithms October 1, 2008

Lecture 8

Lecturer: Michel X. Goemans

Previously, we introduced the dynamic tree data structure and the operations that dynamic trees must support. Today, we take a more detailed look at dynamic trees and describe the efficient implementation of the operations. In doing so, much of our focus will be on the EXPOSE method, an extended splay operation that is essential in all these operations. We show that any sequence of m operations on a dynamic tree with n nodes takes $O((m+n)\log n)$ time.

1 Dynamic Trees

Dynamic trees (also known as link-cut trees) introduced by Sleator and Tarjan are a data structure intended to maintain a representation of a set of rooted trees. We will be able to perform various operations on these trees, to be discussed later. Figure 1 shows an example tree as a virtual tree (left) and a rooted tree (right).

1.1 Rooted Trees

We view rooted trees as unions of node-disjoint (directed) paths. This divides the edges of the tree into two sets. Solid edges are those that are on the node-disjoint paths that the tree is composed of, and dashed edges are those that are not on these paths. Note that each path consisting of solid edges is a directed path (we omit the arrows here) from top to bottom.

1.2 Virtual Trees

The union of disjoint paths described above can be used to represent virtual trees. In a virtual tree, each solid path is represented by a splay tree such that the following conditions hold:

- A successor node in a splay tree is an ancestor in the rooted tree.
- For each splay tree, its largest node is linked to the parent of the root in the rooted tree.
- In the virtual tree, each node has at most one left child, at most one right child, and any number of middle (virtual) children.

There are three kinds of edges in a virtual tree, corresponding to the three types of children a node can have. Left and right children of a node are connected to the node by solid edges, and middle children of a node are connected to it by dashed edges. Note that there can be many virtual trees corresponding to a rooted tree, because there are two different degrees of freedom involved in constructing a virtual tree — the union of disjoint paths could be different, as could the structure of the splay trees corresponding to the paths.

An important consequence of this setup is that rotations in a splay tree do not affect the structure of the rooted tree.

2 The Expose Operation

The EXPOSE(v) operation is an extended splay operation that brings v to the root of the virtual tree without changing the structure of the rooted tree. The important parts of this operation are to

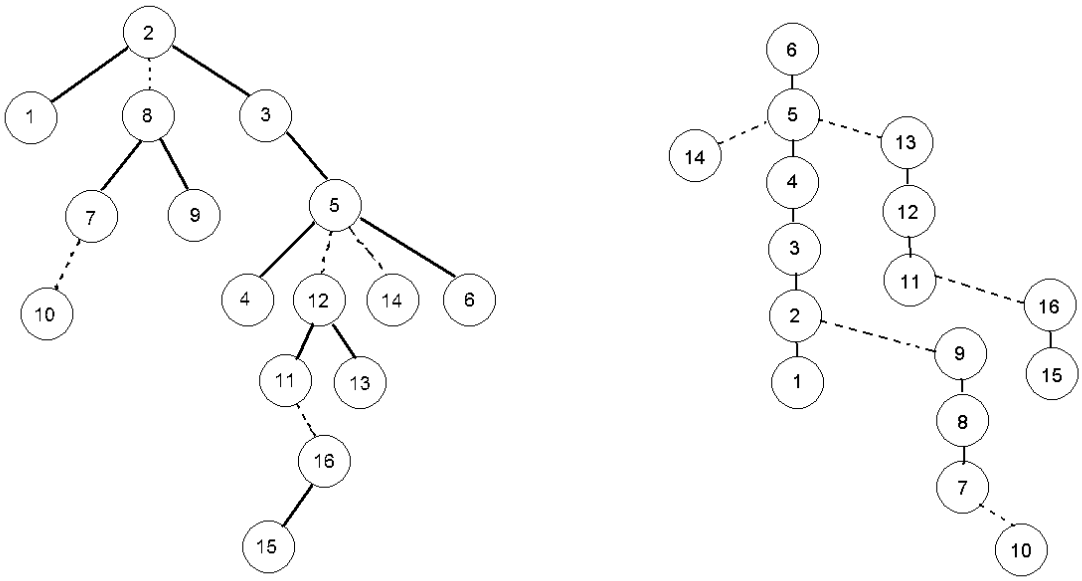

Figure 1: Virtual tree (left) and corresponding rooted tree (right).

make sure that the path from v to the root is solid and that the splay tree representing the path to which v belongs is rooted at v. We can describe this operation in three steps. In our example, we run EXPOSE on node 15.

2.1 Step 1

Step 1 consists of walking from v to the root of the virtual tree. Whenever the walk enters a splay tree (solid edges) at some node w, a SPLAY(w) operation is performed, bringing w to the root of that tree. Middle children are not affected in this step. For instance, we splay nodes 11 and 5 in our example tree as in figure 2. Note that at the end of step 1 of an EXPOSE(v) operation, v will be connected to the root of the virtual tree only by dashed edges.

2.2 Step 2: Splicing

Step 2 consists of walking from v to the root of the virtual tree exchanging along the way each middle edge with the left subtree of the parent. This is illustrated in Figure 3 and called splicing. A middle child of a node w and its left child can be exchanged (without changing the rooted tree) only if w is the root of its splay tree. This justifies our execution of step 1 first since at the end of step 1 all edges from v to the root are middle edges.

Splicing is a valid operation on virtual trees. Indeed, referring to Figure 3, the left subtree of w in the splay tree corresponds to the part of the solid path that is below w in the rooted tree; this is because w is the root of its splay tree. Exchanging that solid subpath with the solid path corresponding to the splay tree rooted at v still leaves the rooted tree decomposed into a node-disjoint union of paths.

Note that after performing this operation on every edge to the root of the virtual tree, there will be a solid path from the root of the rooted tree to the node being exposed.

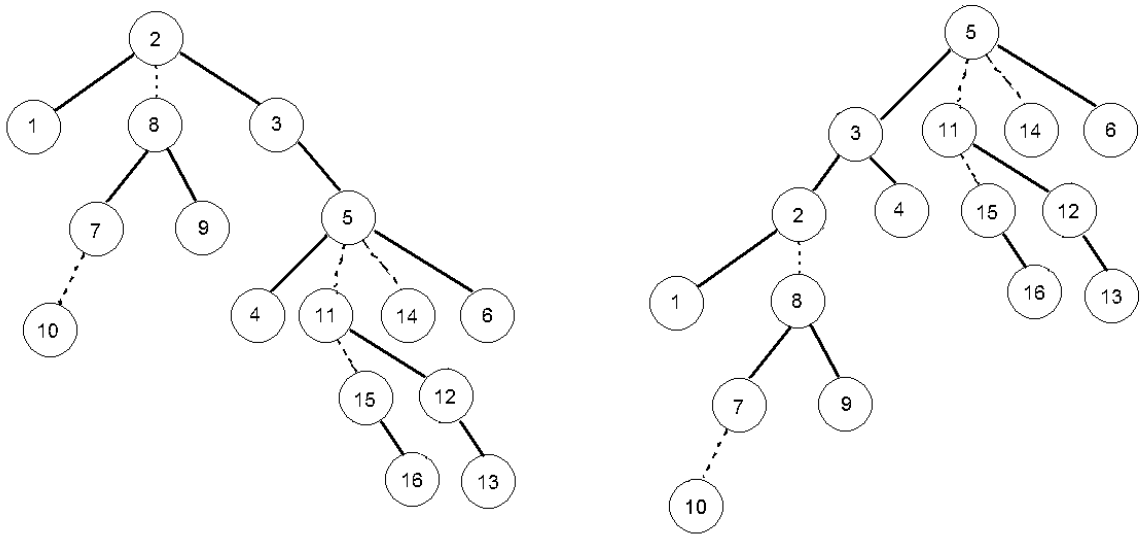

Figure 2: Walking Up and Splaying. The virtual tree after splaying 15 and 11 is shown on the left. The virtual tree on the right is at the end of step 1, after splaying also node 5.

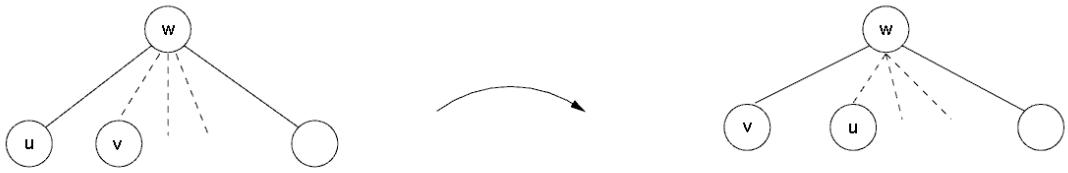

Figure 3: Splicing. w needs to be the root of its splay tree.

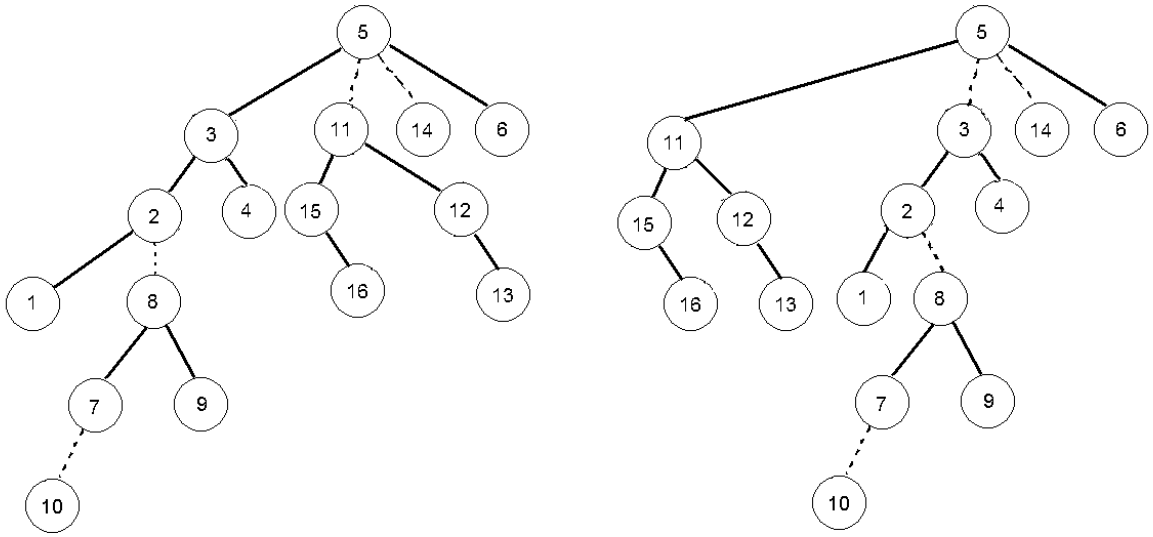

Figure 4: Left virtual tree is after first splicing, the right virtual tree is the one at the end of step 2.

The result of splicing every node on the path to the root for our example is illustrated in Figure 4.

2.3 Step 3

Step 3 consists of walking from v to the root in the virtual tree, splaying v to the root. Note that in the analysis, we can charge the entire cost of step 2 to the final splaying operation in step 3. Figure 5 shows the relevant splay tree before and after this step.

3 Operations on Dynamic Trees

We will now describe the desired operations on a dynamic tree and how to implement them efficiently using the EXPOSE method just defined. Some of these operations require keeping track of different costs in the tree, so we first consider an efficient way of doing this.

3.1 Maintaining Cost Information

When performing operations on the dynamic tree, we need to keep track of $\text{cost}(x)$ for each node x, and we need to be able to find the minimum cost along paths to the root of the rooted tree. If such a path is the prefix of a path corresponding to a splay tree, it seems that, knowing the minimum cost in any subtree of any our splay trees might be helpful. So, in addition to $\text{cost}(x)$, we would like to keep track of the value $\text{mincost}(x)$, given by

$$\text{mincost}(x) = \min\{\text{cost}(y) \mid y \text{ in the subtree rooted at } x \text{ of } x\text{'s splay tree}\}.$$

We'll see that, instead of maintaining $\text{cost}(x)$ and $\text{mincost}(x)$, that it will be easier to maintain the following two quantities for every node x:

$$\Delta\min(x) = \text{cost}(x) - \text{mincost}(x)$$

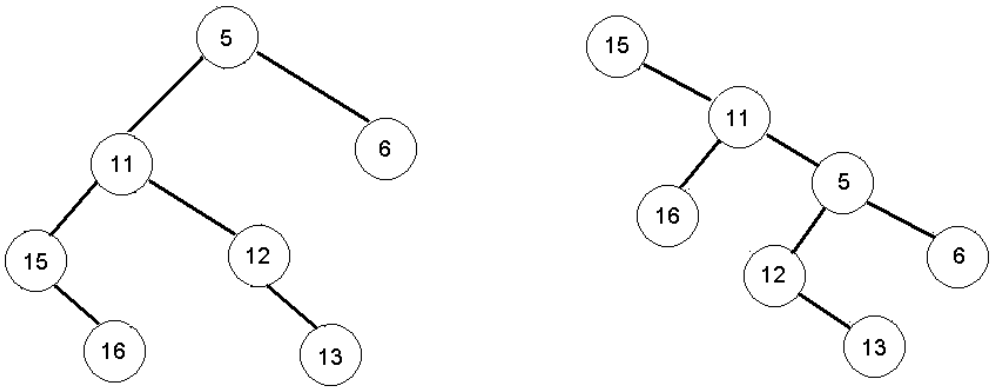

Figure 5: Splaying on Virtual Tree.

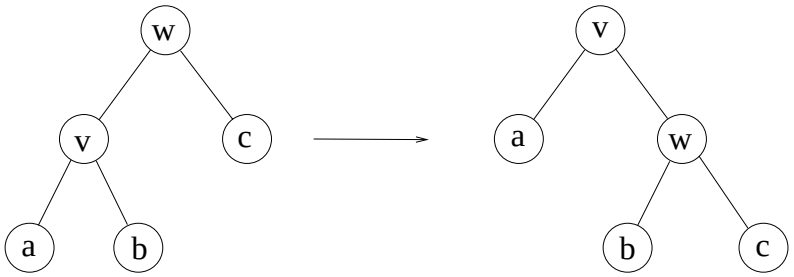

Figure 6: Rotation.

and

$$\Delta\,\mathrm{cost}(x) = \begin{cases} \mathrm{cost}(x) & \text{if } x \text{ is the root of a splay tree,} \\ \mathrm{cost}(x) - \mathrm{cost}(p(x)) & \text{otherwise.} \end{cases}$$

Observe that, if x is the root of a splay tree, then $\mathrm{cost}(x) = \Delta\,\mathrm{cost}(x)$ and $\mathrm{mincost}(x) = \Delta\,\mathrm{cost}(x) - \Delta\,\mathrm{min}(x)$. This fact, combined with the EXPOSE operation, shows that we can find $\mathrm{cost}(x)$ and $\mathrm{mincost}(x)$ given $\Delta\,\mathrm{min}(x)$ and $\Delta\,\mathrm{cost}(x)$, so it is sufficient to maintain the latter.

We now claim that we can update $\Delta\,\mathrm{min}(x)$ and $\Delta\,\mathrm{cost}(x)$ in $O(1)$ time after a rotation or a splice, which will allow us to maintain $\mathrm{cost}(x)$ and $\mathrm{mincost}(x)$ in $O(1)$ time.

We first consider a rotation, see Figure 6 for the labelling of the nodes. Let $\Delta\,\mathrm{cost}(x)$ and $\Delta\,\mathrm{cost}'(x)$ correspond to before and after the rotation, respectively. Similarly define $\Delta\,\mathrm{min}(x)$ and $\Delta\,\mathrm{min}'(x)$. Observe that during a rotation, only the nodes b, w and v have their $\Delta\,\mathrm{cost}(x)$ change. One can check that the updates are as follows:

$$\begin{aligned} \Delta\,\mathrm{cost}'(v) &= \Delta\,\mathrm{cost}(w) + (\mathrm{cost}(v) - \mathrm{cost}(w)) \\ &= \Delta\,\mathrm{cost}(w) + \Delta\,\mathrm{cost}(v), \\ \Delta\,\mathrm{cost}'(w) &= -\Delta\,\mathrm{cost}(v), \\ \Delta\,\mathrm{cost}'(b) &= \Delta\,\mathrm{cost}(b) + (\mathrm{cost}(v) - \mathrm{cost}(w)) = \Delta\,\mathrm{cost}(b) + \Delta\,\mathrm{cost}(v). \end{aligned}$$

Before showing the corresponding updates for $\Delta\,\mathrm{min}(x)$, observe that $\Delta\,\mathrm{min}(x)$ and $\Delta\,\mathrm{cost}(x)$

satisfy the following equation; here x is any node and l is its left child and r is its right child:

$$\begin{aligned}
\Delta\min(x) &= \text{cost}(x) - \text{mincost}(x) \\
&= \text{cost}(x) - \min(\text{cost}(x), \text{mincost}(l), \text{mincost}(r)) \\
&= \max(0, \text{cost}(x) - \text{mincost}(l), \text{cost}(x) - \text{mincost}(r)) \\
&= \max(0, \Delta\min(l) - \Delta\,\text{cost}(l), \Delta\min(r) - \Delta\,\text{cost}(r)). \qquad (1)
\end{aligned}$$

Furthermore, the minimum of the subtree can be located by knowing which term attains the maximum in the last expression.

Back to the updates for $\Delta\min(x)$. The only subtrees that change are those of w and v, and so only those $\Delta\min$ values change. Using (1), one can see that

$$\begin{aligned}
\Delta\min'(w) &= \max(0, \Delta\min(b) - \Delta\,\text{cost}'(b), \Delta\min(c) - \Delta\,\text{cost}(c)) \\
\Delta\min'(v) &= \max(0, \Delta\min(a) - \Delta\,\text{cost}(a), \Delta\min'(w) - \Delta\,\text{cost}'(w)).
\end{aligned}$$

Notice that $\Delta\min'(v)$ depends on $\Delta\min'(w)$ that was just computed.

Similar when we perform the splicing step given in Figure 3, Δ cost only change for v and u and only $\Delta\min(w)$ changes. The updates are:

$$\begin{aligned}
\Delta\,\text{cost}'(v) &= \Delta(\mathit{cost}(v)) - \Delta(\mathit{cost}(w)), \\
\Delta\,\text{cost}'(u) &= \Delta\,\text{cost}(u) + \Delta\,\text{cost}(w), \\
\Delta\min'(w) &= \max(0, \Delta\min(v) - \Delta\,\text{cost}'(v), \Delta\min(z) - \Delta\,\text{cost}(z)).
\end{aligned}$$

3.2 Implementation of Operations

We now describe the implementation of each of the desired operations on a dynamic tree, making extensive use of the EXPOSE operation.

- MAKE-TREE(v)

  Simply create a tree with the single node v.

- FIND-ROOT(v)

  First, run EXPOSE(v). Then follow right children until a leaf w of the splay tree containing v is reached. Now, SPLAY(w), and then return w.

- FIND-COST(v)

  First, run EXPOSE(v). Now v is the root, so return $\Delta\,\text{cost}(v) = \text{cost}(v)$. Note that the actual computations here were done by the updates of $\Delta\,\text{cost}(v)$ and $\Delta\min(x)$ within the SPLAY and SPLICE operations.

- FIND-MIN(v)

  First, run EXPOSE(v). Now, let's rewrite (1):

  $$\Delta\min(v) = \max\{0, -\Delta\,\text{cost}(\text{left}(v)) + \Delta\min(\text{left}(v)), -\Delta\,\text{cost}(\text{right}(v)) + \Delta\min(\text{right}(v))\}.$$

  If $\Delta\min(v) = 0$, then SPLAY(v) and then return v, as the minimum is achieved at v. Else, if $-\Delta\,\text{cost}(\text{left}(v)) + \Delta\min(\text{left}(v)) > -\Delta\,\text{cost}(\text{right}(v)) + \Delta\min(\text{right}(v))$, then the minimum is contained in the left subtree and we walk down it recursively. Otherwise, the minimum is contained in the right subtree, so we recurse down the right. Once we have found the minimum, we splay it.

- ADD-COST(v, x)

  First, run EXPOSE(v). Add x to $\Delta\,\text{cost}(v)$ and subtract x from $\Delta\,\text{cost}(\text{left}(v))$. Also update $\Delta\,\min(v)$ (using (1)). (The $\Delta\,\min$ value of other nodes is unchanged.)

- CUT(v)

  First, run EXPOSE(v). Add $\Delta\,\text{cost}(v)$ to $\Delta\,\text{cost}(\text{right}(v))$. Remove the edge $(v, \text{right}(v))$.

- LINK(v, w, x)

  First, run EXPOSE(v) and EXPOSE(w). Then, add the root w as a middle child of v. Add $\Delta\,\text{cost}(w) - x$ to $\Delta\,\text{cost}(\text{right}(v))$ and to $\Delta\,\text{cost}(\text{left}(v))$. Also update $\Delta\,\min(w)$.

4 Analysis of Dynamic Trees

We now give an amortized analysis of cost of operations in these dynamic trees. We will see that any sequence of m dynamic tree operations on n nodes will take $O((m+n)\log n)$ time.

4.1 Potential Function

We will use the following potential function in our analysis, motivated by our analysis of splay trees. For each node x, let $w(x) = 1$ be the weight assigned to x, and define

$$s(x) = \sum_{y \in T_x} w(y),$$

where T_x is the entire virtual tree subtree attached at x. Then, consider $r(x) = \log_2 s(x)$ and take our final potential function to be

$$\phi(T) = 3 \sum_{x \in T} r(x).$$

This differs from the potential function for splay trees in 2 ways. First T_x is defined over the entire virtual tree and secondly we have this additional factor 3. We will see later why the constant factor of 3 was chosen here.

4.2 Runtime of the Expose Operation

We first analyze the runtime of EXPOSE(v), since it is used in all other operations. We look at each step of EXPOSE(v) separately. Let k be the number of middle edges separating v from the root of the entire virtual tree. Equivalently, k is the number of SPLAY operations performed during Step 1.

- **Step 1:** Let $t(v)$ be the root of the splay tree containing v. Recall that the amortized cost of SPLAY(v) was $3(r(t(v)) - r(v)) + 1$ when we used the potential function

  $$\phi_{\text{splay}}(T) = \sum_{x \in T} r(x).$$

  We now have the potential function $\phi(T) = 3\phi_{\text{splay}}(T)$, so the $3(r(t(v)) - r(v))$ term here should be multiplied by 3 to obtain an amortized runtime of $9(r(t(v)) - r(v)) + 1$ for each call of SPLAY(v) (the $+1$ corresponds to the cost of the last zig, if any, and so we do not need to multiply it by 3).

We are using the SPLAY operation on the k nodes $v, p(t(v)), \ldots, (p \circ t)^{k-1}(v)$ in this step, meaning that we get a total amortized runtime of

$$\sum_{i=0}^{k-1} 9\left[r(t((p \circ t)^i(v))) - r((p \circ t)^i(v))\right] + 1 \leq 9[r(\text{root}) - r(v)] + k,$$

since we have that $r(t(p \circ t)^{i-1}(v)) \leq r((p \circ t)^i(v))$, so the sum telescopes. The amortized cost of step 1 is therefore $O(\log n) + k$ (since $r(\text{root}) - r(v) \leq \log n$).

- **Step 2:** Splicing does not change the value of $\phi(T)$, so the amortized cost for this step is the same as its actual cost of k.
- **Step 3:** We are using the SPLAY operation once on node v at distance k from the root, so this has an actual cost of k. Using the fact that our potential ϕ has an additional factor 3 in its definition compared to the splay tree version, we get from the amortized analysis of splaying that:

  $$k + \frac{1}{3}\Delta\phi(T) \leq 3[r(\text{root}) - r(v)] + 1 = O(\log n).$$

  Multiplying by 3, we see that we can also account for the additional cost of $2k$ from steps 1 and 2, and have an amortized time of $O(\log n)$.
- **Total:** We get $O(\log n) + k$ in step 1, k in step 2, and these $2k$ plus step 3 gives $O(\log n)$, for a total of $O(\log n)$.

4.3 Runtimes of all Operations

We can now briefly summarize the runtimes of all other operations in terms of EXPOSE.

- FIND-COST, FIND-ROOT, FIND-MIN, ADD-COST

  Each of these operations requires at most one use of EXPOSE, at most one run of SPLAY, and at most one search of the tree which can be charged to the last splay. Therefore, they each run in $O(\log n)$ amortized time.
- CUT

  We again use EXPOSE once. We now consider the effect of the other actions on the potential function. Removing the edge $(v, \text{right}(v))$ decreases $s(v)$ by $s(\text{right}(v))$ and leaves $s(x)$ unchanged for all other x, so it decreases $\phi(T)$, which we can safely ignore. This gives an amortized runtime of $O(\log n)$.
- LINK

  We use EXPOSE twice. Now, when we link w to v, we see that $r(v)$ increases by $O(\log n)$, and all other $r(x)$ remain unchanged. Hence, this operation increases $\phi(T)$ by $O(\log n)$, giving a total amortized runtime of $O(\log n)$.

With this analysis, we see that every operation has amortized time $O(\log n)$. A sequence of m operations has therefore amortized time $O(m \log n)$. Furthermore, the potential function satisfies

$$\phi(T) = \sum_{x \in T} r(x) \leq \sum_{x \in T} \log n \leq n \log n,$$

meaning that any increase in potential is at most $O(n \log n)$, implying that the total cost is at most $O((m+n)\log n)$. We now have the following theorem.

Theorem 1 *Any m operations on a dynamic tree with n nodes run in $O((m+n)\log n)$ time.*