CS1570 Lecture Notes

Claire Mathieu

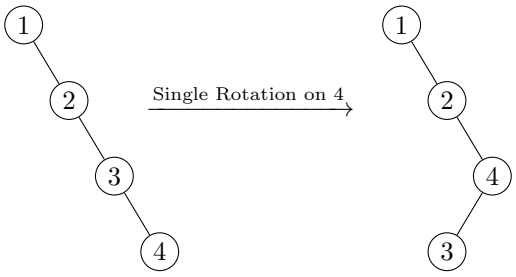
Fall 2026

1 Oct 20: Splay Trees: Amortized Analysis and the Access Lemma

A propos of nothing, Note on Current Literature: Recent lecture on discrete FFT algorithms shows an execution runtime of $\mathcal{O}(n \log n)$. A noted open problem (Problem 130 from OpenAI) claims an improved upper bound of $\mathcal{O}(n(\log n)^{1-\delta})$ where $\delta = 10^{-13}$.

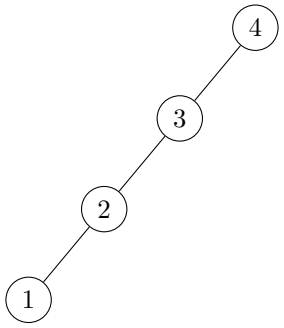
Before analyzing the standard splay operation, which utilizes double-rotations, we examine a simpler, naive alternative: restructuring a tree by repeatedly performing single rotations to move the accessed element x to the root. While intuitive, this naive strategy fails to ensure an efficient tree shape over a sequence of accesses.

Consider a degenerate right-skewed tree of size n, forming a single linked path where the nodes are ordered $1, 2, \ldots, n$ from the root down to the deepest leaf. Suppose a request is made for the deepest node, $n = 4$.

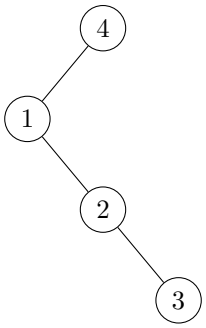


When we perform repeated single left rotations on n with its respective parents to lift it to the root, each single rotation only modifies the local parent-child relationship.

When I asked AI to give me the example (which I had erased from the previous lecture), it did it completely wrong (!) and wrote confidently: " Once node n reaches the root, the remaining path of nodes $1, 2, \ldots, n-1$ *is simply inverted into a left-skewed degenerate path" with the following incorrect picture:*



Instead, here is the correct picture after the splay:



The real cost of this operations sequence is $\Theta(n)$. Crucially, the maximum depth of the tree remains $\Theta(n)$, meaning the tree has not been balanced or significantly shortened. Accessing the node 3 next will incur another $\Theta(n)$ cost, and alternating requests between the deepest nodes creates an adversarial sequence with an average cost of $\Theta(n)$ per operation. This demonstrates that **single rotations alone cannot reduce tree depth effectively**.

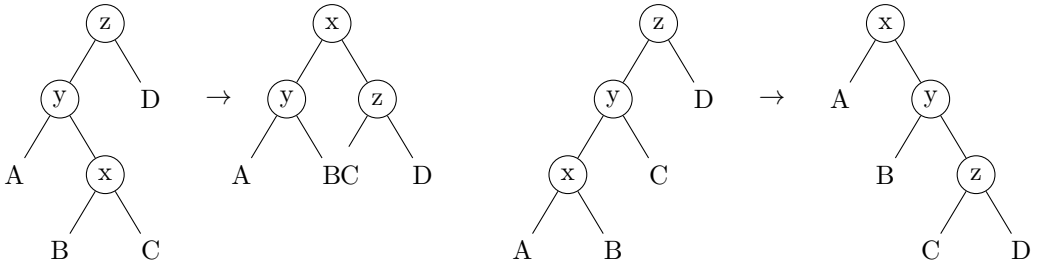
A **splay tree** is a self-adjusting binary search tree where every accessed node is moved to the root via a sequence of structured rotations called a **splay operation**.

To search for an element x, we locate it as in a standard BST and then restructure the tree to bring x to the root using the following loop:

Algorithm 1 Splay Operation

```
while x is not the root do
    if parent(x) == root then
        Perform a single rotation to bring x to the root (the final "zig" step).
    else
        Let y = parent(x) and z = grandparent(x).
        Perform a double-rotation centered at z according to the geometric configuration of x, y, z.
    end if
end while
```

Geometric Rotations Visualized (up to left-right symmetry):



Theorem 1. *A sequence of m requests on a splay tree of size n has a total cost of $\mathcal{O}(m \log n + n \log n)$.*

To prove Theorem 1 we will use an amortized analysis.
We need to be a bit more specific. We normalize the basic unit of (real) work such that:

- The cost of any double-rotation = 1.
- The cost of a single final rotation under the root = 1.

To do an amortized analysis, we define a potential function Φ associated to a tree, capturing some notion of balance. (There exists other methods to do amortized analyses but in this course we are only studying the potential function approach). Let $T(x)$ denote the subtree rooted at node x and its **size** $s(x) = |T(x)|$ be

equal to the number of nodes contained within $T(x)$. Define the **rank** of a node x as: $r(x) = \lfloor \log_2(s(x)) \rfloor$. The **potential** $\Phi(T)$ of a tree T is then defined as the sum of the ranks of all its constituent nodes:

$$\Phi(T) = \sum_{x \in T} r(x)$$

Before moving on to the proof proper, let us explore this potential function a bit. The Theorem has a bound of $m \log n$ for m operations, so the cost should be something like $\log n$ per operation. The idea is that when there is an operation whose real cost is much greater than $\log n$, it's because it has accessed a node that was very ddep in the tree, the tree was very unbalanced, and in that case, the splay operation helps rebalance the tree, and that rebalancing act ought to translate into a reduction of potential that makes up for the real cost incurred.

Potential Bounds: Since $1 \leq s(x) \leq n$, the ranks range from 0 to $\lfloor \log_2 n \rfloor$ and the potential satisfies

$$0 \leq \Phi(T) \leq \mathcal{O}(n \log n)$$

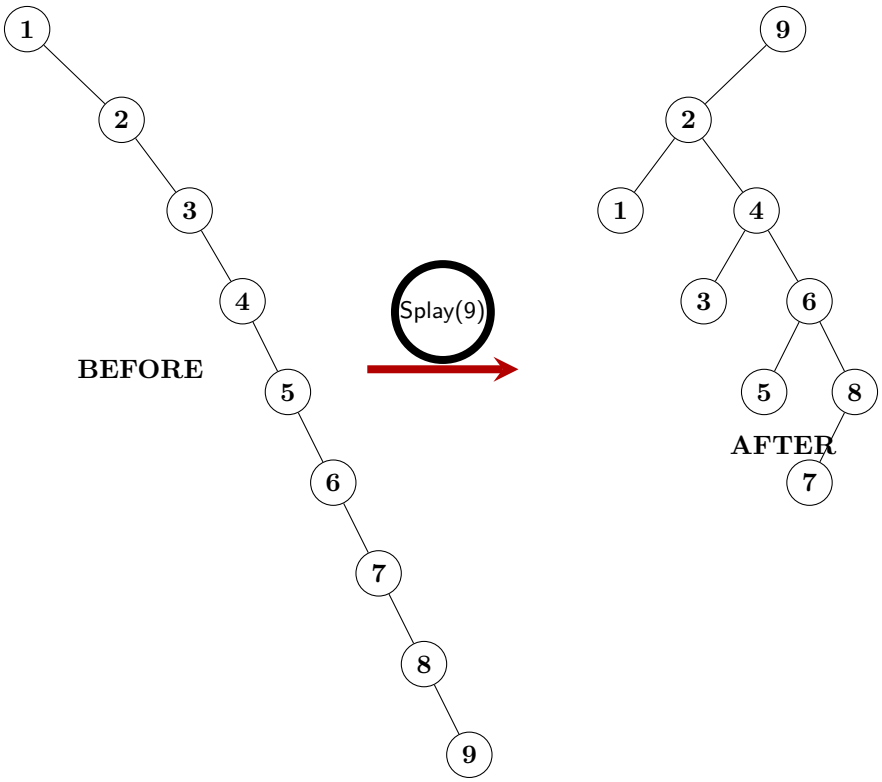
Consider the case of a skew, highly unbalanced tree (see Figure below). For a highly unbalanced right-skewed tree of size n, the potential reaches its maximum value:

$$\Phi_{\text{before}} \approx \sum_{i=1}^{n} \log_2 i \approx n \log_2 n$$

What happens after the splay? The depth has roughly halved. All the off-spine vertices have rank 0. The path contracts, significantly reducing the potential:

$$\Phi_{\text{after}} \approx \sum_{\substack{i=1 \\ i \text{ even}}}^{n} \log_2 i \approx \frac{1}{2} n \log_2 n$$

I had to ask the AI twice to get the following picture. The first time it generated an output tree that was completely wrong after the splay operation. It did not even respect the binary search tree order!



Amortized Cost Framework: The amortized cost of an operation is defined as:

$$\text{Amortized Cost} = \text{Real Cost} + \Delta\Phi$$

Lemma 1 (The Access Lemma)**.** *The amortized cost of serving an access request at a node x in a tree of size n is at most:*

$$1 + 3(r(root) - r(x)) \leq 1 + 3\lfloor \log_2 n \rfloor$$

1.1 Proof of the Main Theorem Assuming the Lemma

Consider a sequence of m requests. The total real cost is given by the sum of individual real costs:

$$\begin{aligned}
\text{Total Real Cost} &= \sum_{\text{requests}} \text{realcost(req)} \\
&= \sum_{\text{req}} (\text{amortizedcost(req)} - \Delta\Phi) \\
&= \sum_{\text{req}} \text{amortizedcost(req)} - \sum_{\text{req}} \Delta\Phi
\end{aligned}$$

Because the potential sum telescopes, we have $\sum_{\text{req}} \Delta\Phi = \Phi_{\text{last}} - \Phi_{\text{initial}}$. Thus, using the Lemma:

$$\begin{aligned}
\text{Total Real Cost} &= \sum_{\text{req}} \text{amortizedcost(req)} - (\Phi_{\text{last}} - \Phi_{\text{initial}}) \\
&\leq \sum_{\text{req}} (1 + 3\lfloor \log_2 n \rfloor) - \Phi_{\text{last}} + \Phi_{\text{initial}}
\end{aligned}$$

Since $-\Phi_{\text{last}} \leq 0$ and $\Phi_{\text{initial}} \leq \mathcal{O}(n \log n)$, substituting the bounds gives:

$$\begin{aligned}
\text{Total Real Cost} &\leq m(1 + 3\log_2 n) + \mathcal{O}(n \log n) \\
&= \mathcal{O}(m \log n + n \log n) \quad \blacksquare
\end{aligned}$$

So all that is left is to prove the Lemma.

1.2 Proof of the Access Lemma

We take an atomic view of the splay operation. A splay operation is actually a sequence of (possibly many) double-rotations followed by possibly one single rotation. Let' s call those *atomic* operations. The total cost of serving a request is the sum of the costs of all double-rotations (1 for each) plus an optional single final rotation (1):

$$\text{Cost(serving a request)} = \sum_{\{\text{double rotations}\}} 1 + (\text{possible final single rotation})$$

Let Φ, r denote the potential and rank function before an atomic step, and Φ', r' denote the potential and rank function after the step.

Atomic Amortized Cost of a Step: The real cost of a single double-rotation step is 1. Observe that by definition of Φ, the change in potential during this step only affects the nodes involved (x, y, and z). So we can take a local view. That is a very nice property of our choice of definition of potential! Thus, the atomic amortized cost is:

$$\text{Amortized Cost}_{\text{atomic}} = 1 + (r'(x) + r'(y) + r'(z)) - (r(x) + r(y) + r(z))$$

We claim that for any double-rotation step, this cost is bounded above:

$$\text{Amortized Cost}_{\text{atomic}} \leq 3(r'(x) - r(x)) \tag{1}$$

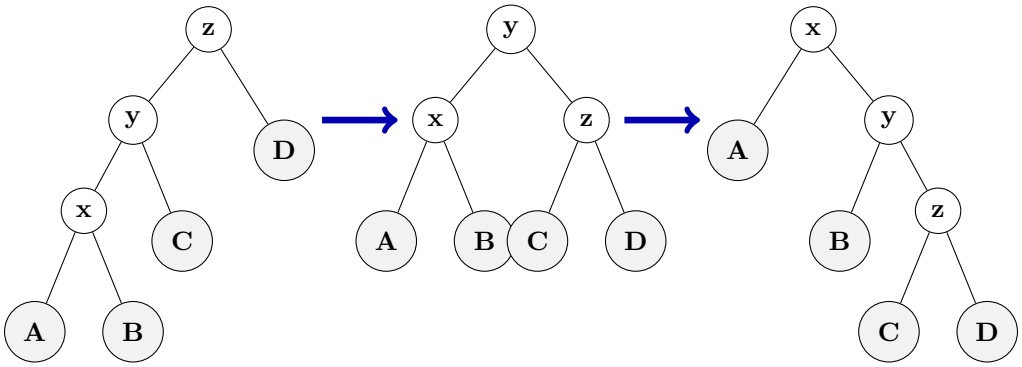
Assume that that's true, and that the final rotation has cost at most $1+3(r'(x)-r(x))$. By summing those bounds across all consecutive atomic steps in a splay operation, the terms telescope, proving the Lemma:

$$1 + \sum 3(r'_i(x) - r_i(x)) \leq 1 + 3(r(\text{root}) - r(x))$$

Thus, it only remains to analyze the amortized cost of a double-rotation (or of the final single rotation).

At this point the AI got totally confused. It as unable to give me a correct prof in spite of my having provided a picture. It thought that $s'(x) > s(x)$ implied $r'(x) > r(x)$, which is wrong. After some back-and-forth, I managed to convince it ("nice catch!", it said...), but then it wanted to redo the proof with a different rank function, without the floor function. Since that was not the proof I did in lecture, I did not want that. I eventually gave up and wrote the end of the proof by myself.

Case 1: Zig-Zig Double-Rotation A zig-zig step occurs when x and y are both left children (the case when they are both right children is symmetric). We perform a right rotation on the edge (y, z) followed by a right rotation on the edge (x, y). Let us draw the intemediate configuration.



The first subcase is if $r(x) = r(z)$. Let $r = r(x) = r(y) = r(z) = r'(x)$. Consider the middle picture. There, the subtree of node x and the subtree of node y both have rank r. *It follows, by definition of rank, that the subtree of node z must have rank $< r$. For example, $r = 9$ means subtrees of size ranging from* 512 *to* 1023*: well, it cannot be that two siblings and their parent all have sizes in that range.* Comparing the middle picture to the right picture, we infer that $r'(z) < r$. Thus:

$$1 + (r'(x) + r'(y) + r'(z)) - (r(x) + r(y) + r(z)) \leq 1 + (r + r + r - 1) - (r + r + r) \leq 0 = 3(r'(x) - r(x)),$$

as desired.

The second subcase is if $r(x) < r(z)$. Then it is rather simple: we have $r'_x = r_z$, but also $r'_y + r'_z \leq 2r'_x$ and $r_x + r_y \geq 3r_x$, so

$$1 + (r'(x) + r'(y) + r'(z)) - (r(x) + r(y) + r(z)) = 1 + (r'(y) + r'(z)) - (r(x) + r(y)) \leq$$

$$\leq 1 + 2r'_x - 2r_x = 1 + 2(r'_x - r_x) \leq 3(r'_x - r_x)$$

where the last inequality is implied by the fact that $r(x) < r(z) = r'(x)$.

Case 2: Zig-Zag Double-Rotation I am going to leave that as an exercise.

Final Step: Last Single Rotation If x is one level below the root, a single final rotation is performed. The real cost is 1, and the change in potential is bounded by $r'(\text{root}) - r(x)$. Thus, the amortized cost of this terminal step is at most: Amortized $\text{Cost}_{\text{final}} \leq 1 + 3(r'(\text{root}) - r(x))$ Summing the bounds from all atomic cases completes the proof of the Access Lemma.