CS 224: Advanced Algorithms Spring 2017

Lecture 8 — Feb 16, 2017

Prof. Jelani Nelson *Scribe: Tiffany*

1 Overview

In the last lecture we covered the properties of splay trees, including amortized $O(\log n)$ time for operations, static finger, dynamic finger, and others.

In this lecture we analyze splay trees, and begin discussion of online algorithms.

2 Splay Tree Analysis

2.1 Notation

Define a weight function on nodes $w : \{nodes\} \to \mathbb{R}_{\geq 0}$. We will write our potential function in terms of w, and by taking different weight functions will allow us to prove different properties, such as amortized $O(\log n)$ operations and static optimality later on in this lecture. For now, think of $w(x) = 1$.

Additionally, define

$$s(x) \coloneqq \sum_{\substack{y \in x \text{ subtree} \\ \text{including} x}} w(y)$$

and let the rank of a node x be $r(x) \coloneqq \lg(s(x))$. It's not clear including a log here is well motivated, and indeed the original authors of the paper say as much.

The potential function on a particular state S of the splay tree is

$$\phi(S) \coloneqq \sum_{x \in T} r(x)$$

Additionally, let $W = \sum_{x \in T} w(x)$ be the size of the root.

2.2 Analysis of splay(x) taking $O(\log n)$ time

Set $w(x) = 1$.

Claim 1. *The amortized cost of splay(x) is*

$$\tilde{t}_{splay} \leq 3 \cdot (r(root) - r(x)) + 1 \leq 3 \log(n) + 1$$

since the root has weight n.

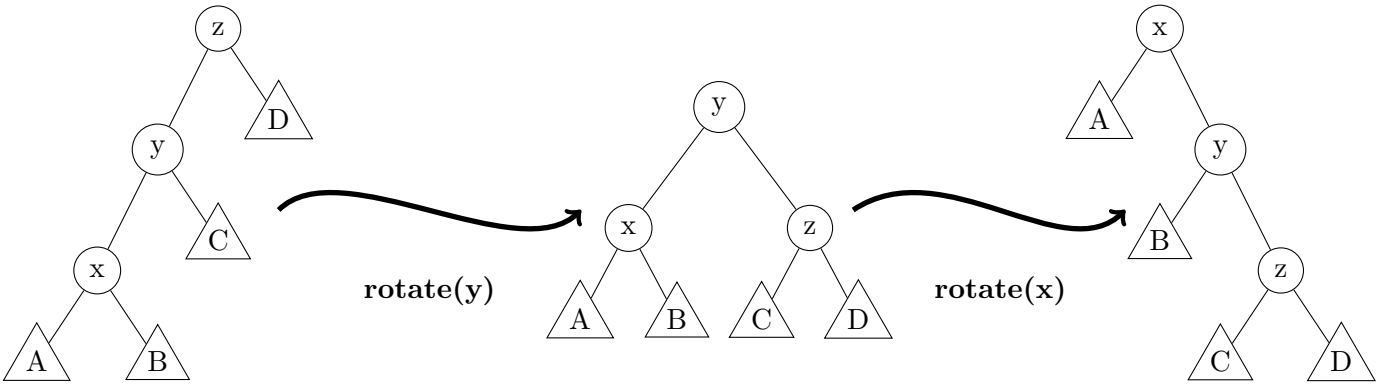

Figure 1: Zig-zig step in splay operation (Case 2)

Claim 2. *Suppose we are doing one of the cases of (1) zig, (2) zig-zig, or (3)zig-zag. Then amortized cost of performing each of these operations is*

- *Case (1):* $3(r'(x) - r(x)) + 1$ *where* $r'(x)$ *is the rank of* x *after the operation.*
- *Cases (2), (3):* $3(r'(x) - r(x))$

Note that Claim 2 implies Claim 1 because of a telescoping sum of the operations up the tree, so the final cost will be about $3(r(root) - r(x))$ where $r(x)$ is the rank of the node that x was located before splaying.

Proof. We will just do the zig-zig case (case 2), which as a reminder is when $x = y.left, y = z.left$ or the symmetric case. Other cases are similar.

After the operation we know that $r'(x) = r(z)$. Using the definition of amortization,

$$\text{amortized cost} = \text{actual cost} + \Delta\Phi$$

The actual cost is just 2 rotations. We also know that since x, y, z are the only ranks that change, then using $r'(x) = r(z)$

$$\begin{aligned}\Delta\Phi &= (r'(x) + r'(y) + r'(z)) - (r(x) + r(y) + r(z)) \\ &= r'(y) + r'(z) - r(x) - r(y) \\ &= r'(x) + r'(z) - 2r(x) \end{aligned} \tag{1}$$

where the last step follows due to $r(x) \leq r(y), r'(y) \leq r'(x)$ due to the parental relationship between x and y before and after the operation.

We try to bound $r(x) + r'(z)$:

$$\begin{aligned}
\frac{r(x) + r'(z)}{2} &= \lg\left(\sqrt{s(x)s'(z)}\right) \\
&\leq \lg\left(\frac{s(x) + s'(z)}{2}\right) && \text{by AM-GM} \\
&\leq \lg\left(\frac{s'(x)}{2}\right) = r'(x) - 1
\end{aligned}$$

where the last line follows since $s(x) = w(x) + s(A) + s(B), s'(z) = w(z) + s(C) + s(D)$, and $s'(x) = w(x) + w(y) + w(z) + s(A) + s(C) + s(D) \geq s(x) + s'(z)$.

Therefore, we have that

$$r'(z) \leq 2r'(x) - r(x) - 2$$

Subsituting into equation (1), we get

$$\Delta\Phi = r'(x) + r'(z) - 2r(x) \leq r'(x) + 2r'(x) - r(x) - 2 - 2r(x) = 3(r'(x) - r(x) - 2$$

and therefore

$$\tilde{t}_2 = 2 + 3(r'(x) - r(x)) - 2 = 3(r'(x) - r(x))$$

as desired. □

So we can say splay and query take $O(\lg n)$, since query is simply walking down the tree and a splay. However, insert must be dealt with differently since we increased the potential function.

2.3 Analysis of insert(x)

The amortized cost of insert is $\Delta\Phi$ from inserting k levels down and $O(splay) = O(\log n)$. It suffices to bound $\Delta\Phi$ from the insert by $O(\log n)$ to achieve the desired amortized runtime.

Suppose in order to insert x, its parents above it in the BST are $y_1 = x.parent, y_2 = y_1.parent, ...y_k = root$. Then

$$\begin{aligned}
\Delta\Phi &= \sum_{j=1}^{k} r'(y_j) - r(y_j) \\
&= \sum_{j} \lg\left(\frac{s(y_j) + 1}{s(y_j)}\right) && w(x) = 1 \\
&= \lg\left(\pi_j \frac{s(y_j) + 1}{s(y_j)}\right) \\
&\leq \lg\left[\frac{s(y_2)}{s(y_1)} \cdot \frac{s(y_3)}{s(y_2)} \cdot ... \cdot \frac{s(y_k) + 1}{s(y_k)}\right] && s(y_j) + 1 \leq s(y_{j+1}) \\
&= \lg\left(\frac{s(y_k) + 1}{s(y_1)}\right) \\
&\leq \lg(s(y_k) + 1) = O(r(y_k)) = O(\lg n)
\end{aligned}$$

as desired.

2.4 Static Optimality

We want to be competitive with a BST built for the distribution of queries given, in the case that there are no inserts or deletes (there's a dynamic programming algorithm for constructing this). We claim that if a sequence of queries σ is long enough, splay trees achieve a constant within the optimal static BST, which is proven in the original splay tree paper by Tarjan and Sleator[2].

Formally, given a query sequence σ and fixed BST T,

$$C_{splay}(\sigma) \leq O(n^2 + C_T(\sigma))$$

and once $|\sigma^2| > n^2$, this implies $C \cdot OPT$. Note that we can achieve $n \log n$, which is better than n^2, though we won't prove it here.

Proof. Suppose T puts node$x$ at the level ℓ_x of the tree where $\ell_{root} = 0$. Define $w(X) = \frac{1}{3^{\ell_x}}$.

Recall,

$$\begin{aligned} \sum(\text{amortized costs}) &= \sum(\text{actual costs}) + \Phi_{final} - \Phi_{init} \\ \Rightarrow \sum(\text{actual costs}) &= \sum(\text{amortized costs}) + \Phi_{init} - \Phi_{final} \end{aligned}$$

we will show that $\sum(\text{amortized costs}) = O(OPT)$ and $\Phi_{init} - \Phi_{final} = O(n^2)$, which is sufficient to prove the claim. Here, we relax the condition that Φ must be positive and zero on the initial data structure, since we account for the difference in potential explicitly.

Difference in potential: Note that $\Phi(S) = \sum_x \lg(s(x))$, and we want to know how big of a magnitude this can get. Well, we know that $\frac{1}{3^n} \leq w(x) \leq 1$, so $s(x) \geq \frac{1}{3^n} \Rightarrow \lg(s(x)) \geq -cn$. So we know that

$$\sum_x \lg(s(x)) \geq -cn^2$$

for some constant c. Addditionally, we know since each $s(x) \leq 1$, that

$$\sum_x \lg(s(x)) \leq O(n \log n)$$

Thus, the difference in potential is no more than $O(n^2)$.

Amortized costs: Amortized cost of query(x) is O(amortized cost of splay(x)). It suffices to show that the amoritized cost of splay(x) is the same cost of accessing an item in the perfect BST T, which is ℓ_x.

$$O(root) - r(x)) = O\left(\lg\left(\frac{W}{s(x)}\right)\right)$$

where $W = \sum_z w(z)$.

We want this amoritized cost to be $O(\ell_x)$. We note that $W \leq C$ for some constant C, since

$$W = \sum_z w(z) = \sum_z \frac{1}{3^{\ell_z}} = \sum_{\ell=0}^{n} \frac{1}{3^\ell} \cdot (\#\text{nodes at level } \ell) \leq \sum_\ell \frac{2}{3}^\ell = O(1)$$

We know that $W \geq 1$, so $W = \Theta(1)$. Therefore,

$$\begin{aligned}\lg\left(\frac{W}{s(x)}\right) &= \Theta(1) - \lg(s(x)) \\ &= \Theta(1) - \lg\left(\frac{1}{3^{\ell_x}}\right) \\ &= \Theta(1 + \ell_x)\end{aligned}$$

so we are done, sinc ethe cost is exactly a constant factor off of ℓ_x, the cost in the optimal BST.

□

3 Online Algorithms

Online algorithms are a class of algorithms in which one must make a sequence of irrevocable decisions without knowing the future. However, we want to be competitive the best sequence of decisions in hindsight.

Definition 3. *Let OPT be the optimal strategy given omniscience. We say that a strategyS is r-competitive if for a sequence of events σ*

$$cost_S(\sigma) \leq r \cdot cost_{OPT}(\sigma) + O(1)$$

3.1 Examples

3.1.1 Ski Rental Problem

Suppose you are on a ski trip, and you let your friends dictate how long your group will continue skiing. Your friends will inform you each morning whether you will continue to ski that day, or leave the resort. You need skis, and there are two options:

- rent skis: \$1 per day
- buy skis: \$ B, one time payment at any time

If you knew ahead of time how many days you wil stay at the resort n, your strategy would be to buy the skis on the first day if $n > B$ and rent otherwise. What should your strategy be to minimize your regret?

Strategy: You rent for $B - 1$ days. If by the Bth day you did not leave, buy the skis.

This is optimal for staying $\leq B - 1$ days, and the worst case is if you stay for exactly B days and leave the next day. Then you spent \$2B-1 dollars, but opt spent B, so you're always guaranteed to be within a factor of 2 from the optimal.

Indeed, the ski rental problem has competitve ratio $r = 2$, and no other online strategy can do better, which makes it the optimal.

3.1.2 Free Swag at the Career Fair

You're standing in the middle of a hallway lined with $2n + 1$ rooms at the career fair. Your friend texts you saying that there is some room in which a free laptops (!!!) are being given out by some company who really wants you to work for them. You want to get there as soon as possible, because if you get there too late, all the laptops may be claimed. Walking by a room takes constant time, and looking in as you walk past a room costs no time. What should be your strategy of exploring rooms?

If you knew which room t had the swag in it, you would just walk to t, incurring $|t|$ cost to get there.

Suppose you're at room 0, and there are n rooms on either side of you, which we'll number $1, 2, ...n$ on your right and $-n, -n + 1, ... - 1$ on your left. One strategy might be to go to rooms $1, -1, 2, -2, 4, -4, ...$ until you find the room that you're looking for. This cost will be

$$(1 + 1 + 1 + 1) + (2 + 2 + 2 + 2) + (4 + 4 + 4 + 4) + ... = 4(2^0 + 2^1 + ... + 2^j) \approx 4 \cdot 2^{j+1}$$

The worst case scenario arises when the jackpot room is $t = -2^j - 1$. Then we would pay $4 \cdot 2^{j+1} + 2 \cdot 2 \cdot 2^{j+1} + t = 13t$. So this algorithm is 13-competitive.

There exists a 9-competitive strategy where we traverse the rooms $1, -2, 4, -8, ...$, which we won't analyze here.

4 List Update Problem

This mechanism was originally posed by Sleater and Tarjan[1], before which analysis of this kind was usually done assuming a known distribution of list accesses.

There are n nodes in a linked list. We are given an access sequence σ where $\sigma_i \in [n]$, and we will call query on each of them.

query(x):

- start at head of the linked list
- cost(query(x)) = depth of x in the linked list
- for free, we can move x towards the front of the linked list
- for cost 1, we can swap two adjacent items that are not x.

Note that we can view paging as a list update problem with a different cost function, where

$$\text{cost}(\text{query}(x)) = \begin{cases} 0 & \text{if } x \in \text{ first } k \text{ positions} \\ 1 & \text{otherwise} \end{cases}$$

to simulate having $k < n$ space in cache, and unbounded memory.

4.1 Different Heuristics

- **Move to Front:** move x to the front of the list after querying x (note, this is exactly the Least Recently Used cache eviction strategy in paging)
- **Transpose:** move x one position closer to front
- **Frequency count:** Keeps items stored in decreasing order of access frequency

References

[1] Daniel Dominic Sleator, Robert Endre Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, 1985.

[2] Daniel Dominic Sleator, Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652686, 1985.