# Recitation 13 – *Ordered Sets, Augmented Trees, and Exam II Debrief*

Parallel and Sequential Data Structures and Algorithms, 15-210 (Fall 2014)

*November 11th, 2014*

## 1 Announcements

- Exam II was passed back today. Any questions or concerns?
- *RangeLab* is released! Get started early!

## 2 Exam II *Post-Mortem*


**Mean:** 70.53
**Median:** 71
**Standard Deviation:** 11.39

## 3 Treaps

Remember how we construct treaps: A treap is a rooted BST where each node has both a key and a *priority*. Think about it this way: the key decides where the node ends up from left to right, and the priority decides where it goes in the vertical dimension. The nice thing about this is that two treaps with the same keys and priority will always have the same structure, regardless of the order we add the nodes in. Additionally, if we're smart about how we assign priorities to keys, we can make our trees well balanced.

To see this in action, let's try making a basic BST for this sequence of key/value pairs (add nodes in order from left to right).

$$\langle (1,A),(2,B),(3,C),(4,D),(5,E),(6,F),(7,G),(8,H),(9,I)\rangle$$

Now let's construct a treap, where the tuples are (key, priority, value)

$$\langle (1,5,A),(2,4,B),(3,7,C),(4,3,D),(5,9,E),(6,2,F),(7,6,G),(8,8,H),(9,9,I)\rangle$$

## 4 Treap Analysis

In the lecture notes, you'll see an analysis of the expected depth of a node in a treap. Here, we'll analyze the expected number of children that a node has. The analysis is almost identical, but it'll be a useful exercise in probability and reasoning about trees.

Let's start by turning the question around. Let's find out the probability that node $j$ is an ancestor of node $i$. We'll designate an indicator random variable for this, $A_i^j$. Given this, what's an expression for the expected size of the subtree rooted at node $j$?

So what's the probability of $A_i^j = 1$? This expression should feel very much like something you did for quicksort.

## 5 Union

For two sets of size $m$ and $n$, where $m \leq n$, the cost of union is

$$O\left(m \lg\left(1 + \frac{n}{m}\right)\right)$$

What. Let's walk through the analysis of union to find out where this comes from. Recall that sets are implemented as randomized treaps (each element has a random priority assigned to it when it comes in), and this makes the treap more or less balanced. So for this problem, let's assume that the two treaps we're unioning are balanced.

Remember our two general-purpose treap helper functions: `split` and `join`. `split` takes a treap and a key and splits the treap at that key, giving the value at it (or NONE, if it doesn't exist), and the two subtrees. `join` takes two treaps (where all the left treap's keys are less than the right's), and a value to stick them together with, or NONE, and returns a new treap with all of the keys in each of them. Note that the requirement that one treap has smaller keys than the other means that we can't just do `join` for `union`. Both `split` and `join` have $O(\lg n)$ work, where $n$ is the combined number of nodes in their argument(s).

So let's look at the algorithm for union. When we union $T_1$, which has root $k$, left child $L_1$, and right child $R_1$, and $T_2$, we do this:

1. Split $T_2$ using $k$ into $L_2$ and $R_2$ (we don't care about the element)
2. Union $L_1, L_2$ into $L$ and $R_1, R_2$ into $R$
3. Join $L$, $k$, and $R$.

Assume that at each level, we let $T_1$ be the smaller treap.

Write the recurrence for the work we do in terms of $m$ and $n$

Now let's solve this recurrence, where $n + m = N$:

## 6 Let's Keep Growing Them Trees

**Q: What is an augmented tree?**

**A:** A balanced binary tree (such as a BST or treap) with a designated (associative) operation $f : \alpha \times \alpha \to \alpha$ and an operation `reduceVal`: $T \to \alpha$ which is equivalent to `reduce` $f\ I_f\ T$ where $I_f$ is an identity of $f$.

**Q: How is `reduceVal` different from `reduce`?**

**A:** Unlike `reduce`, `reduceVal` takes $O(1)$ work and span.

**Q: What?? How is this possible? (This one isn't rhetorical)**

**A:**

**Q: How are augmented trees implemented?**

**A:**

## 7 Let's Grow Some Money Trees!

You're working as a consultant for the famous QADSAN market, which wants to support the following query: *given a time range, return the maximum increase in stock value during that range* (i.e. the largest difference between two trades in which the larger amount comes after the smaller amount).

**Q: What keys, elements and function $f$ would you use in an augmented tree to support such queries? Hint:** the reduced value, on which $f$ operates, may include more information than the answer to the query.

**A:**

**Q: Calling `reduceVal` and taking the difference between the max and min will only give the maximum increase across an entire (sub)tree. How can we make this query for a specific range $(t_1, t_2)$ in $O(\lg n)$ work?**

**A:**

# 8 Ordered Sets

**Q: What are ordered sets?**

**A:** Ordered sets are a data structure that make it easy to perform queries like the one above (get the subset of key/value pairs with keys in a given range).

The basic idea is that, as we saw last week, our implementation of sets contains additional information not present in the signature: an ordering on keys. Ordered sets take advantage of this information to implement additional features, listed in the table below (note that, with an ordering on keys, it now makes sense to expose the `split` and `join` operations as part of the ordered set ADT).

| | | | | |
|---|---|---|---|---|
| $\texttt{last}(S)$ | : | $\mathbb{S} \to \mathbb{U}$ | = | $\max\ S$ |
| $\texttt{first}(S)$ | : | $\mathbb{S} \to \mathbb{U}$ | = | $\min\ S$ |
| $\texttt{split}(S,k)$ | : | $\mathbb{S} \times \mathbb{U} \to \mathbb{S} \times bool \times \mathbb{S}$ | = | $(\{k' \in S \mid k' < k\}, k \stackrel{?}{\in} S, \{k' \in S \mid k' > k\})$ |
| $\texttt{join}(S_1, S_2)$ | : | $\mathbb{S} \times \mathbb{S} \to \mathbb{S}$ | = | $S_1 \cup S_2$, assuming $\max S_1 < \min S_2$ |
| $\texttt{getRange}(S, k_1, k_2)$ | : | $\mathbb{S} \times \mathbb{U} \times \mathbb{U} \to \mathbb{S}$ | = | $\{k \in S \mid k_1 \leq k \leq k_2\}$ |
| $\texttt{rank}(S,k)$ | : | $\mathbb{S} \times \mathbb{U} \to int$ | = | $\lvert\{k' \in S \mid k' < k\}\rvert$ |
| $\texttt{select}(S,i)$ | : | $\mathbb{S} \times int \to \mathbb{U}$ | = | $k$ such that $\lvert\{k' \in S \mid k' < k\}\rvert = i$ |
| $\texttt{splitIdx}(S,i)$ | : | $\mathbb{S} \times int \to \mathbb{S} \times \mathbb{S}$ | = | $(\{k \in S \mid k < \texttt{select}(S,i)\},$ $\{k \in S \mid k \geq \texttt{select}(S,i)\})$ |

We saw above how to implement `getRange`. Implementing the last 3 functions efficiently requires a trick from augmented trees. Let's take a look at `rank`, which returns the numerical index of a key in a set according to the ordering. We can find the key in $O(\lg n)$ work, but, naively, finding the number of keys to the left of it would take $O(n)$ work, since we would need to count them all. The way around this is to store the size of each subtree at its root, so we can find the number of keys to the left of a given key by splitting on it and taking (now in $O(1)$) the size of the left tree. The code for `select` is in the lecture notes, and `splitIdx` is similar. As with augmented trees, trees must be constructed with a `makeNode` function that augments new nodes with the sum of the sizes of the two subtrees plus one.