Recitation 6

Treaps

6.1 Announcements

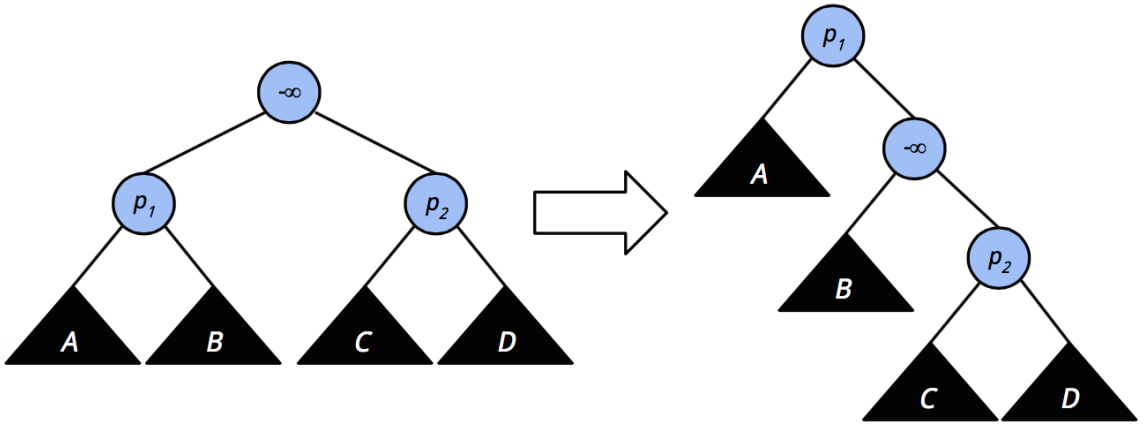
- *FingerLab* has been released, and is due **Thursday night**. It's worth 150 points.
- Midterm 1 is on **Friday**. You are allowed a single, double-sided, 8.5×11in sheet of paper for notes. You must write in **black or blue ink**.
- *RangeLab* will be released **next Thursday**.

6.2 Deletion

Consider the following strategy for deleting a key k from a treap:

1. Locate the node containing k,
2. Set the priority of k to be $-\infty$ (note that if k has children, then this breaks the heap invariant of the treap),
3. Restore the heap invariant by rotating k downwards until it has only leaves for children,
4. Delete k by replacing its node with a leaf.

A "rotation" in this case refers to the process of making one of k's children the root, depending on their relative priorities. For example, if k has two children with priorities p_1 and p_2 where $p_1 > p_2$, we rotate like so:



The case of $p_1 < p_2$ is symmetric. In turns out that this process is equivalent to calling `join` on the children of k. You should convince yourself of this.

We're interested in the following: in expectation, *how many rotations must we perform before we can delete k?*

Let's set up the specifics: we have a treap T formed from the sorted sequence of keys S, $|S| = n$. We're interested in deleting the key $S[d]$. Let T' be the same treap, except that the priority of $S[d]$ is now $-\infty$.

We need a couple indicator random variables:

$$A_j^i = \begin{cases} 1, & \text{if } S[i] \text{ is an ancestor of } S[j] \text{ in } T \\ 0, & \text{otherwise} \end{cases}$$

$$(A')_j^i = \begin{cases} 1, & \text{if } S[i] \text{ is an ancestor of } S[j] \text{ in } T' \\ 0, & \text{otherwise} \end{cases}$$

Task 6.1. *Write R_d, the number of rotations necessary to delete $S[d]$, in terms of the given random variables.*

The number of rotations is equal to the **number of nodes which aren't an ancestor of $S[d]$ in T, but are in T'**. Therefore we have

$$R_d = \sum_{i=0}^{n-1} (A')_d^i - \sum_{i=0}^{n-1} A_d^i$$

Task 6.2. *Give* $\mathbf{E}\left[A_d^i\right]$ *and* $\mathbf{E}\left[(A')_d^i\right]$ *in terms of i and d.*

We have both $A_d^i = 1$ and $(A')_d^i = 1$ if $S[i]$ has the largest priority among the $|d-i|+1$ keys between $S[i]$ and $S[d]$. However, notice that in the latter case, we already know that the priority of $S[i]$ is larger than that of $S[d]$, unless $i = d$. So we only need that $S[i]$ is the largest among the $|d-i|$ significant keys in this range. Therefore:

$$\mathbf{E}\left[A_d^i\right] = \frac{1}{|d-i|+1}$$

$$\mathbf{E}\left[(A')_d^i\right] = \begin{cases} 1, & \text{if } i = d \\ \frac{1}{|d-i|}, & \text{otherwise} \end{cases}$$

Task 6.3. *Compute* $\mathbf{E}[R_d]$. *For simplicity, you may assume* $1 \leq d \leq n-2$.

$$\begin{aligned}
\mathbf{E}[R_d] &= \sum_{i=0}^{n-1} \mathbf{E}\left[(A')_d^i\right] - \sum_{i=0}^{n-1} \mathbf{E}\left[A_d^i\right] \\
&= \left(\sum_{i=0}^{d-1} \mathbf{E}\left[(A')_d^i\right] + 1 + \sum_{i=d+1}^{n-1} \mathbf{E}\left[(A')_d^i\right]\right) - \left(\sum_{i=0}^{d-1} \mathbf{E}\left[A_d^i\right] + 1 + \sum_{i=d+1}^{n-1} \mathbf{E}\left[A_d^i\right]\right) \\
&= \left(\sum_{i=0}^{d-1} \frac{1}{d-i} + \sum_{i=d+1}^{n-1} \frac{1}{i-d}\right) - \left(\sum_{i=0}^{d-1} \frac{1}{d-i+1} + \sum_{i=d+1}^{n-1} \frac{1}{i-d+1}\right) \\
&= \left(H_d + H_{n-d-1}\right) - \left(\,(H_{d+1} - 1) + (H_{n-d} - 1)\,\right) \\
&= 2 + \left(H_d - H_{d+1}\right) + \left(H_{n-d-1} - H_{n-d}\right) \\
&= 2 - \frac{1}{d+1} - \frac{1}{n-d} \\
&\leq 2
\end{aligned}$$

6.3 Additional Exercises

Exercise 6.4. *Implement* `find` *and* `insert` *without using auxiliary BST functions like* `split` *and* `join`*.*

Exercise 6.5. *For treaps, suppose you are given implementations of* `find`*,* `insert`*, and* `delete`*. Implement* `split` *and* `join` *in terms of these functions such that they have the desired logarithmic cost bounds. You'll need to "hack" the keys and priorities; i.e., assume you can do funky things like insert a key with a* specific *priority, or construct a temporary "dummy" key.*

.