Recitation 7

Treaps and Combining BSTs

7.1 Announcements

- *FingerLab* is due **Friday afternoon**. It's worth 125 points.
- *RangeLab* will be released on **Friday**.

7.2 Deletion from a Treap

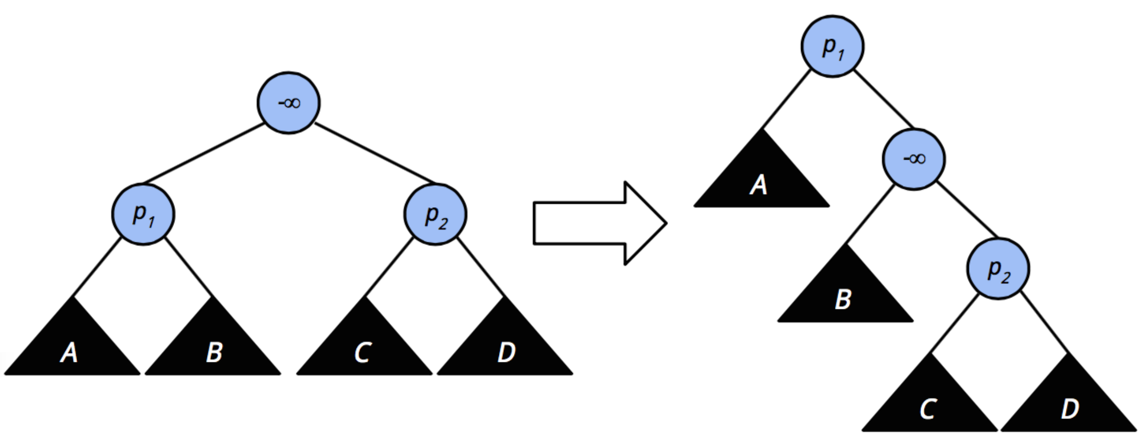
Recall that a treap is a BST with a priority function $p : U \to \mathbb{Z}$, where U is the universe of keys. You should think of p as a random number generator: for each key, it returns a random integer. A treap has two structural properties:

1. **BST invariant**: For every $\texttt{Node}(L, k, R)$, we have $\ell < k$ for every ℓ in L, and symmetrically $k < r$ for every r in R.
2. **Heap invariant**: For every $\texttt{Node}(L, k, R)$, we have that $p(k) > p(x)$ for every x in either L or R.

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:

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

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

Task 7.1. *Write R_d, the number of rotations necessary to delete $S[d]$, in terms of the given random variables.*

Task 7.2. *Give* $\mathbf{E}\left[X_d^i\right]$ *and* $\mathbf{E}\left[(X')_d^i\right]$ *in terms of i and d.*

Task 7.3. *Compute* $\mathbf{E}\left[R_d\right]$*. For simplicity, you may assume $1 \le d \le n-2$.*

7.3 Generalized Combination

In lecture, we discussed `union`, and argued that it has $O\left(m \log\left(\frac{n}{m}+1\right)\right)$ work and $O(\log(n)\log(m))$ span. The latter bound can be improved to $O(\log n + \log m)$ using *futures*[1], but that is outside the scope of this course.

Let's begin by inspecting the code for `union`.

Algorithm 7.4. *BST union.*

```
fun union (T1, T2) =
  case (T1, T2) of
    (_, Leaf) => T1
  | (Leaf, _) => T2
  | (Node (L1, x, R1), _) =>
      let val (L2, _, R2) = split (T2, x)
          val (L, R) = (union (L1, L2) || union (R1, R2))
      in joinMid (L, x, R)
      end
```

What about the functions `intersection` and `difference`? These can be implemented in a similar fashion as `union`, and as such have the same cost bounds. In this recitation, we'll establish this more concretely.

Task 7.5. *Implement a helper function* `combine` *which has* $O\left(m \log\left(\frac{n}{m}+1\right)\right)$ *work and* $O(\log(n)\log(m))$ *span for BSTs of size* n *and* m, $n \geq m$. *Use* `combine` *to implement* `intersection` *and* `difference`. *Conclude that all three of the set functions have the same cost bounds.*

Task 7.6. *Consider a function* `symdiff` *where* (`symdiff` (A, B)) *returns a BST containing all keys which are either in* A *or* B, *but not both. Implement* `symdiff` *in terms of* `combine`.

[1] http://dl.acm.org/citation.cfm?id=258517

7.4 Additional Exercises

Exercise 7.7. *Describe an algorithm for inserting an element into a treap by "undoing" the deletion process described in Section 7.2.*

Exercise 7.8. *For treaps, suppose you are given implementations of* `find`*,* `insert`*, and* `delete`*. Implement* `split` *and* `joinMid` *in terms of these functions. 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.*

Exercise 7.9. *Given a set of key-priority pairs* $(k_i, p_i) : 0 \leq i < n$ *where all of the* k_i*'s are distinct and all of the* p_i*'s are distinct, prove that there is a unique corresponding treap* T*.*

.