Recitation 6

Treaps

6.1 Announcements

- Midterm 1 is on Friday. You are allowed a single, double-sided, 8.5 × 11 in sheet of paper for notes.

- FingerLab is due next Friday, Mar 3.
6.2 Example

Recall that a treap is a BST with a priority function $p : U \rightarrow \mathbb{Z}$, 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 node $(L, k, R)$, we have $\ell < k$ for every $\ell \in L$, and symmetrically $k < r$ for every $r \in R$.

2. **Heap invariant**: For every node $(L, k, R)$, we have that $p(k) > p(x)$ for every $x$ in either $L$ or $R$.

**Task 6.1.** Build a treap from the following keys and priorities using two different strategies, and observe that the resulting treap is the same in both cases.

1. Run quicksort, creating a new node every time a pivot is chosen.

2. Beginning with an empty tree, sequentially insert keys in priority-order. Each newly inserted key should be placed at a leaf.

<table>
<thead>
<tr>
<th>$k$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(k)$</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
6.3 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:

![Rotation Diagram]

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:

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

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

**Task 6.2.** Write \( R_d \), the number of rotations necessary to delete \( S[d] \), in terms of the given random variables.

**Task 6.3.** Give \( E[X^i_d] \) and \( E[(X')^i_d] \) in terms of \( i \) and \( d \).

**Task 6.4.** Compute \( E[R_d] \). For simplicity, you may assume \( 1 \leq d \leq n - 2 \).
6.4 Additional Exercises

**Exercise 6.5.** Describe an algorithm for inserting an element into a treap by “undoing” the deletion process described in Section 6.3.

**Exercise 6.6.** 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 6.7.** 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 \(T\).