Problem 1.1 [Here Trees]. Describe and analyze a data structure for storing an ordered set of keys, initially empty. Your data structure should support the following operations and time bounds, where $n$ is the number of keys in the set currently represented by the data structure:

(a) $\text{predecessor}(k)$ in $O(\log n)$ time: find the largest key $\leq k$ that is in the set, and return a pointer to the node in the data structure representing that key.

(b) $\text{successor}(k)$ in $O(\log n)$ time: symmetrically

(c) $\text{predecessor-of-here}(k, x)$ in $O(1)$ time: given a pointer to the node $x$ representing a key $k$, find the largest key $< k$ that is in the set, and return a pointer to the node representing that key. (If there is no such key, return null.)

(d) $\text{successor-of-here}(k, x)$ in $O(1)$ time: symmetrically

(e) $\text{insert-after}(k, x)$ in $O(1)$ amortized time: given a pointer to the node $x$ representing the largest key $< k$, insert $k$ into the data structure (if it doesn’t already exist), and return a pointer to the node representing $k$.

(f) $\text{delete-here}(k, x)$ in $O(1)$ amortized time: given a pointer to the node $x$ representing a key $k$, delete $k$ from the data structure.

(g) $\text{split-here}(k, x)$ in $O(1)$ amortized time: given a pointer to the node $x$ representing a key $k$, destructively split the data structure into two, one containing all keys $\leq k$ and the other containing all keys $> k$. (Future operations should depend on the newly split sizes.)

(Each cost can be amortized over all operations, not just split-here operations. You should already be comfortable with amortization from a prerequisite class. If not, we recommend that you talk with the course staff for advice.)

$\text{Hint}$: Start from $(a, b)$-trees.

$\text{Hint}$: In defining a potential function to amortize split-here, think about what changes about the split edges.