Recitation 15 — Leftist Heaps and Practice Problems

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2013)

May 1, 2013

Today's Agenda:
- Leftist Heaps
- Weight-Biased Leftist Heaps
- Practice Problems

1 Heaps

Recall from lecture that a priority queue data structure can be implemented using a heap. A min-heap is a tree satisfying the property that the key stored at every node is less than or equal to the keys of all of its descendants. A max-heap is one where the key stored at every node is greater than or equal to the keys of all of its descendants.

A useful operation of heaps is the *meld* operation, which joins two heaps into one. An implementation of meld traverses the right spine of each tree, so we would like to guarantee that the right spine is not too long. In particular if our two trees are of size n and m we would like to guarantee that the right spines of the two trees are of length $O(\log n)$ and $O(\log m)$, respectively. In this case, melding the two trees would require $O(\log n + \log m)$ work and span.

1.1 Leftist Heaps

A leftist heap is a type of heap which supports efficient meld operations. It has the property that the right spine of any subtree is at most as long as the subtree's left spine. The code from lecture is shown here:

```
datatype $PQ$ = Leaf | Node of (int × key × $PQ$ × $PQ$)
function rank Leaf = 0
  | rank (Node($r$, _, _, _)) = $r$
function makeLeftistNode ($v$, $L$, $R$) =
  if (rank($L$) < rank($R$))
  then Node(1 + rank($L$), $v$, $R$, $L$)
  else Node(1 + rank($R$), $v$, $L$, $R$)
function meld ($A$, $B$) =
  case ($A$, $B$) of
    (_, Leaf) ⇒ $A$
  | (Leaf, _) ⇒ $B$
  | (Node(_, $k_a$, $L_a$, $R_a$), Node(_, $k_b$, $L_b$, $R_b$)) ⇒
      case Key.compare($k_a$, $k_b$) of
        LESS ⇒ makeLeftistNode ($k_a$, $L_a$, meld($R_a$, $B$))
      | _ ⇒ makeLeftistNode ($k_b$, $L_b$, meld($A$, $R_b$))
```

We define the *rank* of a node x as

$$\mathsf{rank}(x) = \#\text{ of nodes on the right spine of the subtree rooted at } x,$$

and more formally:

$$\begin{aligned}\mathsf{rank}(\mathsf{leaf}) &= 0\\ \mathsf{rank}(\mathsf{node}(\_,\_,R)) &= 1 + \mathsf{rank}(R)\end{aligned}$$

Recall that all nodes of a leftist heap have the "leftist property". That is, if $L(x)$ and $R(x)$ are the left and right children of x, then we have:

$$\boxed{\textbf{Leftist Property: } \text{For all node } x \text{ in a leftist heap, } \mathsf{rank}(L(x)) \geq \mathsf{rank}(R(x))}$$

This is why the tree is called leftist: for each node in the heap, the rank of the left child must be at least the rank of the right child. That way, the right spine of a leftist heap is kept short as most of the entries will amass on the left.

Let's review the following lemma from lecture.

Lemma 1.1. *In a leftist heap with n entries, the rank of the root node is at most* $\log_2(n+1)$.

In words, this lemma says *leftist heaps have a short right spine,* about $\log n$ in length.

Before proving Lemma 1.1 we will first prove a claim that relates the number of nodes in a leftist heap to the rank of the heap.

Claim: If a heap has rank r, it contains at least $2^r - 1$ entries.

To prove this claim, let $n(r)$ denote the number of nodes in the smallest leftist heap with rank r. It is not hard to convince ourselves that $n(r)$ is a monotone function; that is, if $r' \geq r$, then $n(r') \geq n(r)$. With that, we'll establish a recurrence for $n(r)$. By definition, a rank-0 heap has 0 nodes. We can establish a recurrence for $n(r)$ as follows: Consider the heap with root node x that has rank r. It must be the case that the right child of x has rank $r-1$, by the definition of rank. Moreover, by the leftist property, the rank of the left child of x must be at least the rank of the right child of x, which in turn means that $\mathsf{rank}(L(x)) \geq \mathsf{rank}(R(x)) = r-1$. As the size of the tree rooted x is $1 + |L(x)| + |R(x)|$, the smallest size this tree can be is

$$\begin{aligned}n(r) &= 1 + n(\mathsf{rank}(L(x))) + n(\mathsf{rank}(R(x)))\\ &\geq 1 + n(r-1) + n(r-1) = 1 + 2\cdot n(r-1).\end{aligned}$$

Unfolding the recurrence, we get $n(r) \geq 2^r - 1$, which proves the claim.

Proof of Lemma 1.1. To prove that the rank of the leftist heap with n nodes is at most $\log(n+1)$, we simply apply the claim: Consider a leftist heap with n nodes and suppose it has rank r. By the claim it must be the case that $n \geq n(r)$, because $n(r)$ is the fewest possible number of nodes in a heap with rank r. But then, by the claim above, we know that $n(r) \geq 2^r - 1$, so

$$n \geq n(r) \geq 2^r - 1 \implies 2^r \leq n+1 \implies r \leq \log_2(n+1).$$

This concludes the proof that the rank of a leftist heap is $r \leq \log_2(n+1)$. $\square$

1.2 Weight-Biased Leftist Heaps

A variant of the leftist heap is the *weight-biased leftist heap*, described by Cho and Sahni (1996). It preserves the invariant that for any node, the number of nodes in its left subtree is greater than or equal to the number of nodes in its right subtree. This was shown to perform better in practice than traditional leftist heaps.

We define the *weight* of a node x as

$$\mathsf{weight}(x) = \#\text{ of nodes in the subtree rooted at } x \text{ (including } x \text{ itself)},$$

and more formally:

$$\begin{aligned} \mathsf{weight}(\mathsf{leaf}) &= 1 \\ \mathsf{weight}(\mathsf{node}(L, \_, R)) &= 1 + \mathsf{weight}(L) + \mathsf{weight}(R) \end{aligned}$$

If $L(x)$ and $R(x)$ are the left and right children of x, then we have:

Weight-Biased Leftist Property: For all node x in a weight-biased leftist heap, $\mathsf{weight}(L(x)) \geq \mathsf{weight}(R(x))$

This can be implemented with slight modifications to the code presented above (do this as an exercise). We will show that the right spine of a weight-biased leftist heap is still short.

Lemma 1.2. *In a weight-biased leftist heap with n entries, the rank of the root node x is at most $\log_2(n+1)$ (equivalently $\log_2(\mathsf{weight}(x)+1)$).*

Proof of Lemma 1.2 (adapted from Cho and Sahni). We will use induction on $\mathsf{weight}(x)$. The base case is a leaf node, which has a weight of 1 and a rank of 0. This satisfies the lemma, as we have $0 \leq \log_2(1+1)$.

Inductive Hypothesis: Now assume that the lemma holds for all $k < n$.

By the Weight-Biased Leftist Property for a heap of size n we have

$$\mathsf{weight}(R(x)) \leq (n-1)/2$$

Applying the inductive hypothesis, we have

$$\mathsf{rank}(R(x)) \leq \log_2((n-1)/2+1)$$

Now

$$\mathsf{rank}(x) = 1 + \mathsf{rank}(R(x)) \leq 1 + \log_2((n-1)/2+1) = 1 + \log_2(n+1) - 1 = \log_2(n+1)$$

□