Recitation 15 – *Leftist Heaps*

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2015)

April 21st, 2015

1 Meldable Priority Queues and Leftist Heaps

Q: What is a meldable priority queue?

A: An ADT for priority queues that supports meld, an operation that combines two priority queues into one.

Q: How are priority queues typically implemented?

A: With heaps: tree-based data structures that only maintain a partial ordering on their keys.

Q: What are the two major properties of a *binary* heap?

A: The *shape property* requires that the tree is a complete binary tree. The *heap property* enforces a partial ordering on the keys: in a *min*-heap, the key at each node must be less than both of its descendants. In a *max*-heap, it must be greater.

Q: What are the two major properties of a *leftist* heap?

A: The *heap property* is the same as before. The *leftist property* requires that for every node x with children L and R, $\text{rank}(L) \geq \text{rank}(R)$. We define $\text{rank}(x)$ as the number of nodes in the right spine of x.

Q: Why do leftist heaps improve the cost of meld?

A: Lemma 20.3 from lecture states, "In a leftist heap with n entries, the rank of the root node is at most $\log_2(n+1)$."

We know $\text{meld}(A, B)$ only traverses the right spine of both A and B, so it follows that the work of meld is bounded by $O(\log|A| + \log|B|)$.

The code for meld on *min*-heaps from lecture is given below.

```
datatype PQ = Leaf | Node of (int × key × PQ × PQ)

fun rank Leaf = 0
  | rank (Node(r, _, _, _)) = r

fun 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)

fun 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))
```

Consider the following code.

```
fun singleton v = Node(1, v, Leaf, Leaf)
val Q = Seq.reduce meld Leaf (Seq.map singleton S)
```

Suppose $S = \langle 3,5,2,1,4,6,7,8 \rangle$. Draw the tree structure of Q.

Write and solve work and span recurrences for the given code in terms of $|S| = n$.

2 Maintaining the Median

Suppose you want to construct a data structure M representing a set of integers which supports fast median queries. Specifically, we want the following:

	Work	Span
`fromSeq(`S`)`	$O(\|S\|)$	$O(\log^2 \|S\|)$
`median(`M`)`	$O(1)$	$O(1)$
`insert(`M`,`k`)`	$O(\log \|M\|)$	$O(\log \|M\|)$

Describe how to implement this structure.

3 Bonus Which Has Nothing To Do With PQs

You have just been asked to design the latest air-traffic-control system. One constraint is that planes must stay at least 1 km away from one another. Describe an algorithm that, given the (x, y, z) coordinates of n planes, determines if any are within 1 km of each other. It must run in $O(n)$ work and $O(\log^2 n)$ span. Obscure hints: $\sqrt{3} = 1.71$, and $5^3 = 125$.