Advanced Algorithms, Fall 2011

Prof. Bernard Moret

Amortized Analysis of Skew Heaps

A skew heap is, in effect, a somewhat crippled leftist tree. By removing the information about the length of the rightmost path from each node, we make it impossible to maintain the leftist property; but by enforcing a swap of left and right children at every node along the merge path, we create a melding operation that mimics that used in leftist trees well enough to ensure a logarithmic amortized running time.

We begin by defining a few terms. The *weight* of some node x is the total number of nodes in the subtree rooted at node x (including the root). A *heavy* node is a node whose weight is greater than half of its parent's weight. We will call the sibling of a heavy node a *light* node. In the exceptional case where the two children have equal weight (and thus neither child is heavy), we will call both children light—the rationale for assimilating such nodes to the siblings of heavy nodes will become apparent in the next assertion.

We define the potential of a skew heap to be twice the number of heavy nodes that are also right children. This is in line with the idea that a skew heap is a kind of "approximation" to a leftist tree: the "bad" nodes, which contribute to the potential, are nodes that violate the leftist property, that is, nodes that cause a lean to the right.

Consider some node x and some descendant node y. We claim that the number of light nodes on the path from x to y is at most $\log_2 \frac{w(x)}{w(y)}$. Why is that? A light node has, by definition, less than half of the weight of its parent—that is immediate for the sibling of a heavy node, but for two children of equal weight, each has weight $(w(p)-1)/2$, where $w(p)$ is the weight of the parent, and thus it is also true. (This is why we assimilated such matched nodes to the siblings of heavy nodes.) Thus, every time the path from x to y goes through a light node, the number of remaining nodes is cut in half; but we started with $w(x)$ nodes; so after k light nodes, we are down to at most $2^{-k}w(x)$ nodes and yet y has weight $w(y)$; setting $w(y) \leq 2^{-k}w(x)$ gives us the desired relationship.

In particular, if T_1 has n_1 nodes and T_2 has n_2 nodes, then the rightmost path of T_1 has at most $\log_2 n_1$ light nodes and the rightmost path of T_2 has at most $\log_2 n_2$ light nodes. These two paths are all that matter in melding T_1 and T_2. Now,, consider what happens in the melding: we merge the two rightmost paths, resulting in some new rightmost path of unknown length k. Of these k nodes on the merged path (before swapping children), at most $\log n_1 + \log n_2$ are light nodes—the number cannot have increased as a result of the merging of the two paths, but it could have decreased, since the merge path is now longer and thus many nodes on it have had their weight increased, some to the point where they have become heavy. Now, we go back and swap the left and right pointers at every node on this merged path. The result is that every node that was light is exchanged with its sibling; this sibling was a left child and so did not count in the potential, but the swap makes it a right child and most of these siblings are heavy (an exceptional few might have the same weight as their sibling), so that these nodes could contribute an increase in the potential of at most $2(\log n_1 + \log n_2)$. The other nodes on the merge path are all heavy, and there are at least $k - \log n_1 - \log n_2$ of them; they contributed to the potential before the swap, as they were right children, but the swap turns them into left children and so their contribution is now removed; thus at least $2(k - \log n_1 - \log n - 2)$ is removed from the potential. Overall, the potential decreases by at least $2(k - 2\log n_1 - 2\log n_2)$. The real cost of the merge operation was $2k$ (go down the merge path, then come back up), so the sum of the real cost and the change in potential is at most $2k - 2(k - 2\log n_1 - 2\log n_2) = 4(\log n_1 + \log n_2)$. In other words, we can set the amortized cost to 4 times the log of the size of the input and obey our master inequality $a(n) \leq f(n) + \Delta\Phi$.

Note that, in fact, there is no need to traverse the merge path and swap on the way back up: we can do the swap on the way down, cutting the real cost down to k, use a potential function that counts each heavy right child just once instead of twice, and set the amortized cost per operation to twice the log of the size of the input.

Insert and Deletemin are just Meld operations preceded by some constant work and thus also take logarithmic amortized time. We just need to verify that neither does weird things to the potential. In the case of Insert, creating a new skew heap of one node does not alter the potential, since a tree of one node has no heavy right child. In the case of Deletemin, removing the root does not alter the potential, since the root cannot be heavy (no parent!), but in case the right child of the root had been heavy (and thus, as a right child, contributed 1 to the potential), the potential goes down by one, as this right child becomes a root of its own. The decrease by one does not change the overall analysis.

Finally, note that a skew heap could degenerate into a single right-leaning caterpillar, in which case every node other than the root and the leaves is a heavy right child—thus the potential of a tree of weight n (n odd) could be as large as $(n-3)/2$. But also note that, at the next Meld, everyone of these heavy right nodes will become a left child and thus no longer contribute to the potential—again, we see that processing a "worst-case" structure (the right-leaning caterpillar) takes a lot of real work, but causes a very large drop in the potential, compensating ("paying") for the work.