Short Notes: Amortized Analysis of Fibonacci Heap Using Accounting Method

Palash Dey
Indian Institute of Technology, Kharagpur
palash.dey@cse.iitkgp.ac.in

In this note, we will prove that the amortized time complexities of insertion, extract-min, and decrease key operations on an n-node Fibonacci heap are respectively $\mathcal{O}(1), \mathcal{O}(\log n), \mathcal{O}(1)$. We will use accounting method for proving this. We will maintain the following invariant throughout the run of the algorithm.

Invariant: Every node in the root list has \$1 and every marked node has \$2 stored with it.

Consider a sequence of n operations where every operation is either insert or extract-min or decrease key operation. We will prove this claim using induction on n. We will use the fact that the maximum degree of any node in an n-node Fibonacci heap is at most $\log_\varphi n$ where $\varphi = \frac{\sqrt{5}-1}{2}$.

For $n = 0$, the Fibonacci heap is empty and thus the invariant holds. So the induction base case holds.

Let us assume that the invariant holds for every sequence of $n - 1$ operations.

Suppose the n-th operation is an insertion operation. We charge \$2 for the operation. The actual cost of insertion is $\mathcal{O}(1)$ which we pay using \$1 and store the remaining \$1 with the newly inserted element. Hence, the loop invariant holds after the insertion operation also.

Suppose the n-th operation is an extract-min operation. Suppose h_1 and h_2 are respectively the number of nodes in the root list before and after the n-th operation. The actual cost of the operation is $\mathcal{O}(h_1)$. We pay this $\mathcal{O}(h_1)$ actual cost with the $\$h_1$ stored with the h_1 nodes in the root list. We charge $\$\lceil 1 + \log_\varphi n \rceil$ for the extract-min operation which we store, \$1 in every node in the root list. This is enough since $(1 + h_2)$ is at most the maximum degree plus one, that is $\$\lceil 1 + \log_\varphi n \rceil$. Hence, the loop invariant holds after the extract-min operation also.

Suppose the n-th operation is a decrease key operation. We charge \$4 for every decrease key operation. The decrease key operation causes at most one cut operation and one or more cascading cut operations. Suppose the decrease key operation causes k cascading cut operations. Since every node cut through cascading cut operation was marked before the n-th operation, they had \$2 stored with them. Cutting every such node and making it part of root list and updating H.min is needed takes $\mathcal{O}(1)$ actual cost which we pay by spending \$1 from the \$2 stored with the node being cut in the cascading cut. The remaining \$1 is stored with that node which is required to satisfy

Statutory warning: This is a draft version and may contain errors. If you find any error, please send an email to the author.

the invariant. The actual cost of decreasing the key, cutting it if needed, and updating H.min if needed incurs $\mathcal{O}(1)$ actual cost which we pay by spending \$1 from the \$4 that we have charged the decrease key operation. Note that we still have \$3 left with us. If the node on which decrease key has been performed, becomes part of the root list, then we spend \$1 from the remaining \$3 to store with it. Note that, we still has at least \$2 remaining with us. Notice that, at most one unmarked node can be marked in the decrease key operation. If that happens, then we store the remaining \$2 with that node which was unmarked before the n-th and got marked during the n-th operation. Hence, the invariant continue to hold after the n-th operation.

Hence, the actual cost of any sequence of n operations is fully paid off by the total amount that we have charged. This proves that the amortized time complexities of insertion, extract-min, and decrease key operations on an n-node Fibonacci heap are respectively $\mathcal{O}(1)$, $\mathcal{O}(\log n)$, $\mathcal{O}(1)$.