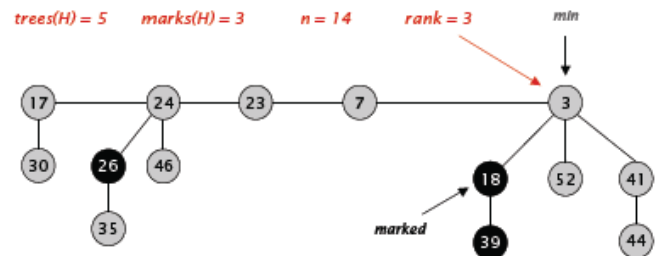


- Set of Heap ordered trees (each parent smaller than children).
- Maintain pointer to minimum element (FIND-MIN takes $O(1)$ time).
- Set of marked nodes (if one of its children has been removed).
- n - number of nodes in the heap.
- $\text{RANK}(x)$ - number of children of node x .
- $\text{RANK}(H)$ - Max rank of any node in heap H .
- $\text{TREES}(H)$ - Number of trees in heap H .
- $\text{MARKS}(H)$ - number of marked nodes in H .



CS 430 Lecture 22 Activities

Fibonacci Heaps

Fibonacci heaps which support heap operations that do not delete elements in constant amortized time. From a theoretical standpoint, Fibonacci heaps are especially desirable when the number of EXTRACT-MIN and DELETE operations is small relative to the number of other operations performed. This situation arises in many graph algorithms.

In essence, a Fibonacci heap is a “lazy” binomial heap in which the necessary housekeeping is delayed until the last possible moment: deletion.

1. See <https://www.cs.usfca.edu/~galles/JavascriptVisual/FibonacciHeap.html> and <https://www.cs.princeton.edu/~wayne/cs423/fibonacci/FibonacciHeapAnimation.html> to help describe how each operation is done, and a rough estimate on its run time:

Make-Heap : $O(1)$ Makes a single node heap.

Insert : $O(1)$ Make single node heap, put in the (bidirectional linked) list to left of current min. Update min pointer if necessary.

Minimum : $O(1)$ We maintain a pointer to the min.

Union : $O(1)$ Link the two root lists together¹, and then update the min pointer to point to the smaller of the two previous mins.

¹You don't need to walk either to do that.

Operation	Binary Heap	Binomial Heap	Fibonacci Heap	Note that the times indicated for the Fibonacci heap are amortized times while the times for binary and binomial heaps are worst-case per-operation times.
MAKE-HEAP	$\theta(1)$	$\theta(1)$	$\theta(1)$	
INSERT	$\theta(\log n)$	$\theta(\log n)$	$\theta(1)$	
MINIMUM	$\theta(1)$	$\theta(\log n)$	$\theta(1)$	
EXTRACT-MIN	$\theta(\log n)$	$\theta(\log n)$	$\theta(\log n)$	
UNION	$\theta(n)$	$\theta(\log n)$	$\theta(1)$	
DECREASE-KEY	$\theta(\log n)$	$\theta(\log n)$	$\theta(1)$	
DELETE	$\theta(\log n)$	$\theta(\log n)$	$\theta(\log n)$	

Extract-Min : Find min is $O(\log n)$ ² because we have a pointer to the min. Remove min from root list (saving the value to return). If the min had children, put those children in the root list. CONSOLIDATE the root list³

Decrease-Key : $O(1)$ Reduce value at that node, move it and its subtree to root list, check if new min value pointer. Breaking that binomial heap by removing a subtree⁴. Wait for a marked node to lose a 2nd child, move to root list, don't CONSOLIDATE yet.

Delete :

²The name Fibnoacci comes from the fact that the base in the $\log n$ is the golden ratio. The maximum degree of any node in the root list is also related to Fibonacci.

³Use array of rank on last node

⁴Do not CONSOLIDATE yet.