The Hollow Heap Data Structure: An Introduction, Run-Time Analysis, and Comparison to the Fibonacci Heap

Alisha Sprinkle and Courtney Dixon

December 8, 2019

1 Abstract

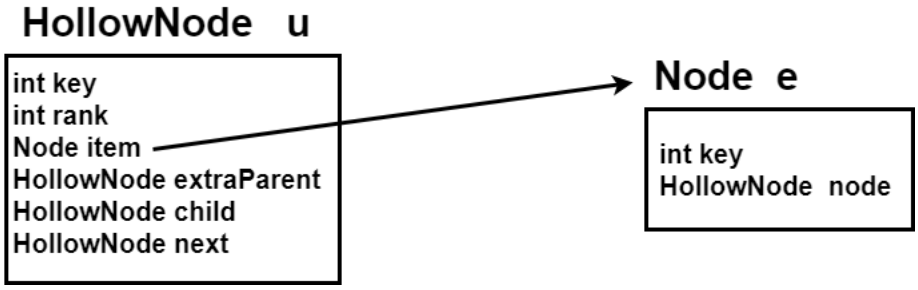
Hollow Heaps were introduced by Hansen, Kaplan, Tarjan, and Zwick in 2015. This data structure performs operations with the same amortized efficiency as the Fibonacci Heap introduced by Tarjan and Fredman. Hollow Heaps perform all operations characteristic of a heap, however, Hollow Heaps use lazy deletion and a directed acyclic graph rather than trees. The goal of this paper is to explain the Hollow Heap data structure precisely, analyze the amortized running time of the data structure, implement the data structure in Java, and compare the efficiency of Hollow Heaps to the renown Fibonacci Heap data structure.

2 Hollow Heap: the data structure

"Clearly explain the data structure"

A Hollow Heap is a heap-ordered data structure. Therefore, a Hollow Heap maintains the heap property; the key of Node a is less than or equal to the key of Node b for every directed arc(a,b) in the structure where Node a is the parent of Node b. The typical properties of nodes still hold. A node that does not have a parent is a root and a node that does not have any children is a leaf. The root has the minimum key in the Hollow Heap.

Hollow Heaps are just as efficient as Fibonacci Heaps. All the operations of a heap, excluding two, take $O(1)$ time in the worst case and amortized with Hollow Heaps. The two heap operations that take longer than constant time are *delete* and *deleteMin*, which each take $O(\log n)$ amortized time with Hollow Heaps. The *decreaseKey* operation in Hollow Heaps uses lazy deletion and reinsertion. Other heaps use a tree or set of trees, but Hollow Heaps use a directed acyclic graph (dag) instead. A Hollow Heap is made up of nodes that hold items and not the typical nodes that are items. The node structure is depicted in the image below.



The authors presented three variants of the Hollow Heap data structure: Multi-root, One-root, and Two-parent. The focus in this paper is the Two-Parent Hollow Heap.

2.1 Methods

- makeHeap()
- findMin(HollowHeap h)

- insert(Node e, int key, HollowHeap h)
- deleteMin(HollowHeap h)
- meld(HollowNode g, HollowHeap h)
- decreaseKey(HollowNode e, int key, HollowHeap h)
- delete(HollowNode e, HollowHeap h)
- makeNode(wNode e, int key)
- link(HollowNode v, HollowNode w)
- addChild(HollowNode v, HollowNode w)
- doRankedLinks(HollowNode u)
- doUnrankedLinks()

3 Amortized Analysis via the Accounting Method

"Amortized Analysis via Accounting Method"

Op #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
makeHeap																				
copy																				
total																				
cumulative																				
findMin																				
copy																				
total																				
cumulative																				
insert																				
copy																				
total																				
cumulative																				
deleteMin																				
copy																				
total																				
cumulative																				
meld																				
copy																				
total																				
cumulative																				
decreaseKey																				
copy																				
total																				
cumulative																				
delete																				
copy																				
total																				
cumulative																				

4 Hollow Heap Implementation

The authors presented three variants of the Hollow Heap data structure: Multi-root, One-root, and Two-parent. Our implementation is of Two-Parent Hollow Heaps. Our programs are written in the Java programming language. The UML diagram below is for the Hollow Heap class. All other UML diagrams can be found in the figures section of this paper.

5 Hollow versus Fibonacci: A Comparison

"Compare the two data structures in terms of practical efficiency"

5.1 Similarities

1. heap operations in $O(1)$ and $O(\log n)$
2. simple to implement
3.

5.2 Dissimilarities

1. Tree(s) versus Dag
2. no cascading cuts
3.

6 Figures

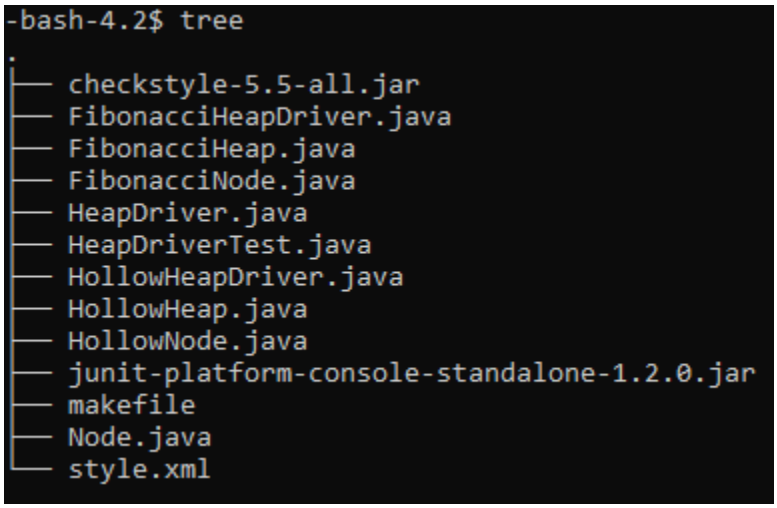


Figure : Directory Tree

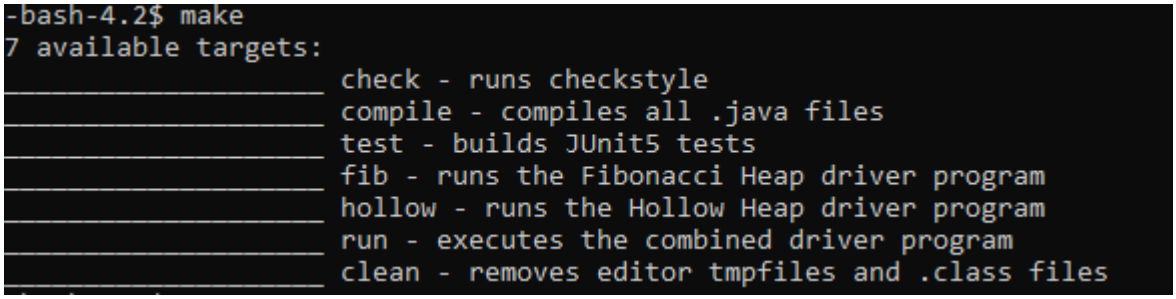


Figure : Makefile Targets