CMPUT403-README

Alex Dong

April 2020

1 Brief Description

Hollow heaps are heaps more easily implementable than their twin Fibonacci Heaps. Both the time complexities of the Fibonacci and Hollow heaps are O(1) for inserting new items. The time complexity for the binary heap (heapq in Python for example), is that of O(log n) for pushing/popping. The deletion of an item is O(log n) amortized for the Hollow Heap (and Fibonacci Heap). The time complexities are based on the optimizations via ranked linkages and hollow nodes, Melding (the merge of two 'Hollow Heaps') creates a faster implementation that the heapify counterpart - think of it similar to union-find where we just take their roots, and append them to each other based on the priority, as the property of minimum heap is retained.

2 Run-time

As mentioned above in the description, the distinctive runtime between the regular binary heap and the hollow heap is the insertion of new items. The time complexity for the binary heap is O(log n) while the hollow heap is O(1). This is due to the melding and linking within Hollow Heaps. Assigning and linking the roots of heaps is the optimization created as the minimum-heap property is retained. The deletion and getMinimum() functionality are about the same as the binary heap (Amortized O(logn) deletion, O(1) lookup).

3 Resources

(Main Information Source - Two-parent heap psuedo and info) - https://arxiv.org/pdf/1510.06535.pdf

(Python heapq module in C) - https://stackoverflow.com/questions/29244014/python-heapq-python-and-c-implementation-which-one-is-used

4 Instructions

Standard Python libraries used. The only other module required is NumPy. The folder should hold the main implementation file and the test file named respectively. To run the tests in Python3:

python3 HollowHeapTest.py

We can simply import the Hollow Heap structure otherwise from importing it normally:

from HollowHeap import HollowHeap, Item

Note: import item if you would like to utilize the decreaseKey property, we require that the node is known prior to decreasing the key. The locations of these nodes can be stored within some dictionary for O(1) lookup.

5 Assumptions

A few assumptions were made for this situation:

- The values tested were only integers, the keys are the integer values themselves for simplicity sake. However, since the key is used as the priority, we can implement using customized data structures as the nodes in the heaps are only containers (hence the hollow property)
- There is no customized key comparison used. It is simply just a check for the minimum key
- This implementation is the two-parent hollow heap, there are multiple ideas of hollow heaps such as one parent heaps but they were not created
- In theory, the Hollow Heap should be the faster implementation in inserting values into some sort of priority queue. However, the benchmarks between the binary heap and the Hollow Heaps are very skewed off. This could be most likely from the overhead of instantiating and item class to hold the nodes in the heap. Also, we are comparing the heapq module, which is implemented in C

6 Summary of Files

There is only two files sent over. The main implementation and methods are coded up in HollowHeap.py. We import the HollowHeap and Item classes from this file. The test file is HollowHeapTest.py, with some simple benchmarking and unit tests for the methods used.

7 Interpreting Output

Inserting a new value into the Hollow Heap returns you the actual reference to the item being inserted into the heap. We can store it into a references dictionary to decrease the key. We can reference the key and value of the item via .val and .key. Refer to the HollowHeap.py file for all the properties.