Hollow Heaps

R. Frederic Sauve-Hoover

April 20, 2020

Description

I implement two-parent hollow heaps as described in [Hansen et al., 2015] as a python package. Hollow heaps are a heap implementation with improved complexity compared to more typical priority queues (heapq in python) on par with fibonacci heaps. However, hollow heaps are significantly simpler to implement than fibonacci heaps.

The goal of this is to be able to provide an importable python library containing an implementation of a fast heap and to get some idea of how well it compares to existing python libraries.

Time Bounds

Using the two-parent hollow heap implementation, we have all usual operations: *insert*, *find_min*, *meld*, etc. taking $O(1)$ time worst-case, except for *delete* which takes $O(1)$ per hollow node losing a parent and per link being added, as well as $O(logN)$, where N is how many nodes we have in our dag prior to the *delete* or *delete_min* operation being done (*delete* and *delete_min* are trivially equivalent in time bounds since *find_min* is $O(1)$). The more exact proofs for the time bounds are in the paper and go into significantly more detail than I am in this brief overview.

For operations such as *find_min* we can trivially see that they're $O(1)$ since they're simply looking acessing attributes of our root. For *insert* and *meld* it's less obvious, but we're simply linking the new node to our root, which is clearly $O(1)$, with the ranking and re-ordering being left until the *delete* operation.

Resources

I follow the algorithm provided in [Hansen et al., 2015], which provides much of the basis for my code.

Instructions

To use the package, import **hollow_heap** as shown in the snippet below

```
# Element is used if you want to store additional data alongside the
# key
from hollow_heap import HollowHeap, Element

heap = HollowHeap()
e = Element([1,2,3]) # a simple element containing a list
heap.insert(1, e)

# an insertion that doesn't use an element object
heap.insert(5)
```

There aren't any required libraries to install to use **hollow_heap**, however to run tests and benchmarks, run *pip install -r requirements.txt*

To run the tests, run *pytest tests.py*

To run the benchmark, run *python benchmark.py*

Assumptions

There are a few assumptions, mostly to do with the usage of the library. These are things I may improve some other time.

- inserts must be keys that are not already present in the heap
- meld will only be called on heaps with distinct items
- decrease_key is only called on items with a key ¿ k
- delete is only called on an item that exists in the heap

Files in directory

- **hollow_heap.py** The hollow heap library, import the usual python way
- **tests.py** unit tests for the hollow heap library, run with *pytest tests.py*
- **benchmark.py** The benchmarking code, times execution of various sorts comparing heapq, hollow heaps, and builtin sort. Interestingly enough I wasn't even close to the speeds given by heapq and sort (sort isn't surprising at all, it's super optimized), and this is likely in no small part due to implementations of standard python libraries being mostly underlying c code, as well as an implementation bug present in my hollow heaps that I haven't quite narrowed down yet.

Output of program

References

[Hansen et al., 2015] Hansen, T. D., Kaplan, H., Tarjan, R. E., and Zwick, U. (2015). Hollow heaps.