COMP4040 - Project 3 – Priority Queues

Xavier Carpent & Ian Knight

November 25, 2021

Heaps and Beaps

You are already familiar with at least two priority queue implementations: the binary heap, and the Fibonacci heap. There are also other fun constructions, that try to improve some aspect of their complexities or add support for other functionalities. One such variant is the *Beap*, short for *bi-parental heap*... Behold!

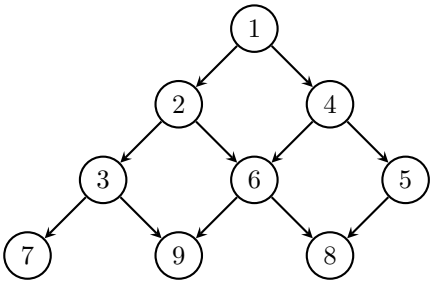

Figure 1: A min-"beap".

The beap operates very similarly to a binary heap for `insert` and `extract`. It has the same heap property, and the same mechanisms for ensuring it. Just like the binary heap, the beap can be implemented *implicitly* for fast access and low memory overhead.

The main structural difference is that each node in a beap has *two* parents. As such, a natural representation of the beap is that of a *grid*. The number of nodes at each level increases *linearly* with the depth (as opposed to exponentially in a typical tree). The height of a beap of n nodes is bounded by about $\sqrt{2n}$, which indicates that the `insert` and `extract` operations are $O(\sqrt{n})$. While significantly better than $O(n)$, this is quite a bit worse than the logarithmic cost of these operations in binary heaps.

The main upside however is that the `max` and `search` operations are also $O(\sqrt{n})$ in beaps, while they are $O(n)$ in binary heaps, making the former a bit more flexible, and more efficient for applications than occasionally need these operations as well.

More information can be found in the original 1979 paper [1].

3-eap[1]

A somewhat natural extension of beaps is a structure in which each node has 3 parents and 3 children. The nodes can then fit on a 3-dimensional grid (limited to a corner of a cube). What can then be said about the complexities of operations on it? Are there situations where this generalization makes sense?

This idea can of course further be extended to a k-eap for $k > 3$, representing the heap as a corner of a k-hypercube...

Note: this construction is different *than the d-ary heap (a generalization of the binary heap where each node has d children) and the K-D heap (a binary heap for multi-dimensional data). There are just (uhm)* heaps *of variants...*

[1] Unfortunately the name "treap" is already taken, by another (related but disconnected) data structure...

Goal of the project

The goal of the project is threefold:

1. Implement the binary heap and the beap in their min- form. You may of course use the implementation you may have started in lab 7. It is quite likely (and expected) that the implementations of binary heap and beap will be similar, given the parallel between the two data structures. You may use an implicit **or** explicit representation, but the former will be worth (a small amount of) extra points. For the beap in implicit representation, you may find it useful to use the *Cantor pairing function* to map coordinates of nodes to index in an array. The necessary operations and expected complexities are:

operation	binary heap	beap
`min`	$O(1)$	$O(1)$
`max`	$O(n)$	$O(\sqrt{n})$
`insert`	$O(\log n)$	$O(\sqrt{n})$
`extract`	$O(\log n)$	$O(\sqrt{n})$
`search`	$O(n)$	$O(\sqrt{n})$

2. Test your implementation *extensively*. This objective is deliberately vague, but you should now have enough tools and insight to put in place a series of experiments and conclude decisively on their results (in the context of algorithms and data structures).

3. Analysis of 3-eaps.

Deliverable

You may use the programming language of your choice among Haskell, Python, Java, and C/C++. You *cannot* use a library for directly implementing either of the data structures in this project. **Given the situation with the first project, we will also be extremely attentive towards plagiarism (either between students or from other sources) when grading, and will escalate appropriately if it comes to it.**

Along with your code, you must *write a report* containing:

1. A brief description about your implementation, with anything of particular note;

2. A summary of the method and results of your testing (this should be the focal point);

3. *(optional, for extra points)* A brief analysis of 3-eaps, the complexity of the main operations for them, and a discussion about their potential usefulness (this may include an implementation of 3-eaps, but this is not expected).

The report must contain the equivalent of a *maximum of two pages* of text (this maximum is a maximum, not a target). This can be supplemented by any (**reasonable**) number of pages of figures, plots, diagrams, etc. as appropriate. Submit your code and your report in a zip file.

References

[1] J Ian Munro and Hendra Suwanda. Implicit data structures for fast search and update. *Journal of Computer and System Sciences*, 21(2):236–250, 1980.