COMP4019 - Lab Session 7 – Binary Heaps

Xavier Carpent & Ian Knight

November 18, 2021

1 Implicit Data Structures

Consider the representation of a *complete m-ary tree* (that is, a tree where each node has k children, and such that all levels are filled, except possibly the last one, which is flushed to the left). See Figure 1 for an example. We have discussed how this type of tree can be represented *implicitly* in an array. For this, we need to map each node in the tree to an index in the array.

1. For a tree of a given branching factor m and a maximum depth d, what size should the array be initialized to?
2. Given a list $(x_1, \ldots x_k)$, where $0 \leq x_i < m$ and $k \leq d$ representing a path to a node in the tree, find the corresponding index in the array;
3. What about the inverse operation?

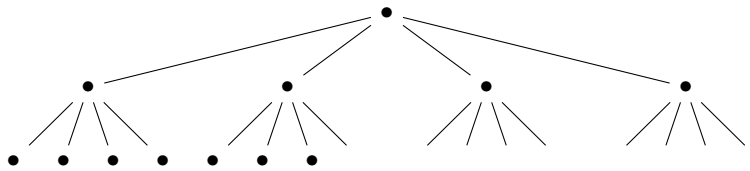

Figure 1: A complete 4-ary tree.

2 Binary Heaps

Implement a binary heap in your favorite programming language (most specifically the operations `insert`, `minimum`, and `extract`). Depending on the programming paradigm you choose, you may need to use a different form of balancing (a *complete* binary tree is natural for implicit array-based implementations, whereas "left-right height difference cap" is more natural for functional implementations).

1. What do you need to change to transform your *min-heap* implementation (as seen in lecture) to a *max-heap* one?
2. Empirically measure the efficiency of your implementation, like in Lab 4.