Outline of an implementation of a priority queue

Tor Nordam

A simple way to implement a priority queue is to use a data structure known as a binary heap. A binary heap is built up as a tree, with one node (the root node) at the top, and each node having at most two child nodes. The elements that should be stored in the queue are kept as nodes in the heap, and they are inserted in such a way that a node is always smaller than or equal to both of its children.

A heap can be stored in a normal array, as shown in Fig. 1. You need only an array that is at least as long as the number of elements to store, and a variable to remember the number of elements. To traverse the array, note that in a array with 1-based indexing, you can always calculate the index of the parent by dividing by 2 (and rounding down to the nearest integer), and you can calculate the index of the two child nodes by multiplying by 2 for the left child, and multiplying by 2 and adding 1 for the right child.

To implement the `enqueue` method, do approximately the following (here `array` is the array where you store the queue, and `N` is the current number of elements in the queue:

- If `N+1 > length(array)`, increase the size of the array (for example double it),
- Insert the new element in position `N+1`, and set `N = N + 1`,
- If the new element is smaller than its parent node, exchange their positions. Repeat until the element is larger than or equal to its parent.

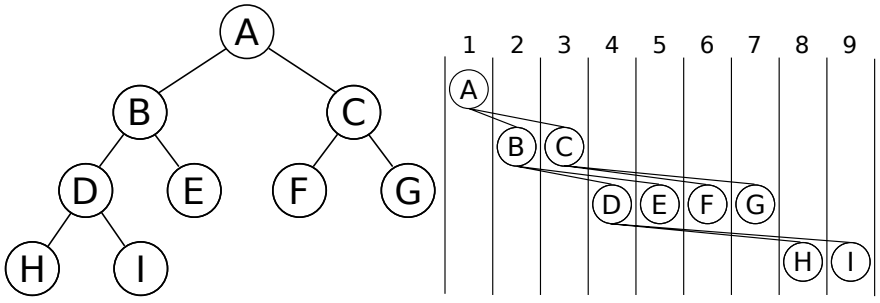

Figure 1: Left: binary heap. Right: the same heap, stored in an array.

To implement the `pop` method, do something like this:

- Make a copy of the element at the root node (first element in the array, this is the smallest one, so this is the element you want),
- Insert the element in position `N` at the root node (overwriting the previous value, which you copied),
- Set `N = N - 1`
- If the element at the root node is larger than any of its two children, exchange its position with the smallest of the two children. Repeat this procedure until the element is smaller than or equal to both it's children.