Irene Avezzu' - 20142 - IAvezzu@unibz.it

Worked together with: Massimo Marcon, Ivana Nworah Bortot

Explain the ideas behind your algorithm. In particular, explain how you design the methods and which auxiliary methods you use.

1) isEmpty: simply if the root is null then it means that the queue is empty

2) insert: to insert a new node it must respect the size-balance and the heap property. First of all, if checking if the queue is empty, in this case, it inserts the new node as root, otherwise, it uses the recursive method insert.

In the recursive method, it starts checking whether the left (and then the right) subtree is empty, if it is it adds the new node in the empty subtree and fixes the node counter (*refreshCount*) and checks that it maintains the status of the heap (*heapifyUpward*). If both subtrees are not empty it recalls the method on the smaller subtree (current.right/left).

refreshCount is a support method to my recursive insert method, it starts from the node passed as parameter and proceeds upward until it exits the tree, checks whether the node is a left/right child and increases the left/right counter.

heapifyUpward is a support method to my recursive insert method, it starts from the node passed as parameter and proceeds upward until the root checks that each child is either null or smaller than the parent

3) extractMax: the max node is always in the root position so that is the value we need to extract but by removing this node we need to reorganize the whole queue to make sure all its properties remain true. it stores the max node's value into a variable and set as the key of the root the min node's value (*extractLeaf*) and then it proceeds to make sure the heap property is respected, to do so it uses a support method (*heapifyDownward*).

extractLeaf is a support method for my extractMax procedure.

It proceeds downward until a leaf (node with no child node) choosing always the subtree with the least nodes. Once it reaches a leaf it checks whether the current node is the root (current.parent = null) or not. If it's just necessary to reset the root of the queue otherwise check if the current node is a left or right child and set the parent child accordingly, it also refreshes the count of nodes of the parent (same support method as ex 2.2).

heapifyDownward is a support method for my extractMax procedure. It starts from the node passed as parameter until it exits the tree (current=null). It looks if there is a child with a key greater than the one of the current node (if both children are greater it picks the greatest one) and set it as the nextNode. If no such node is found then it means that the subtree of the current node is a heap, otherwise, it proceeds by swapping (*swap*) the two nodes and continuing the process of heapify.

swap is a support method for my extractMax procedure, specifically in the heapifyDownward method. If one of the two passed nodes is the root then the swap can be simplified. It switches the whole node and not the single attribute.

replace is a support method for my extractMax procedure, specifically in the swap method. Move the data from the fromNode into the toNode. It replaces the attribute instead of the pointer to the node.