Double-Ended Priority Queues

The abstract datatype double-ended priority queue is a priority queue, which, in addition to initialization, *insert*, *findmin*, and *deletemin*, also supports the operations *findmax* and *deletemax*.

Here we consider a concrete data structure, called a symmetric min/max heap (SMM heap), which implements this abstract datatype.

First some notation. If x is a node of a binary tree, then $x.k$, $x.l$, $x.r$, and $x.p$ refer to the node's key, left child, right child, and parent, respectively; if they exist. T_x denotes the subtree with x as root and $y \in T_x$ denotes that y belongs to the subtree T_x.

A symmetric min/max heap is defined via the following two contraints:

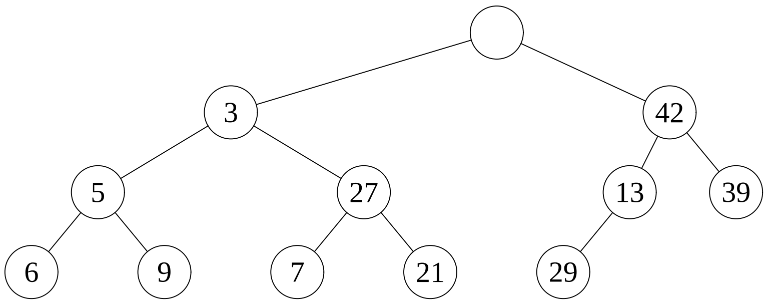
Structural Invariant

> The structure is a binary tree where the root is empty and all other nodes are filled in layer by layer from left to right (as in a standard heap).

Ordering Invariant

> For all nodes x, we require that
>
> $x.l.k = \min\{y.k \mid y \in T_x \wedge y \neq x\}$ and $x.r.k = \max\{y.k \mid y \in T_x \wedge y \neq x\}$.

The illustration below shows an example SMM heap:



Question a: How can *findmin* and *findmax* be implemented? □

The structural invariant is easy to maintain during updates (exactly as it is the case for the standard heap). With regards to the ordering invariant, it would be more convenient with a formulation in terms of local constraints. We define the following properties that a node x can have:

$$L(x):\ x.k \geq x.p.p.l.k \qquad R(x):\ x.k \leq x.p.p.r.k$$

We also define that a node *has* the property if one of the involved fields is undefined, e.g., a child of the root has the properties even though (because) $x.p.p$ does not exist. This convention is used below.

Question b: Argue that $L(x)$ and $R(x)$ hold for all nodes x in an SMM heap. □

Now, we consider the opposite direction, i.e., the establishment of the SMM heap properties through the local constraints.

Question c: Show by induction in the height that if for all nodes u in a tree fulfilling the structural invariant we have

$$P(u):\ L(u) \wedge R(u) \wedge u.l.k \leq u.r.k,$$

then the tree is an SMM heap. □

The operation *insert* can be implemented as shown below. Root(x) is **true**, if and only if x is the root. Swap(x, y) switches the keys of x and y and returns its second argument.

```
Insert the key in a new node $x$,
added to the structure according to the structural invariant
if “$x$ is a right child” and $x.k < x.p.l.k$:
    $x$ = Swap($x$, $x.p.l$)
while not $P(x)$:
    if not $L(x)$:
        $x$ = Swap($x$, $x.p.p.l$)
    elif not $R(x)$:
        $x$ = Swap($x$, $x.p.p.r$)
```

Question d: Argue that if an insertion is made into an SMM heap via the algorithm above, then the resulting structure will again be an SMM heap. □

Question e: Write an algorithm in the style of the one above, but now for the operationen *deletemin*. □