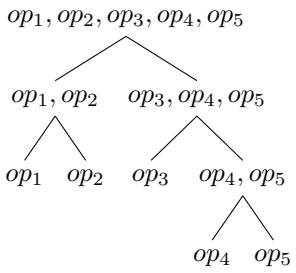
Introduction

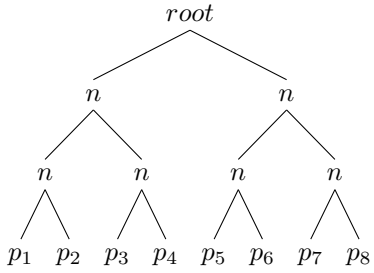
Queue (Q): We are going to implement a MEMD Queue using T.

CAT Tree (CAT): A tree for storing some operation on Q by order. Every leaf In each node there is a tuple stores (#enqueues, #dequeues, #null response dequeues) of the node's subtree.

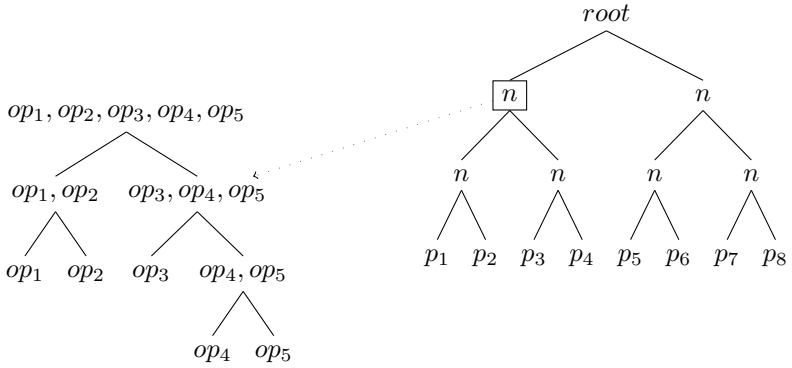


In order to compute deq_i faster we augment the tree with these parameters. At each node we store

Tournament Tree (T): Each process is assigned to one of the Tournament Tree leaves. Every node contains a pointer to a node of the CAT Tree.



Each node in the Tournament Tree points to a node in the CAT tree.

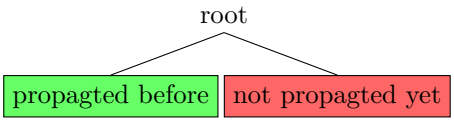


Algorithm

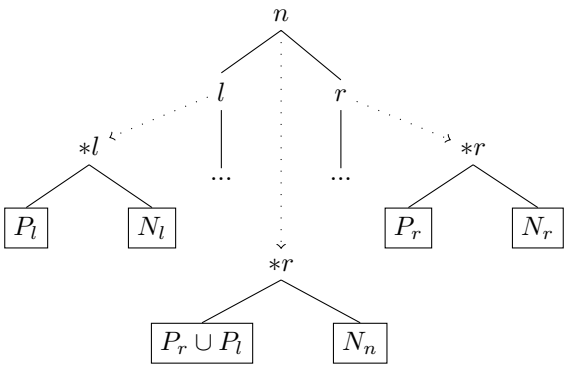
Our idea is to replace the strings in our universal construction with trees in order to make DO() operations faster. Since for *dequeue* queries you only have to know that earliest element added (if it exists). We told earlier the ordering of operations in the CAT tree before.

Merging children of node n

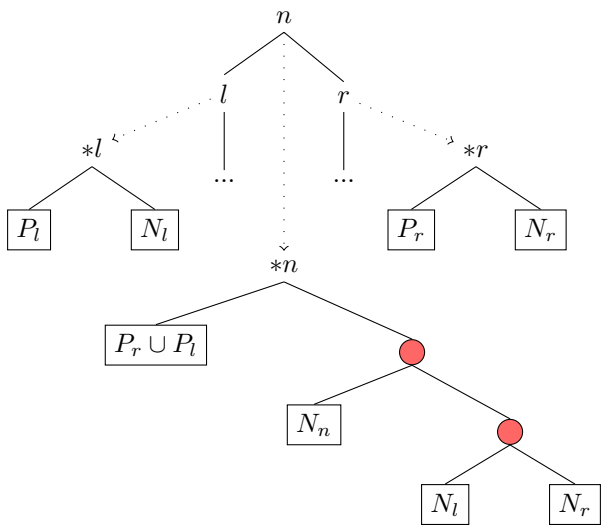
In each node of CAT tree all the propagated operations are descendants of its left child and all not propagated operations are descendants of the right child.



Initial Configuration:



After Merging:



Algorithm 1 Main Algorithm

```
    Shared Objects
        Tournament Tree
        CAT Tree
    Local Objects

 1: function Do(operation op)
 2:     l= p's assigned leaf in tree
 3:     Add_l(op)
 4:     Propagate(l.parent)
 5:     return Compute(op)
 6: end function

 7: function Propagate(node n)
                    ▷ Propagates n operations
 8:     if n==root then return
 9:     else
10:         d=Merge(n)
11:     end if
12:     Propagate(n.parent)
13: end function

14: function Merge(node n)
        ▷ Merges two children of n & updates n*,
    augmented values
15: end function

16: function Compute(op)
17:     if op.type=enq then
18:         return
19:     elsereturn enq_{#enqs(op.l)−#effective−deqs(op.l)}
20:     end if
21: end function

22: function enqs(l)
                              ▷ Returns #enqs before l
23: end function

24: function enq(i)
                                        ▷ Returns enq_i
25: end function
```

Correctness

TODO