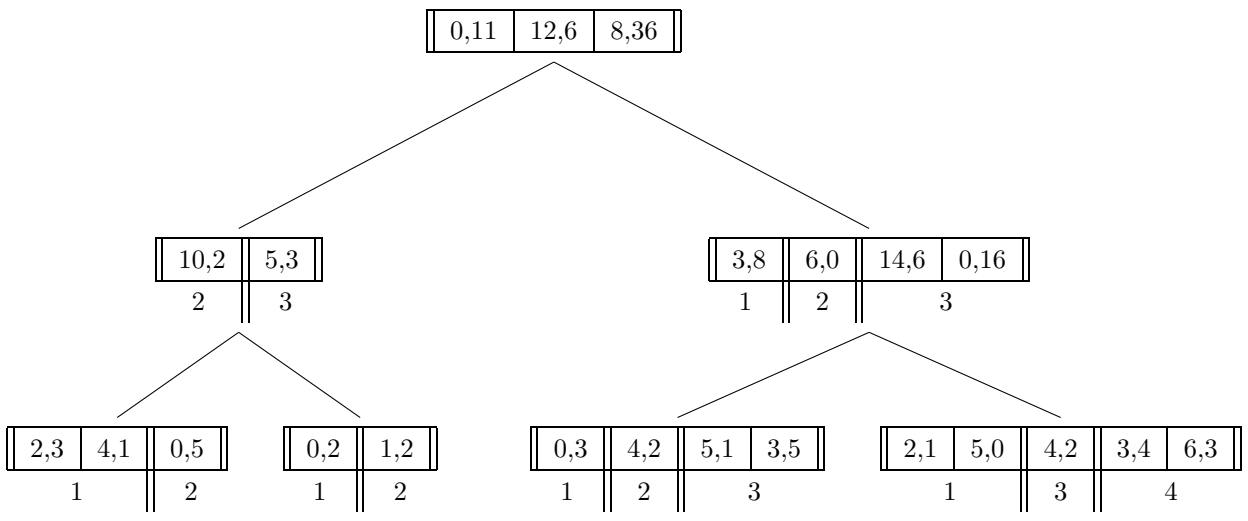
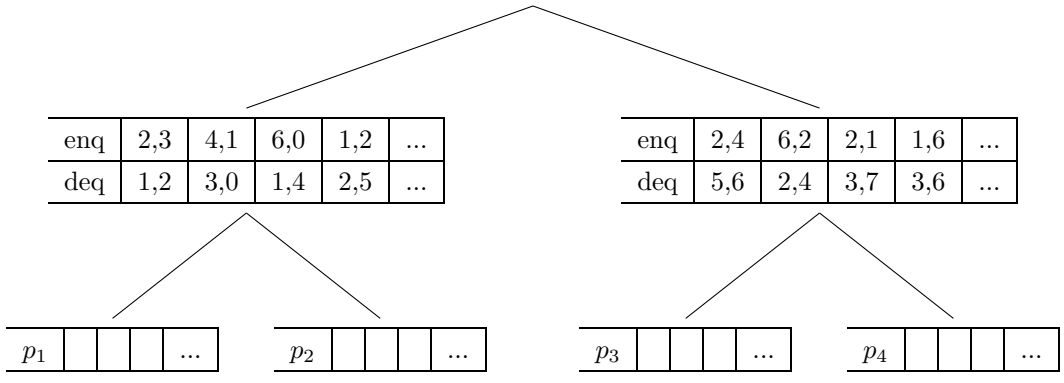
## Design




| enq | 5,6 | 5,8 | 6,3 | 3,7 | 10,2 | 1,2 | 2,3 | ... |
|---|---|---|---|---|---|---|---|---|
| deq | 3,11 | 3,6 | 5,10 | 7,9 | 3,1 | 2,0 | 9,8 | ... |
| rdeq | 11 | 9 | 13 | 10 | 4 | 2 | 14 | ... |
| size | 0 | 4 | 0 | 0 | 8 | 9 | 0 | ... |




## Description

In our model each process has an array which it appends its outgoing operation to it. Then there is a shared tournament tree that its leaves are operation lists of processes. Nodes of the tournament tree contain list of blocks. each block is a summarization of the operation propagated together. it has 4 numbers that tell us how many operation from which type has propagated from which child. In each block we order operation by this ordering(enqueus from left child, enqueues from right child, dequeues from left child, dequeues from right child) and by this rule we can find i-th operation type and the child that it has propagated up from.

Propagate procedure: First we create a block of newly added operation to children of the given node, then try to apen it to the last of the block lists. If it fails we try again.

**Algorithm 1** Main Algorithm

```
1: function Do(operation op)
2:     add p to this.ops
3:     Propagate(this.ops)
4:                                          ▷ When is op added to the root?
5:     if op is a deq then
6:         before-size: size of the block before block containing op
7:         e: #enqs in the block containing op
8:         d: #deqs in the block containing op before op
9:         if before-size + e - d < 1 then
10:            return null
11:        else
12:            d: #rdeqs before the op in all the ordering
13:            return enq(d+1) #d+1th enq value in all the enqs
14:        end if
15:    end if
16: end function

17: function Propagate(node n)
18:    b=Create-Block(n)
19:    if !tryAppend(b, n) then tryAppend(b, n)
20:    end if
21:    Propagate(n.parent)
22: end function

23: function Create-Block(n)
                ▷ constructs block of the new operation in children of n. if n is root it has extra fields: size, rdeqs.
24: end function

25: function tryAppend(b, n)
                                        ▷ tries to append b to the last of the n's list.
26: end function

27: function Index(operation op, level ∈ nodes, type ∈ {block, operation})
    ▷ returns index of op in the given level, e.g Index(op, root, block) return ordering of the block containing op
    in the root blocks.
28: end function

29: function Access(i, level ∈ nodes, type ∈ {enq, deq})
                                    ▷ returns i-th operation of given type in the given node subtree.
30: end function

31: function Prefix-Sum(i, level ∈ nodes, type∈ {enq, deq, rdeq})
    ▷ computes how many of the given type operations are before the ith operation in the given level. For rdeq it
    will only get root level.
32: end function
```