Priority Queues

Companion slides for
The Art of Multiprocessor Programming
by Maurice Herlihy & Nir Shavit

Priority Queue

- Multiset of items
  - with associated priority(score)
- Methods
  - add() a new element
  - removeMin() an element with minimum score
- Bounded / Unbounded

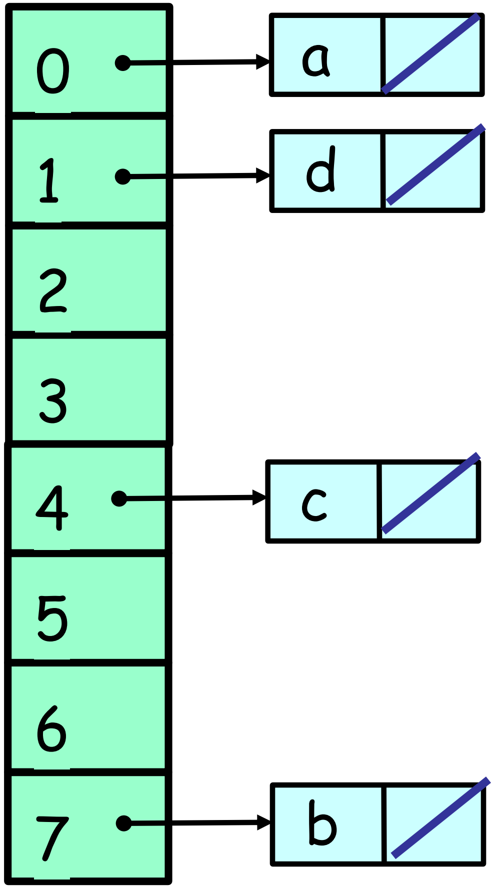
Priority Queue

- Array-based bounded
- Tree-based bounded
- Heap-based unbounded
- SkipList-based unbounded

Concurrent Priority Queues

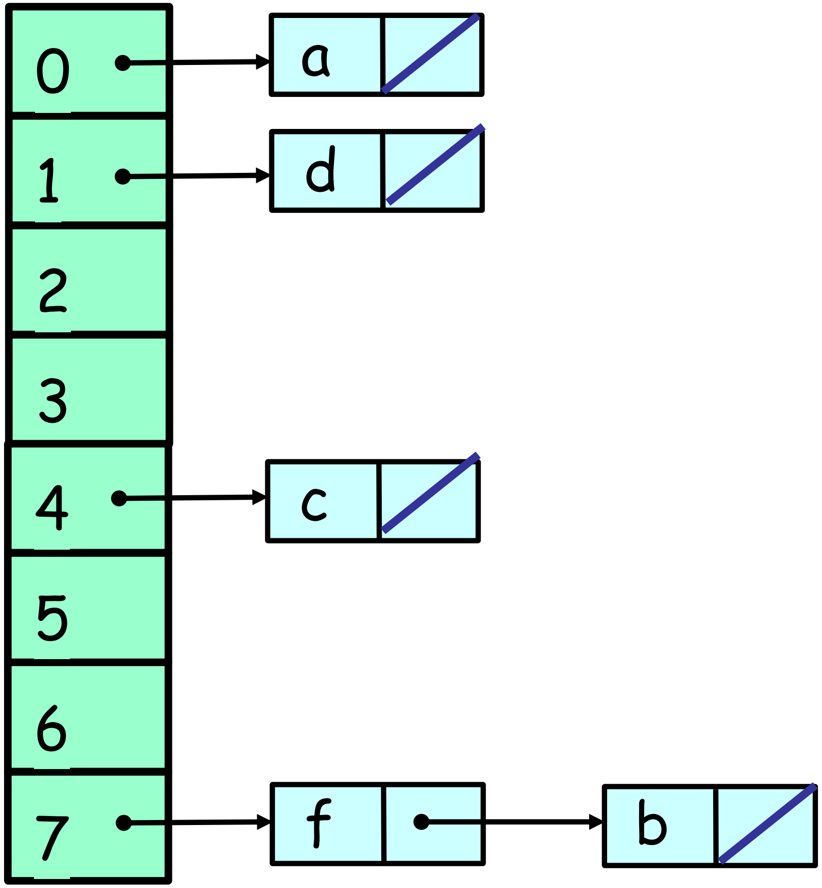
- When there's overlapping add() and removeMin(), what does it mean for an item to be in the set ?
- Linearizability - instant effect
- Quiescent consistency
  - With no additional calls, when all pending method calls complete, the values they return are consistent with some valid sequential execution

Array based P-Q



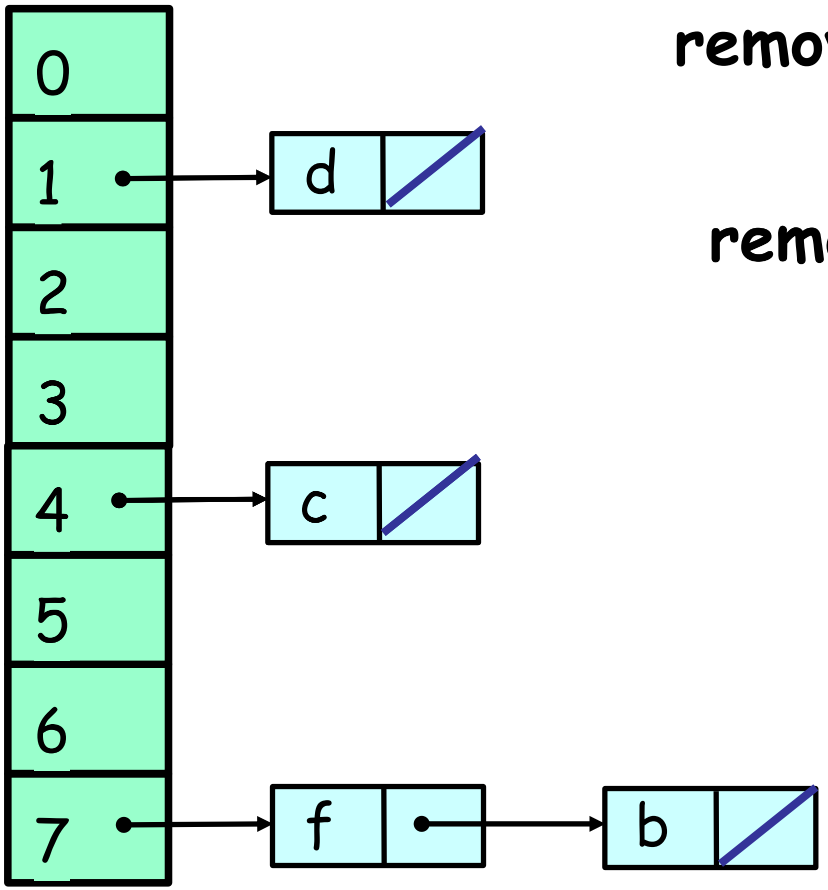
Add(7,f)

Array based P-Q



removeMin()

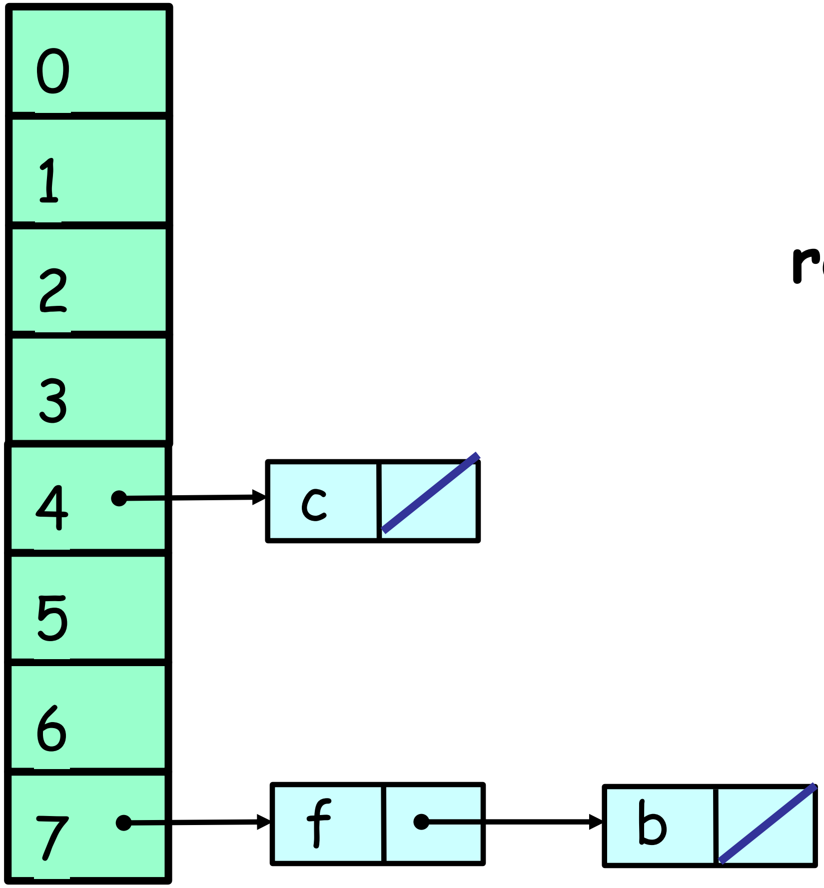
Array based P-Q



removeMin() returns a

removeMin()

Array based P-Q



removeMin() returns d

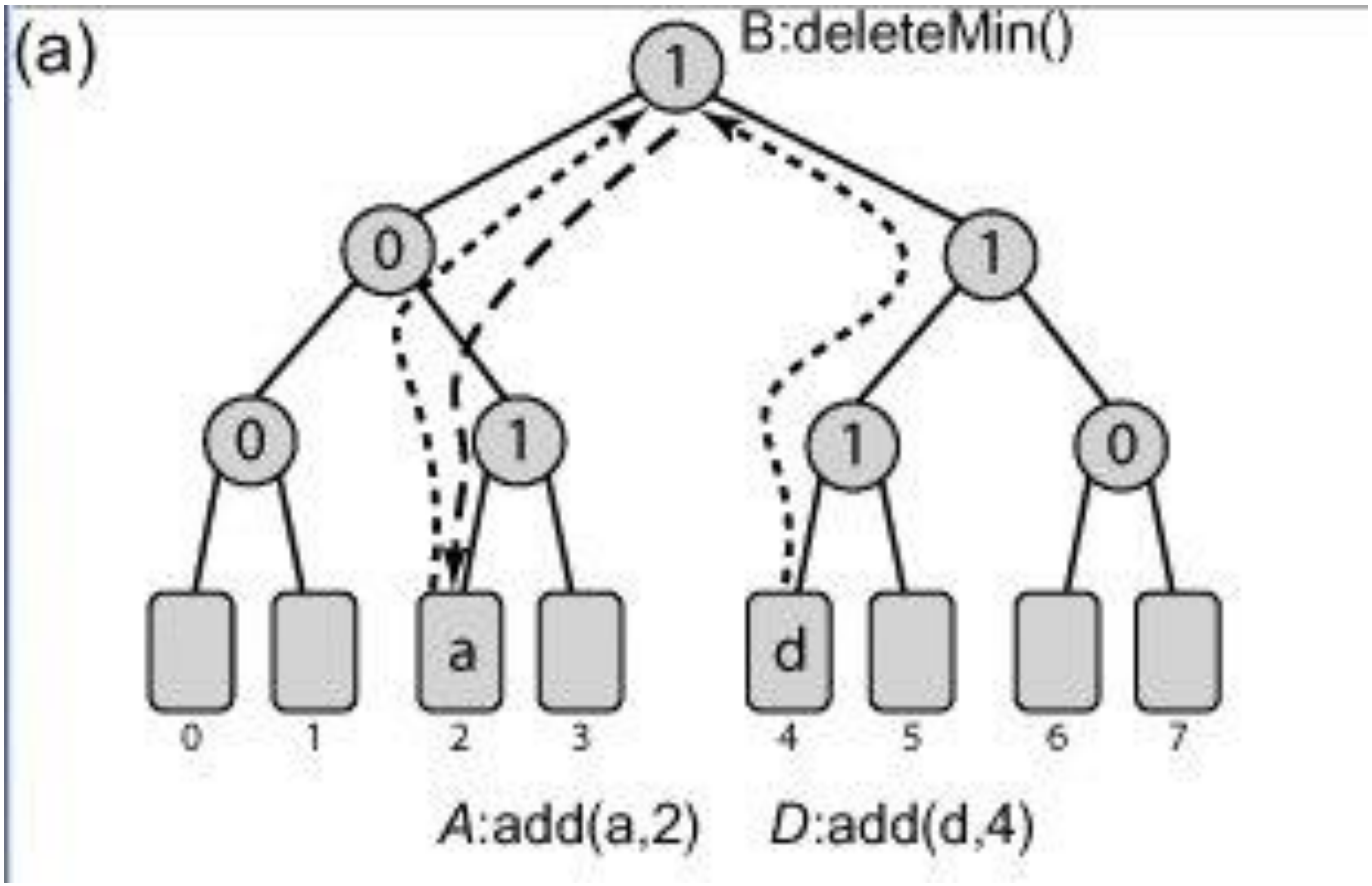
Array-based bounded P-Q

- Use a bounded array of Bins
  - Each Bin has items with the same priority
  - Put()
  - Get()
- Add(item) puts the item into the Bin
  - with the same priority
- removeMin() searches the Bin from the highest priority

Tree-based bounded P-Q

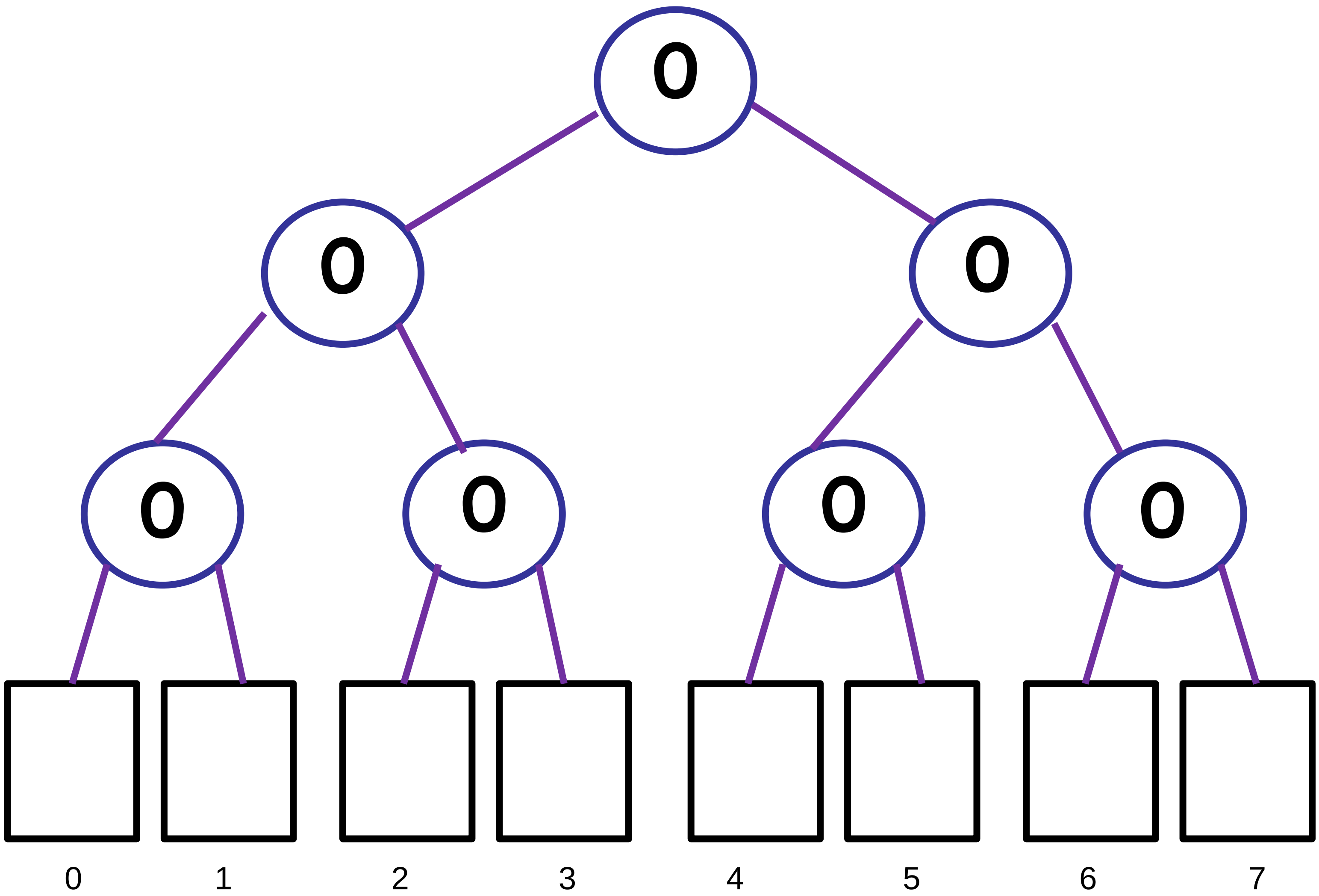
- Lock-free quiescently consistent
- A binary tree
  - Each internal node has a bounded shared counter indicating # of items in its left subtree
  - Each leaf has a Bin containing items with the same priority

Tree-based bounded P-Q

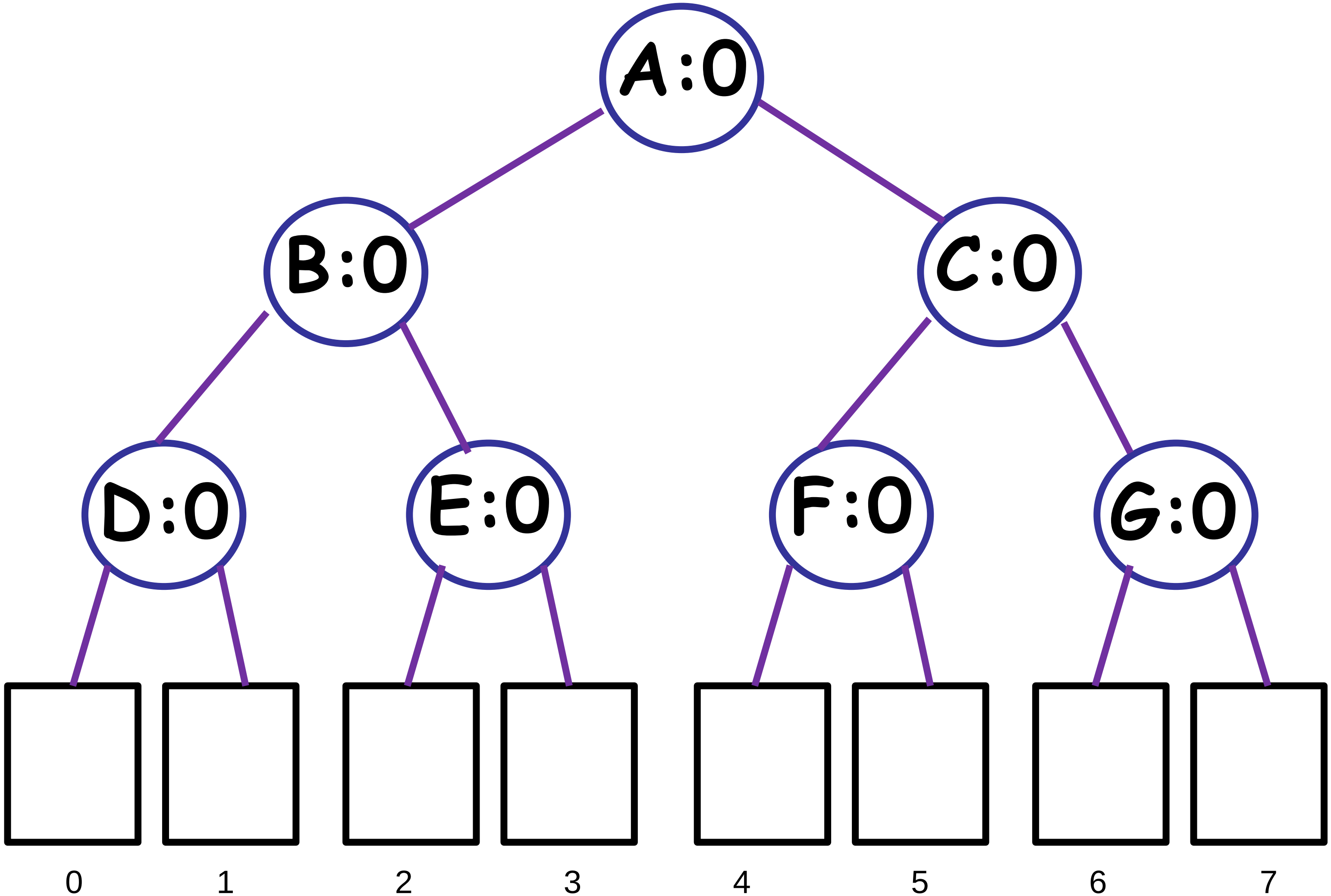


0 1 2 3 4 5 6 7 8

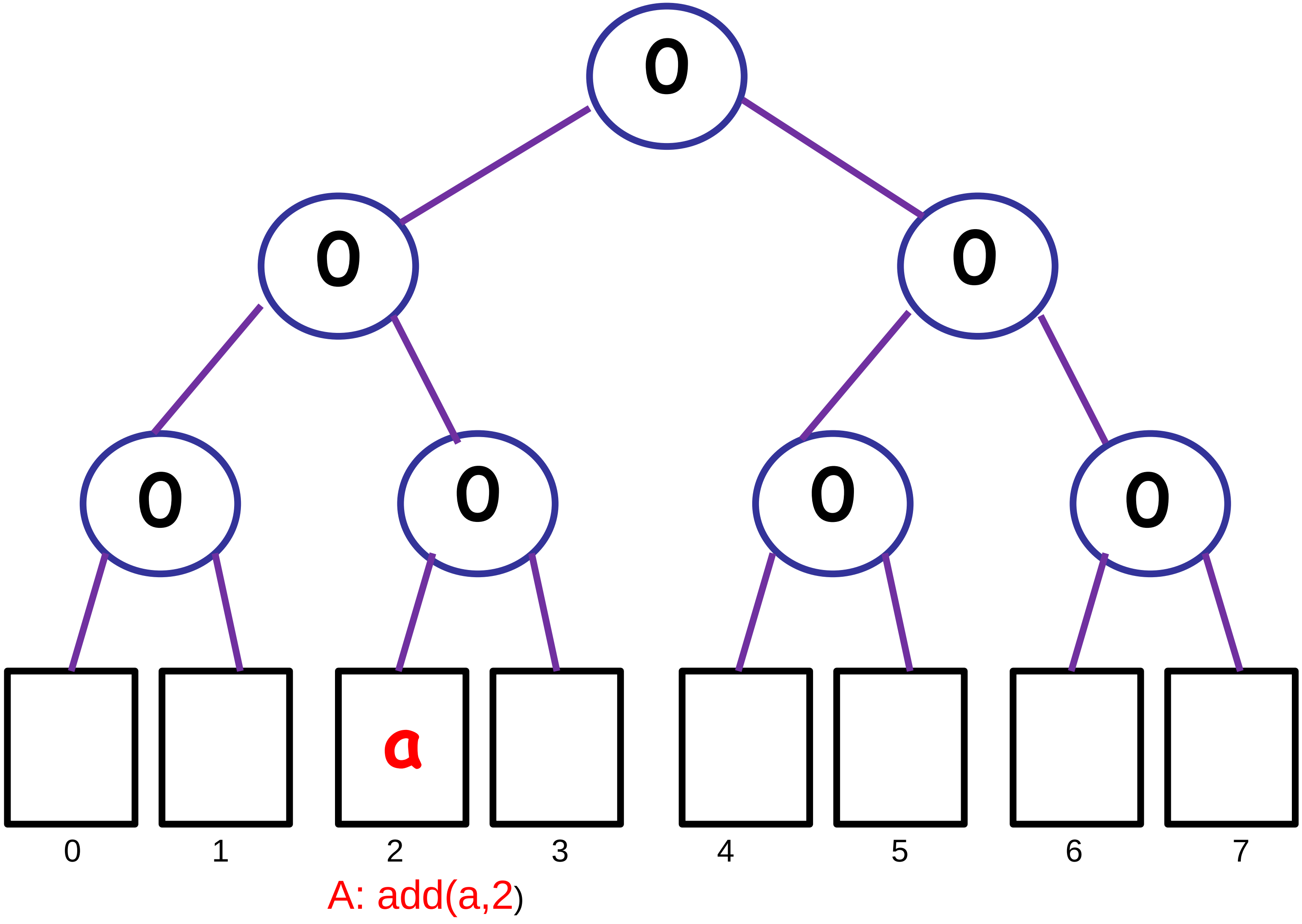
Tree-based bounded P-Q



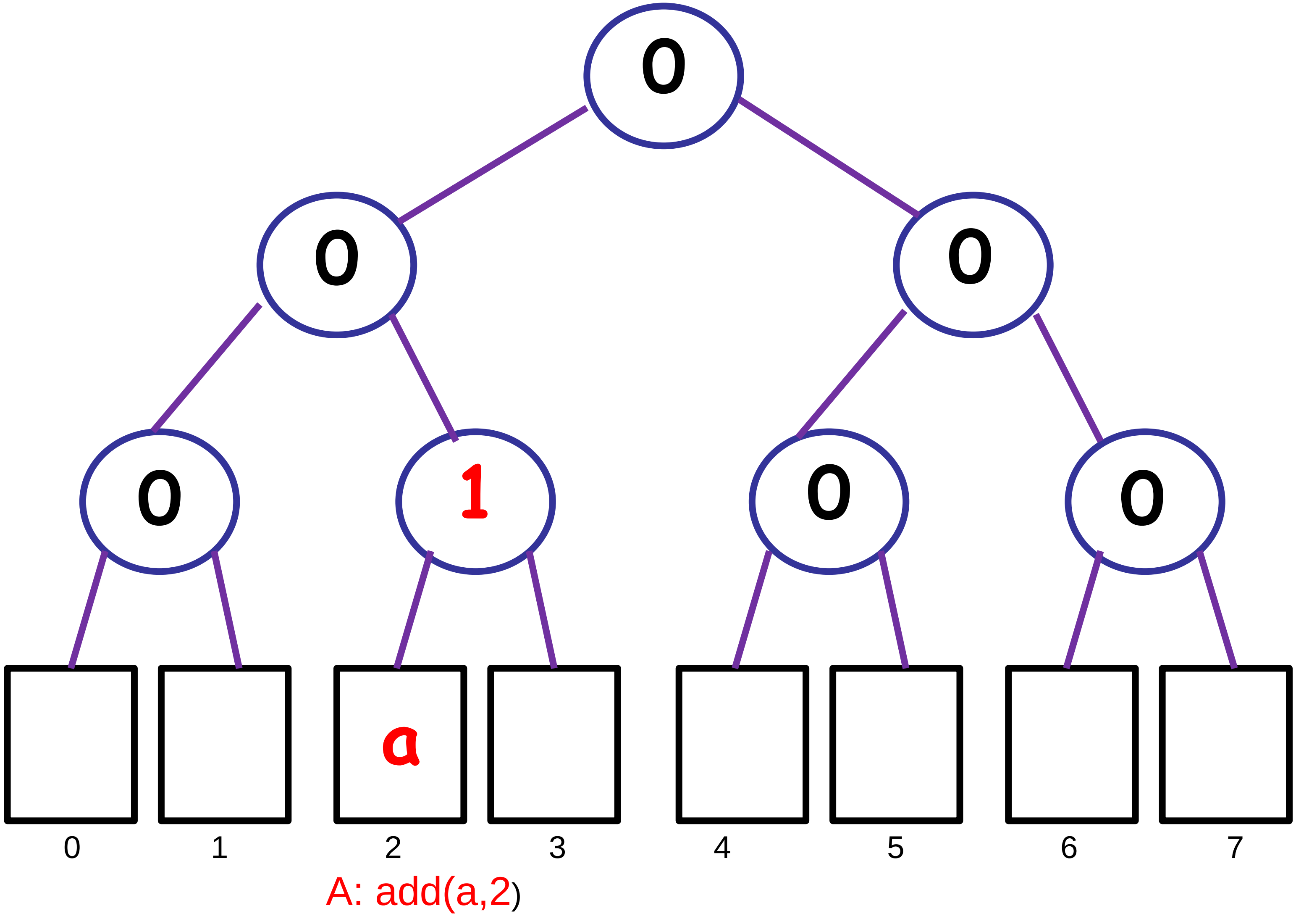
Tree-based bounded P-Q



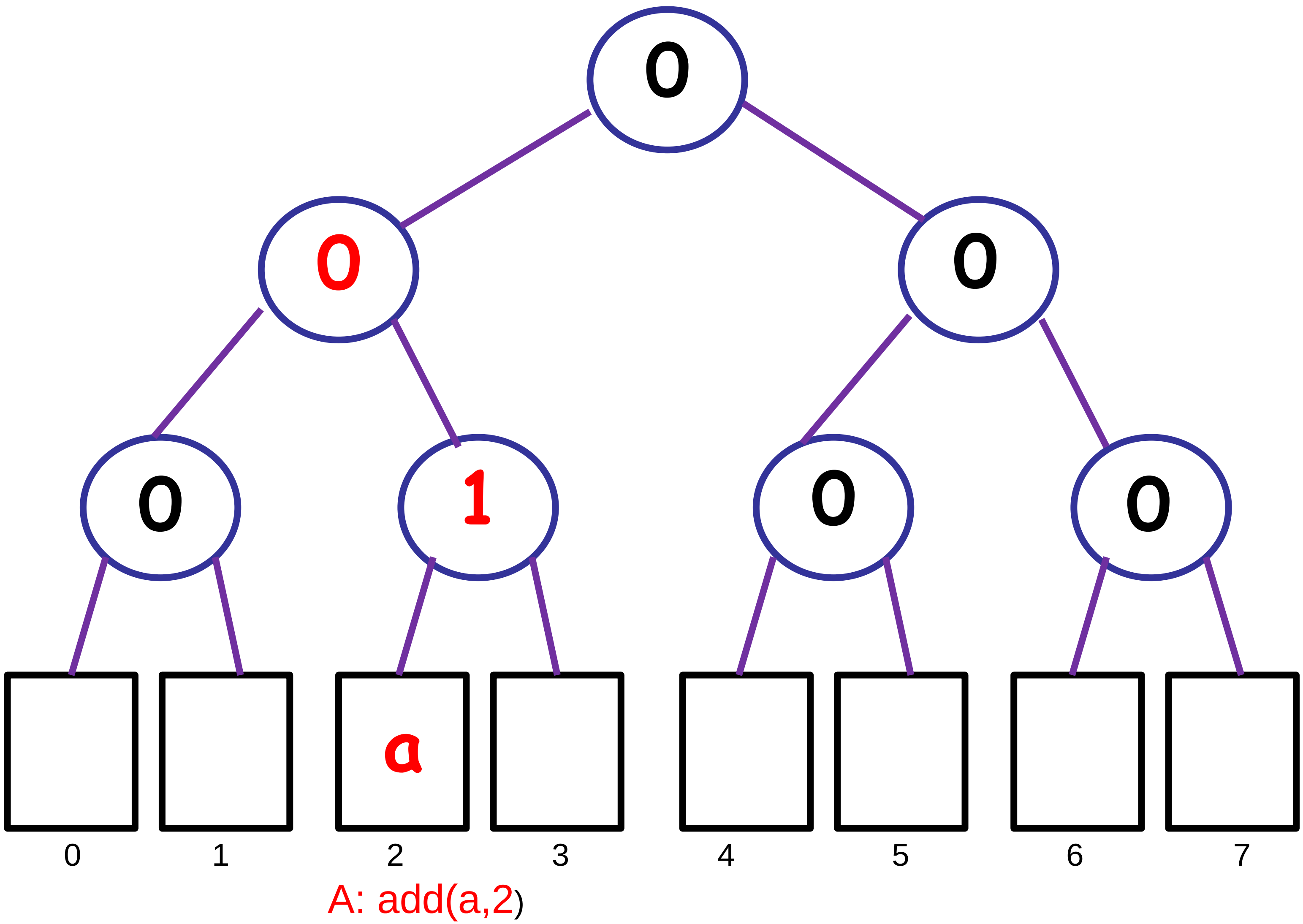
Tree-based bounded P-Q



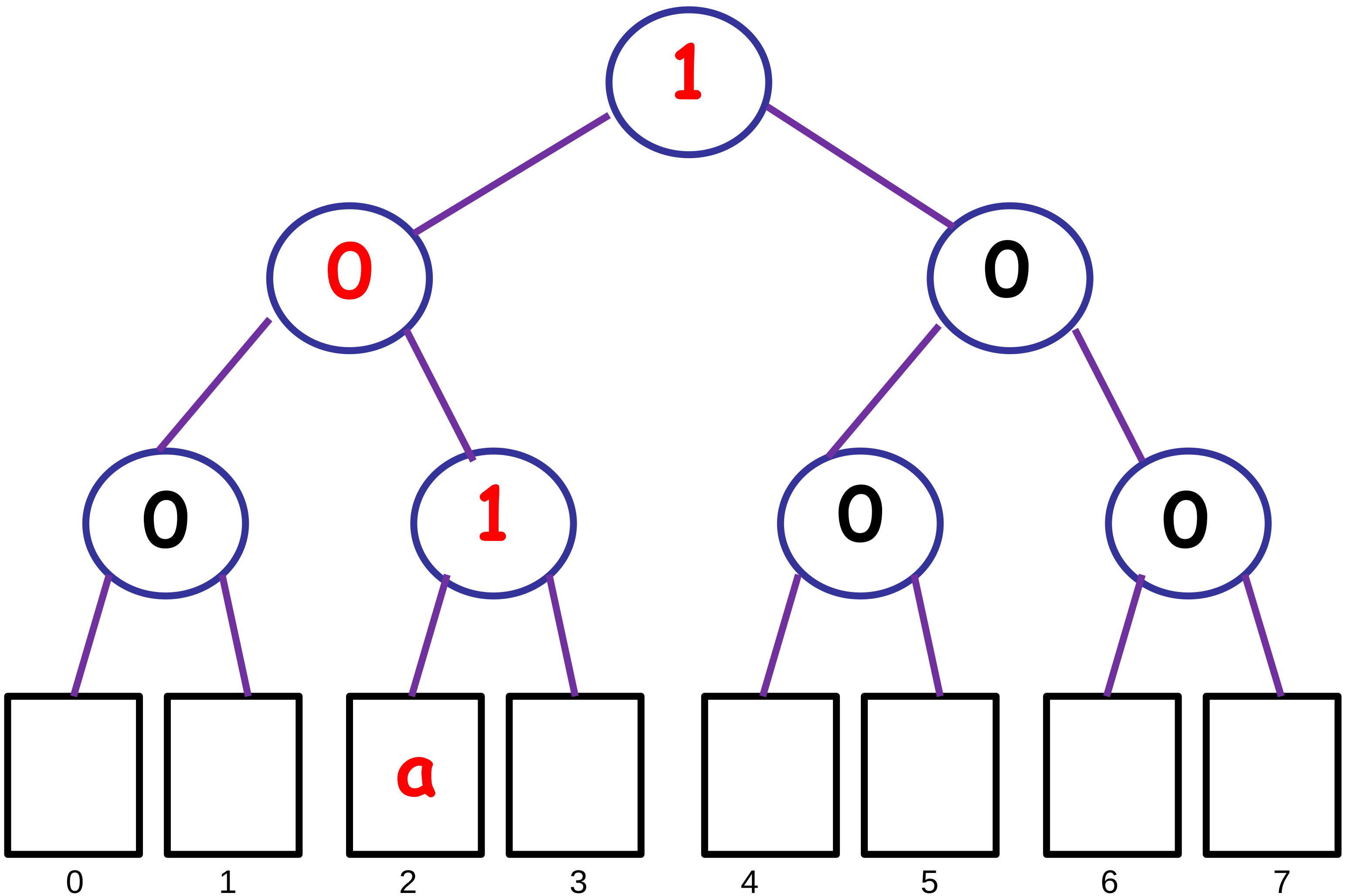
Tree-based bounded P-Q



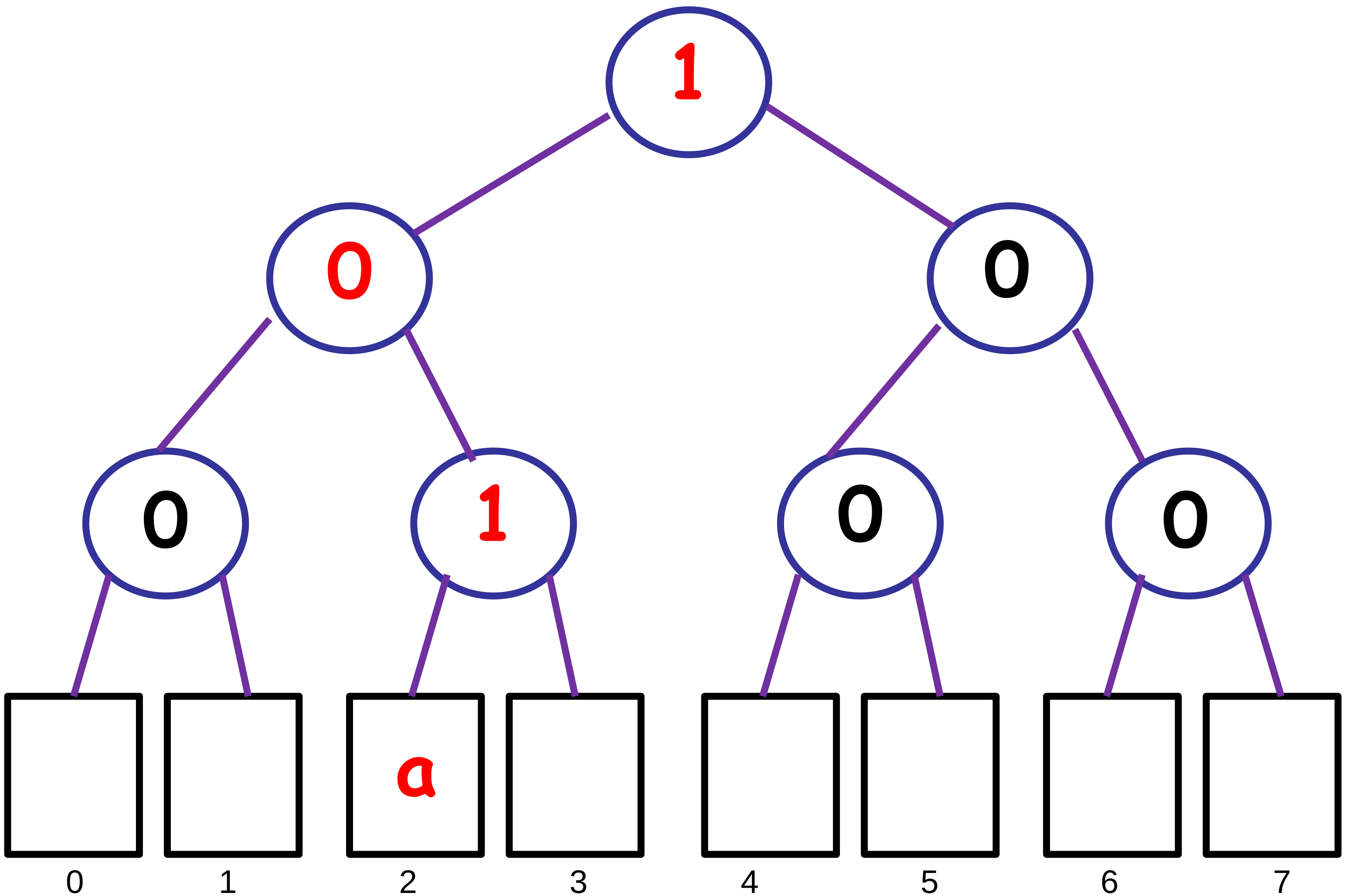
Tree-based bounded P-Q



Tree-based bounded P-Q

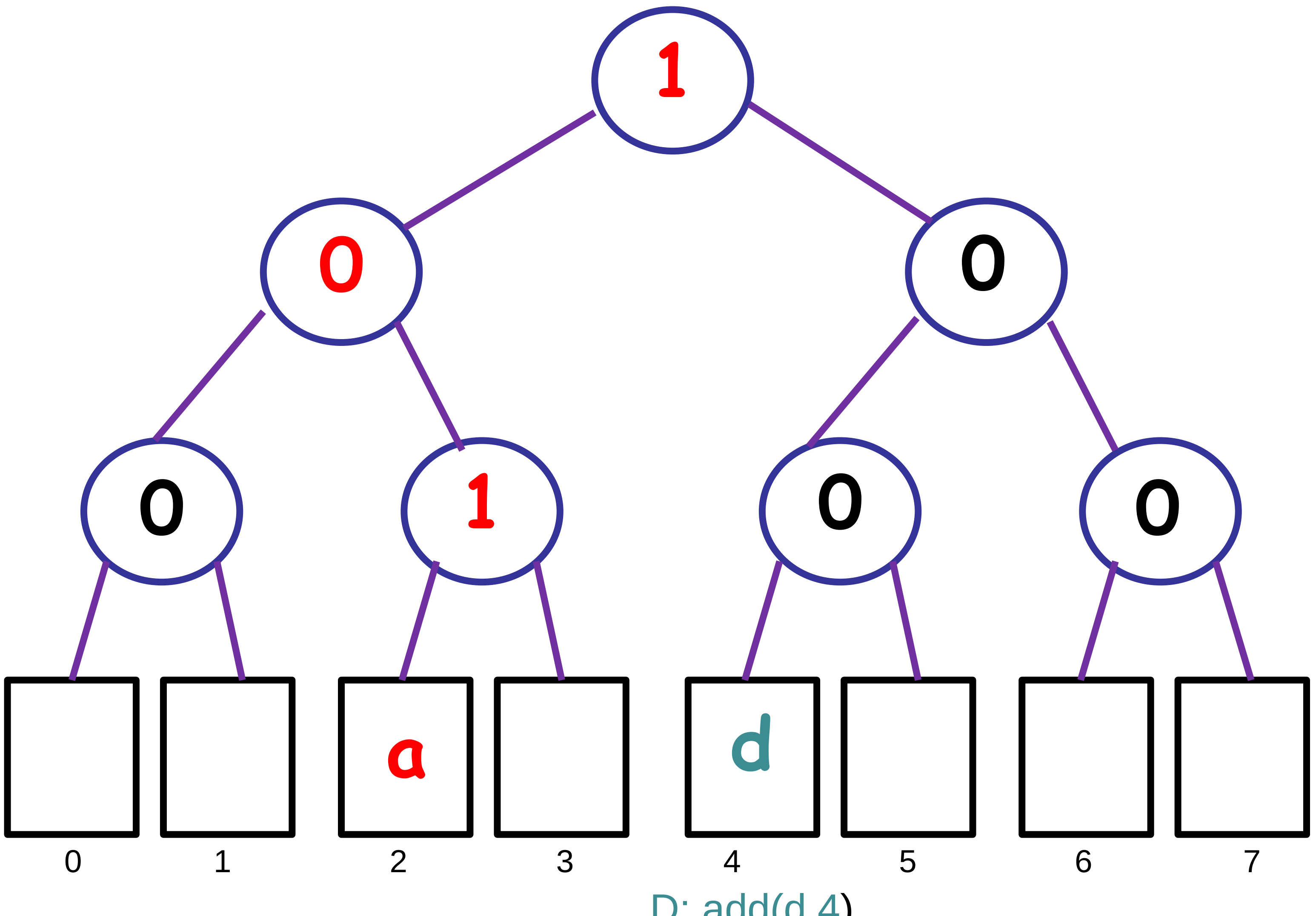


Tree-based bounded P-Q

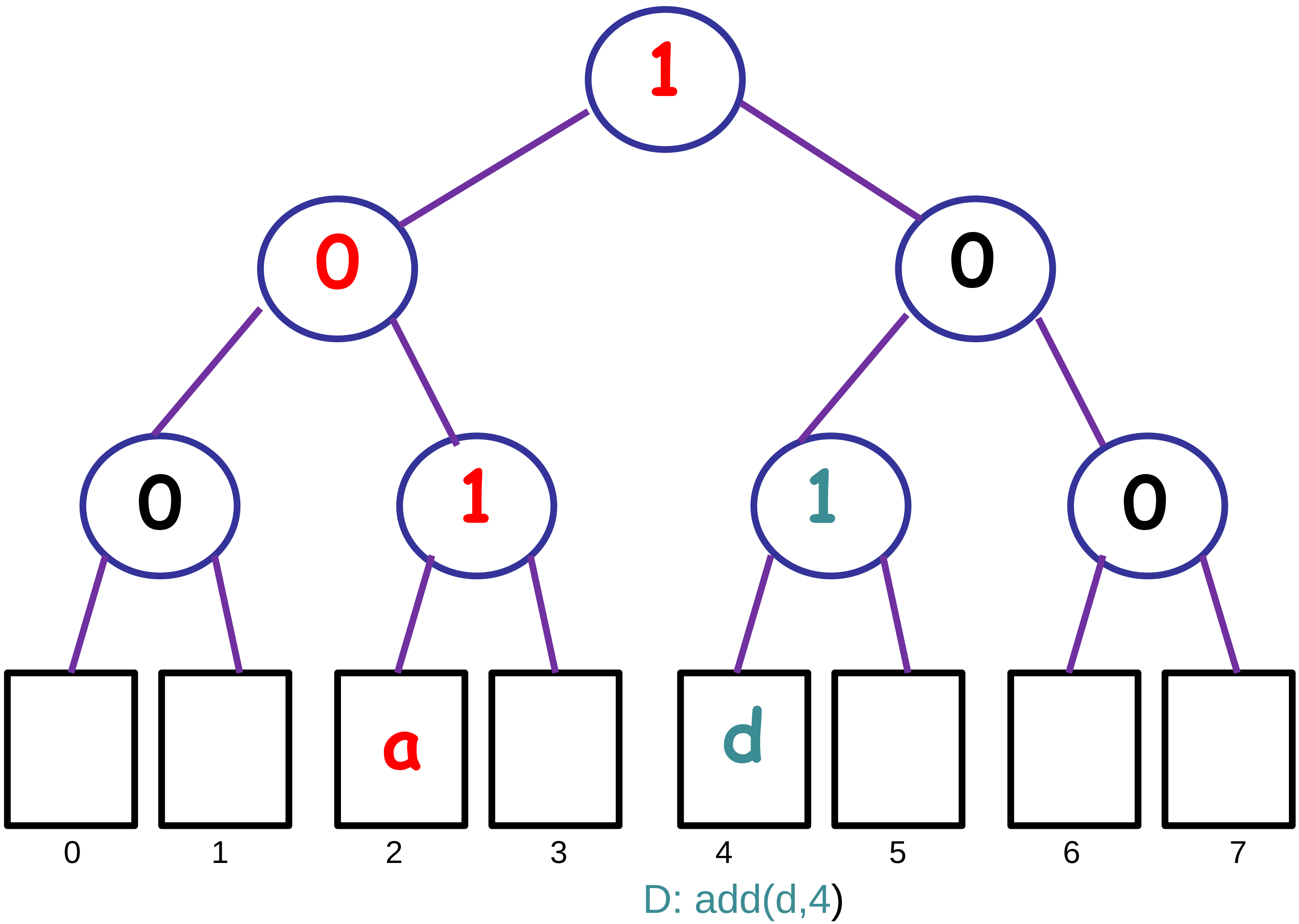


D: add(d,4)

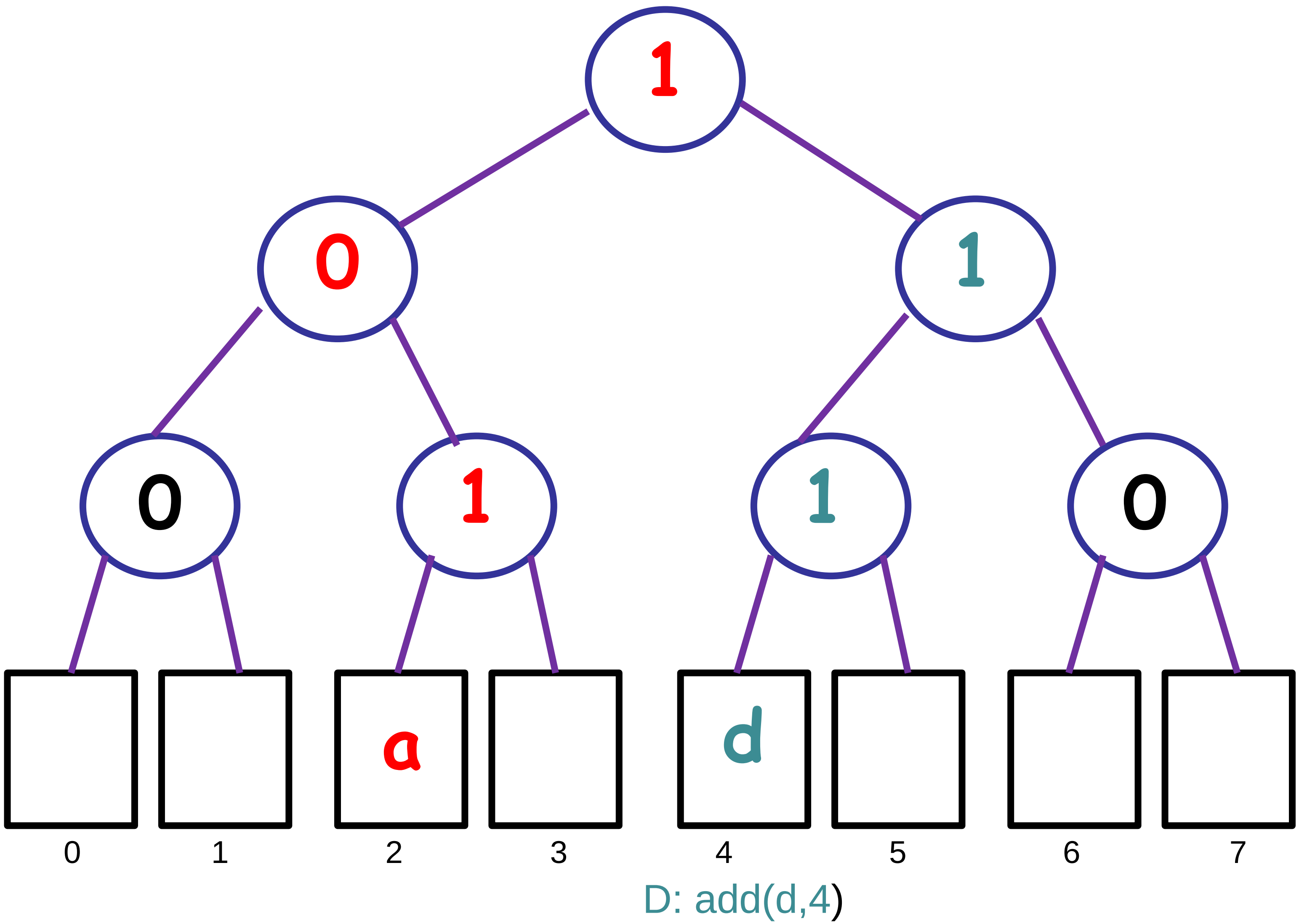
Tree-based bounded P-Q



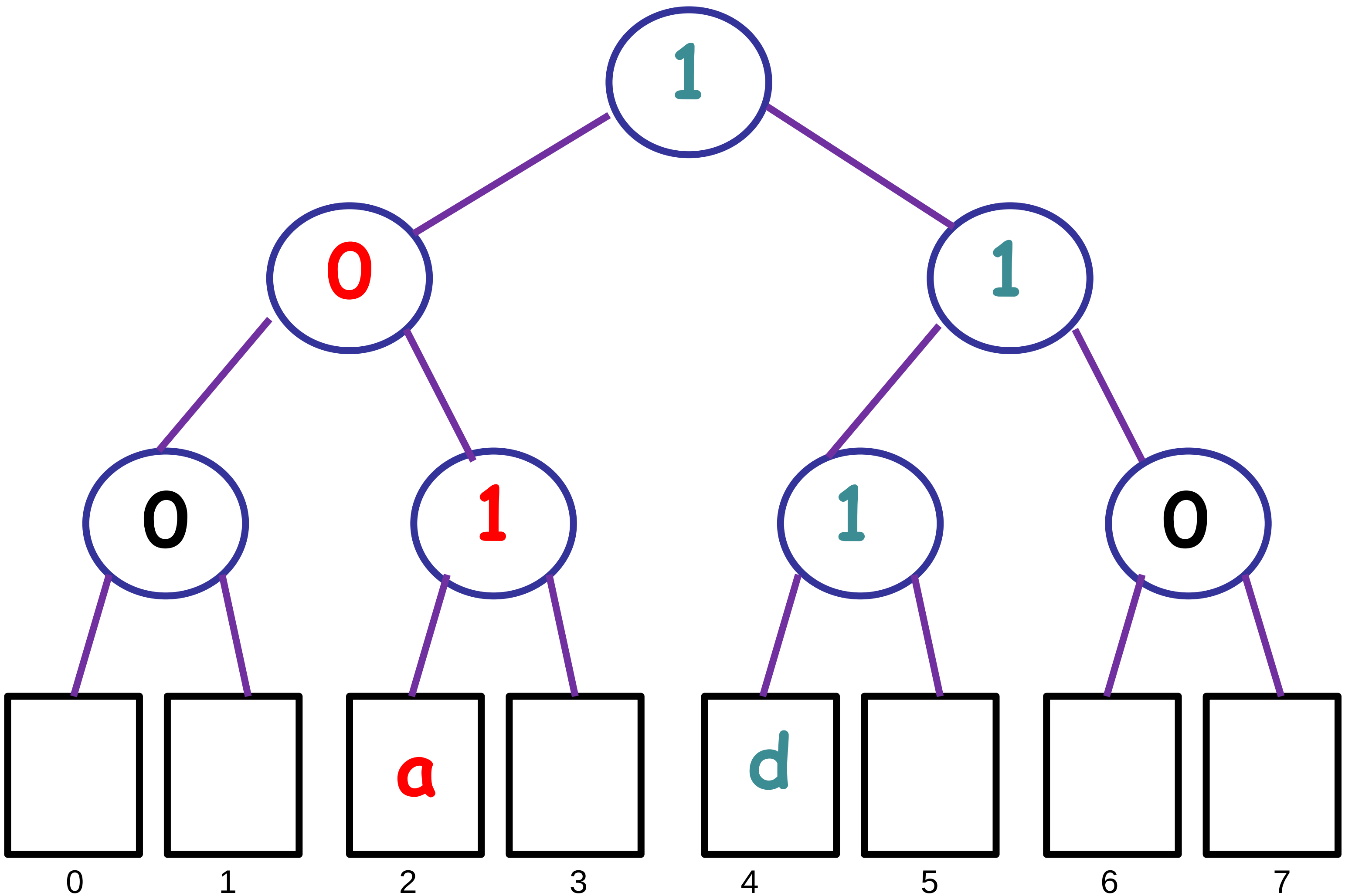
Tree-based bounded P-Q



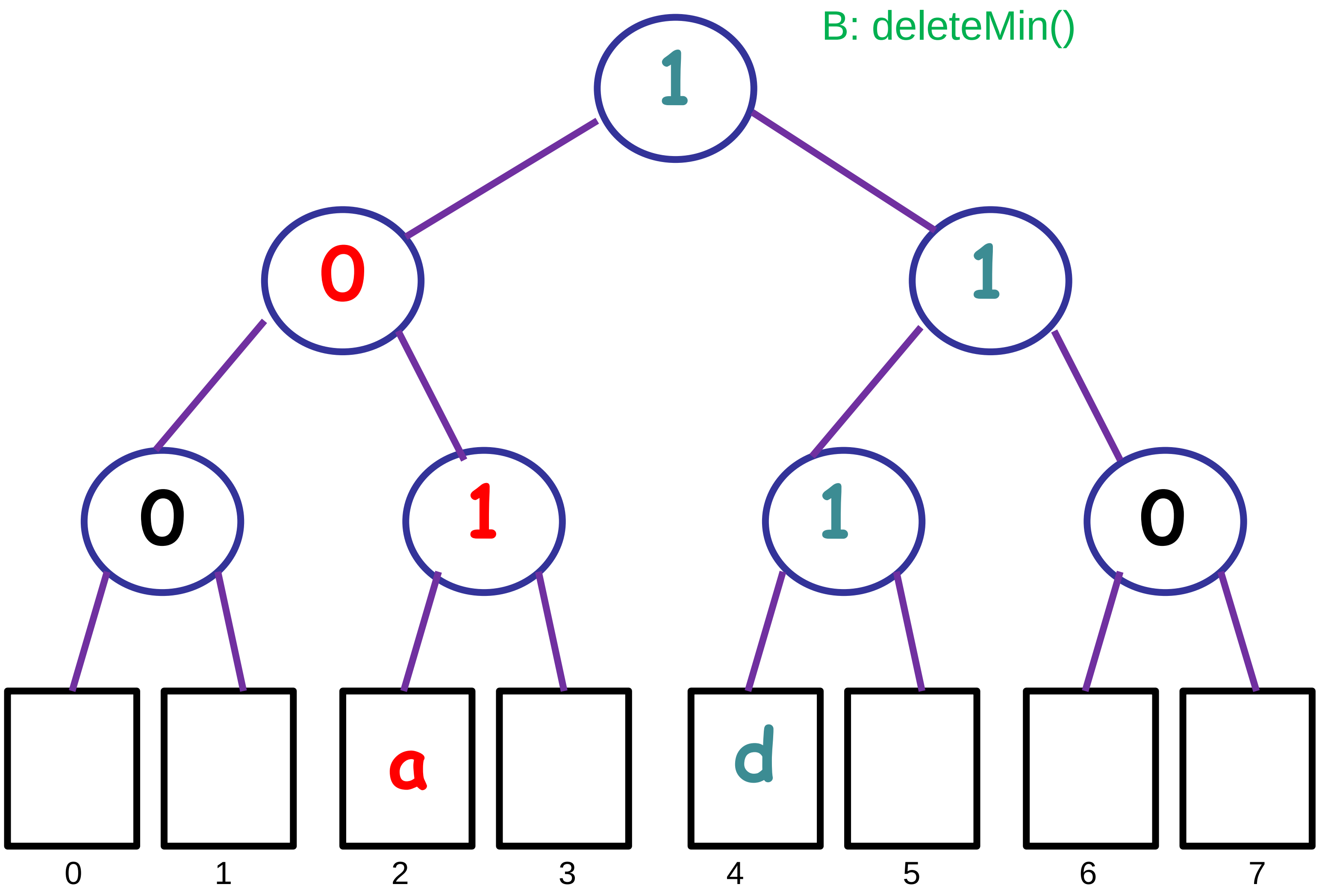
Tree-based bounded P-Q



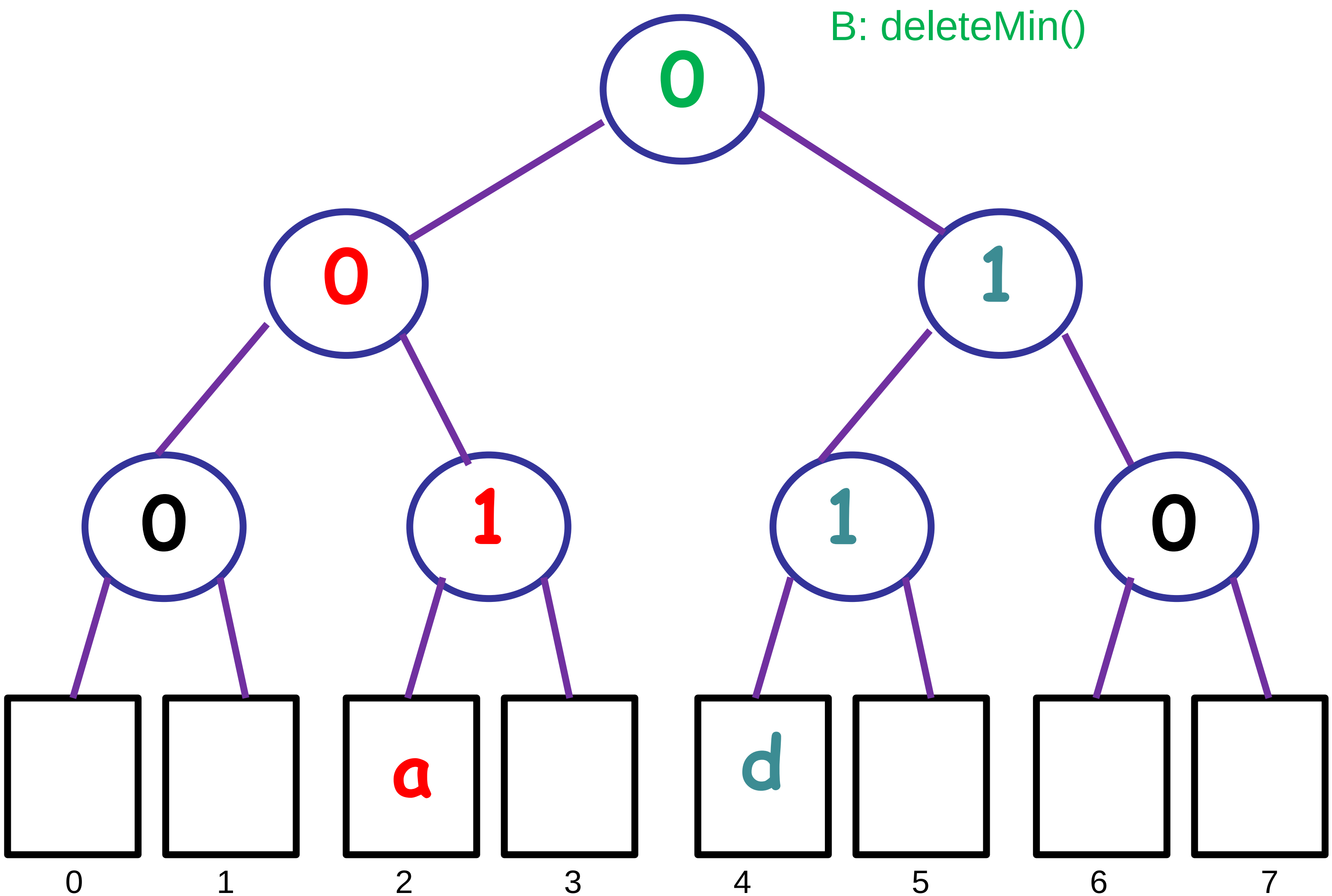
Tree-based bounded P-Q



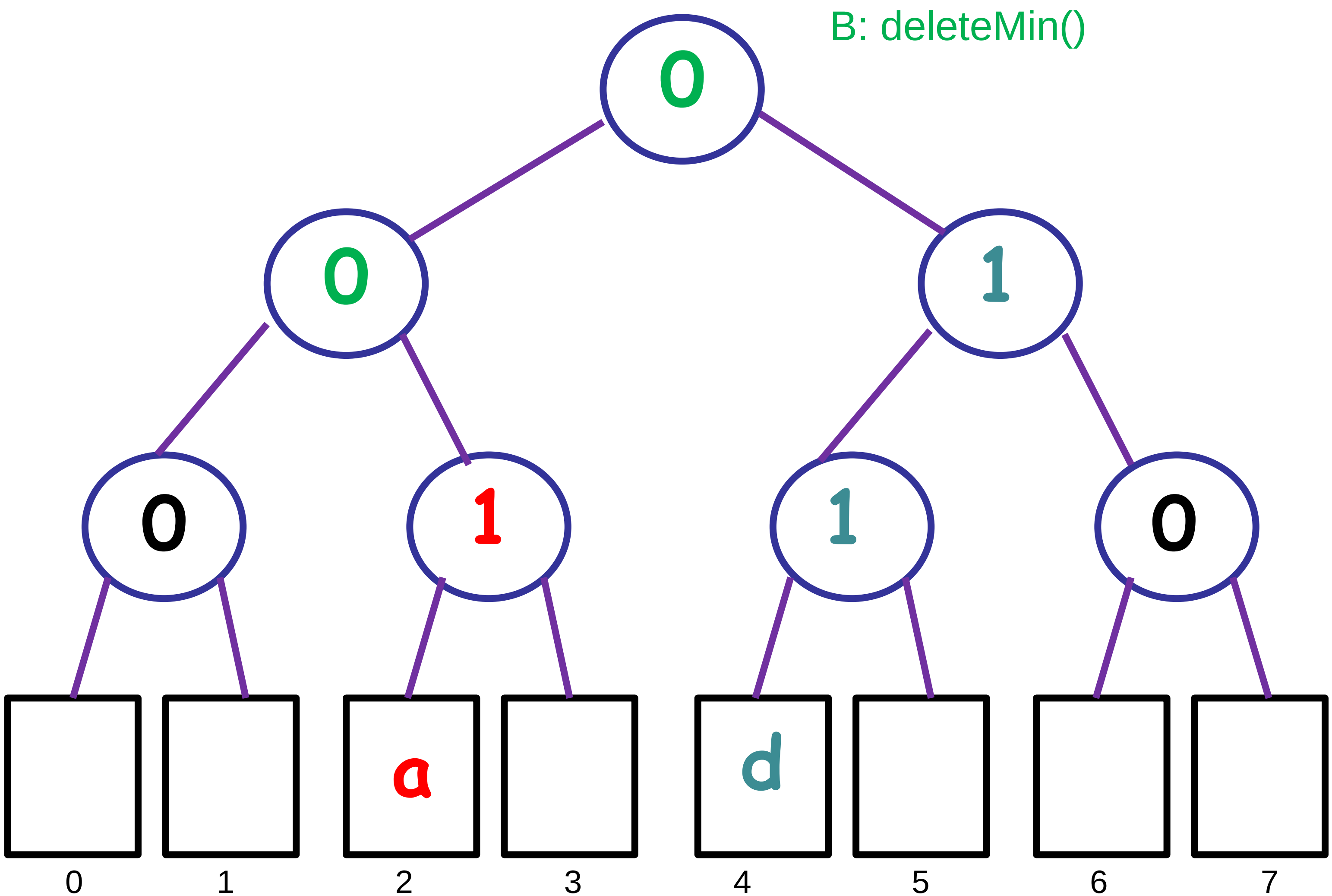
Tree-based bounded P-Q



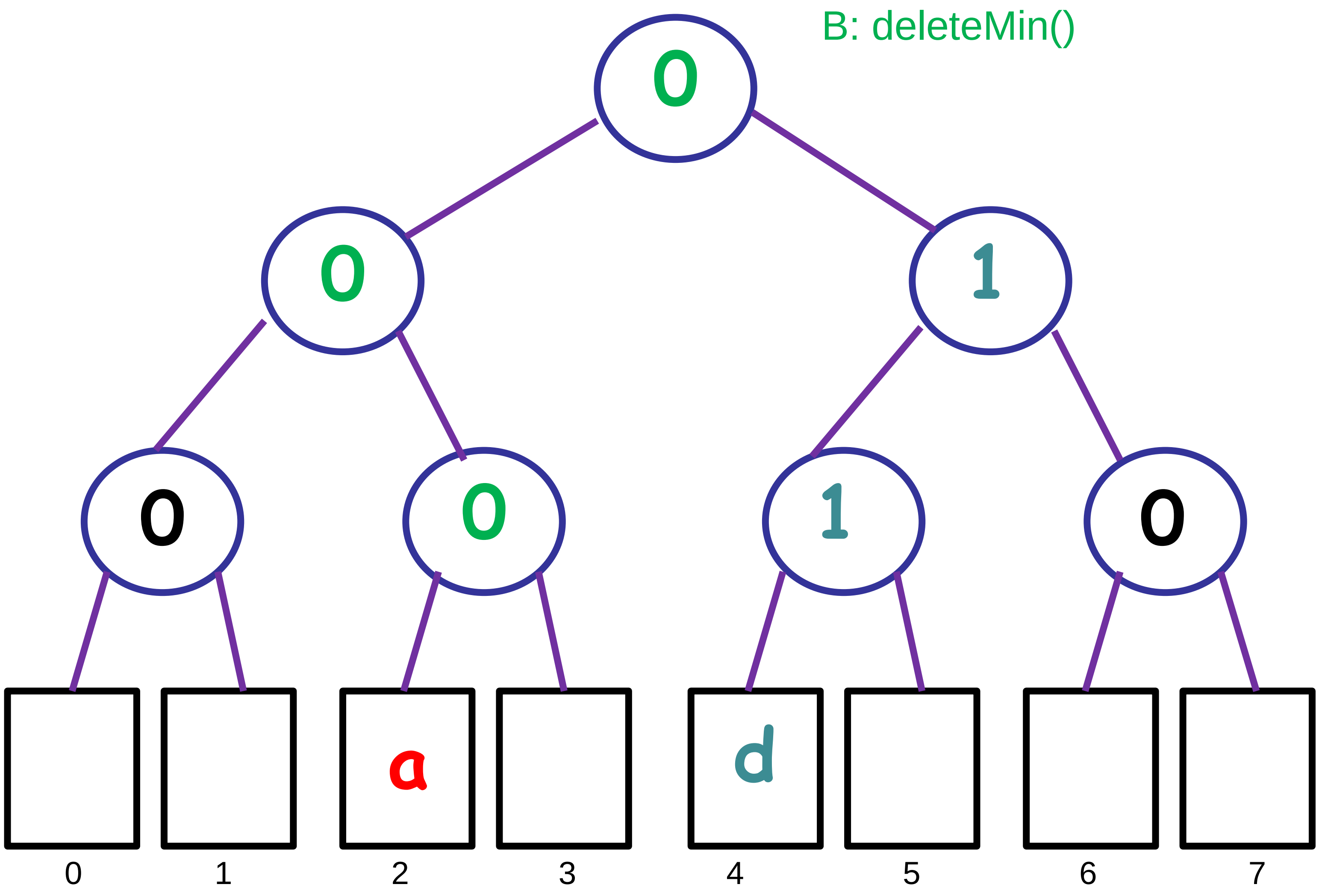
Tree-based bounded P-Q



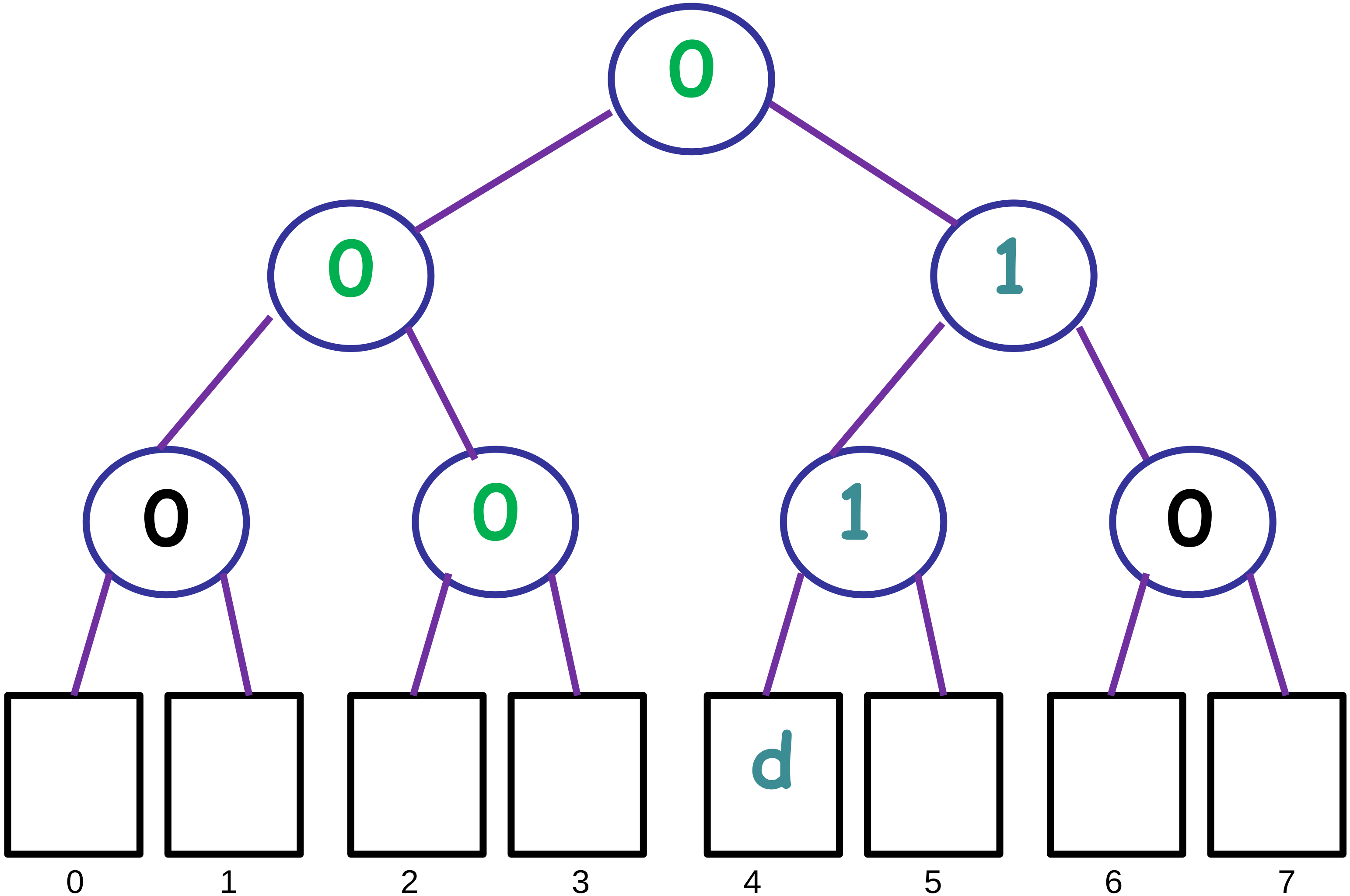
Tree-based bounded P-Q



Tree-based bounded P-Q



Tree-based bounded P-Q



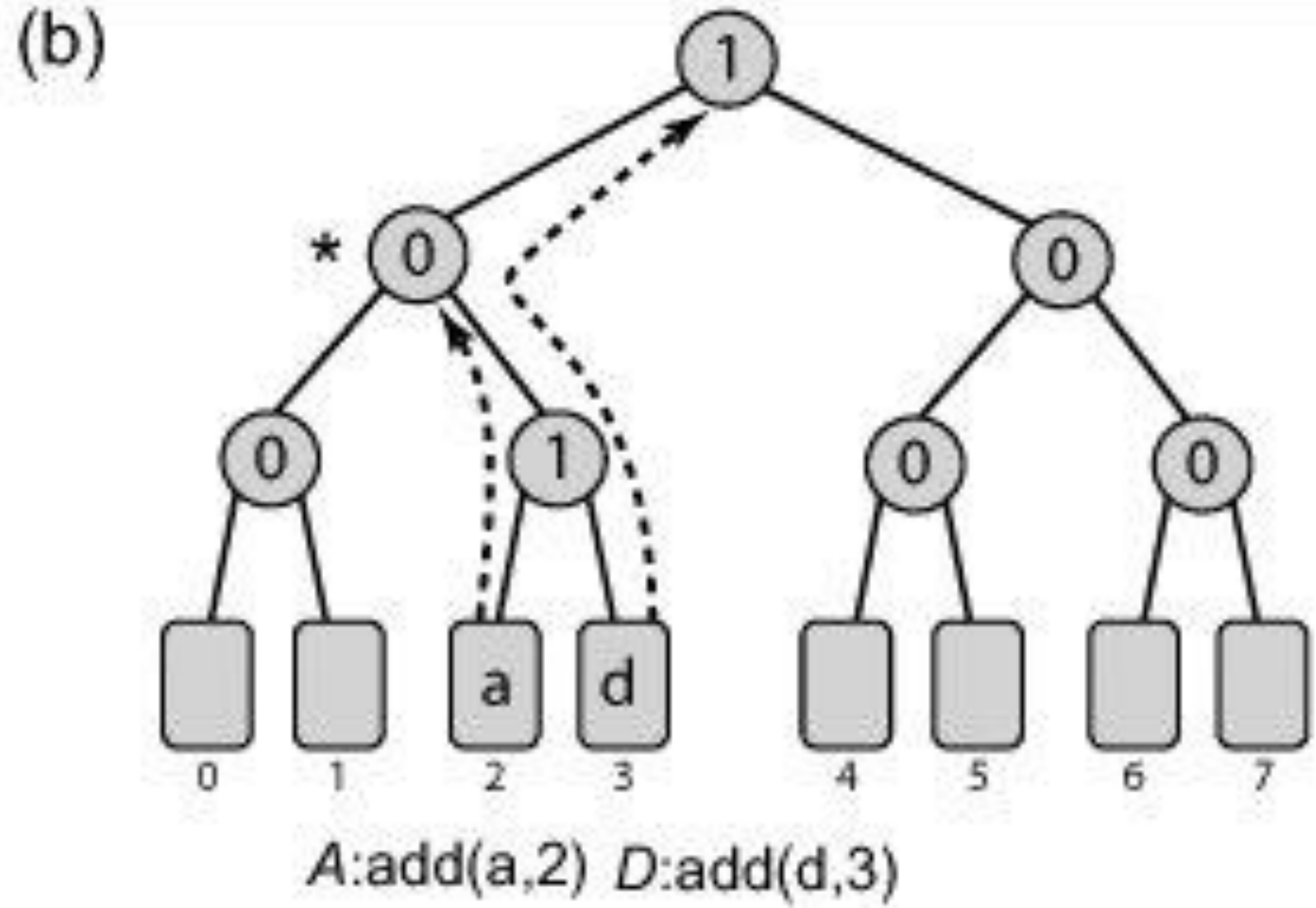
a = deleteMin()

Does it still work concurrently ?

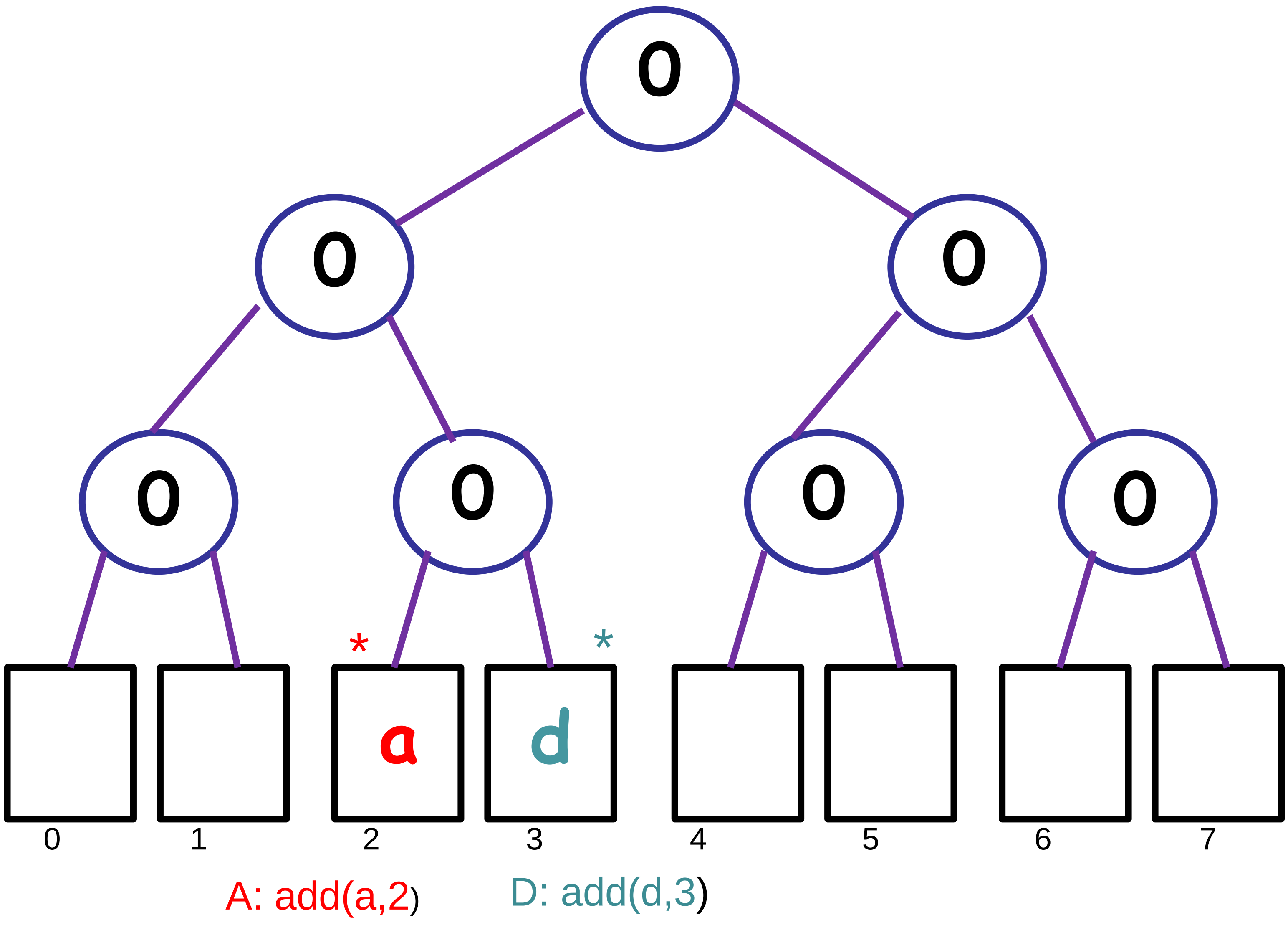
- Lock-free quiescently consistent

- A: add(a,2)
- D: add(d,3)

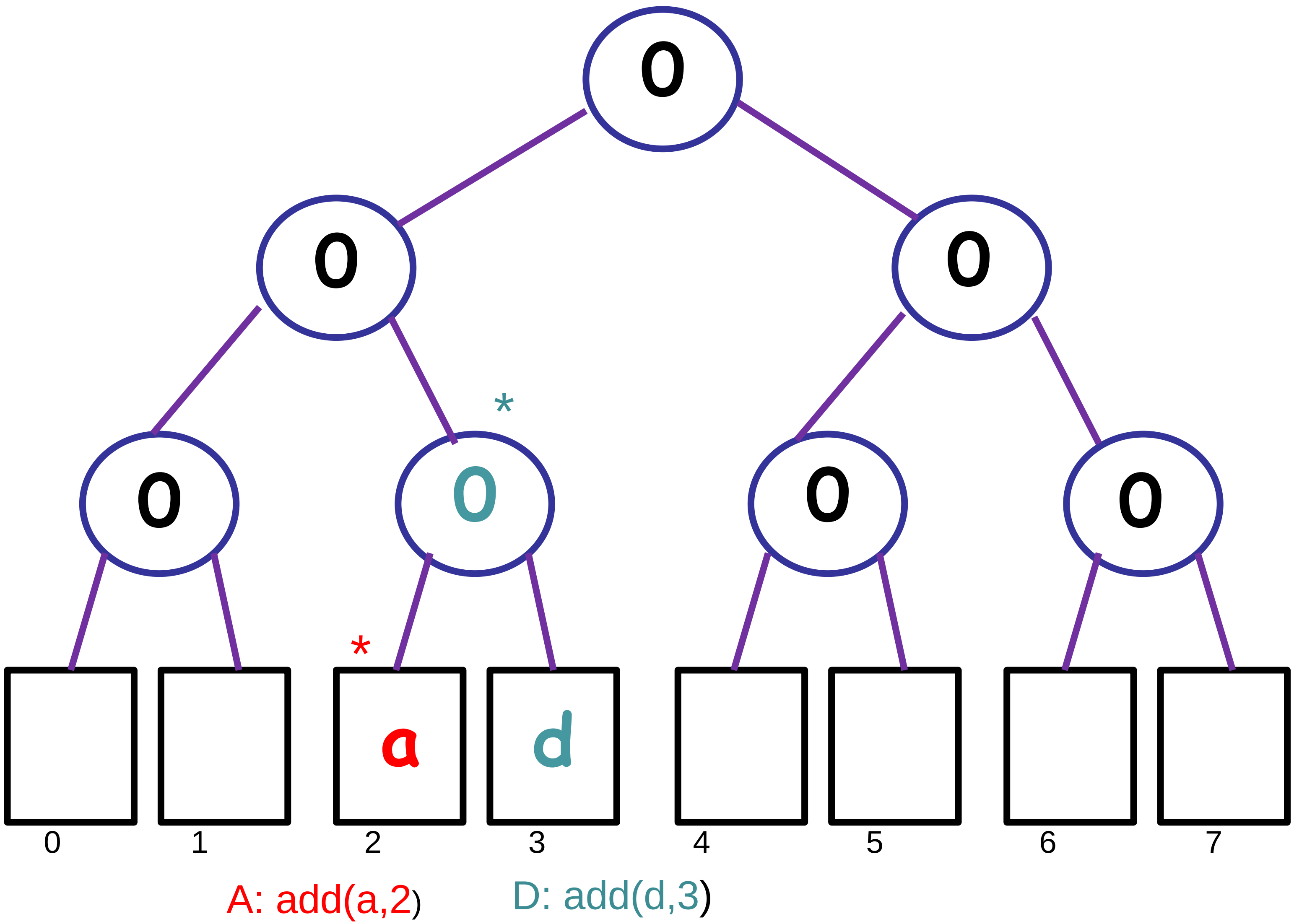
Tree-based bounded P-Q



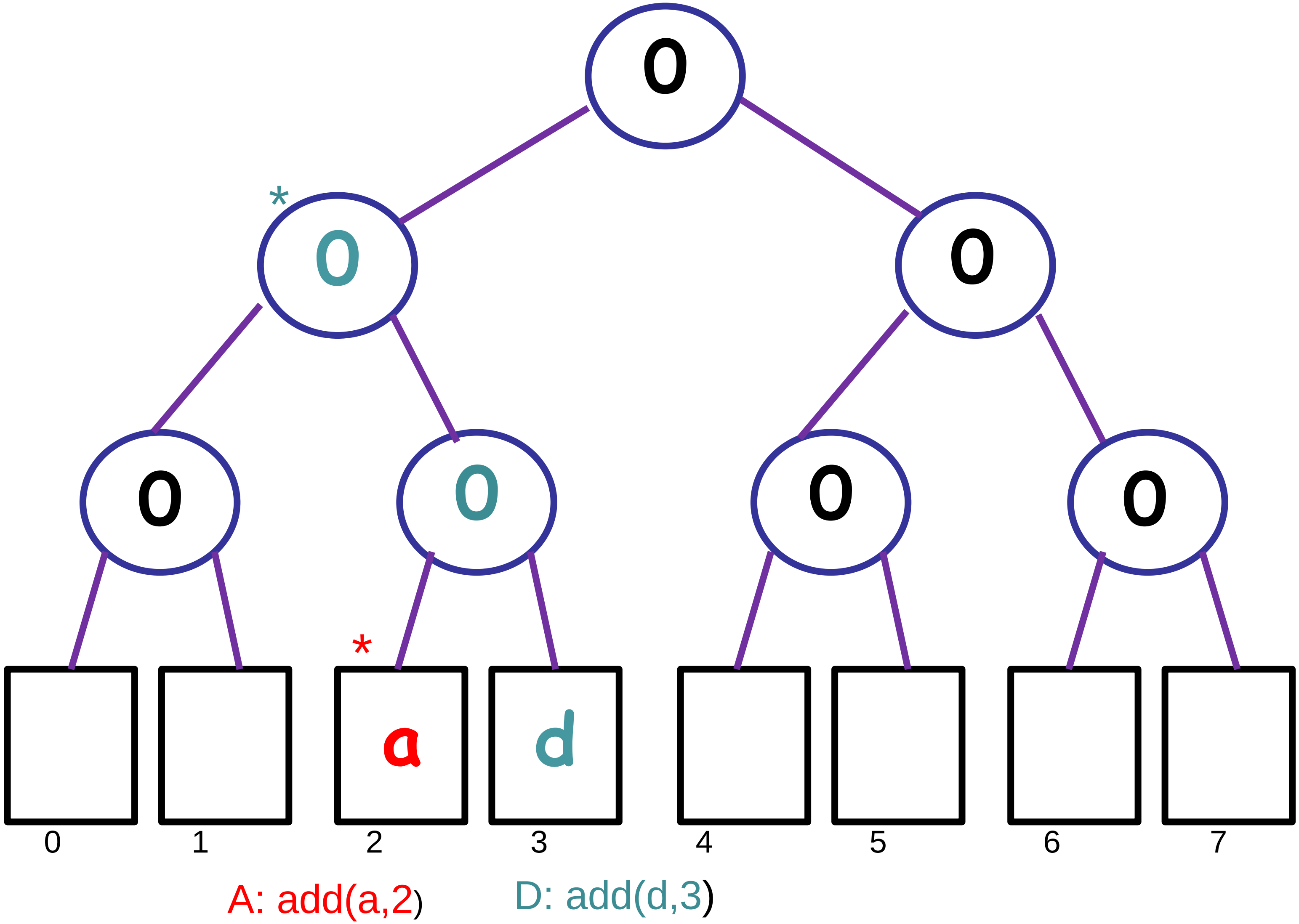
Tree-based bounded P-Q



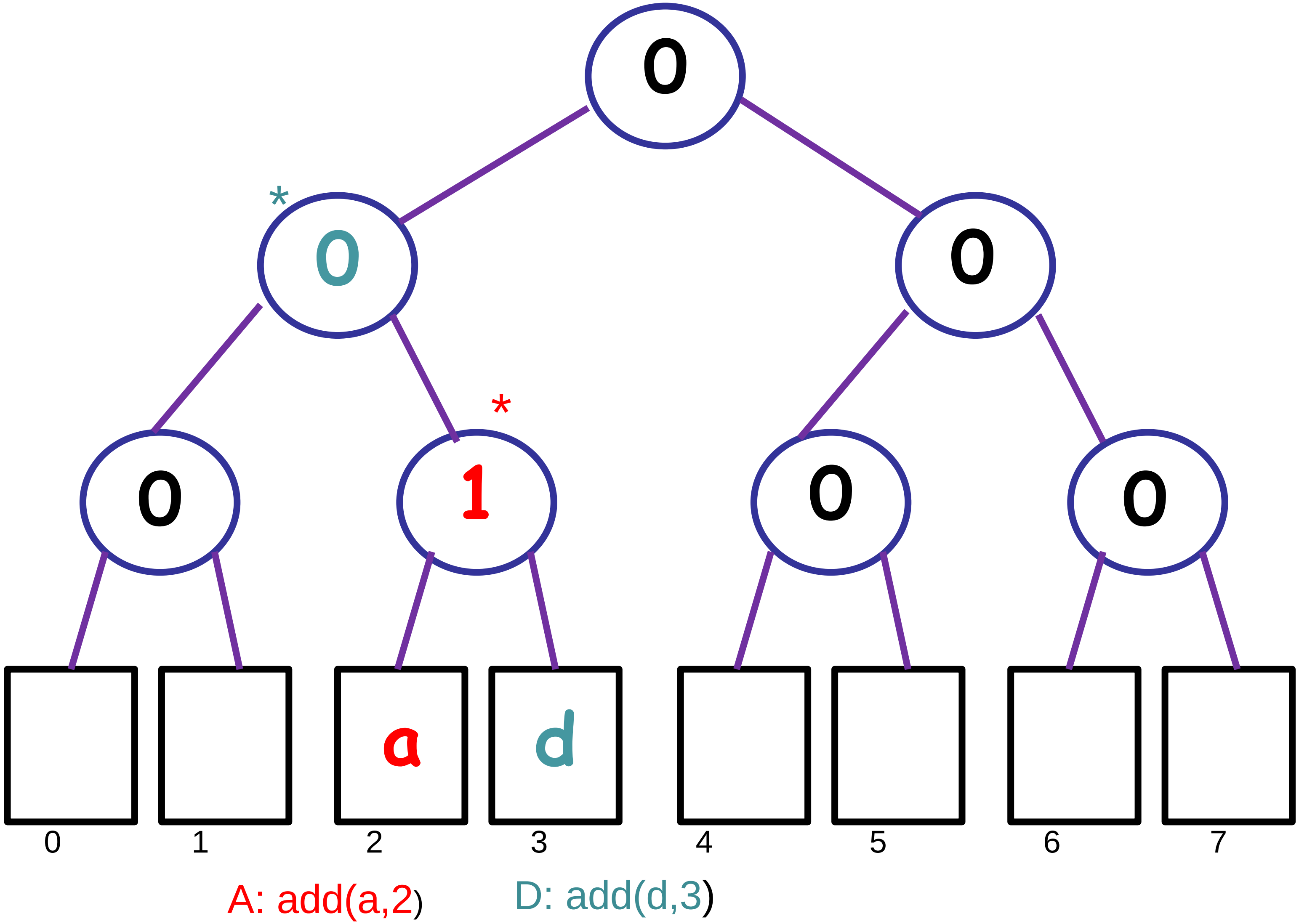
Tree-based bounded P-Q



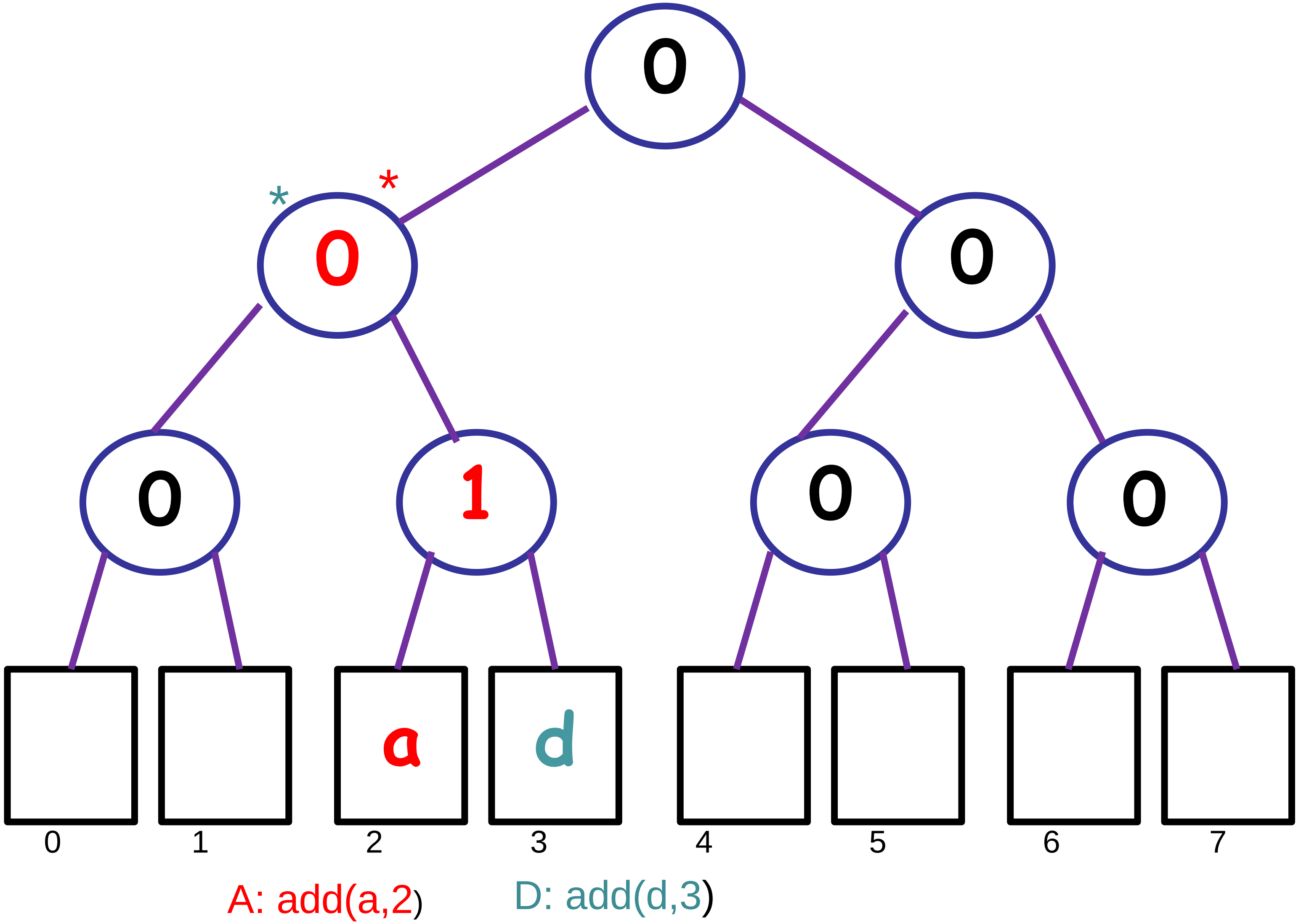
Tree-based bounded P-Q



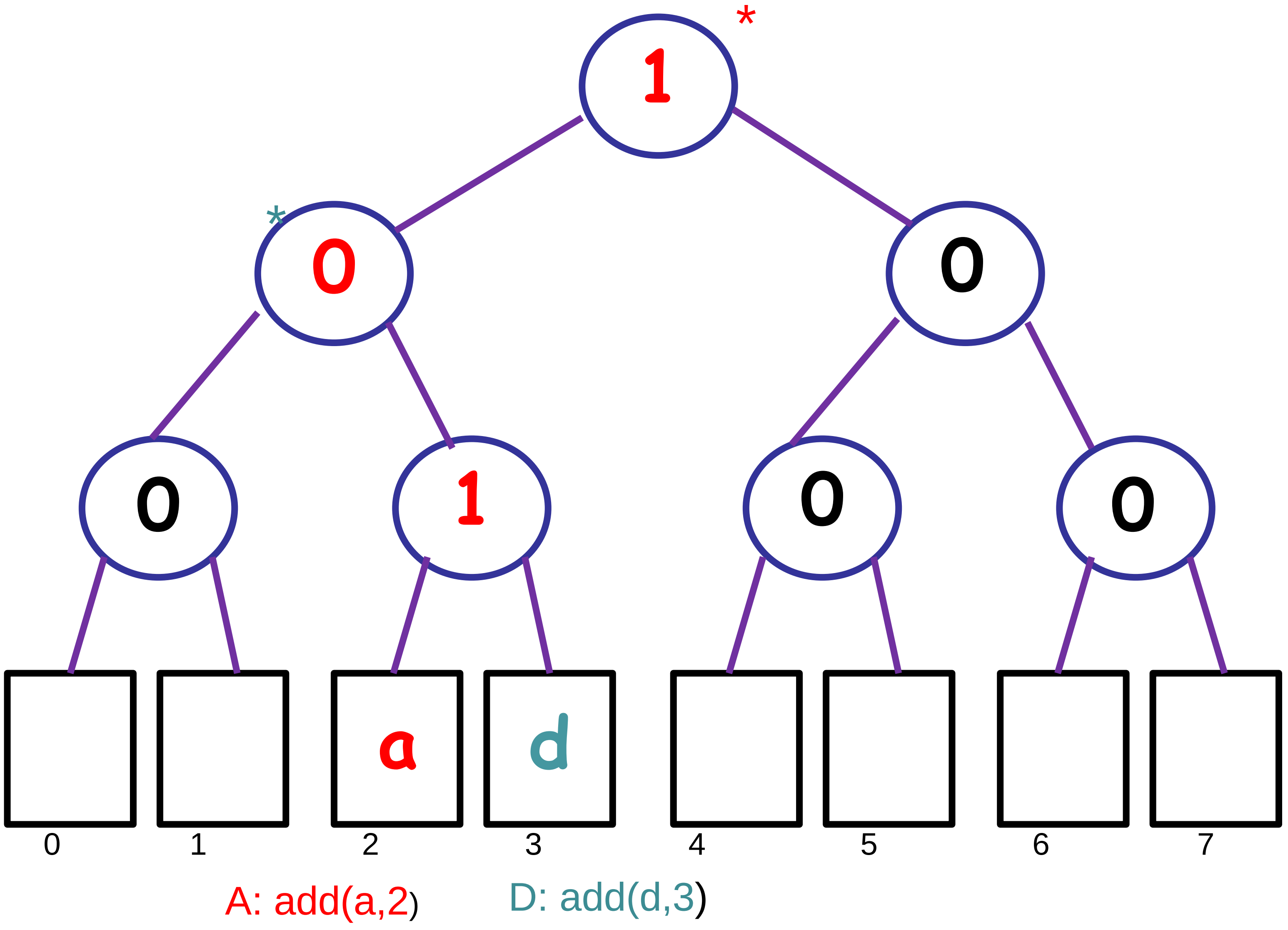
Tree-based bounded P-Q



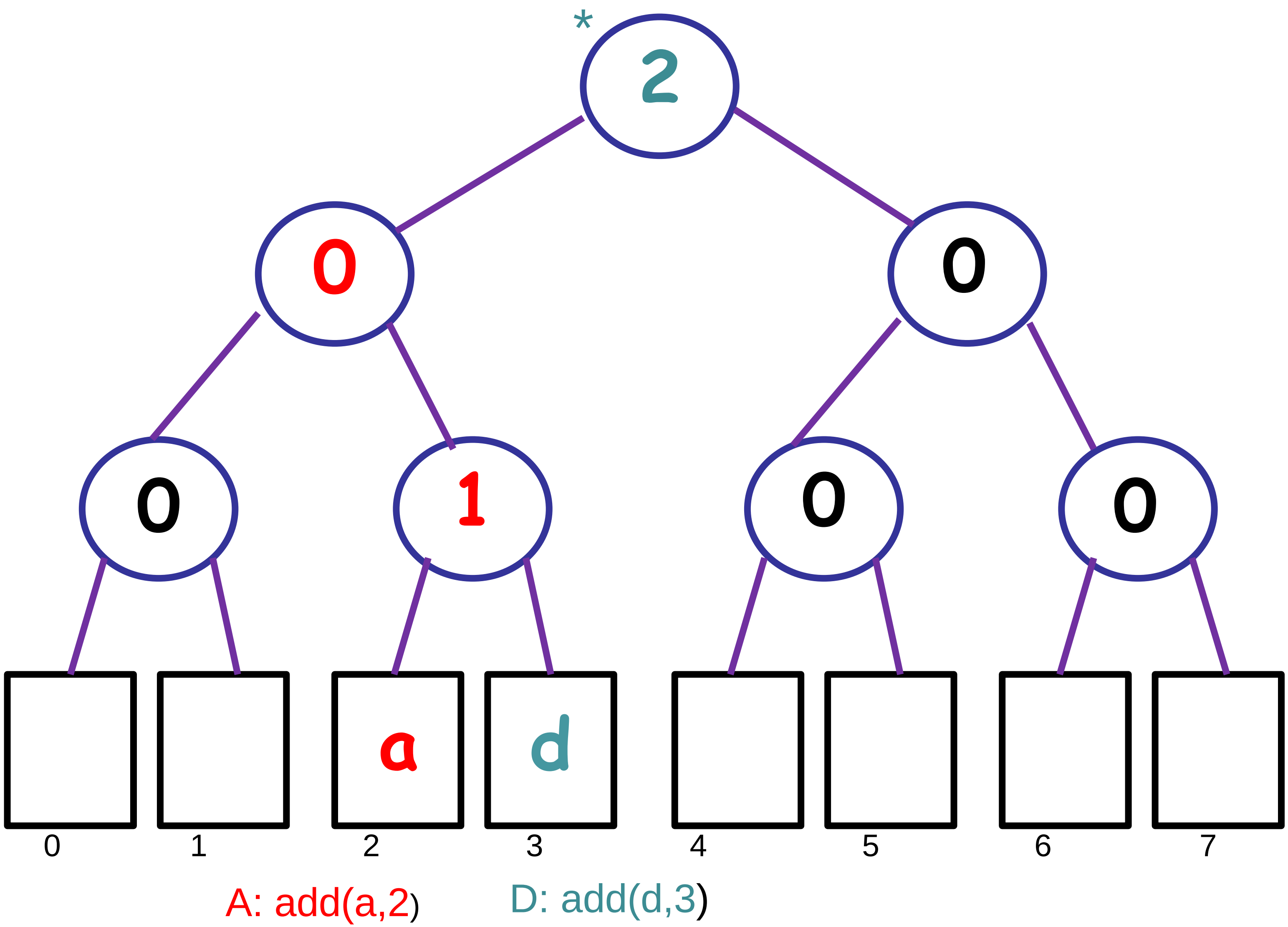
Tree-based bounded P-Q



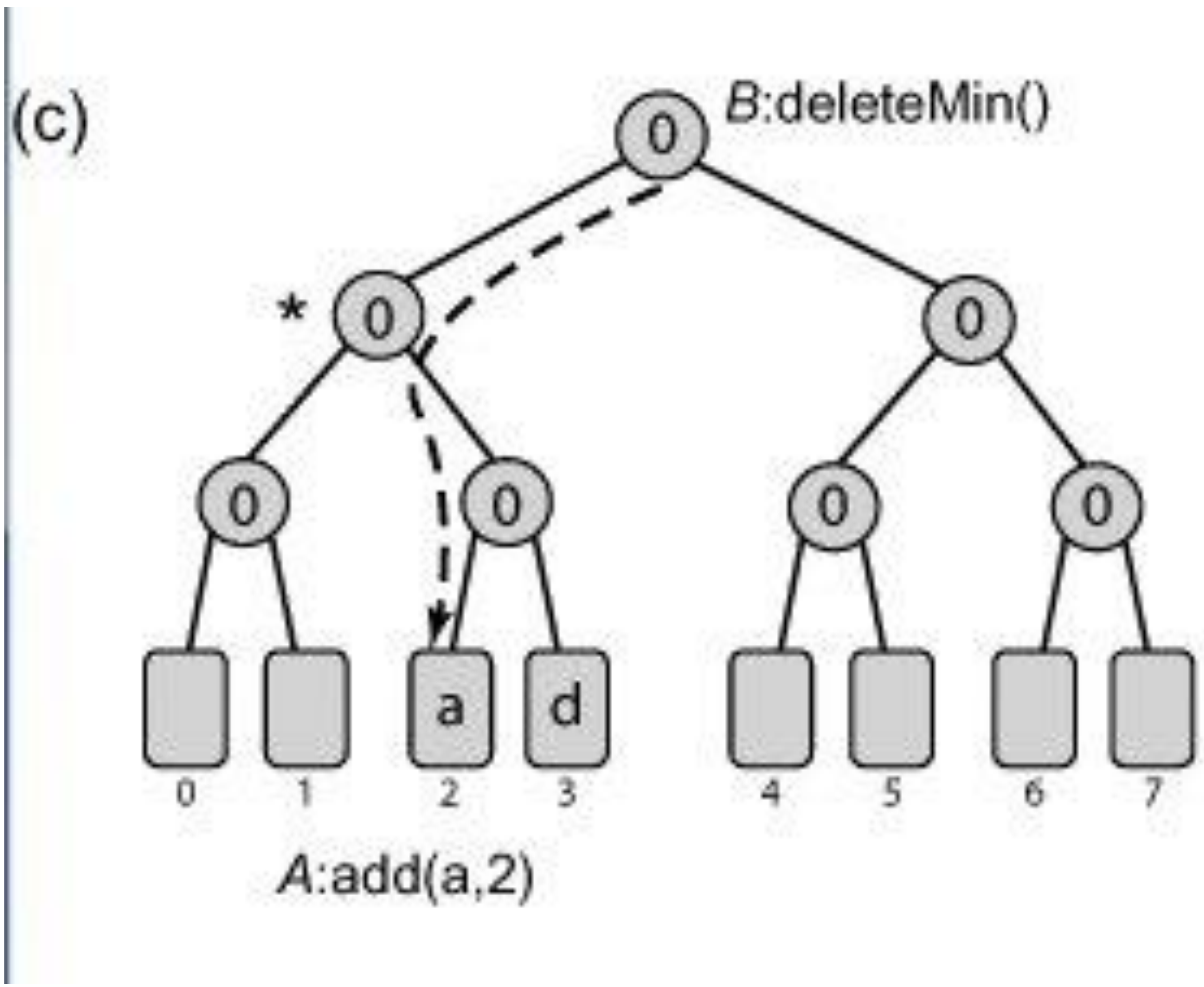
Tree-based bounded P-Q



Tree-based bounded P-Q



Tree-based bounded P-Q

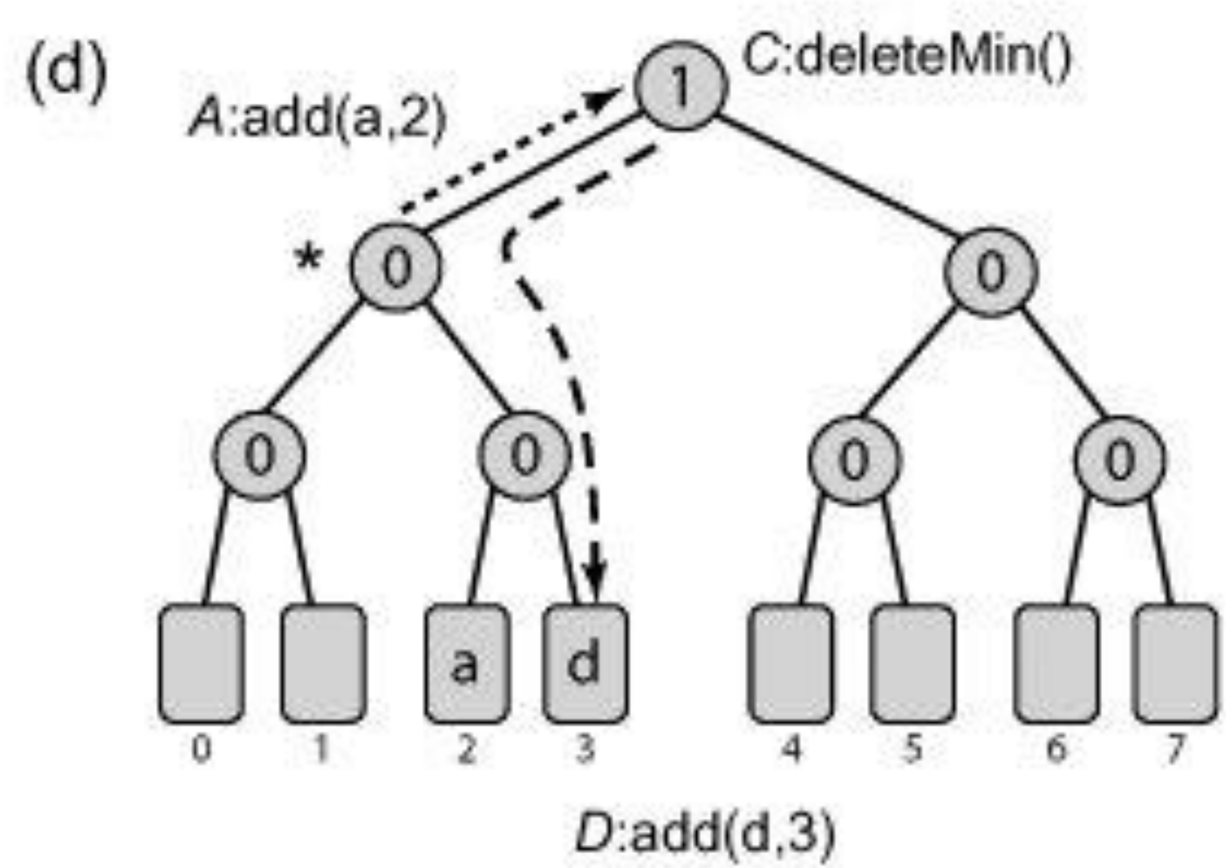


Does it still work concurrently ?

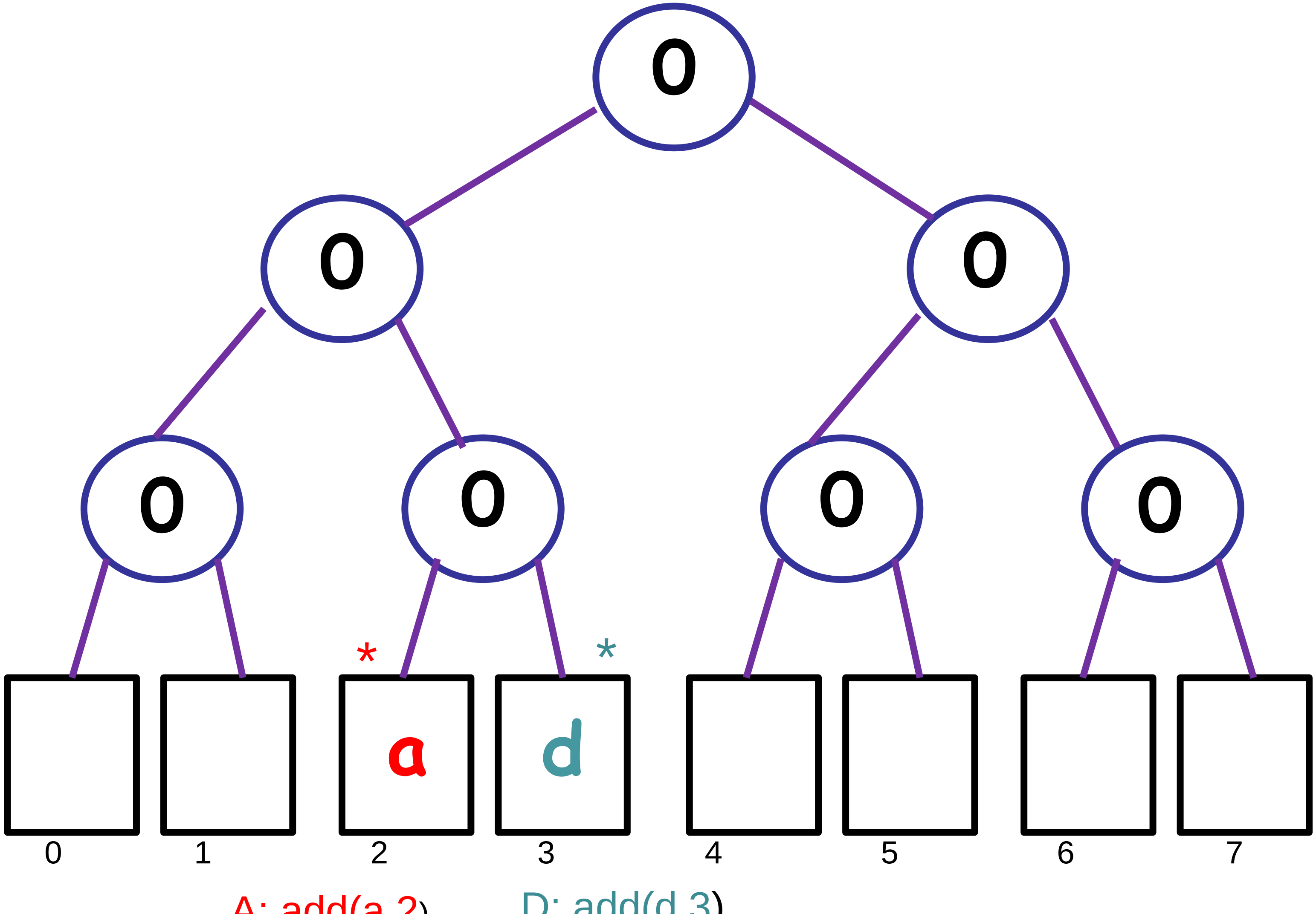
- Lock-free quiescently consistent

- A: add(a,2)
- D: add(d,3)
- B: deleteMin()

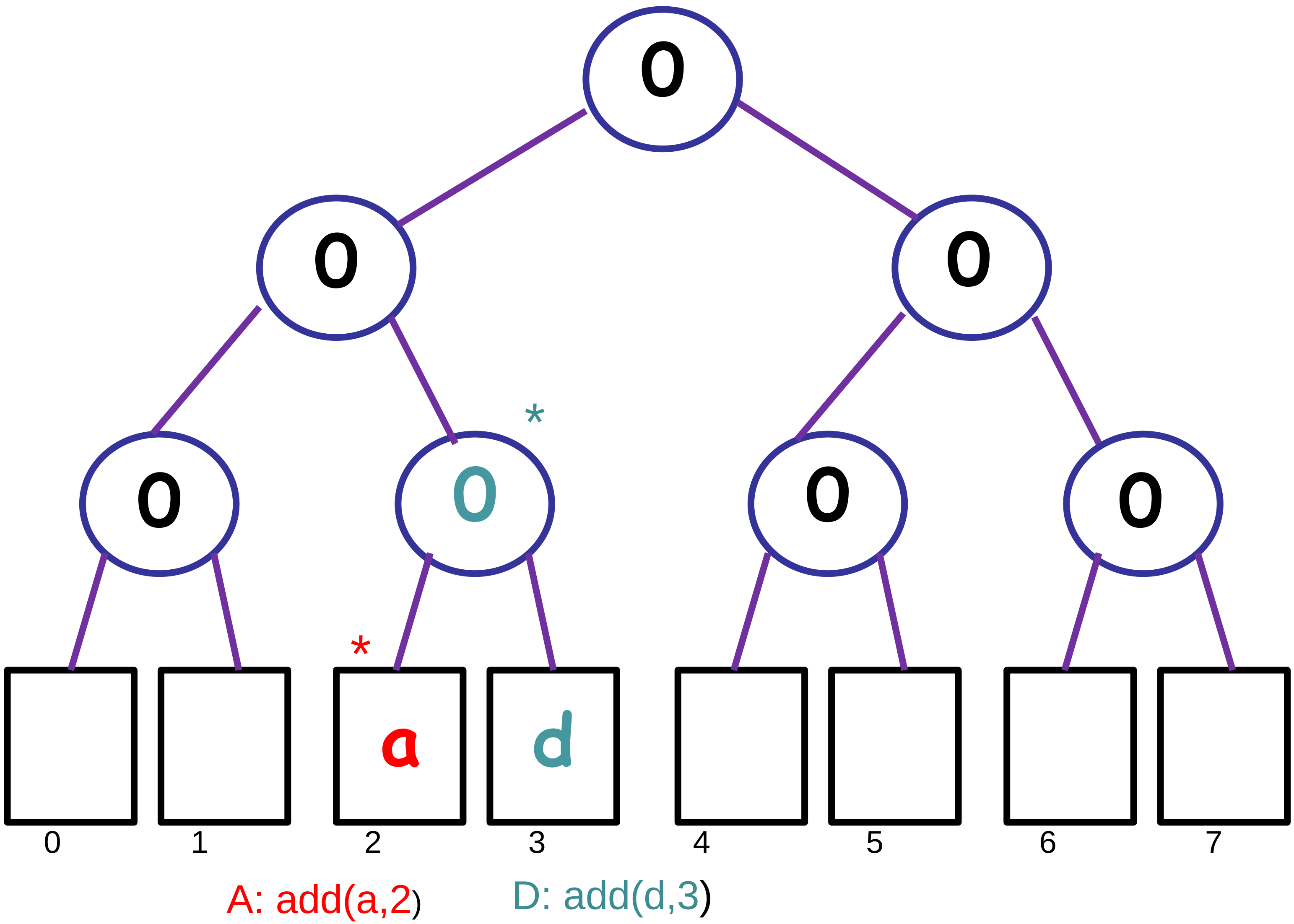
Tree-based bounded P-Q



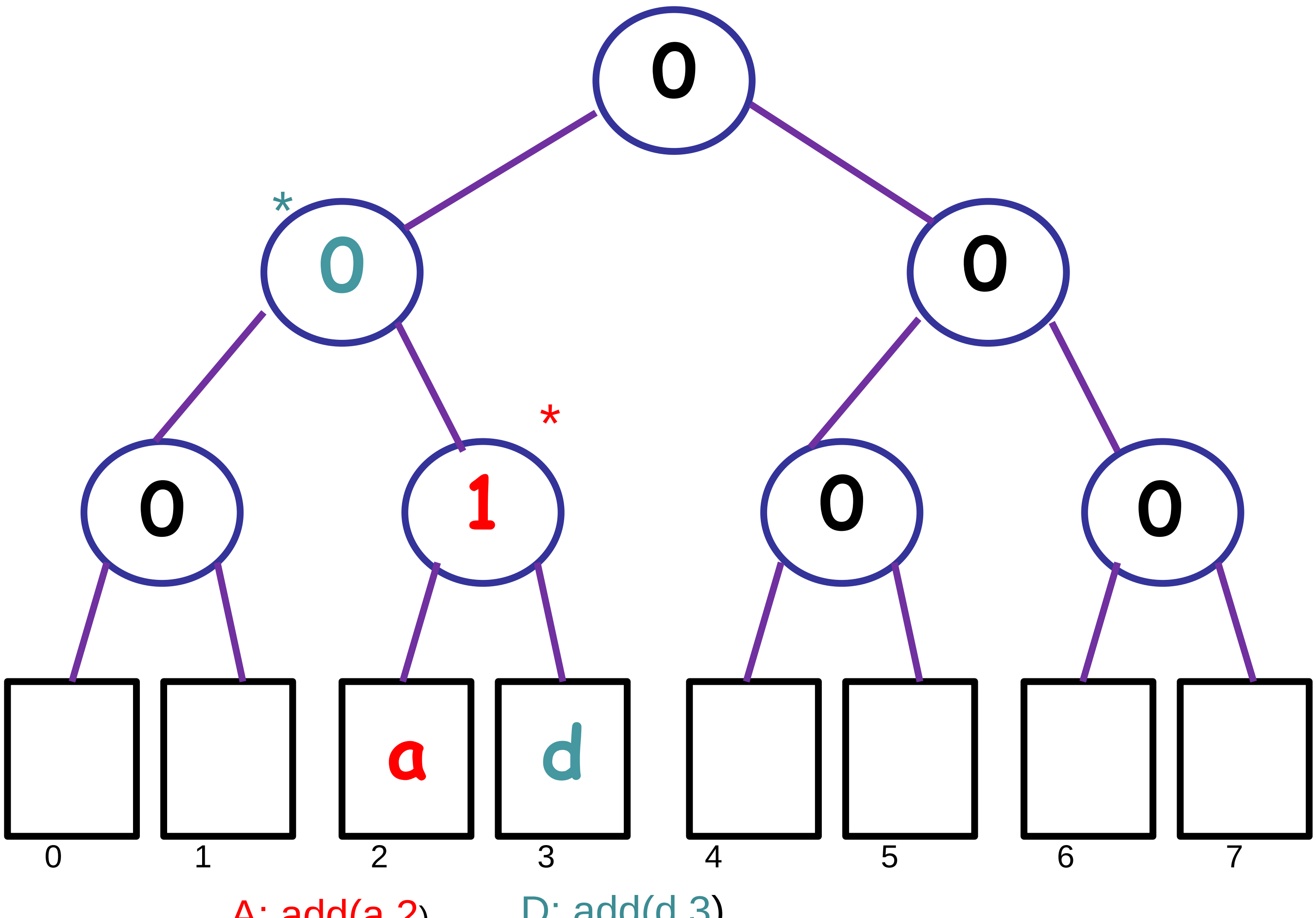
Tree-based bounded P-Q



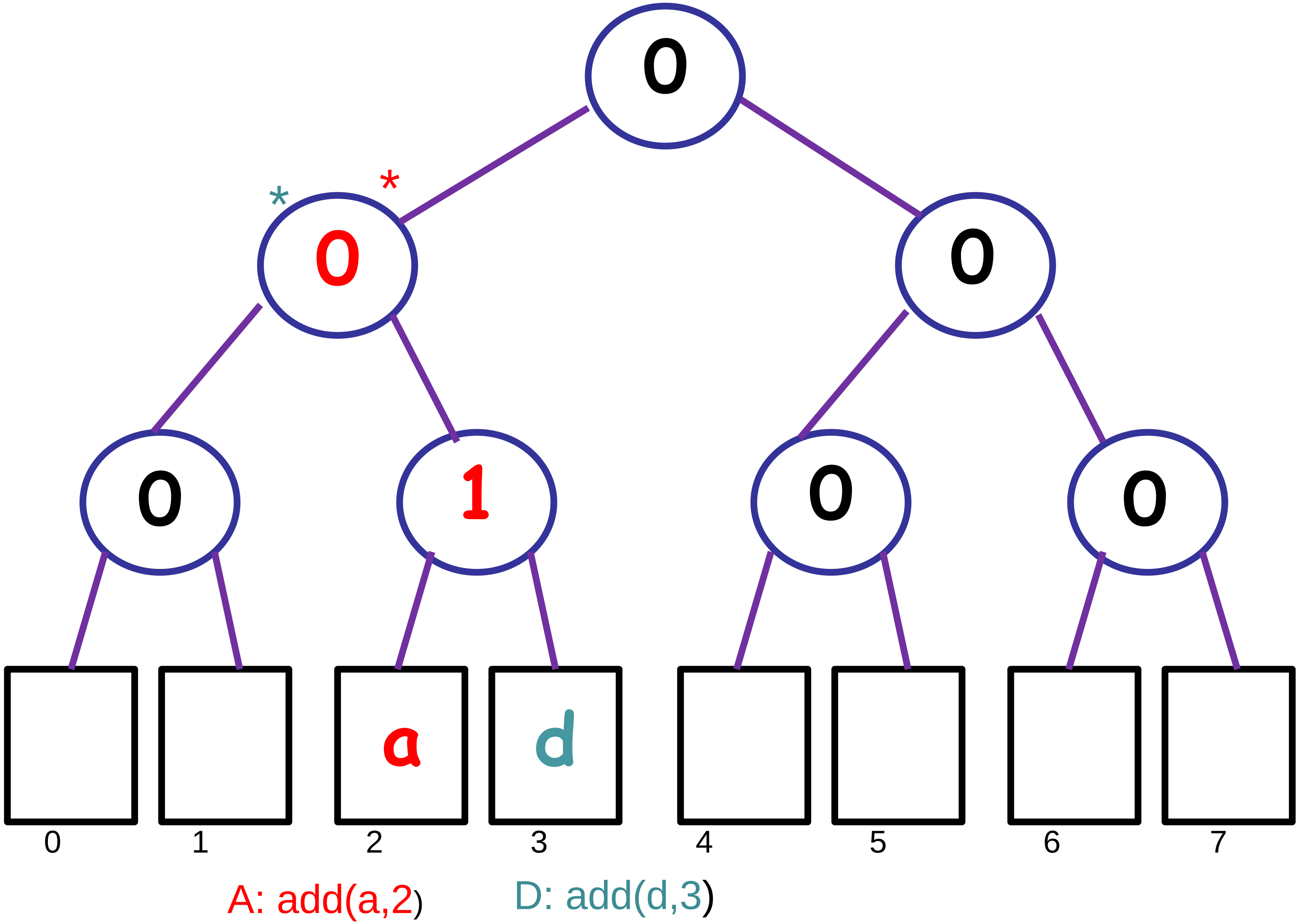
Tree-based bounded P-Q



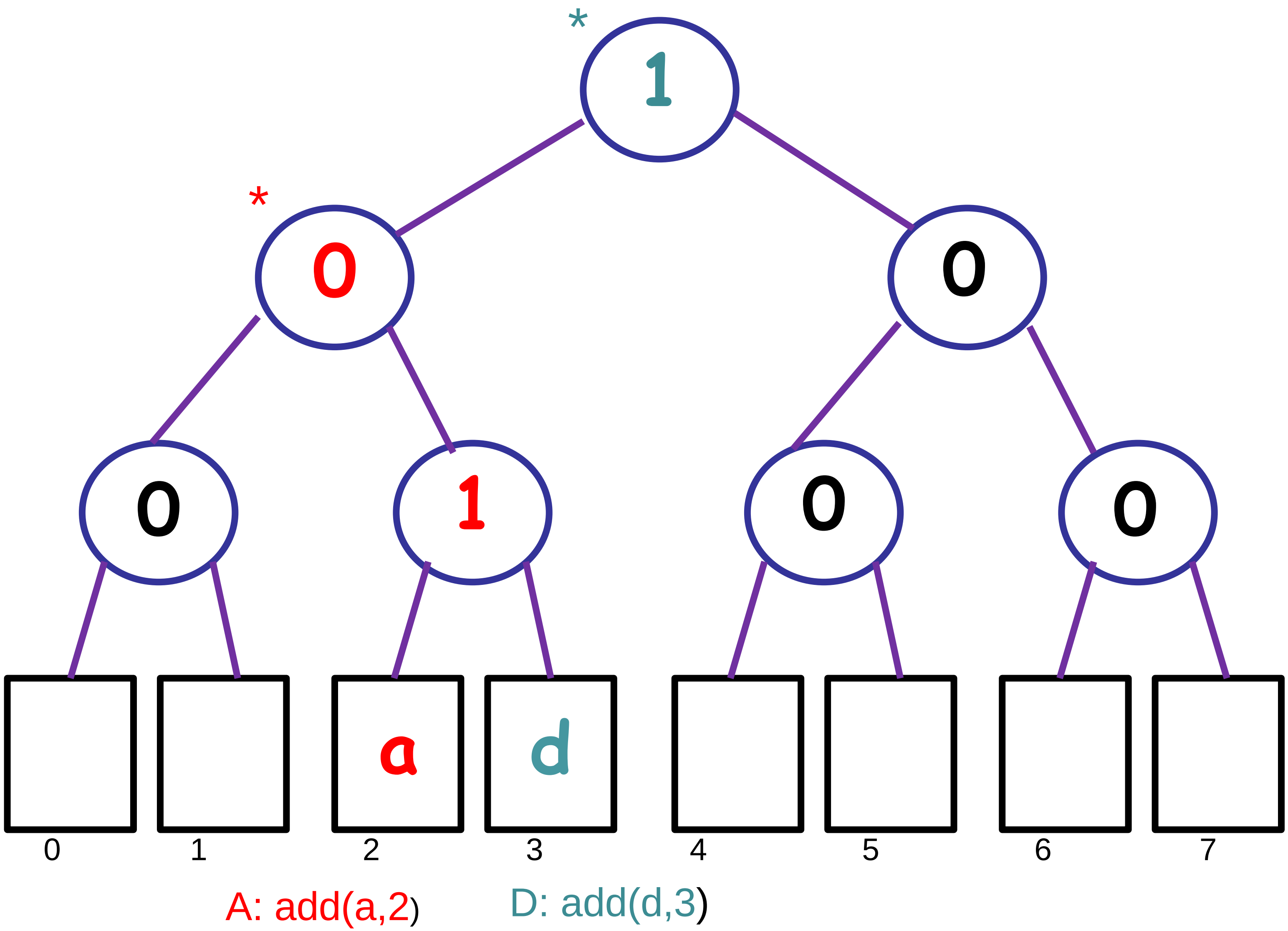
Tree-based bounded P-Q



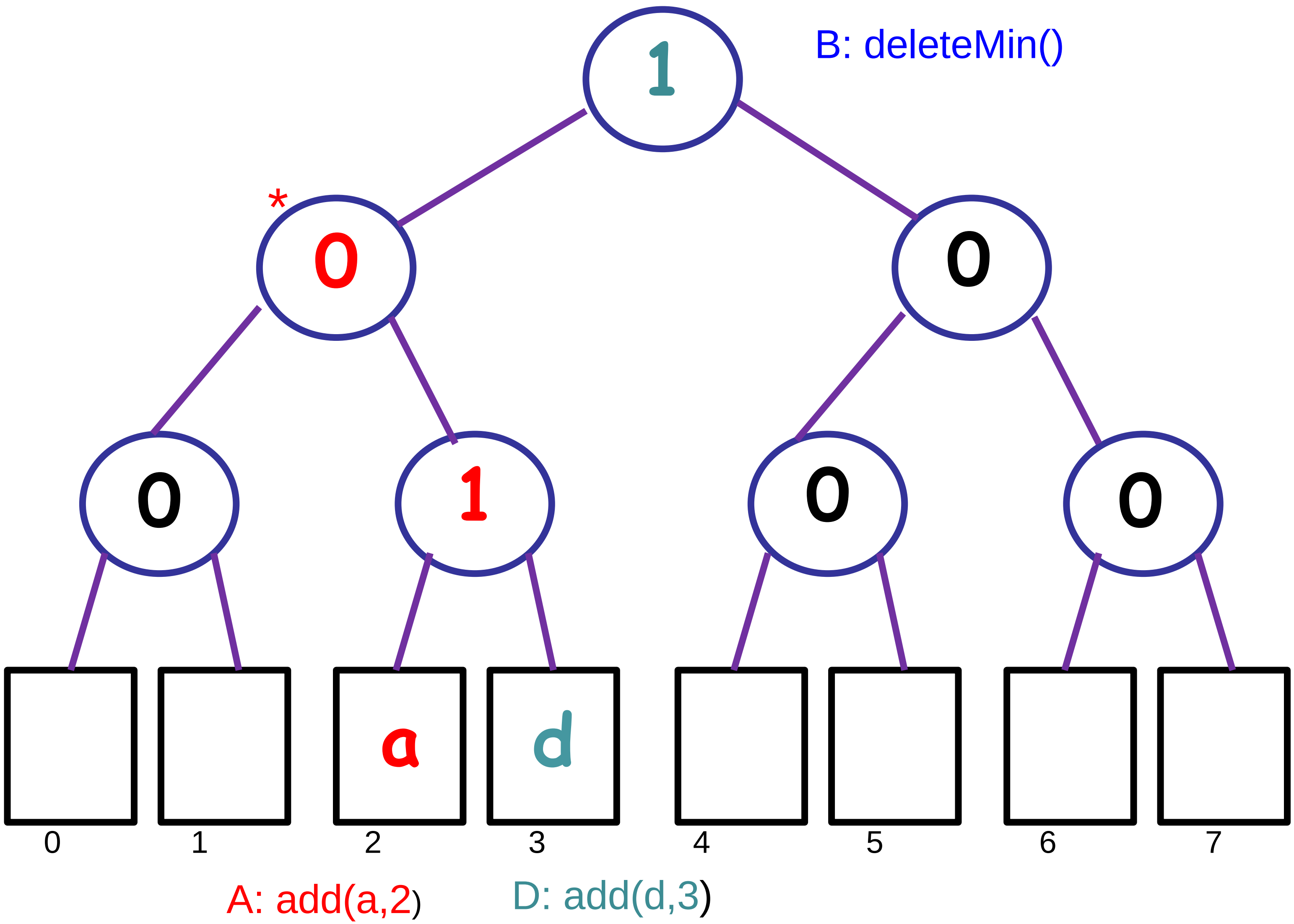
Tree-based bounded P-Q



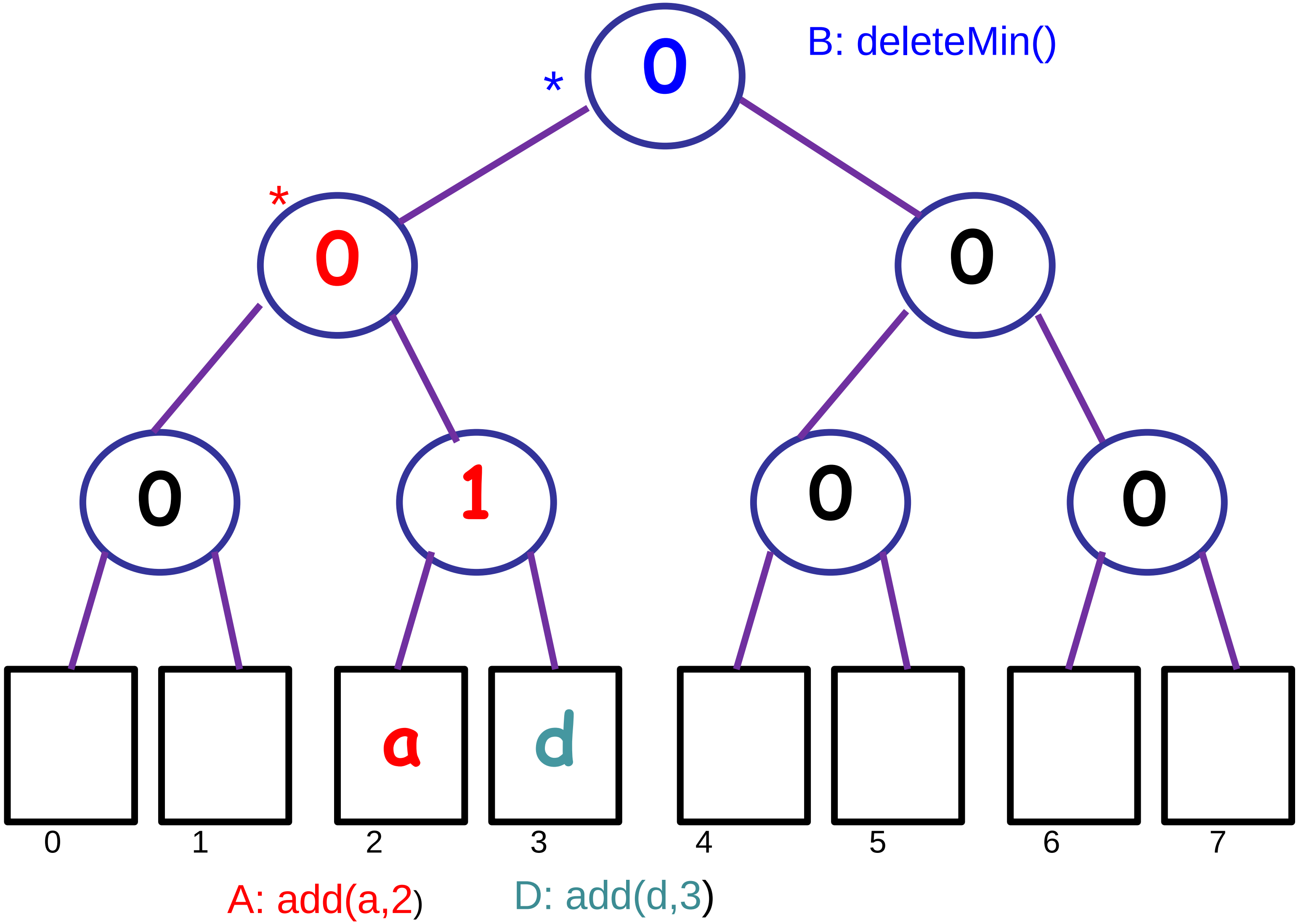
Tree-based bounded P-Q



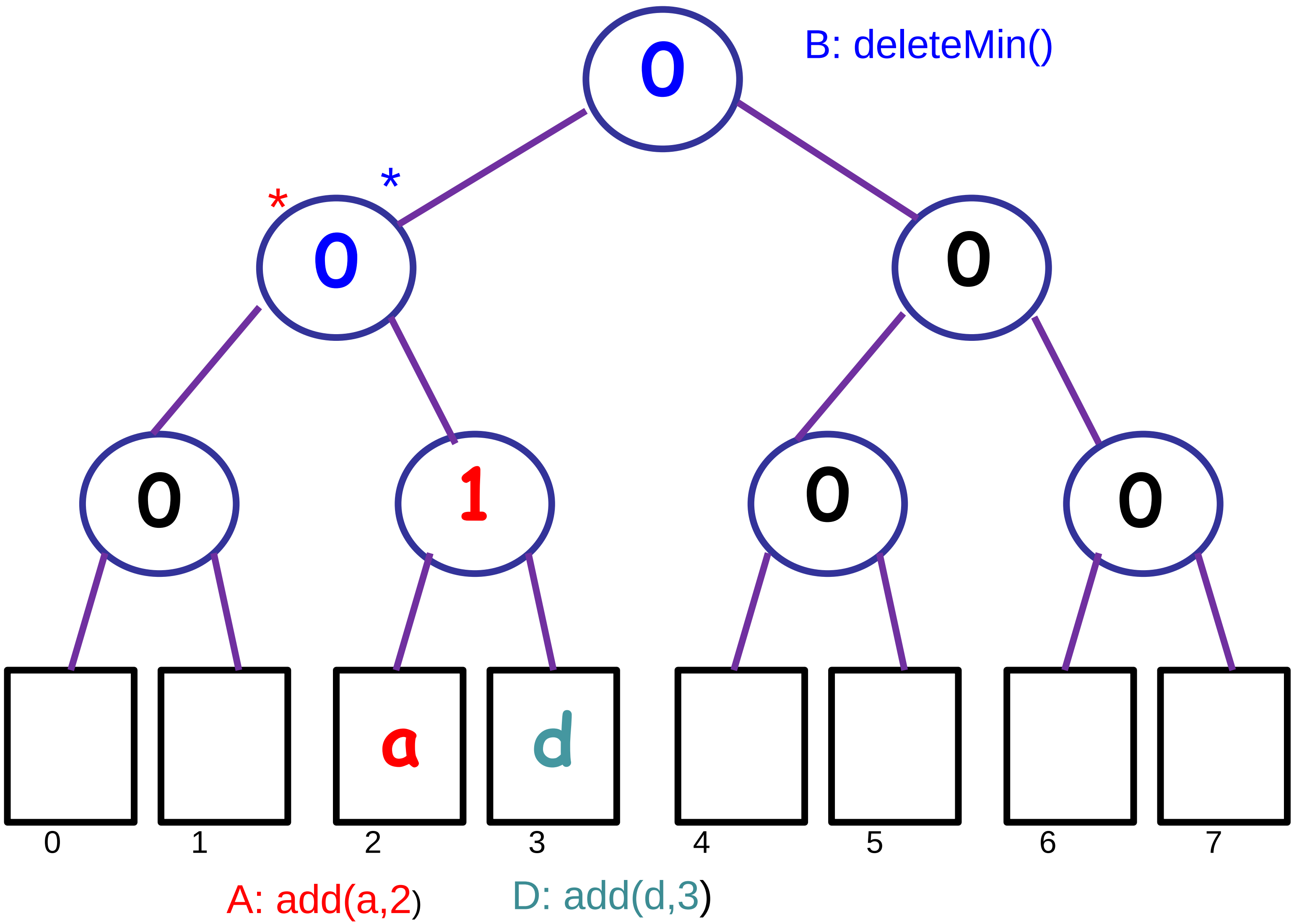
Tree-based bounded P-Q



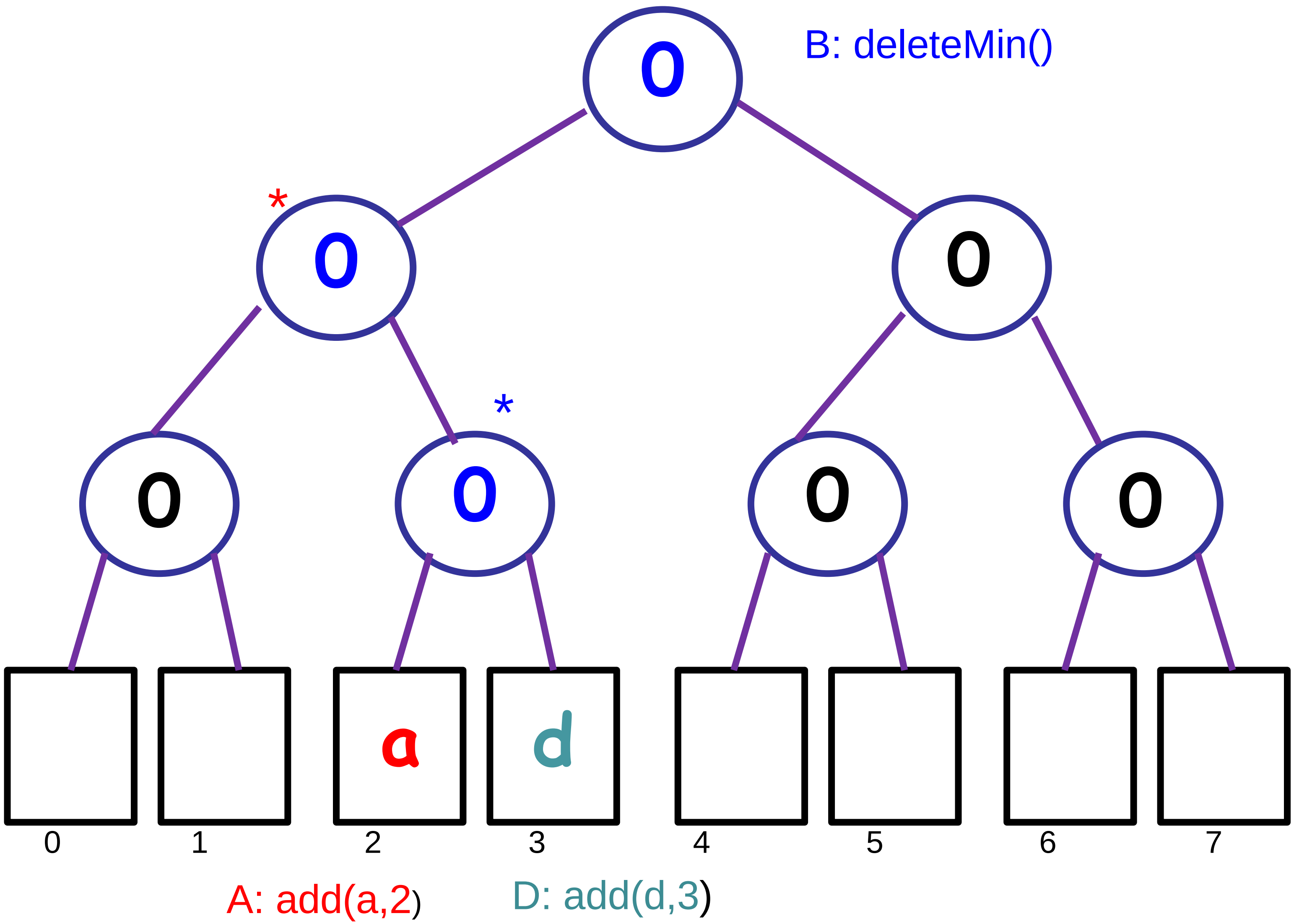
Tree-based bounded P-Q



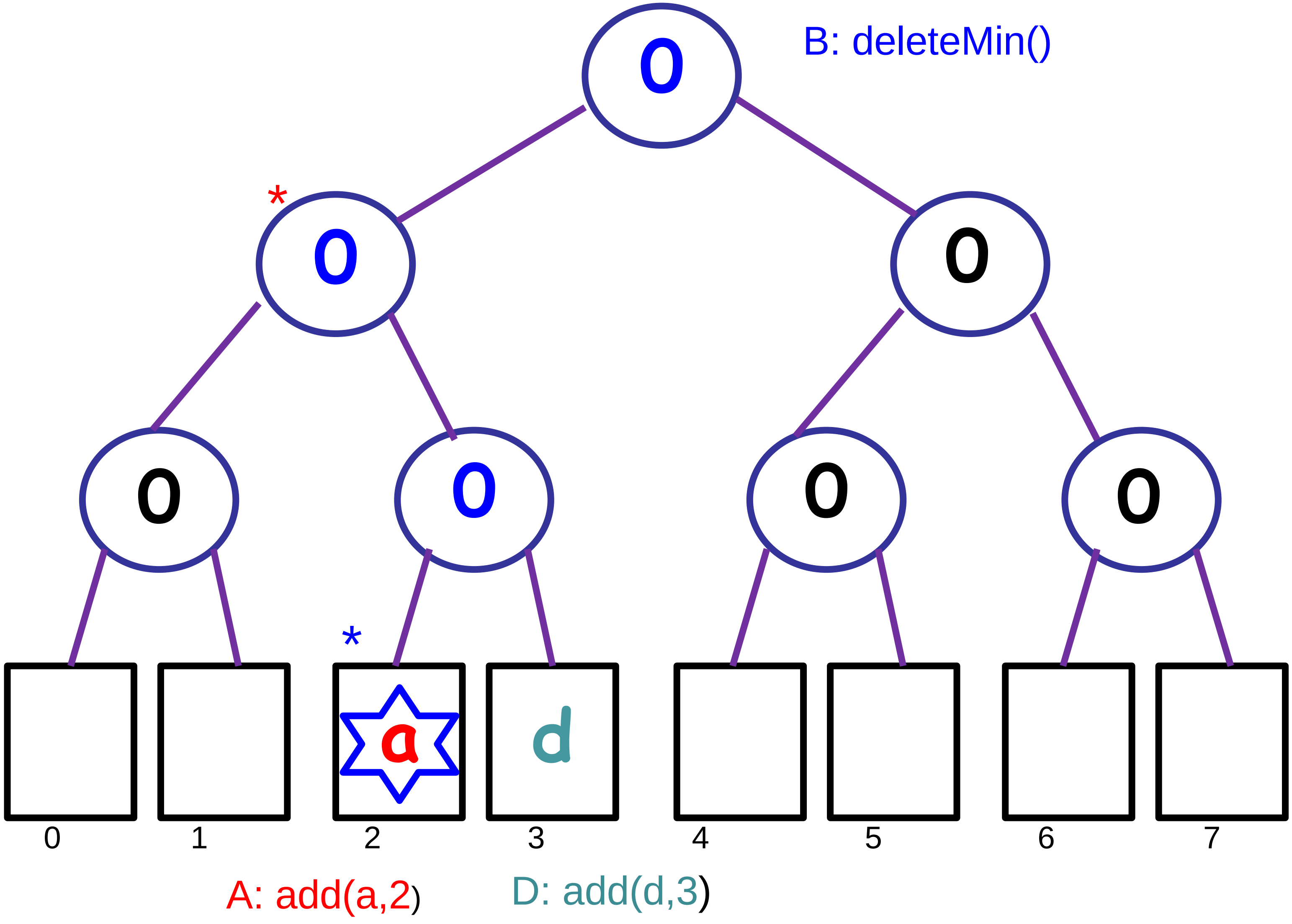
Tree-based bounded P-Q



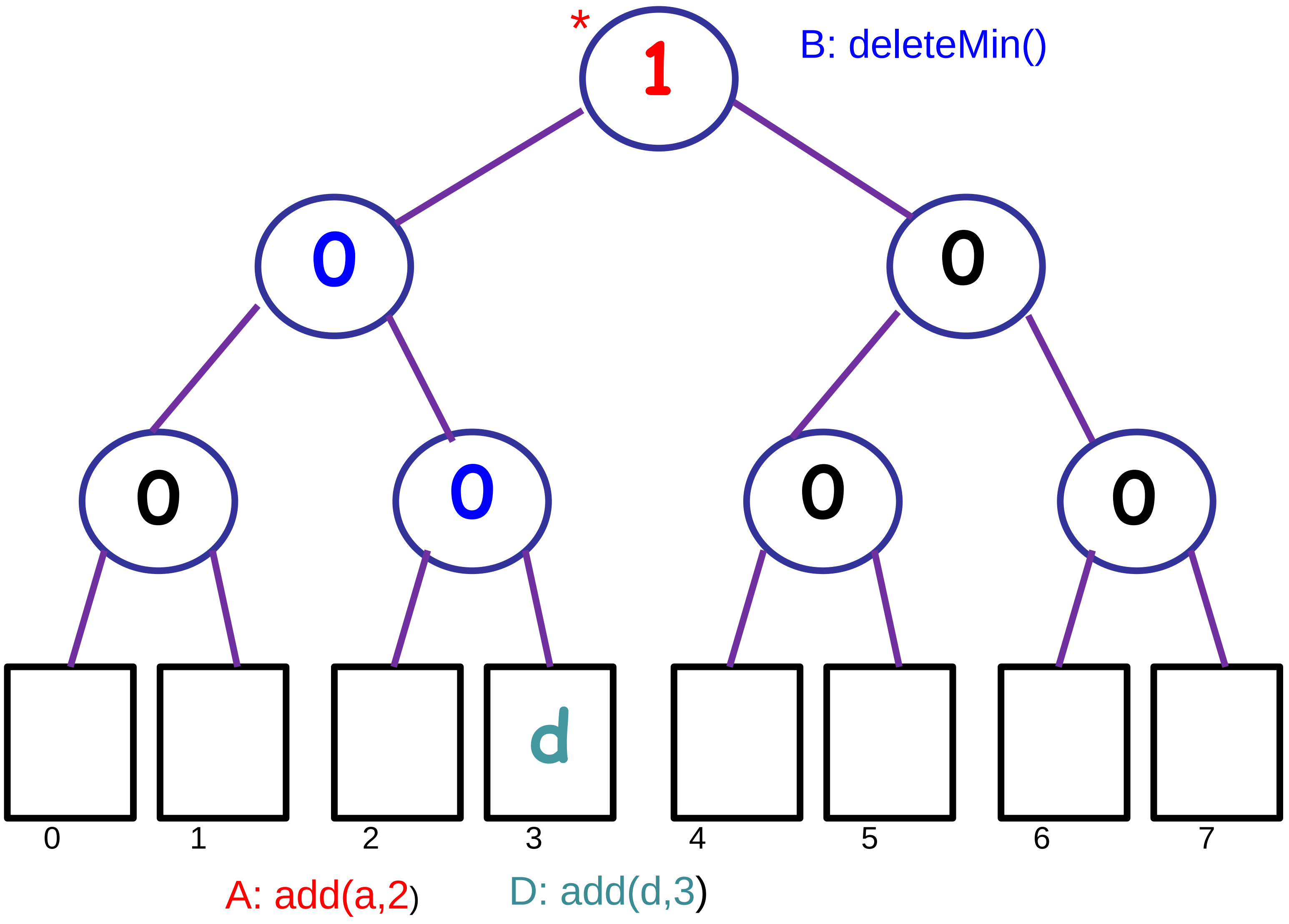
Tree-based bounded P-Q



Tree-based bounded P-Q



Tree-based bounded P-Q



Add()

- Add(x,k)
  - ads x to the bin at the kth leaf
  - Increment node counters in leaf-to-root order

RemoveMin()

- Traverse the tree from root-to-leaf order
- Finds the leaf with highest priority whose bin is not empty
- At each node, if the counter is zero it goes to the right
- Otherwise, decrement the counter and goes to the left

Tree-based bounded P-Q

- It's not linearizable
  - Threads traversing the tree may overtake other thread
- Add() and removeMin() are lock-free
  - If the bins and counters are lock-free
  - Both takes finite steps (bounded by tree depth)

Heap (sequential)

- a complete binary tree with nodes whose priority is greater than all its children's
- removeMin()
  - removes and returns the root of the tree
  - rebalances (root to leaf)
- Add()
  - appends the item at the end of the list
  - rebalances (leaf to root)

Concurrent Heap

- For concurrency
  - Both add() and removeMin() rebalnaces as a sequence of atomic steps to be interleaved
- heaplock
  - for removing the root
- heapnode
  - lock
  - status
    - EMPTY, AVAILABLE, BUSY

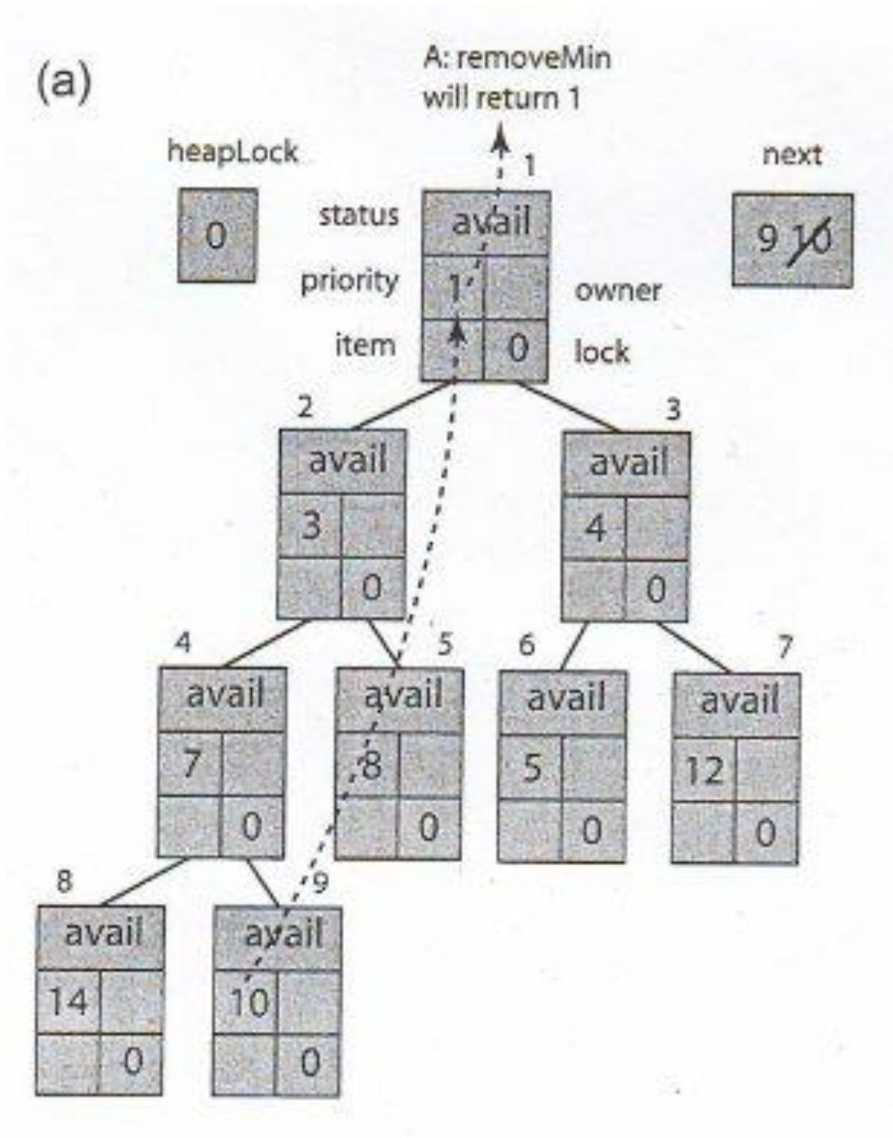
Concurrent Heap

- removeMin()
  - Acquires heapLock, decrements the next, locks top & bottom and releases heapLock
  - Get the top value, swaps top & bottom, mark the bottom Empty and unlocks it
  - The top is percolated down holding the lock
  - When we swap, we lock both
- add()

Concurrent Heap

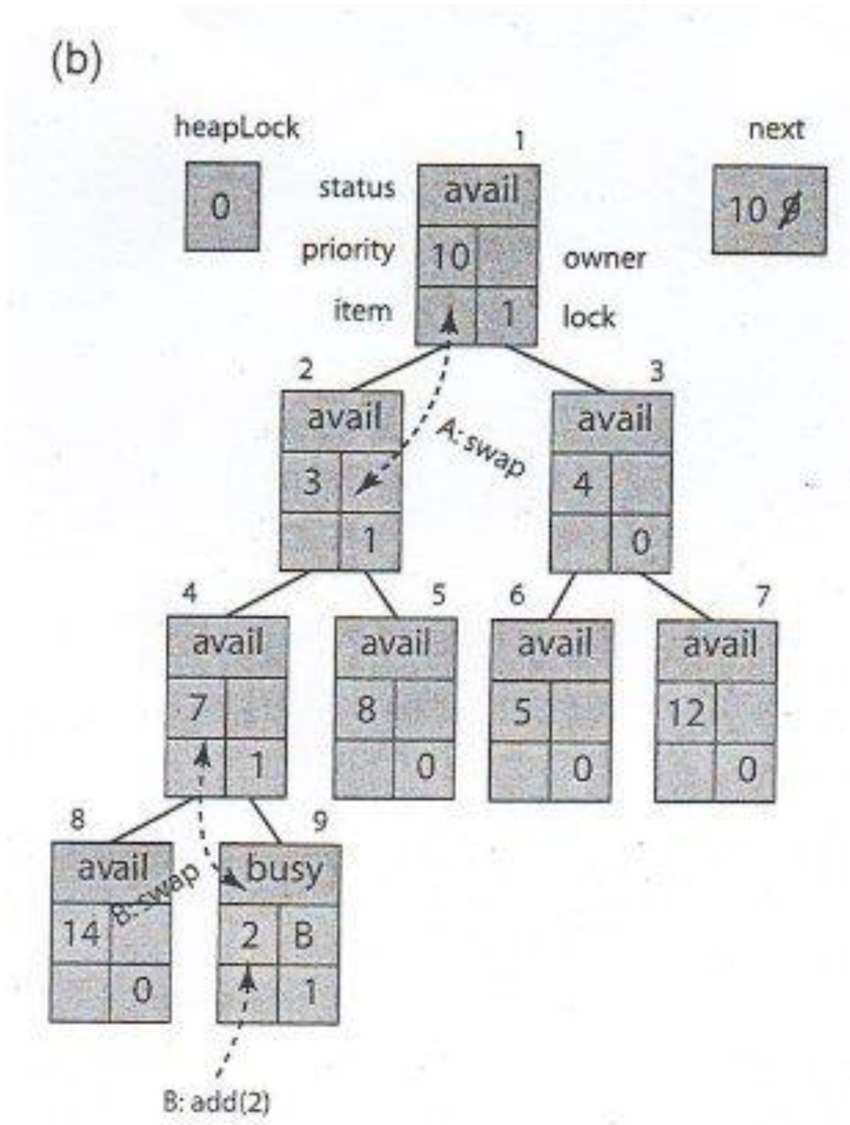
- add()
  - Acquires heapLock, increments the next, locks and initialize the child(Busy, owner), and releases heapLock, child lock
  - The child is percolated up the tree
  - It locks the parent and the child
  - If parent is Available and child is owned by the caller, has high priority, then swap

A: removeMin()



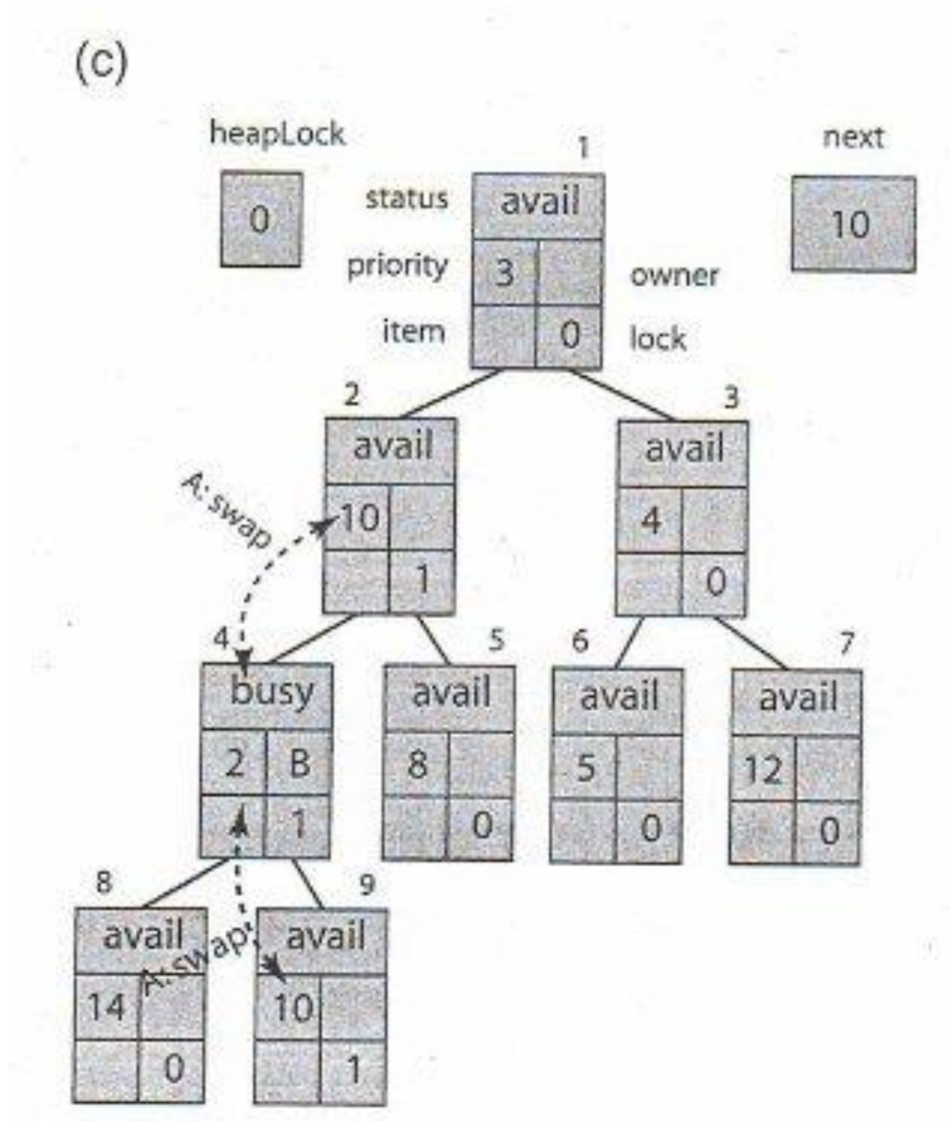
(a)
A: removeMin
will return 1
heapLock
0
next
9 10
1
status
priority
item
owner
lock
avail
1
0
2
3
avail
3
0
avail
4
0
4
5
6
7
avail
7
0
avail
8
0
avail
5
0
avail
12
0
8
9
avail
14
0
avail
10
0

B: add(2)

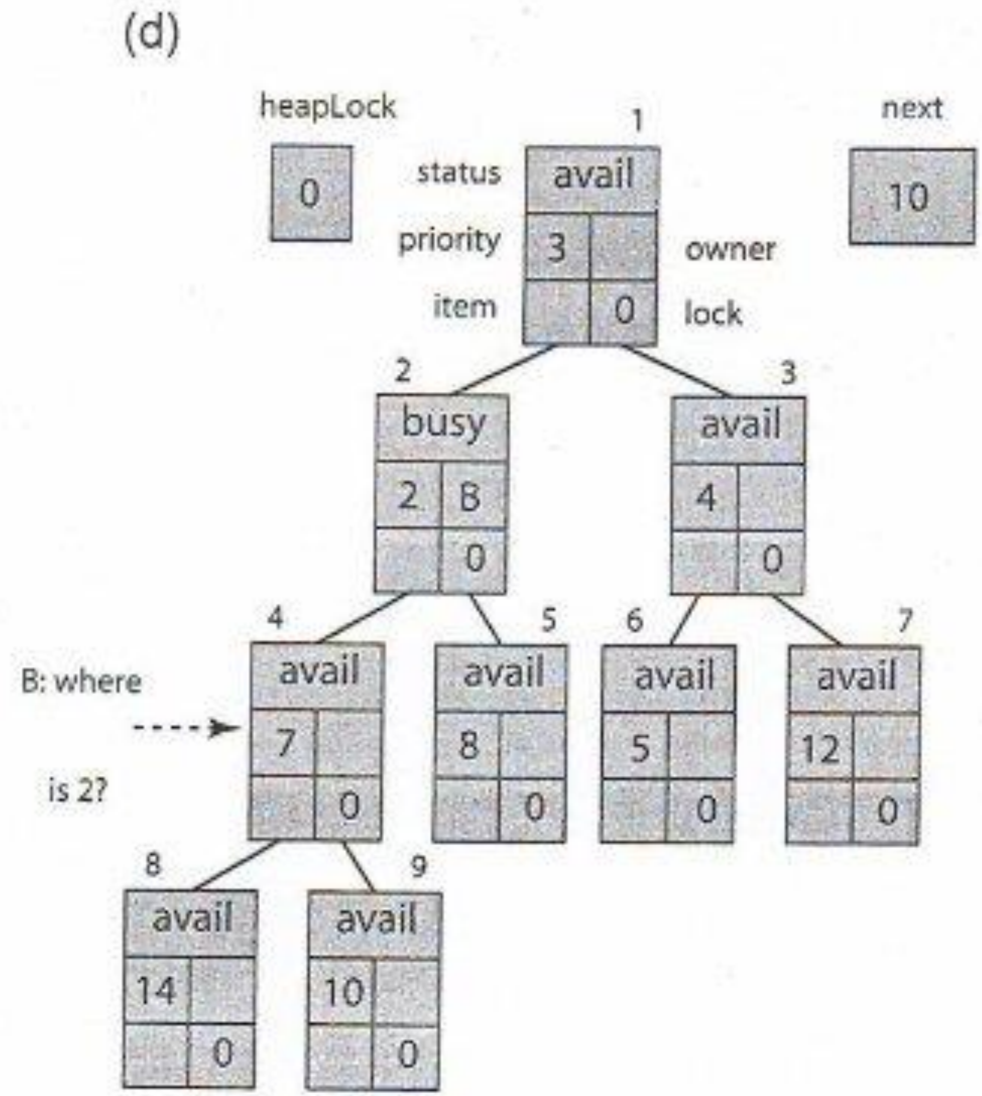


(b)
heapLock
0
1
status
avail
priority
10
owner
item
1
lock
next
10 ∅
2
avail
3
1
A: swap
3
avail
4
0
4
avail
7
1
5
avail
8
0
6
avail
5
0
7
avail
12
0
8
avail
14
0
B: swap
9
busy
2
B
1
B: add(2)

A: percolates down(10)



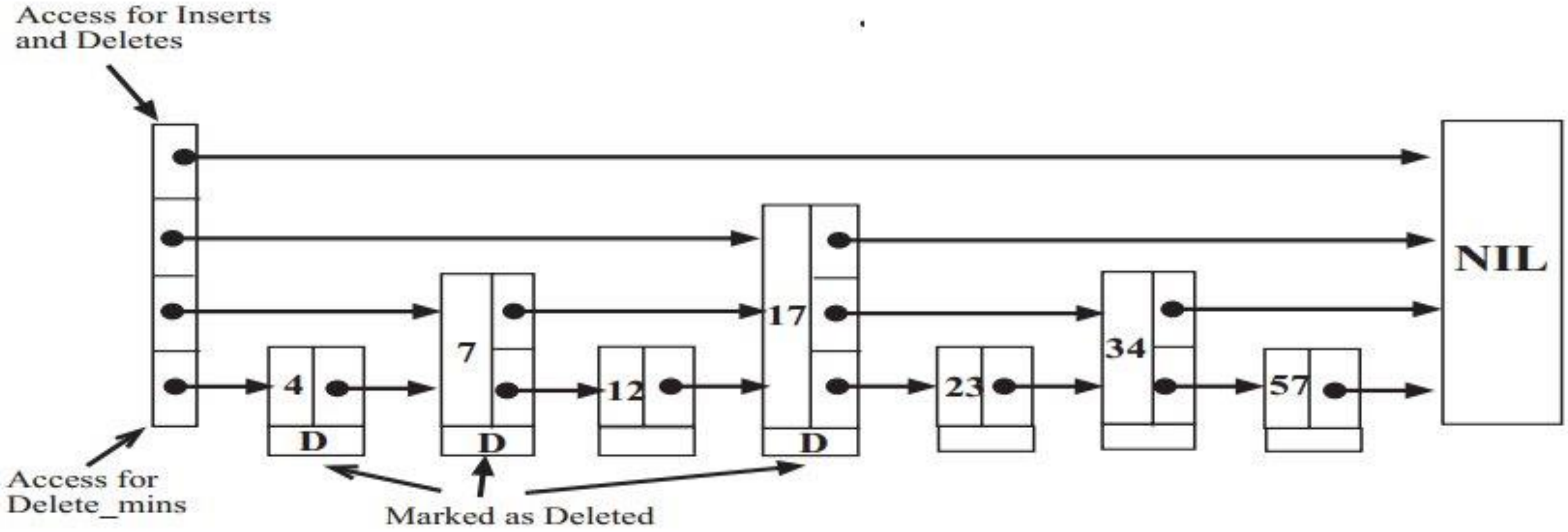
B: looking for 2 to move up



Skiplist-based unbounded P-Q

- No rebalancing is required !
- PrioritySkipList
  - sorted by priority, highest in the front
  - removing is done lazily, findAndMarkMin()
- remove()
  - Physical remove
  - Logarithmic time
- add()

Skiplist-based unbounded P-Q



Skiplist-based unbounded P-Q

- Quiescently consistent, Not linearizable
- Lock-free
- A thread can fail repeatedly if other threads repeatedly succeed
- Contention
  - Multiple threads traverse together
  - Physical removing (neighbors, probably)
- Usually performs better than heap-based priority queue