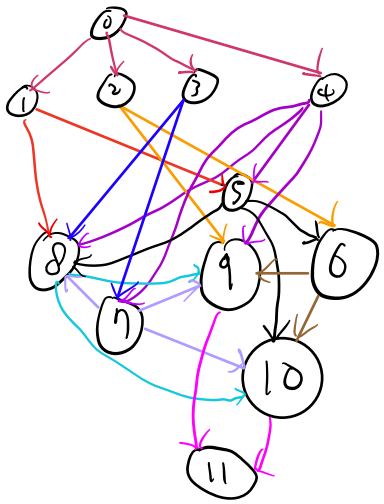
$n = \{1, \ldots, N\}$

N = total # of nodes

nO = nth node operation



lastnode = 11

q := heap()

Processed = { }

q = [0]

Processed = {0 : x}

q = [1, 2, 3, 4]

processed = {0:x, 1: x|20}

q = [2,3,4,8,5]

processed = {0:x, 1: x|20, 2: x|20}

q = [3,4,8,5,9,6]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30}

q = [4,8,5,9,6]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40}

q = [8,5,9,6,7,9]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40, 5:x|50|40∩10}

q = [8,9,6,7,9,10]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40, 5:x|50|40∩10,
6: x|60|(50|40∩10) ∩ 20}

q = [8,9,7,10]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40, 5:x|50|40∩10,
6: x|60|(50|40∩10) ∩ 20, 7: x|40 ∩ 30}

q = [8,9,10]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40, 5:x|50|40∩10,
6: x|60|(50|40∩10) ∩ 20, 7: x|40 ∩ 30,
8: x|(70|40∩30) ∩ (50|40∩10) ∩ 40 ∩ 10 ∩ 30}

q = [9,10]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40, 5:x|50|40∩10,
6: x|60|(50|40∩10) ∩ 20, 7: x|40 ∩ 30,
8: x|(70|40∩30) ∩ (50|40∩10) ∩ 40 ∩ 30 ∩ 10,
9: x|(80|…) ∩ (70|…) ∩ (60|…) ∩ 40 ∩ 20}

q = [10, 11]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40, 5:x|50|40∩10,
6: x|60|(50|40∩10) ∩ 20, 7: x|40 ∩ 30,
8: x|(70|40∩30) ∩ (50|40∩10) ∩ 40 ∩ 30 ∩ 10,
9: x|(80|…) ∩ (70|…) ∩ (60|…) ∩ 40 ∩ 20,
10: x|(80|…) ∩ (70|…) ∩ (60|…) ∩ (50|…)}

q = [11]

processed = {0:x, 1: x|20, 2: x|20, 3: x|30, 4:x|40, 5:x|50|40∩10,
6: x|60|(50|40∩10) ∩ 20, 7: x|40 ∩ 30,
8: x|(70|40∩30) ∩ (50|40∩10) ∩ 40 ∩ 30 ∩ 10,
9: x|(80|…) ∩ (70|…) ∩ (60|…) ∩ 40 ∩ 20,
10: x|(80|…) ∩ (70|…) ∩ (60|…) ∩ (50|…),
11: x|(100|…) ∩ (40|…)}

q = [ ]

return processed[lastnode]

time complexity : checks all the elements and the two extra nodes.
$O(N+2) \Rightarrow O(N)$

space complexity : stores tensor for each node and the two extra nodes.
$O(N) \Rightarrow O(N) \Rightarrow O(NT)$

each tensor has
$B \times L \times W \times H = T$ data types

M = Module dict of Node operations for each node

N = total # of nodes, lastnode = final node

Pseudocode: nodes = [all nodes including extra nodes], x = input tensor

```
Initialize M{ node : NodeOperation(indegee of node, in channel, out channel)
                for node in nodes}

Initialize h = heap
Initialize processed = dict{}

q.push(0)
while q not empty:
    node = q.pop()
    in_nodes = node.get_in_nodes()                    → prep in_node dict for each node
    processed[node] = M.at(node)[concat(in_nodes)]
    neighbors = node.getneighbors()
    for neighbor in neighbors:
        if neighbor not in q:
            q.push(neighbor)

return processed[lastnode]
```