(3)

n = {1,..., N} No total H of nodes no= nth note operation

lastnode= 11 9:= heap()

q = (0) 9 = [1, 2,3,4]

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

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

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

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

9 = (8,9,7,10) 9=[8,9,10]

9=(9,10)

9=[10,11]

9=[11]

4-[]

Processed = { 3 Processed = {o; x}

processed={0:x, 1: x | 10}

processed = {0: x, 1: x(10, 2: 2/20}

processed = {0: x, 1: x 120, 2: 2 120, 3: 2 130} processed = {0:x,1:x120,2;2120,3;2130,4:x140}

processed = {0: x, 1: x 120, 2: x 120, 3: x 130, 4: x 140, 5: x 150 140 10}

processed = {0:x, 1: x(10,2; x(20,3; x(30,4;x(40,5;x(50)40)10) 6: x1601(50140010) 1 20}

processed = {0:x, 1: x(10,2: x(20,3: x(30,4:x(40,5:x)50)40)10, 6: 2(601/50140010) 1 20,7: 21 40 1 70}

processed = {0: x, 1: x[10,2: x[20,3: 2[30,4:x[40,5:x[50]40]]] 6:x(60(50140010) 1 20,7:x140 1 30,

8:x/(nol 40n30) n (50 (40n10) n 40 n10 n 30} processed = {0:x, 1: x(10,2; x(20,3; x(30,4.x(40,5;x)50)40)10,

6:x1601/50140n10) 1 20,7: 21 40 n 30, 8:x/(no/40n30) n (50 (40n10) n 40 n30 n 10, 9: x1(801...) n (101...) n (601...) n 40120}

processed = {0: x, i: x[10,2; x[20,3; x[30,4.x[40,5;x[50]40]]]

6:x(60(50)40010)1 20,1:x(40 n 30, 8:x((10(40030)1 (50 (40010) n 40 130 n 10, 9: x1(801 ···) n (101 ···) n (601 ···) n 40 n 20, 10: x1 (801...) ~ (101...) ~ (601...) ~ (501...) }

processed = {0: x, 1: x(10,2: 2|20,3: x(30,4.x(40,5:x)50)40 Alo,

6:x(601/50140010) 1 20,7:x140 1 30, 8:x/(no/40n30)n (so (40n10)n 40 n30 n 10, 9: 21(801 ···) n (101 ···) n (601 ···) n 40 n 20,

10: 261 (801 ···) n (101 ···) n (601 ···) n (501 ···),

11: x[(1001...) n (401...)}

return processed [lastnode]

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

space omplexity: Stores tensor for each node

and the two extra nodes.

each tensor has BXLXWXH = T data types

o(N) ⇒ o(N) ⇒ o(NT)

```
M = Module dict of Node operations for each node
              N=total # of nodes, last-node = final node
Pseudocode: nodes = [all nodes including extra nodes], > = input tensor
   Initialize M & node: Node Operation (indegee of node, in channel, out channel)
                   for node in nodes}
   Initialize h = heap
   Initialize processed = dict &
                                     prep in-node dict for each node
   4.push (0)
   while q not empty:
       node = q.pop()
       in-nodes = node. get -in-nodes()
       processed [node] = M.at(node) (concot (in-nodes))
       neighbors= node. getneigh bors ()
       for neighbor in neighbors:
           if neighbor not in a:
             q. push (neighbor)
  return processed [last mode]
```