① Every opening braket has the number that of the prev opening braket incremented by 1

(2) For closing braket it will take the number of corresponding opening braket.

Stack ⇒ Valid Parenthesis

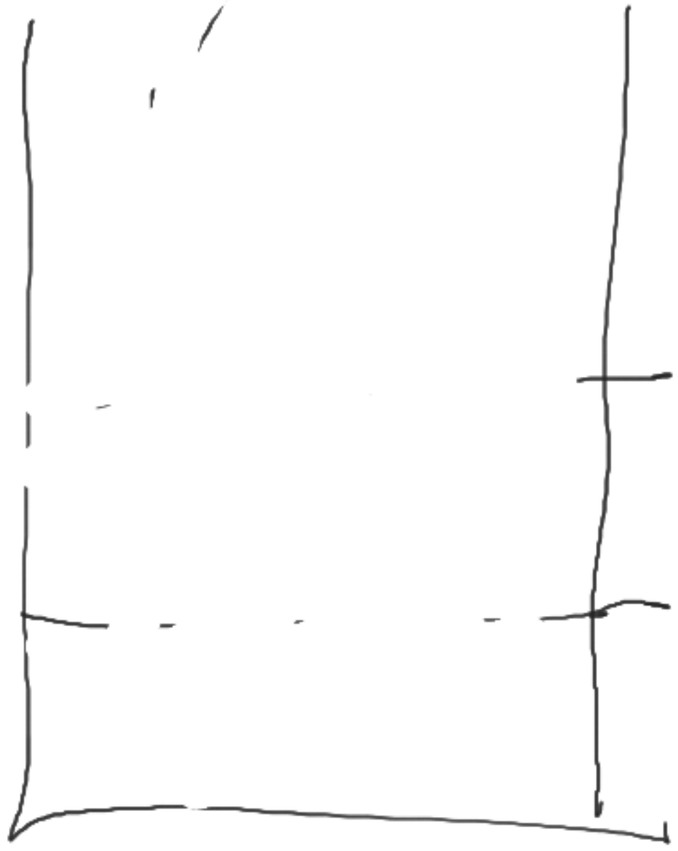
① As soon as you find an opening bracket push that in stack (Modi ⇒ push the no. instead of the bracket)

② For closing bracket pop the opening

③ Ignore the char if it is not a bracket

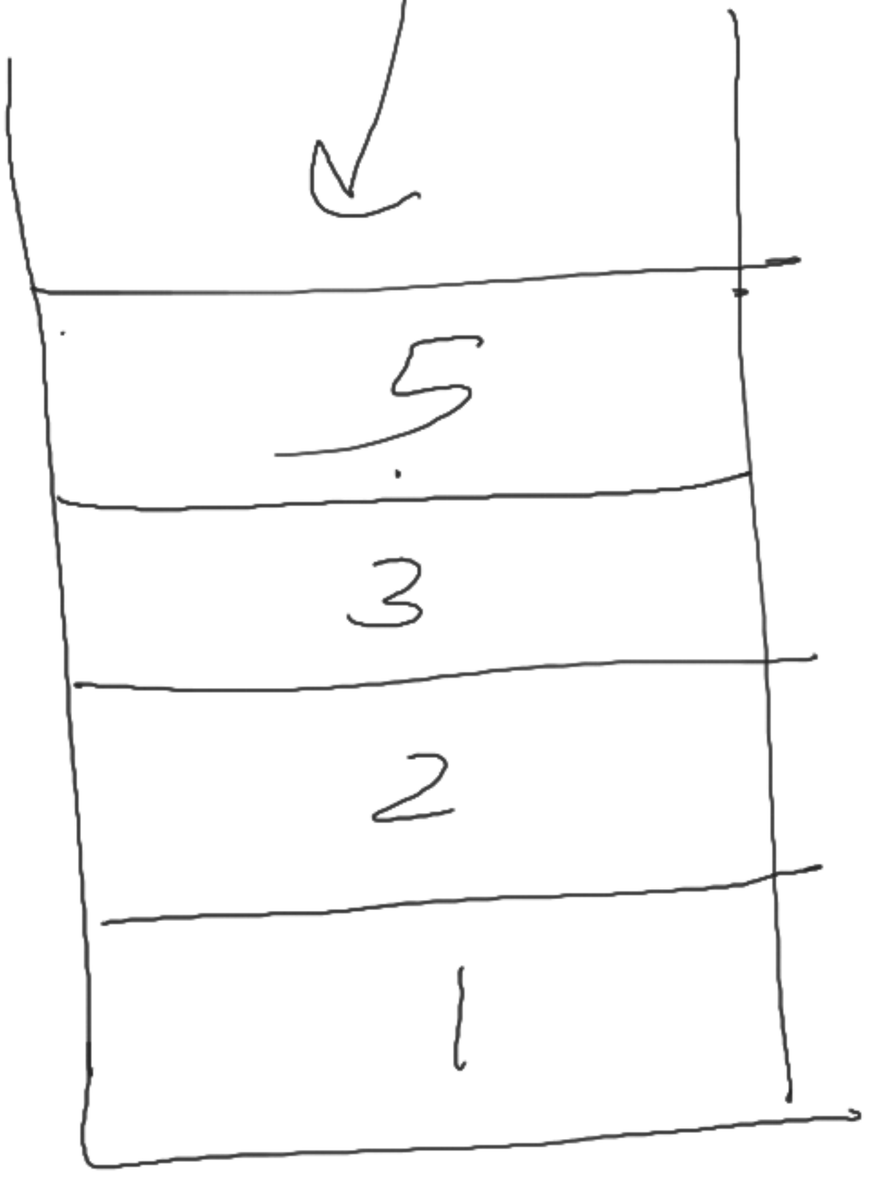
(aa(bdc))p(dee)â€

cnt => 4



d(1,2,2,1,3,3)

Opening Bra



cnt => 0

[1, 2, 3, 4, 5]

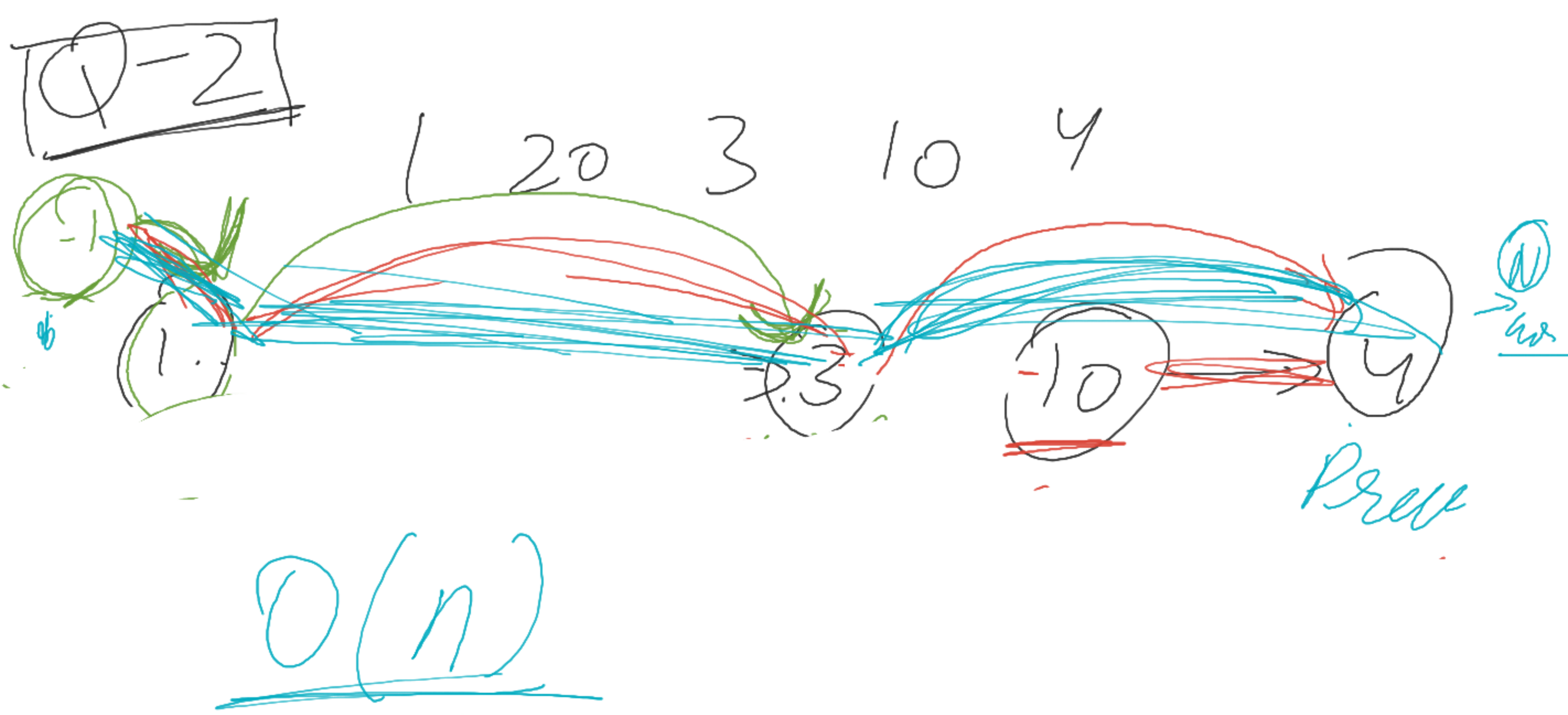
① Print top in the case of closing braket.

② Print int in the case of opening braket.

cnt = 0

1) Before pushing increment cnt

push


Q-2
1 20 3 10 4
1
1
3
10
4
Prev
O(n)

① Purpose is because we don't know the start of the ons U.

If any number is not div by 2

so just move the pointer

Deletion of Node

just break the link b/w

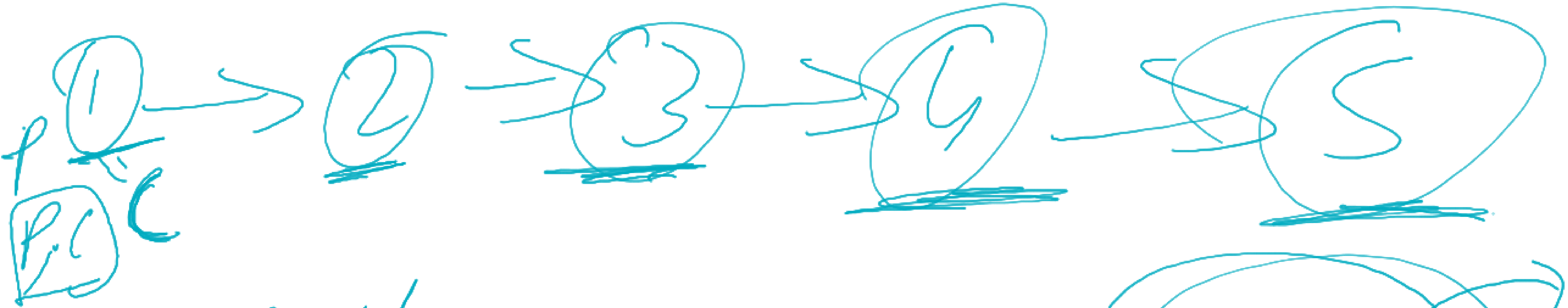
Prev & Curr

i.e

Make a link b/w

prev & curr. next

I'll move my prev when I
am sure that the num is
not div by 10

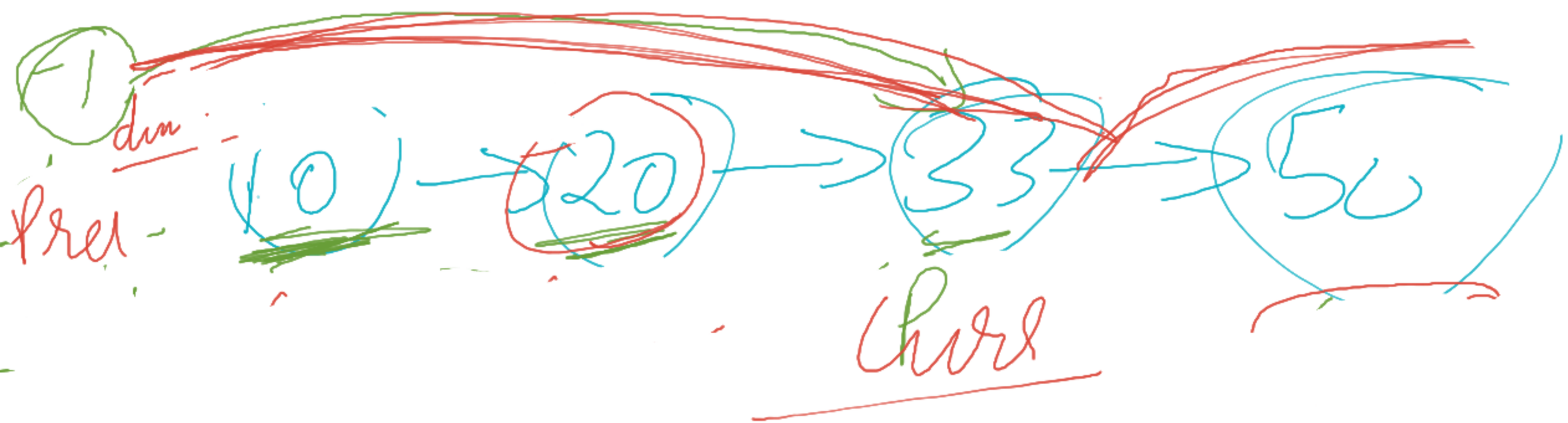


Either you moving curr

or

" " moving both

+1



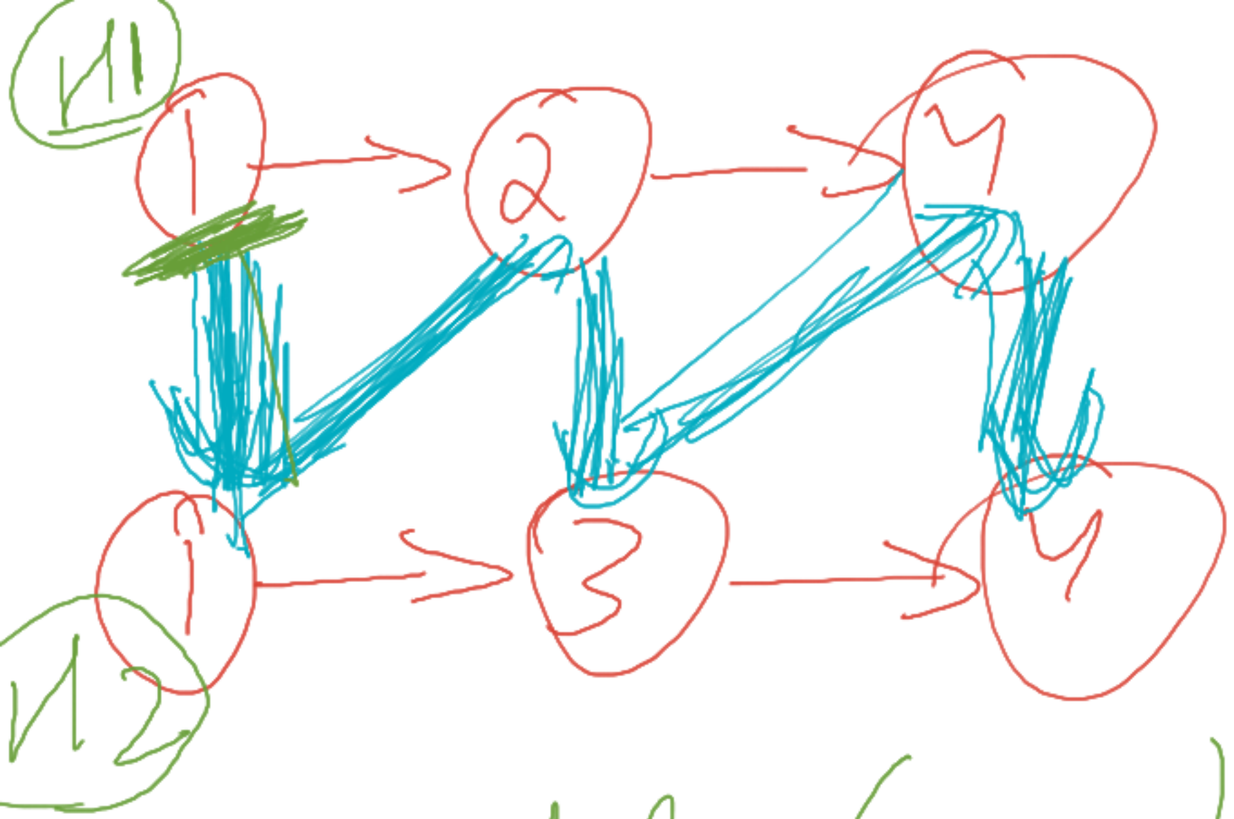
2 ITR

prev at dummy

curr at head

curr traverses the LL

prev is at last valid Node



temp

dummy (-1)

itr

whil ( )

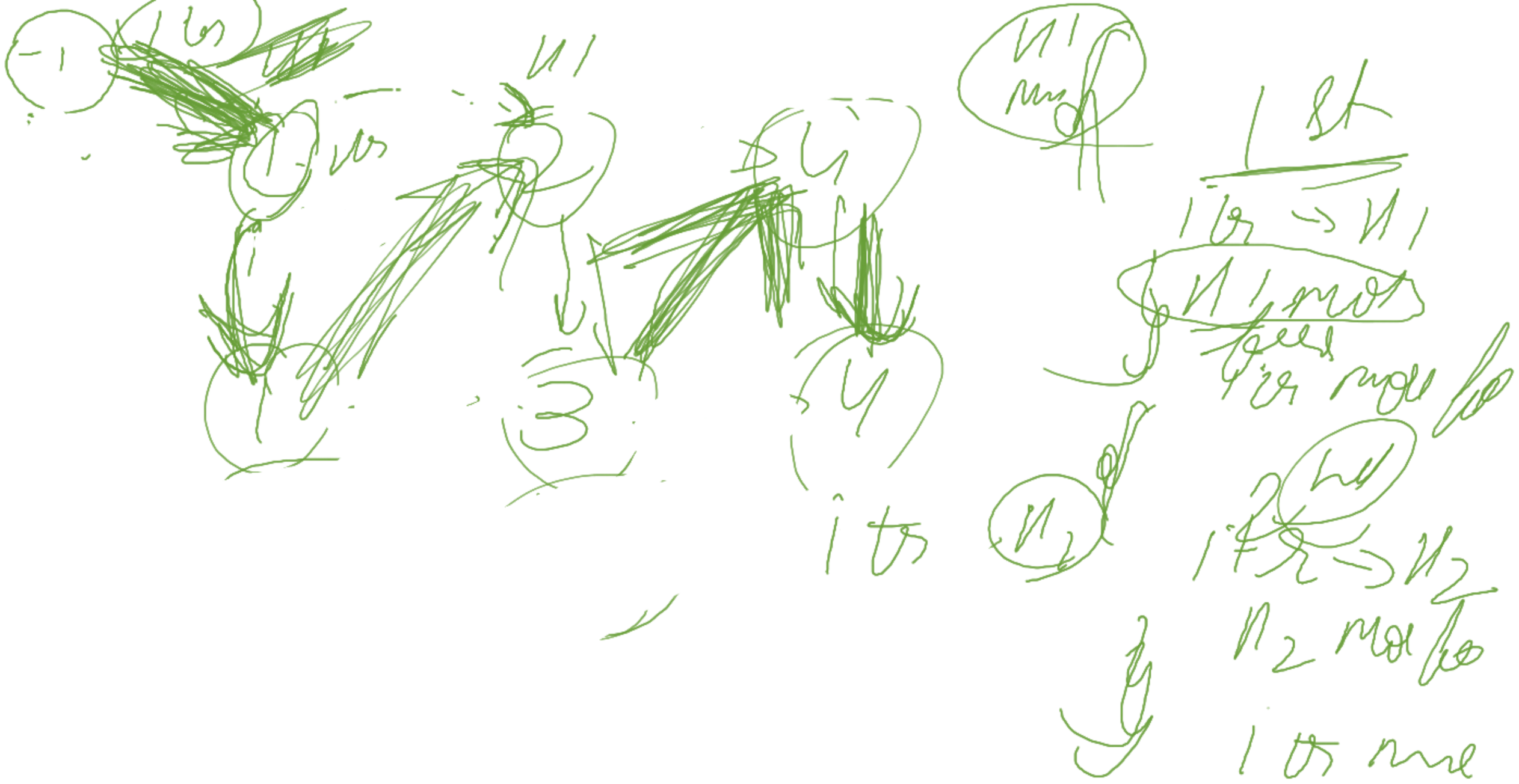
{ itr.next = H1 {

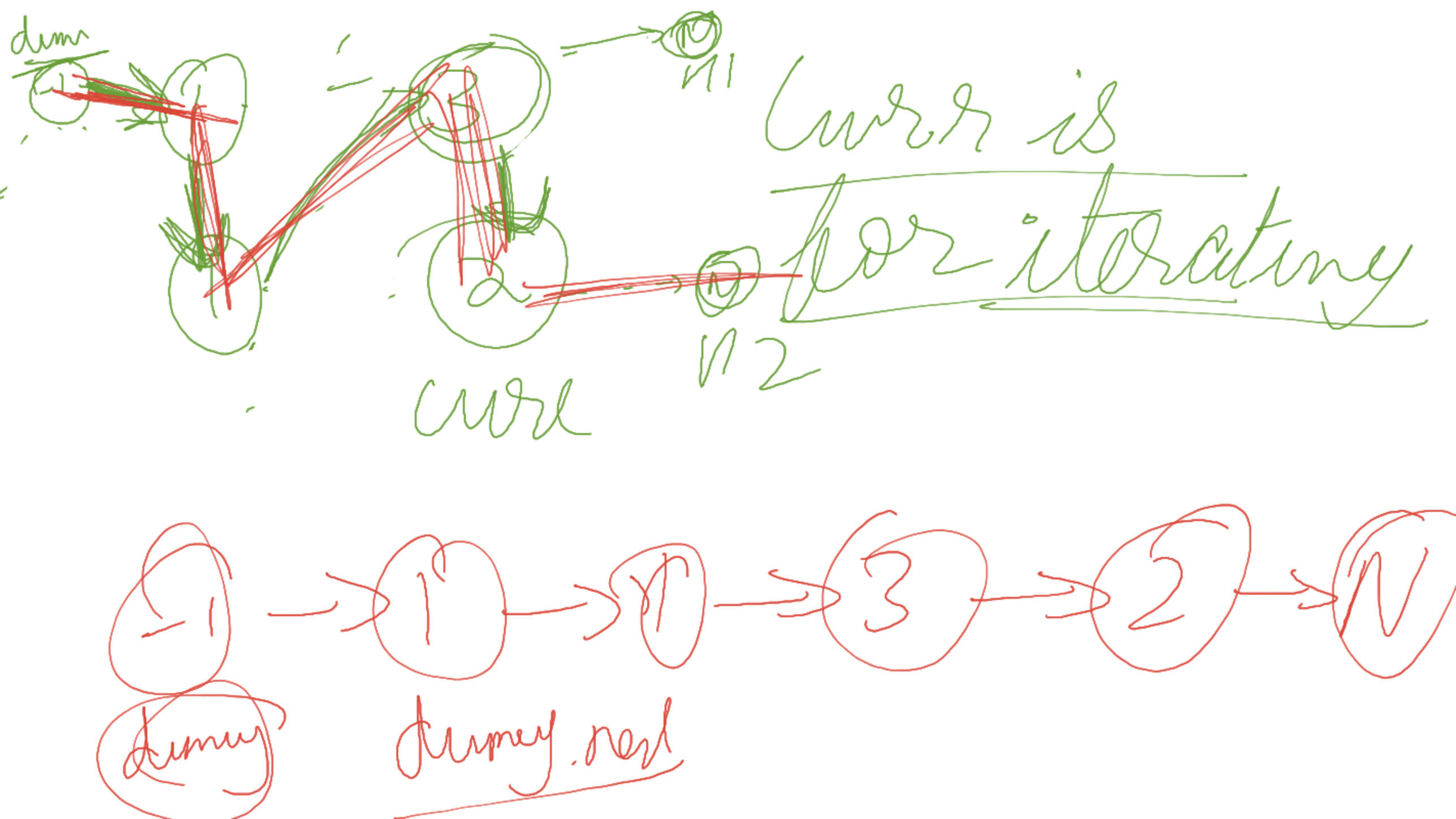
H1 = H1.next

}

ts = itr.next

}




dum
curr
curr is
for iterating
-1
1
1
3
2
N
dummy
dummy.next

2 set oper, 1st

1. link curr & H1
2. H1 moves ahead
3. curr moves ahead

2nd

1. link curr & H2
2. H2 moves ahead
3. curr moves ahead