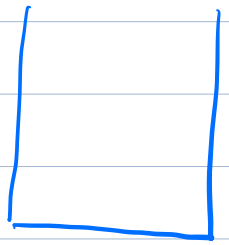
STACKS ⇒

stack of plates

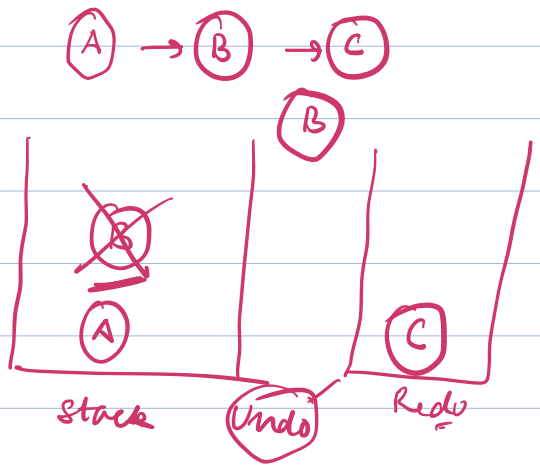
stack of chairs



Open containers

* Recursion.

* Back and next buttons in web browsers.



* Undo and Redo functionalities in Doc

* Parantheses Balancing.

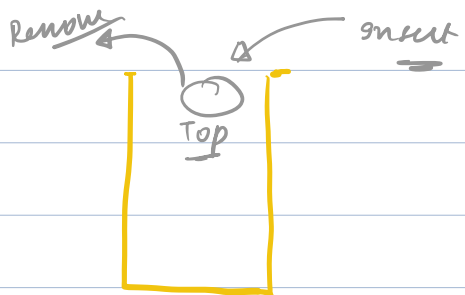
* Expression Evaluation.

Any DS → store Data

→ Insertion

→ Deletion

→ Access some Data



Lists ⇒ Dynamic Arrays

Lists are an alternative of Stack DS in python

append → Insert in End O(1)

[-1] → O(1)

l = [1, 2, 3]

l[-1] → 3

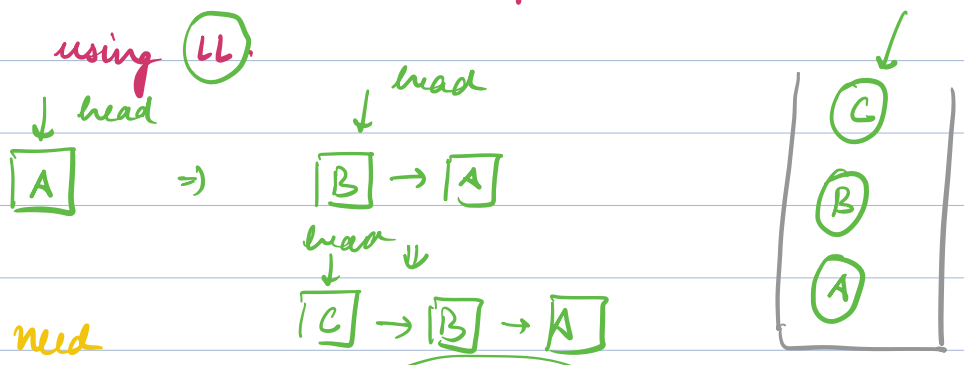
arr[-1] ✓

pop(). → O(1)

In LL.: Can you insert element at the end of LL in O(1)?

→ In the beginning of list

(*) ✓ Stacks can also be implemented using LL.



We don't need random access of data

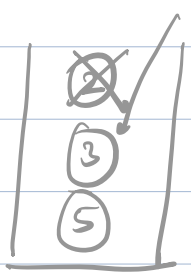
head.val ⇒ value of topmost

Stack ⇒ push, pop, top

↓ ↓ ↓

append pop [-1]

push(5)
push(3)
push(2)
pop().
top() → 3 ✓



LIFO

↓

Last In First Out

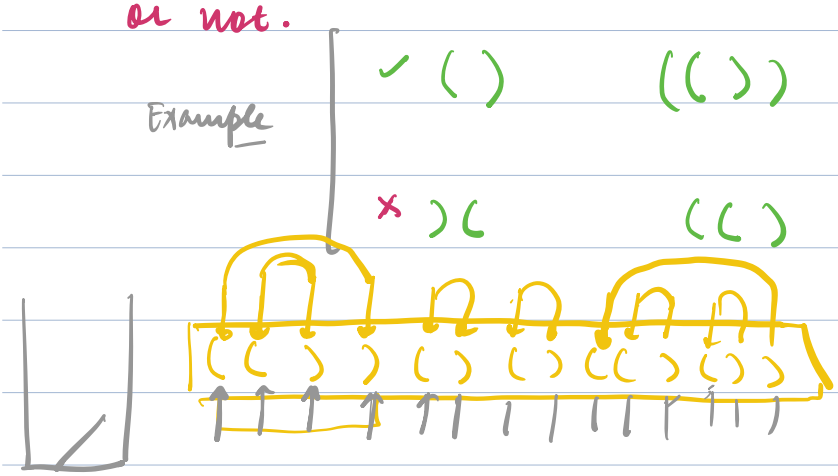
Paranthesis Balancing

You are given a sequence of brackets '(' ')'
You have to tell if the sequence is balanced or not.

Example

✓ ( ) (( )) (( ) ( ) ( ))

✗) ((( )) (( )



✓ ( )

Soln 1

Algo:
- → if opening → push '(' in stack
- → if closing → (1) if st. empty → false
  (2) pop out the '(' in the stack
- → In the end → stack should be empty

```
def isBalanced(s):
    st = []
    for x in s:
        if x == '(':
            st.append('(')
        else:                          (2)
            if len(st) == 0:
                return False
            else:
```

→ Optimised
SC → O(1)

```
cnt = 0
for x in s:
    if x == '(':
        cnt += 1
    else:
    → if cnt == 0:
        return false
    else
```

st. pop(.)

return `len(st) == 0`

cnt -= 1

return cnt == 0

( )) (

↑ ↑ ↑

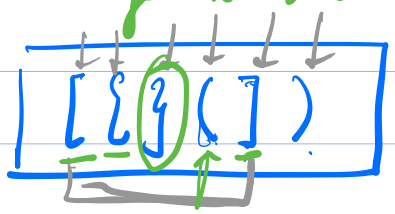
cnt = 0

↙

False

Instead of just '(' and ')'

Now we have [ ] ( ) { } → Tell if the string is balanced or not.

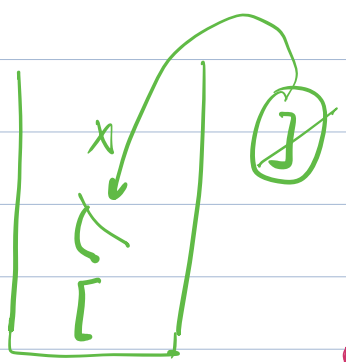


→ Not balanced.

cnt 1 cnt 2 cnt 3

0 1 2

Failing because order is important.



'[' ']'

( ) [ ] { }

↑ ↑ ( )

[({) (})]

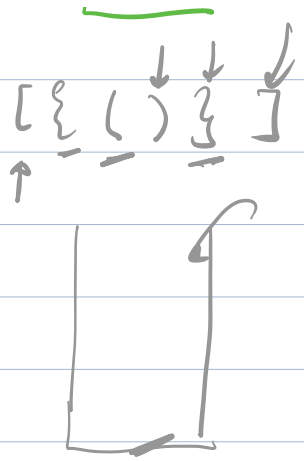
↑ ↑ ↑ X

'{'

'('

'['

```
st = [ ]
for x in s:
    if x == '(' or x == '{' or x == '[':
        st.push(x)
    → else:
        if x == ']' and st[-1] == '[':
```



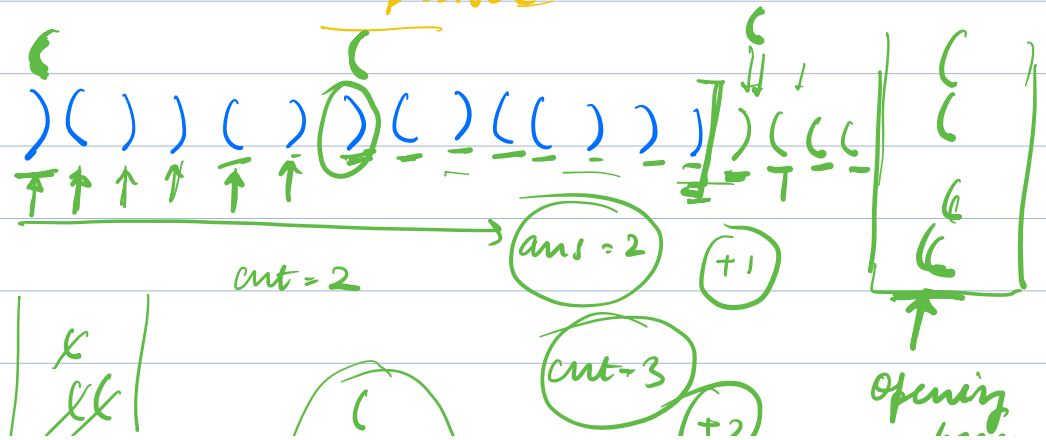
```
            st.pop()
        elif x == '}' and st[-1] == '{'
            st.pop()
        elif x == ')' and st[-1] == '('
            st.pop()
        else:
            return False
    return len(st) == 0
```

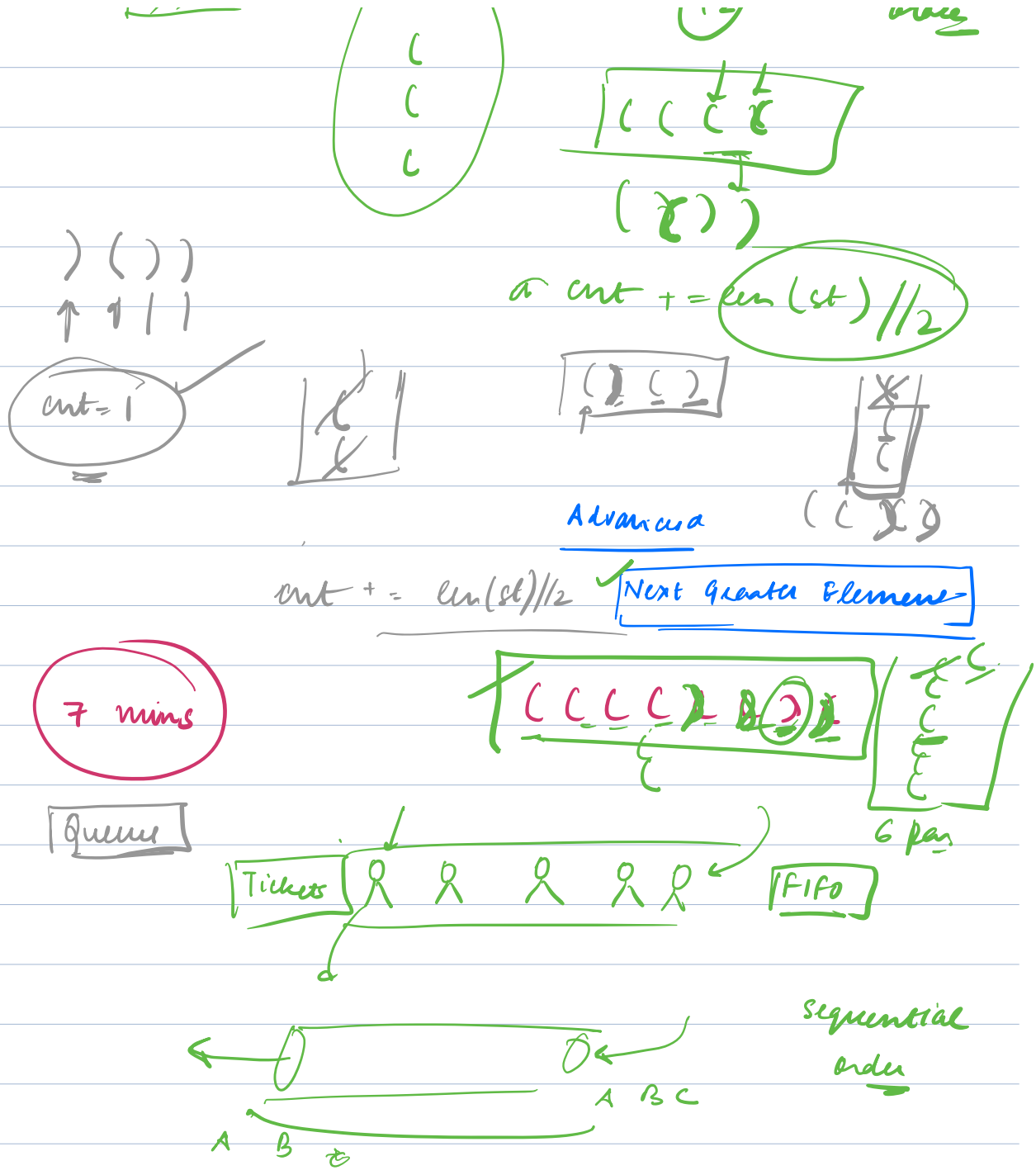
Q You are given a sequence of parentheses ⇒ ( )
You can reverse any paranthesis. How many minimum paranthesis would you have to reverse to balance the string. Return -1 if not possible.

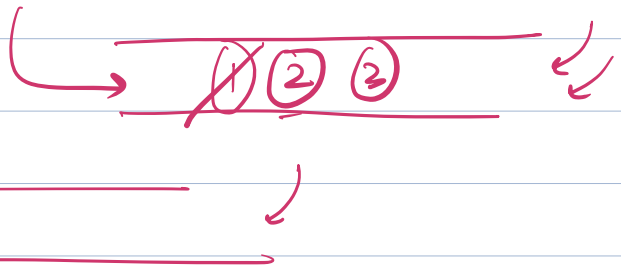
")(" ⇒ 2 ✓

( ) (( ))) ( ) (⇒ 2 ✓

✓ If length is odd ⇒ not possible




cnt += len(st)//2
cnt = 1
Advanced
cnt += len(st)//2
Next Greater Element
7 mins
Queues
Tickets
FIFO
6 pax
Sequential order
A B C
A B
push()
pop()
front()
push(1)
push(2)
push(3)
pop()
front() → (2)
Applications
MQ



import collections

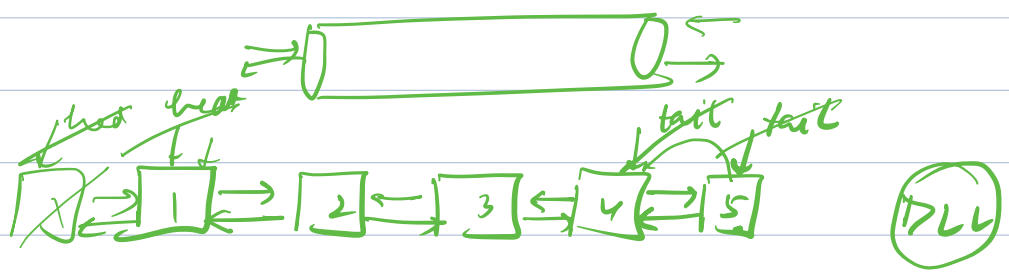
collections.deque.

1, 2, 3

qu = collections.deque([1,2,3])

qu.append(4) → (R) appendleft() → (L)

qu.popleft() → (L) pop() → (R)



DLL

class Node:

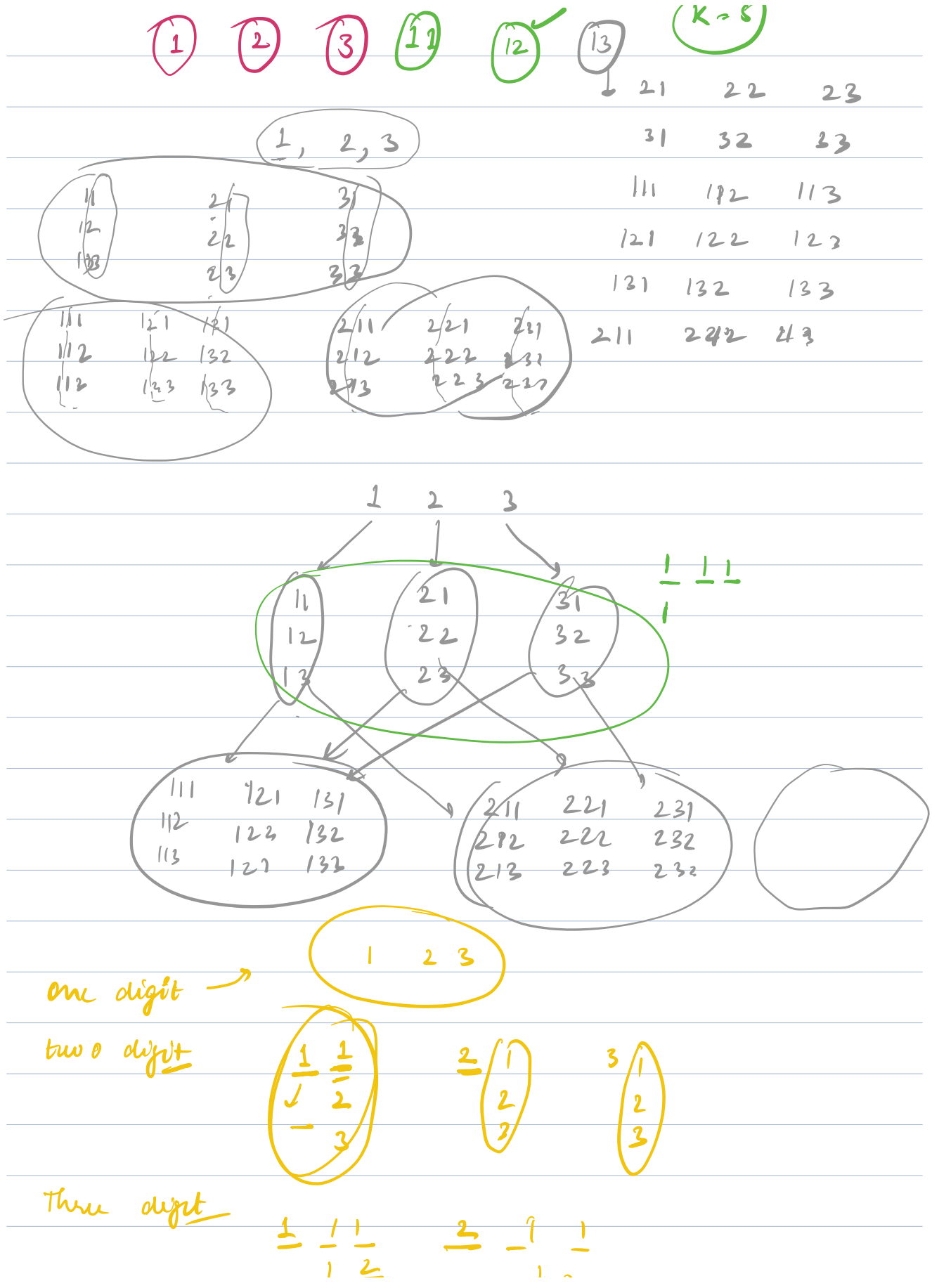
init:

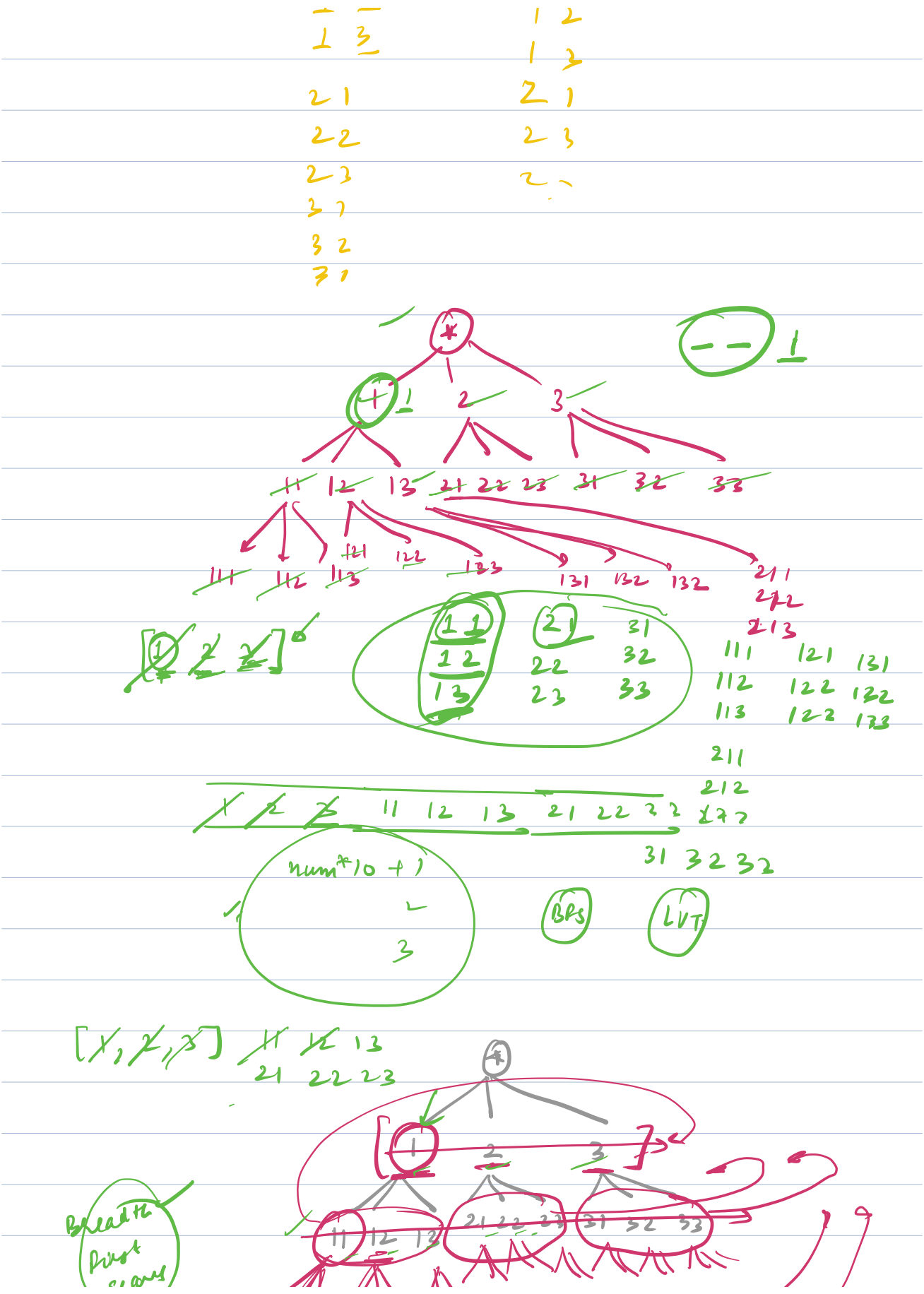
self.next

self.prev

Q: I have given you 3 digits → 1, 2, 3

I want you to tell me the K^{th} smallest number that you can form using 3 digits only.


1 2 3 11 12 13
K = 5
21 22 23
31 32 33
111 112 113
121 122 123
131 132 133
211 212 213
221 222 223
231 232 233
1, 2, 3
1 2 3
one digit
two digit
Three digit


num*10 + i
BFS
LVT
Breadth
First
Search

Level order Traversal ⇒ Queue

k^{th} smaller

import collections

qu = collections.deque([1,2,3])

[1,2,3]

[2,3]

k=5 → k=4 → k=3 → k=2 → k=1

while k > 0

x = qu.popleft()

qu.append(10*x+1)

qu.append(10*x+2)

qu.append(10*x+3)

k -= 1

return x

x=1

x [11, 12, 13]

21 22 23

31 32 33

x=2 111 112 113

x=3

x=11

x=12 ✓

k=4

[1, 2, 3] k=1 k=2

k=4 k=3 k=2 k=1 k=0

00

11

00

5 → Base 3

$0, 1, 1^2, 1^3$

2^3

10 20

11 21

12 22

100

101

102