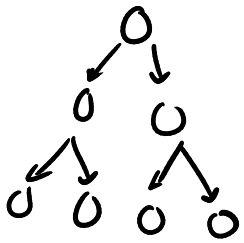
Linked list.

Linked list → Binary Tree → graph

```
class node {
    int data
    node next
}
```

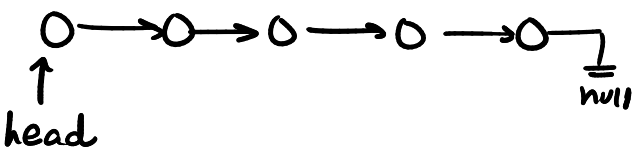
```
class node {
    int data
    node left, right
}
```

```
class node {
    int data
    node neighbour[]
}
```



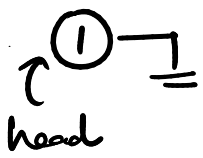
Tree — 2D structure

1D structure.

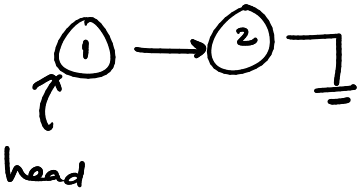


Node head = null ; // empty list.

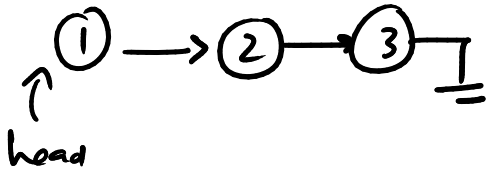
head = new Node(1)



head.next = new Node(2)



head.next.next = new Node(3)



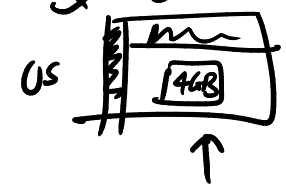
head.next.next.next = new Node(4). // 1 → 2 → 3 → 4

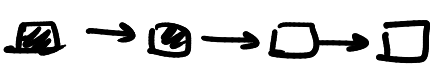
? → how to do better?

CRUD.

1

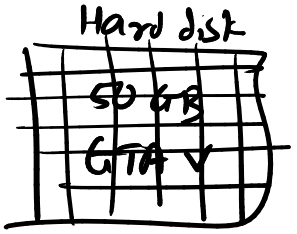
RAM → 16 GB RAM





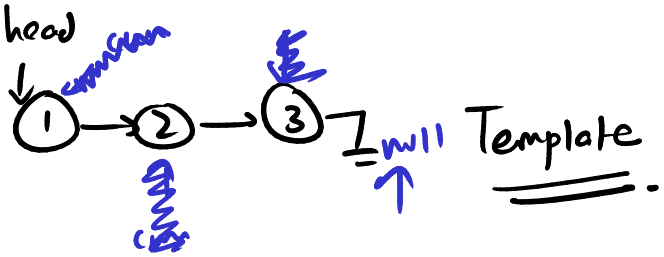
Why LL? | No need of this DS today.

Hard disk



C R U D. ① Read head → always need it | lose it X

logic : start from head | go to next as long as you are not null.



Template.

```
traverse (head):
    curr = head
    while (curr != null):
        print (curr.data)
        curr = curr.next
```

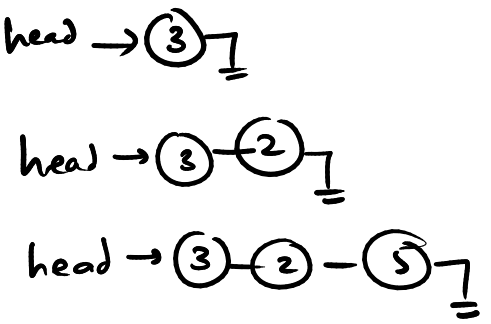
1. count the no. of nodes → length of LL.
2. search for a node
3. min node & max node

Read:

O(n)

2. Create. stream of data (one number at a time)

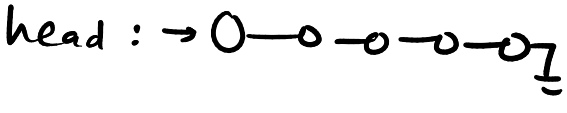
```
insert (head, new_data): {
    Node new_node = new Node(new data)
    // list is empty
    if head == null :
        head = new_node
        return
    Node curr = head
    while curr.next != null :
        curr = curr.next
    curr.next = new_node
}
```



n length LL

TC $O(n^2)$

even / odd nodes



even! ——

odd : ——

if curr.data is even:
    insert (even, curr.data)
else: insert (odd, curr.data)

③ Update. Update = 1 deletion + 1 insertion

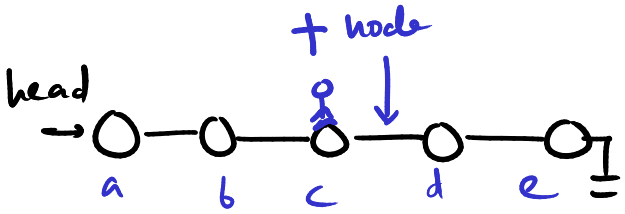
update (head, 3, 5) 3 → node | update to 5.

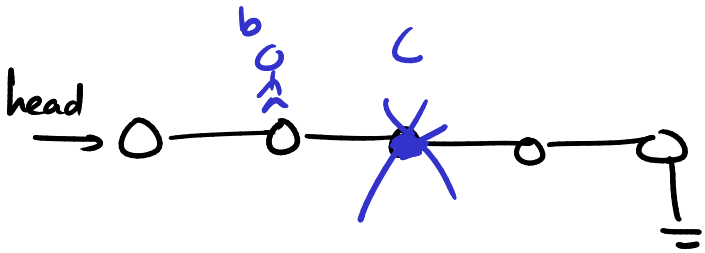
1. search for 3. In the same node, curr.data = 5.

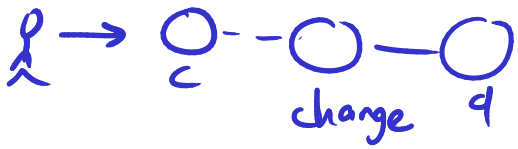
priority to 1st occurrence in case of dups.

Insert a node in between.

Thumb rule: Whatever change in U, you need to be on previous node.







insert (head, data, position):

position: ① on which you new node should sit.

new_node = new Node(data)

if !(position < 0 and > len): return

position: ② this is the node on which you stand, new node position +1

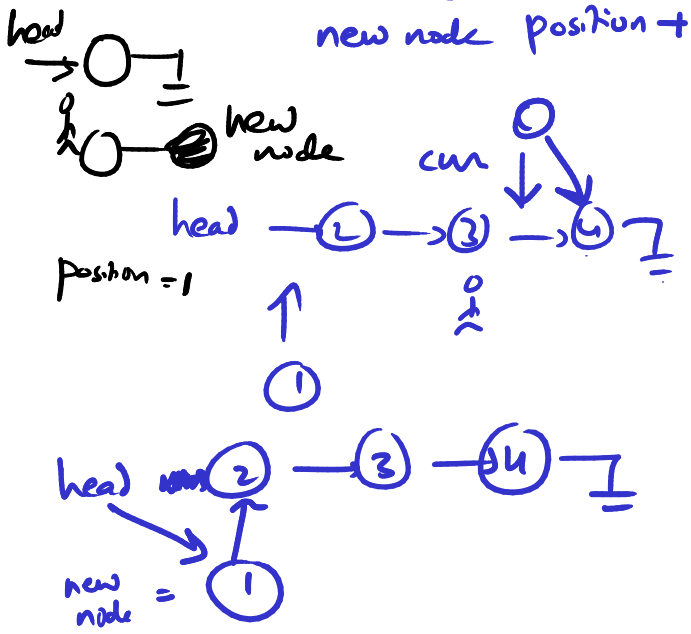
if position == 0:
    new_node.next = head
    head = new_node
    return

curr = head ; position -- ;

while (position--):
    curr = curr.next

new_node.next = curr.next

curr.next = new_node

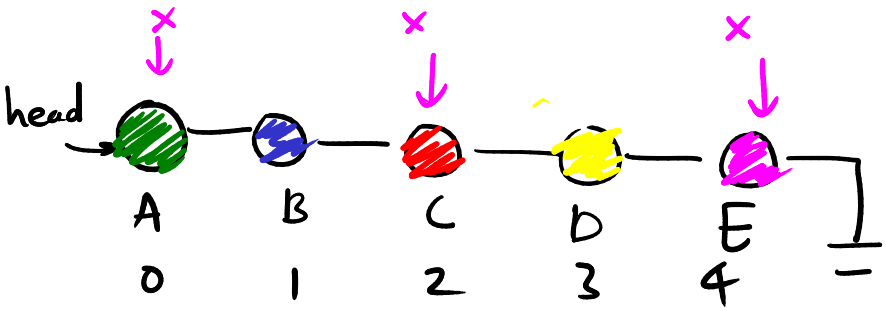


Any operation : Linked list

CORNER CASES. ✓

Think:
1. What if list is empty ?
2. What if list just a single node ?
3. Insert at the every start .
4. Can I do in the middle?
5. Can I do it in the end ?

④ DELETE.



cur_node index delete

① delete index = 0
(delete head node)

② delete index = i

③ delete index = len
(delete last node)

ans.

Ⓐ cur = head
head = head.next
cur.next = null

Ⓒ position = 2
position --; cur = head;
while (position --)
cur = cur.next;
cur → blue

Ⓔ //get one node before
cur.next = null

next_node = curr.next
curr.next = next_node.next
next_node.next = null
next node → red

— all threw nodes → garbage collected —

Else:

1. loops in LL
2. Intersections in LL
3. Sorting of LL
4. Reversal of LL

End of LL.