LINKED LISTS (Background & Intro)

Problem with Arrays

① How to implement Round-Robin Scheduling?

P_0	P_1	P_2	P_3	P_4	P_5
10	5	3	15	10	8

Time = 5

P_1	P_2	P_3	P_4	P_5	P_0
5	3	15	10	8	5

P_2	P_3	P_4	P_5	P_0
3	15	10	8	5

P_1 completes

P_3	P_4	P_5	P_0
15	10	8	5

P_2 completes

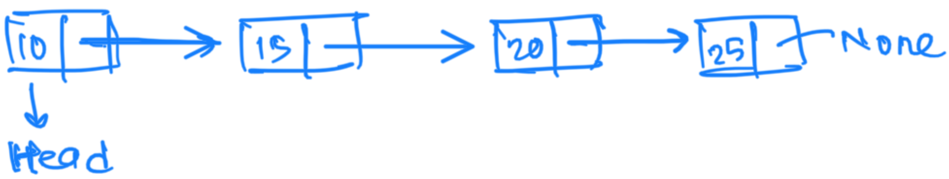
This becomes difficult with arrays, as you have to pick 0th item & insert in the end.

② Given a sequence of items, whenever we see an item x in the sequence we need to replace it with 5 instance of 'y'

I/P → d e a x q r x p y

O/P → d e a y y y y y q r y y y y y p y

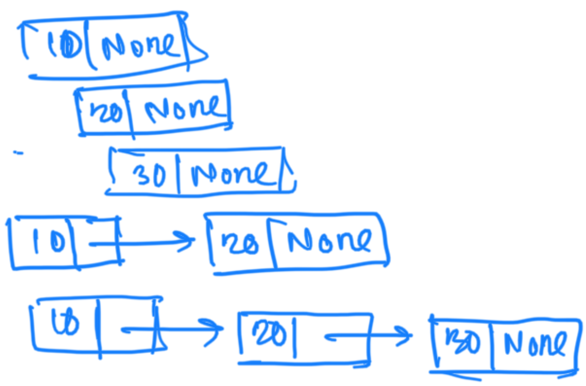
Linked Lists



Simple Implementation:


```
class Node:
    def __init__(self, k):
        self.key = k
        self.next = None

temp1 = Node(10)
temp2 = Node(20)
temp3 = Node(30)
temp1.next = temp2
temp2.next = temp3
head = temp1
```



Shorter Implementation

```
head = Node(10)
head.next = Node(20)
head.next.next = Node(30)
```

Applications of LL

1. Worst case insertion at end & begin are O(1).
2. Insertions & deletions in middle are O(1) if we have reference to previous node.
3. Worst case deletion from beginning is O(1).
4. Round Robin implementation.
5. Merging two sorted linked list is faster than arrays.
6. Implementation of simple memory manager where we need to link free blocks.
7. Easier implementation of Queue & Deque data structures.

Traversal

head

```
def printlist(head):
    curr = head
    while curr != None:
        print(curr.key, end = " ")
        curr = curr.next
```

Insertion at the beginning

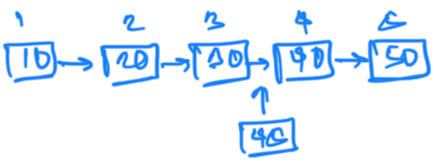
```
class Node:
    def __init__(self, key):
        self.key = key
        self.next = None.
    def insertBegin(head, key):
        temp = Node(key)
        temp.next = head
        return temp.
```

Insertion at the End

```
def insertEnd(head, key):
    if head == None:
        return Node(key)
    curr = head
    while curr.next != None:
        curr = curr.next
    curr.next = Node(key)
    return head.
```

Insertion at given position

```
def insertPos(head, data, pos):
    temp = Node(data)
    if pos == 1:
        temp.next = head
        return temp.
    curr = head
    for i in range(pos-2):
        curr = curr.next
        if curr == None:
```



$\boxed{\Theta(\min(pos, n))}$

```
            return head
    temp.next = curr.next
    curr.next = temp
    return head
```

Delete first node

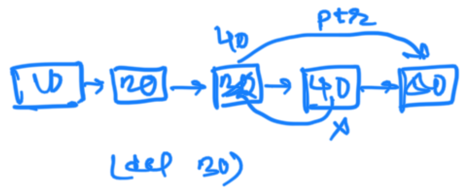
```
def delFirst(head):
    if head == None:
        return None
    else:
        return head.next
```

Delete last node

```
def delLastNode(head):
    if head == None:
        return None.
    if head.next == None:
        return None
    curr = head
    while curr.next.next != None:
        curr = curr.next
    curr.next = None
    return head
```

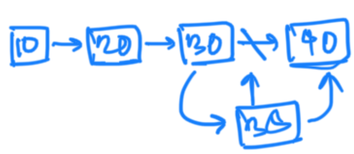
Delete Node using pointer

```
def deleteNode(ptr):
    temp = ptr.next
    ptr.data = temp.data
    ptr.next = temp.next
```



Insertion in Sorted list

```
def sortedInsert(head, val):
    temp = Node(val)
    if head == None:
        return temp
    elif val < head.data:
        temp.next = head
```



```
    return temp
else:
    curr = head
    while curr.next != None & curr.next.data < val:
        curr = curr.next
    temp.next = curr.next
    curr.next = temp
    return head
```

Middle of Linked List

Brute force

- Iterate through and count nodes.
- 2nd pass, iterate till count//2 & print

Optim approach

- Two pointers, slow & fast

```
def printMiddle(head):
    if head == None:
        return
    slow = head
    fast = head
    while fast != None and fast.next != None:
        slow = slow.next
        fast = fast.next.next
    print(slow.data)
```