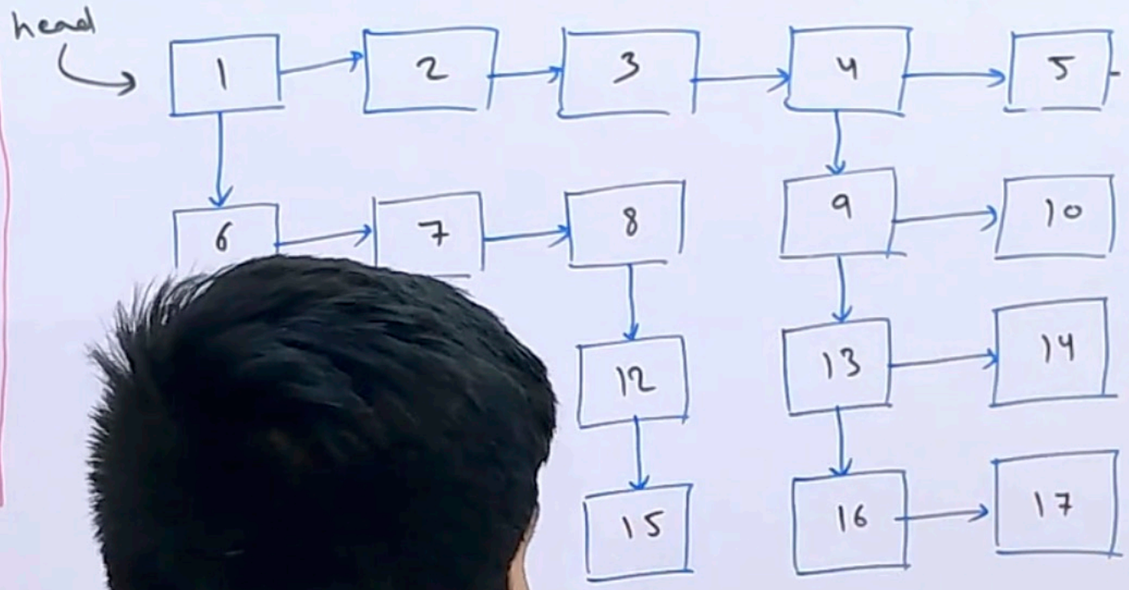
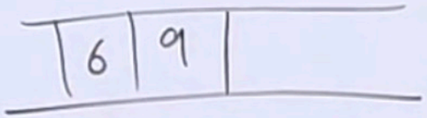
Q. Flatten a Multilevel Linked List

```
Node {
    int data
    Node next;
    Node down;
}
```



Queue FIFO



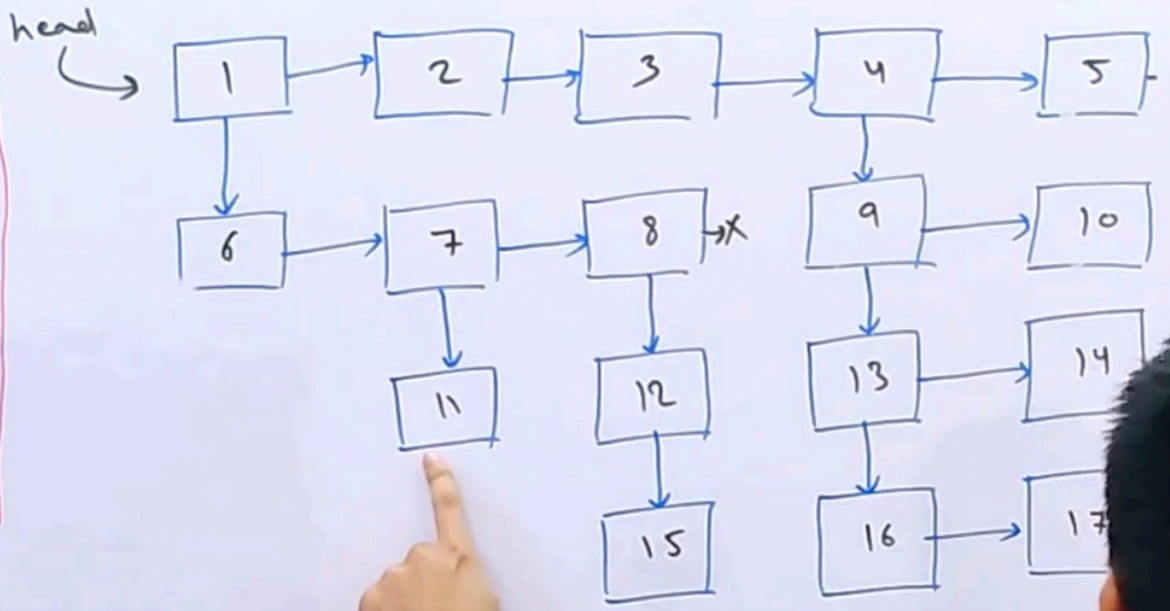
1, 2, 3, 4, 5

Output:

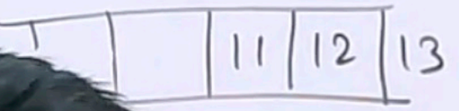
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17

Q. Flatten a Multilevel Linked List

```
Node {
    int data
    Node next;
    Node down;
}
```



Queue FIFO

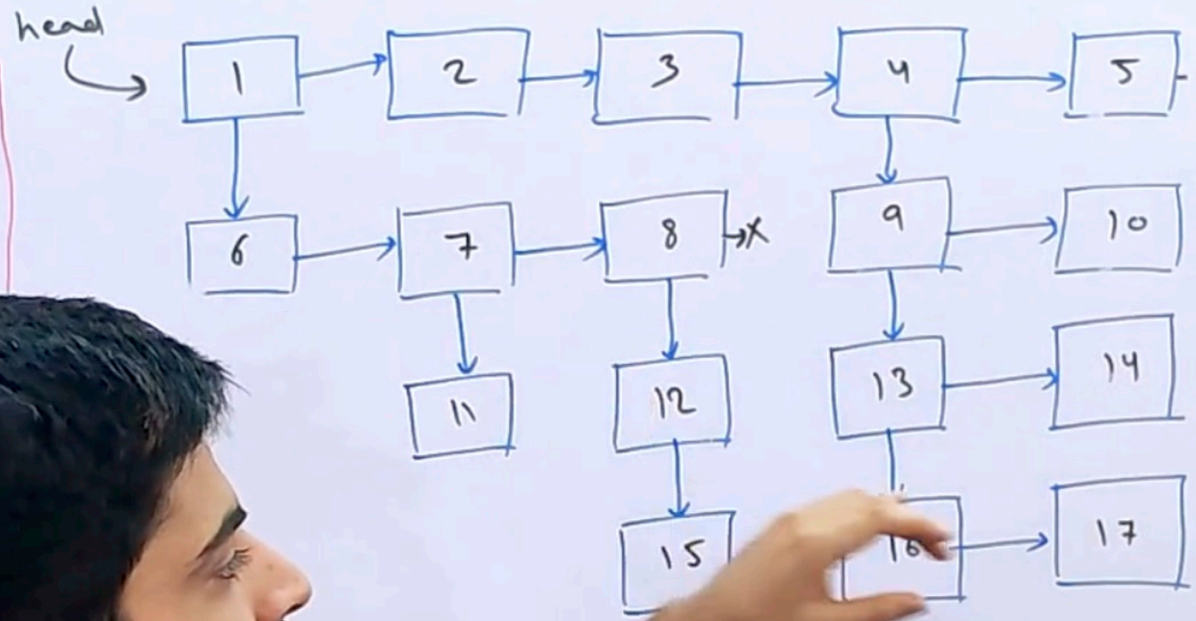


Output:

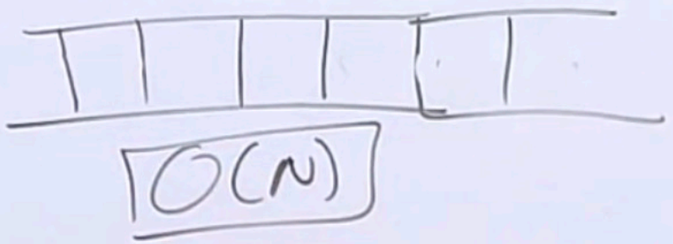
1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ... 2, 13, 14

... 5, 6, 7, 8, 9, 10

```
Node {
    int data
    Node next;
    N
}
```



Queue FIFO



1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

12, 13, 14, 15, 16, 17

Output

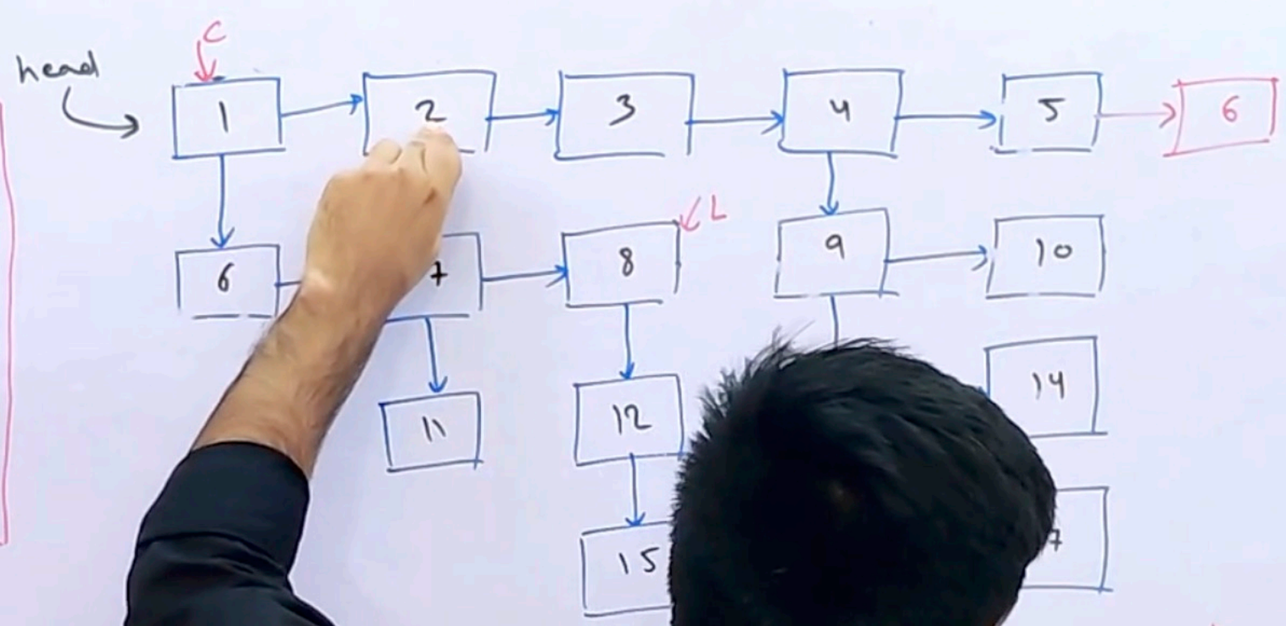
1

11, 12, 13, 14, 15, 16, 17

```java

class Main
{
    // Function to convert a multilevel linked list into a singly linked list
    public static Node convertList(Node head)
    {
        Node curr = head;
        Queue<Node> q = new ArrayDeque<>();

        // process all nodes
        while (curr != null)
        {
            // last node is reached
            if (curr.next == null)
            {
                // dequeue the front node and set it as the next node
                // of the current node
                curr.next = q.poll();
            }

            // if the current node has a child
            if (curr.child != null) {
                q.add(curr.child);
            }

            // advance the current node
            curr = curr.next;
        }

        return head;
    }
```

Q. Flatten a Multilevel Linked List

```
Node {
    int data
    Node next;
    Node down;
}
```

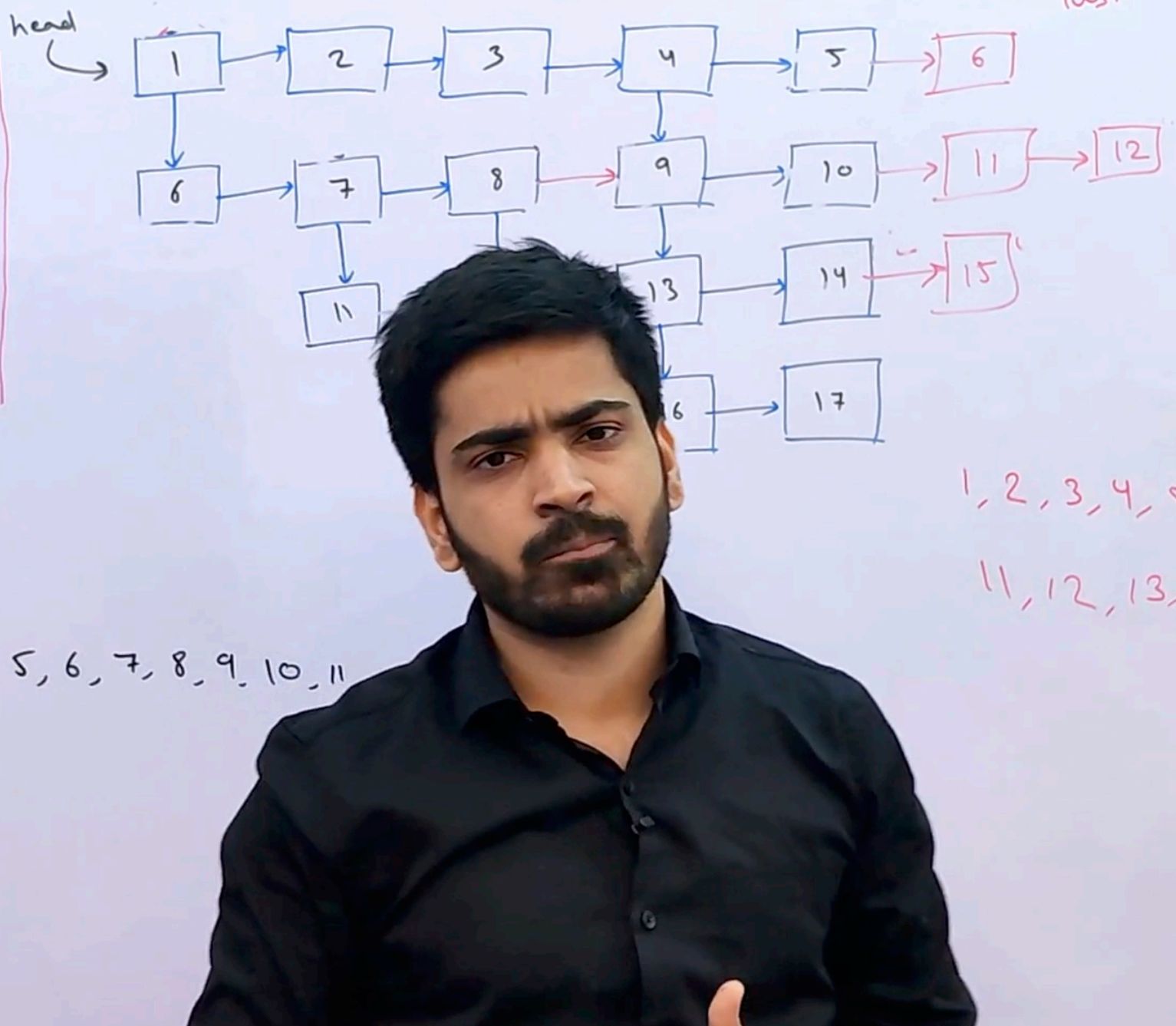


Output:

1, 2, 3, 4, 5, 6, 7, 8, 9,

Q. Flatten a Multilevel Linked List

```
Node {
    int data
    Node next;
    Node down;
}
```



Output:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

11, 12, 13, 14, 15, 16, 17

```
if (node == null) {
    return;
}

Node tmp = null;

/* Find tail node of first level linked list */
Node tail = node;
while (tail.next != null) {
    tail = tail.next;
}

// One by one traverse through all nodes of first level
// linked list till we reach the tail node
Node cur = node;
while (cur != tail) {

    // If current node has a child
    if (cur.child != null) {

        // then append the child at the end of current list
        tail.next = cur.child;

        // and update the tail to new last node
        tmp = cur.child;
        while (tmp.next != null) {
            tmp = tmp.next;
        }
        tail = tmp;
    }
```