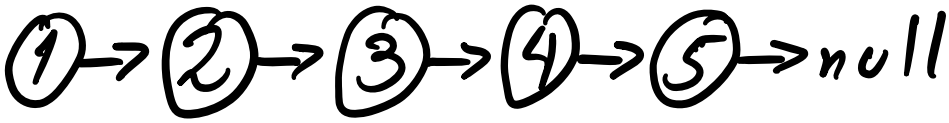
Reverse list

Generar un algoritmo que retorne una lista al reves ejemplo:

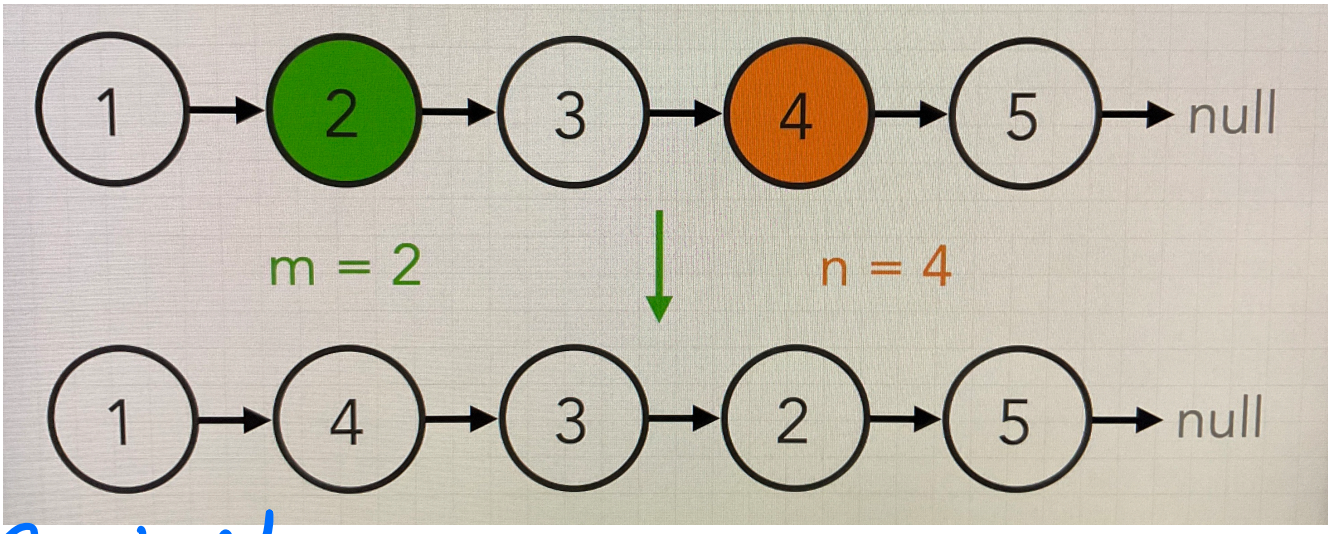


Pseudocódigo

```
func reverse_list (head: Node)->List;
    prev=null
    current = head
    while current!=null;
       next=current.next
       current.next=prev
       prev = current
       current = next
   return List(prev)
```

Intercambiar valores de un lista

Dados dos valores que se encuentren en una lista intercambiar dichos valores en la lista



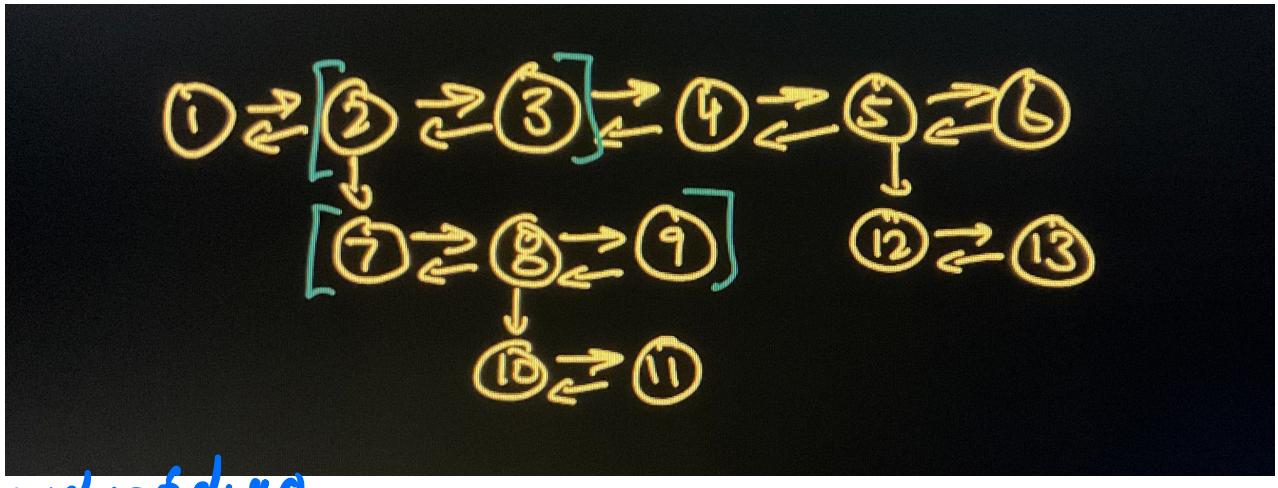
Pseudocódigo

```
func reverse_between(head, m, n):
    current_pos=1
    current_node=head
    start = head
    while current_pos<m:
        start=current_node
        current_node = current_node.next
        current_pos++
    prev  = null
    tail = current_node
    while (current_pos>=m && current_pos<=n):
        next=current_node.next
        current_node.next = prev
        prev = current_node
        current_node = next
```

```
        current_pos ++

    start.next = prev
    tail.next = current_node
    if m > 1 :
        return head
    else:
        return prev
```

Mergear listas doblemente enlazadas

Teniendo una lista doblemente enlazada combinar las listas que tengan nodos hijos para dejar una sola lista enlazada.

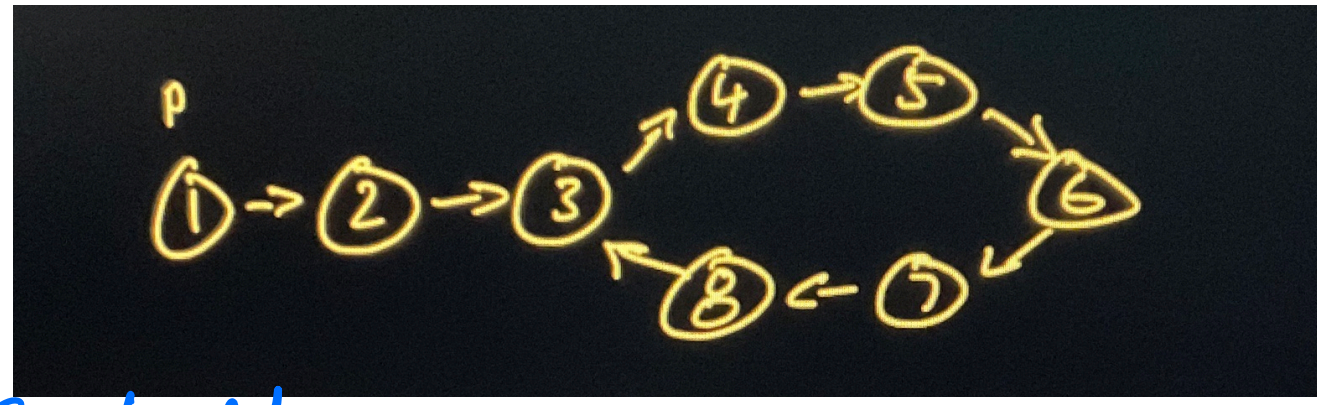


Pseudocódigo

```
func flatten(head: Node) -> Node:
    if (head is None):
        return head

    current_node = head

    while (current_node is not None):
        if current_node.child is None:
            current_node = current_node.next

        else:
            tail = current_node.child

            while(tail.next is not None):
                tail = tail.next

            tail.next = current_node.next

            if (tail.next is not None):
                tail.next.prev = tail
```

```
        current_node.next = current_node.child
        current_node.next.prev = current_node
        current_node.child = None

    return head
```

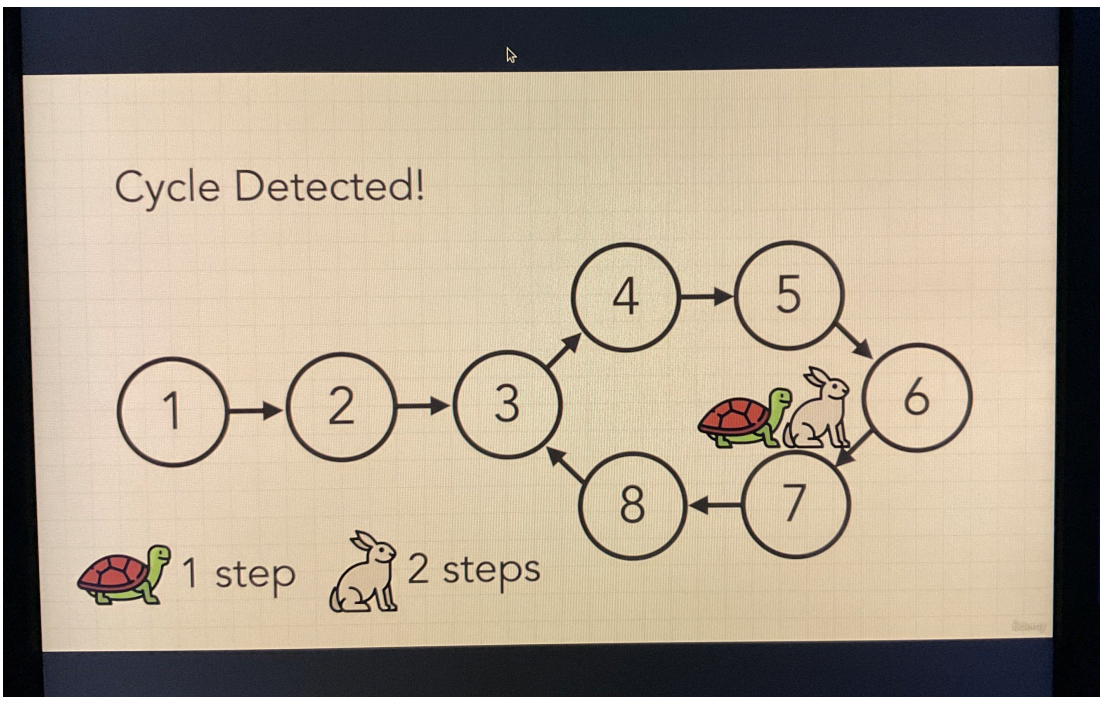
Detectar si es una lista ciclica



Pseudocódigo

```
Func find_cycle(head: Node):
    let current_node = head
    const seen_nodes = set()
    while (!current_node in seen_nodes):
        if current_node.next is None:
            return None
        seen_nodes.add(current_node)
        current_node = current_node.next
    return current_node
```

Floyd's Tortoise algoritmo (detección ciclica).



Pseudocódigo

```
Func find_cycle(head: Node) -> Node:
    tortoise = head
    hare = head
    while (true):
        hare = hare.next
        tortoise = tortoise.next
        if hare is None or hare.next is None:
            return False
        else:
            hare = hare.next
```

```
if tortoise == hare
    break
p1 = head
p2 = tortoise
while p1 != p2:
    p1 = p1.next
    p2 = p2.next
return p1
```