In [37]:
```
class Node:

    def __init__(self, data):
        self.val = data
        self.next = None
        self.prev = None
        self.child = None
```

In [39]:
```
# multi-level-ll-with-multi-child

node1 = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)
node6 = Node(6)
node7 = Node(7)
node8 = Node(8)

head = node1
node1.next = node2
node2.prev = node1
node2.child = node3
node3.next = node4
node4.prev = node3
node2.next = node5
node5.prev = node2
node5.child = node6
node6.next = node7
node7.prev = node6
node5.next = node8
```

In [40]:
```
temp = head
while temp is not None:
    print(temp.val, end=" ")
    temp = temp.next
```

```
1 2 5 8
```

In [41]:
```
class Solution:

    def flatten(self, head: 'Optional[Node]') -> 'Optional[Node]':

        if head is None:
            return None

        if head.child is not None:

            # save the reference of the next node before going to child node
            temp = head.next

            # get the flattened list connected from child to its end
            flatten_list_head = self.flatten(head.child)

            # connect the head with the flattened child list
            head.next = flatten_list_head
            flatten_list_head.prev = head

            # now remove child reference so that doubly ll is valid
            head.child = None

            # now we need to traverse to the end of the child ll
            # so that we can connect end of child ll with the node
```

```
            # that is next to head node
            flatten_temp = flatten_list_head
            while flatten_temp.next is not None:
                flatten_temp = flatten_temp.next

            # connecting end of child ll with node next to head
            flatten_temp.next = temp
            if temp is not None:
                temp.prev = flatten_temp

            # once done we can continue this process from node next to head

        else:
            # if there is no child just keep nodes connected linearly
            self.flatten(head.next)

        return head
```

In [42]:
```
new_ll = Solution().flatten(head)
```

In [43]:
```
temp = new_ll
while temp is not None:
    print(temp.val, end=" ")
    temp = temp.next
```

```
1 2 3 4 5 8
```

after adding self.flatten in if

In [31]:
```
# multi-level-ll-with-multi-child

node1 = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)
node6 = Node(6)
node7 = Node(7)
node8 = Node(8)

head2 = node1
node1.next = node2
node2.prev = node1
node2.child = node3
node3.next = node4
node4.prev = node3
node2.next = node5
node5.prev = node2
node5.child = node6
node6.next = node7
node7.prev = node6
node5.next = node8
```

In [32]:
```
temp = head2
while temp is not None:
    print(temp.val, end=" ")
    temp = temp.next
```

```
1 2 5 8
```

In [44]:
```
class Solution2:

    def flatten(self, head: 'Optional[Node]') -> 'Optional[Node]':
```

```
        if head is None:
            return None

        if head.child is not None:

            # save the reference of the next node before going to child node
            temp = head.next

            # get the flattened list connected from child to its end
            flatten_list_head = self.flatten(head.child)

            # connect the head with the flattened child list
            head.next = flatten_list_head
            flatten_list_head.prev = head

            # now remove child reference so that doubly ll is valid
            head.child = None

            # now we need to traverse to the end of the child ll
            # so that we can connect end of child ll with the node
            # that is next to head node
            flatten_temp = flatten_list_head
            while flatten_temp.next is not None:
                flatten_temp = flatten_temp.next

            # connecting end of child ll with node next to head
            flatten_temp.next = temp
            if temp is not None:
                temp.prev = flatten_temp

            # once done we can continue this process from node next to head
            self.flatten(temp)
        else:
            # if there is no child just keep nodes connected linearly
            self.flatten(head.next)

        return head
```

In [45]:
```
new2 = Solution2().flatten(head2)
```

In [46]:
```
temp = head2
while temp is not None:
    print(temp.val, end=" ")
    temp = temp.next
```

```
1 2 3 4 5 6 7 8
```

In [ ]: