

Swap two Nodes of LL

Problem Level: Medium

Problem Description:

Given a singly linked list of integers along with two integers, 'i,' and 'j.' Swap the nodes that are present at the 'i-th' and 'j-th' positions.

Sample Input 1:

1 (number of test cases)

3 4 5 2 6 1 9 -1

3 4

Sample Output 1:

3 4 5 6 2 1 9

Approach to be followed:

We need to maintain 4 variables in this problem. What we can do is maintain the p1, c1, p2 and c2 node variables that would store the address of the two nodes to be swapped and their previous nodes. Let's assume that i is smaller than j or assign the smaller value to i and greater one to j.

The basic idea behind swapping the nodes will be:

S. No.	Step	Operation
1.	Take the jth node to ith position.	p1.next = c2
2.	Take the ith node to jth position.	p2.next = c1
3.	Create a temporary node variable to store the node next to c2.	temp=c2.next
4.	Update the next of c2.	c2.next = c1.next
5.	Update the next of c1.	c1.next=temp

Some cases that we need to take care of:

(We are taking i as the smaller number and j as the bigger one, and that all the 4 variables $p1, c1, p2$ and $c2$ can be traversed by the learner)

1. $i==0$

In this case, there wouldn't be any $p1$ variable, and we need to update the head as well. Operations to be performed in this case:

- $p2.next=c1$
- $temp=c1.next$
- $c1.next=c2.next$
- $c2.next=temp$
- $head=c2$

2. $i==0$ and $j==1$

For this case, we have to update the next carefully. Also, head would be updated:

- $head=c2$
- $temp=c2.next$
- $c2.next=c1$
- $c1.next=temp$

3. $j-i=1$

This means both the nodes are alternate, which means $p2$ and $c1$ will be the same node. Carefully update the next variables.

- $p1.next=c2$
- $temp=c2.next$
- $c2.next=c1$
- $c1.next=temp$

4. All other conditions

Swap the nodes like we discussed earlier:

- $p1.next=c2$
- $p2.next=c1$
- $temp=c2.next$
- $c2.next=c1.next$
- $c1.next=temp$

Pseudocode:

```
function swapNodes( head, i, j){
    // Initialize x and y as the minimum and maximum value of i and j.
    Initialize x=minimum of (i,j)
    Initialize y=maximum of (i,j)
    Initialize p1=head
    Initialize p2=head
    Initialize c1=head
    Initialize c2=head
    // If x is 0
    if(x==0):
```

```

c1=head
// If y is 1, that is x is 0 and y-x is 1.
if(y==1):
    c2=head<-next
    head=c2
    Initialize temp=c2<-next
    c2<-next=c1
    c1<-next=temp

// Only x is 0.
else:

    while(y!=1):
        p2=p2<-next
        y--

    c2=p2<-next
    p2<-next=c1
    Initialize temp=c1.next
    c1<-next=c2<-next
    c2<-next=temp
    head=c2

// If y-x is 1. That is both the nodes are alternate.
else if(y-x==1):
    // Traverse to the xth node of the Linked list
    while(x!=1):
        p1=p1.next
        x--

    c1=p1<-next
    c2=c1<-next
    p1<-next=c2
    Initialize temp=c2<-next
    c2<-next=c1
    c1<-next=temp

// All other cases.
else:

    while(x!=1):
        p1=p1<-next
        x--

    c1=p1<-next
    while(y!=1):
        p2=p2<-next
        y--

```

// This traversing step can be done more optimally by using a single loop for p1 and p2

```

c2=p2->next
p1->next=c2
p2->next=c1
Initialize temp=c2->next
c2->next=c1->next
c1->next=temp

```

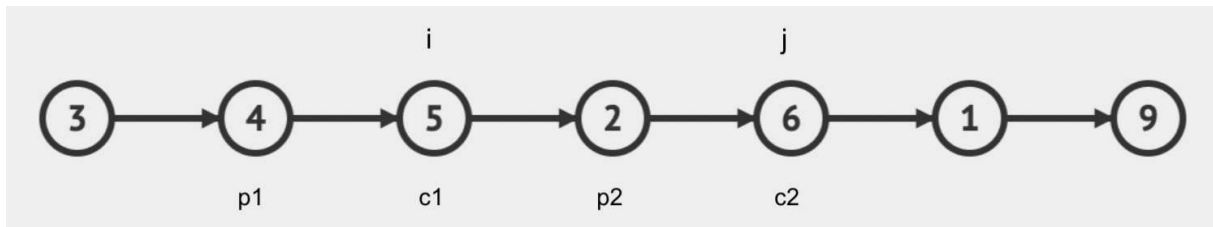
```

return head
}

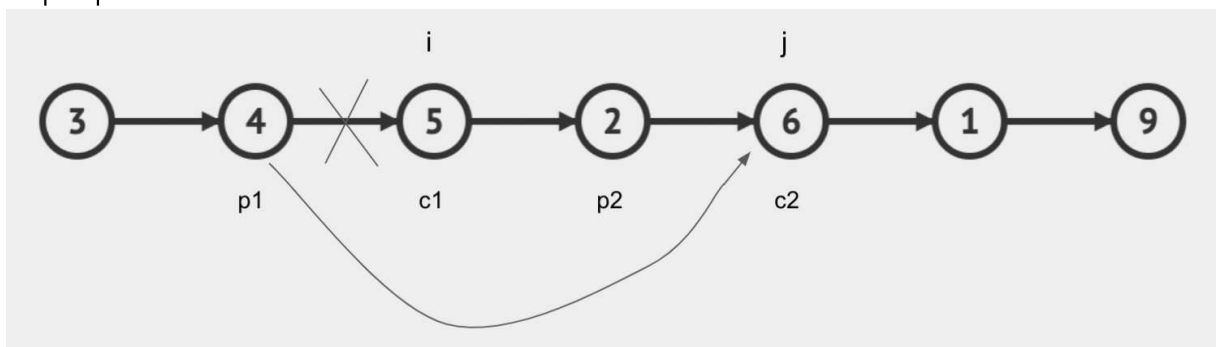
```

Dry Run:

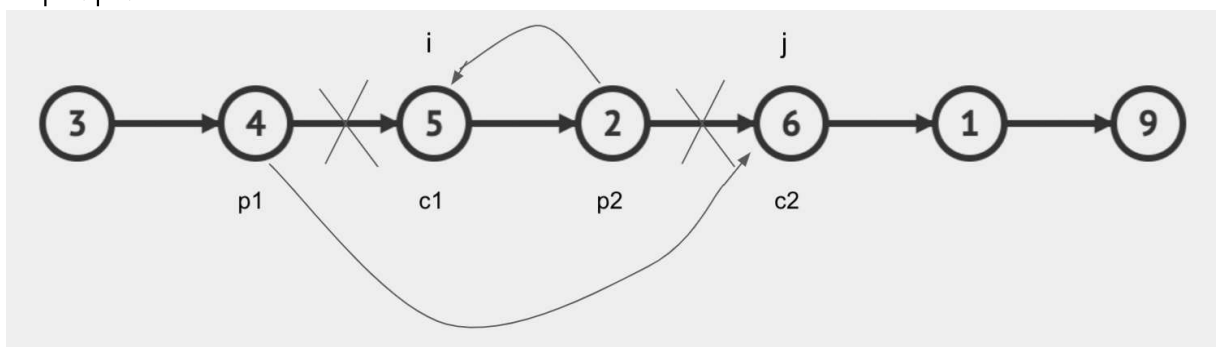
Input:
3 4 5 2 6 1 9 -1
2 4



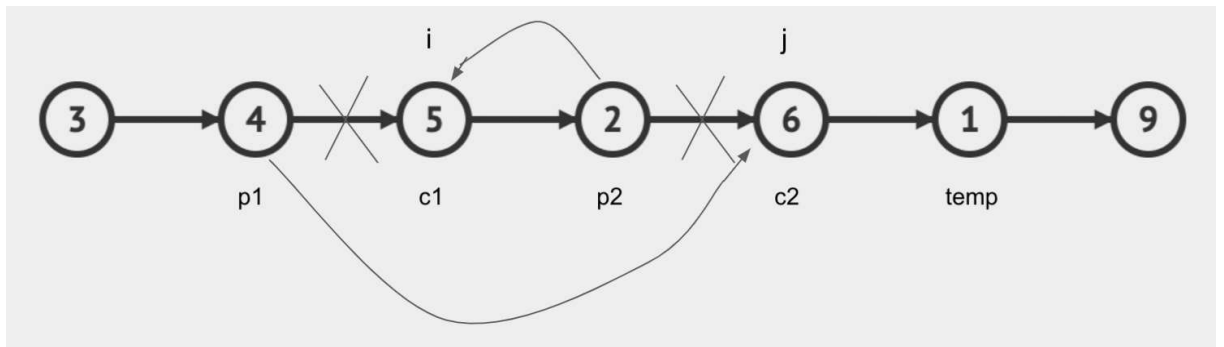
- Step 1: p1.next = c2



- Step 2: p2.next = c1

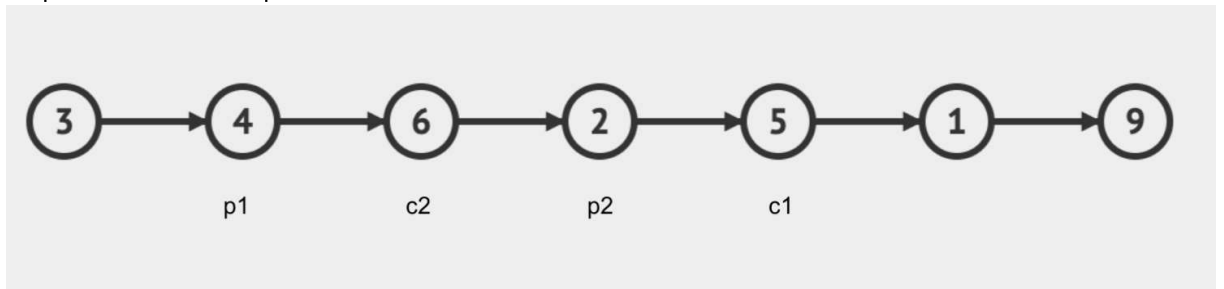


- Step 3: temp = c2.next



4. Step 4: $c2.next = c1.next$

5. Step 5: $c1.next = temp$



Time Complexity: $O(N)$, where **N** denotes the number of nodes in the Linked List.