Swap Nodes in Pairs

Given a linked list, swap every two adjacent nodes and return its head.

For example,
Given `1->2->3->4`, you should return the list as `2->1->4->3`.

Your algorithm should use only constant space. You may **not** modify the values in the list, only nodes itself can be changed.

Solution 1

```
public class Solution {
    public ListNode swapPairs(ListNode head) {
        if ((head == null)||(head.next == null))
            return head;
        ListNode n = head.next;
        head.next = swapPairs(head.next.next);
        n.next = head;
        return n;
    }
}
```

written by whoji original link here

Solution 2

```
public ListNode swapPairs(ListNode head) {
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode current = dummy;
    while (current.next != null && current.next.next != null) {
        ListNode first = current.next;
        ListNode second = current.next.next;
        first.next = second.next;
        current.next = second;
        current.next.next = first;
        current = current.next.next;
    }
    return dummy.next;
}
```

written by tusizi original link here

Solution 3

Three different implementations of the same algorithm, taking advantage of different strengths of the three languages. I suggest reading all three, even if you don't know all three languages.

All three of course work swap the current node with the next node by rearranging pointers, then move on to the next pair, and repeat until the end of the list.

C++

Pointer-pointer `pp` points to the pointer to the current node. So at first, `pp` points to `head`, and later it points to the `next` field of ListNodes. Additionally, for convenience and clarity, pointers `a` and `b` point to the current node and the next node.

We need to go from `*pp == a -> b -> (b->next)` to `*pp == b -> a -> (b->next)`. The first three lines inside the loop do that, setting those three pointers (from right to left). The fourth line moves `pp` to the next pair.

```
ListNode* swapPairs(ListNode* head) {
    ListNode **pp = &head, *a, *b;
    while ((a = *pp) && (b = a->next)) {
        a->next = b->next;
        b->next = a;
        *pp = b;
        pp = &(a->next);
    }
    return head;
}
```

Python

Here, `pre` is the previous node. Since the head doesn't have a previous node, I just use `self` instead. Again, `a` is the current node and `b` is the next node.

To go from `pre -> a -> b -> b.next` to `pre -> b -> a -> b.next`, we need to change those three references. Instead of thinking about in what order I change them, I just change all three at once.

```
def swapPairs(self, head):
    pre, pre.next = self, head
    while pre.next and pre.next.next:
        a = pre.next
        b = a.next
        pre.next, b.next, a.next = b, a, b.next
        pre = a
    return self.next
```

Ruby

Again, `pre` is the previous node, but here I create a dummy as previous node of the head. And again, `a` is the current node and `b` is the next node. This time I go one node further and call it `c`.

To go from `pre -> a -> b -> c` to `pre -> b -> a -> c`, we need to change those three references. Here I chain the assignments, pretty much directly saying "`pre` points to `b`, which points to `a`, which points to `c`".

```
def swap_pairs(head)
    pre = dummy = ListNode.new 0
    pre.next = head
    while a = pre.next and b = a.next
        c = b.next
        ((pre.next = b).next = a).next = c
        pre = a
    end
    dummy.next
end
```

written by StefanPochmann original link here

From Leetcoder.