430. Flatten a Multilevel Doubly Linked List (need to review)

link

这个是我最开始的想法, flat head 但是报错了, 因为next可能是null没有考虑, 但是这样复杂度有点高. 因为实际上是操作尾部来决定, 所以应该写一个return尾部的递归.

```
    public Node flatten(Node head) {
        if(head == null) return head;
        if(head.child != null){
            Node child = flatten(head.child);
            head.child = null;
            Node next = head.next;
            head.next = child;
            child.prev = head;
            while(child.next != null){
                child = child.next;
            }
            child.next = next;
//这里报空指针异常, 因为next可能等于null
            if(next != null)
                next.prev = child;
        }
        flatten(head.next);
        return head;
    }
```

新的写法flatten tail, 这里有一个大问题, tail.next == null没有return.

```
    public Node flatten(Node head){
        if(head != null){
            flattenReturnTail(head);
        }
        return head;

    }
public Node flattenReturnTail(Node head){
    if(head.child == null && head.next == null) return head;
    if(head.child != null){
        Node tail = flattenReturnTail(head.child);
        if(head.next != null)
            head.next.prev = tail;
        tail.next = head.next;
        head.next = head.child;
        head.child.prev = head;
        head.child = null;
//这两行有一个大问题tail.next是null没有直接return tail,极大影响效率
// if(tail.next == null) return tail;
```

```
        if(tail.next != null)
            return flattenReturnTail(tail.next);
    }
    return flattenReturnTail(head.next);
}
```

更好的新的写法:

```
   public Node flatten(Node head) {
        flatLast(head);
        return head;
    }
    private Node flatLast(Node node) {
        if (node == null) return null;
        Node child = node.child, next = node.next, childLast = flatLast(child),
    nextLast = flatLast(next);
        node.child = null;
        if (childLast != null) {
            node.next = child;
            child.prev = node;
            node = childLast;
        }
        if (nextLast != null) {
            node.next = next;
            next.prev = node;
            node = nextLast;
        }
        return node;
    }
```

或者

```
    private Node flattentail(Node cur) {
        if (cur.child != null) {
            Node tail = flattentail(cur.child);
            tail.next = cur.next;
            if (cur.next != null) cur.next.prev = tail;
            cur.next = cur.child;
            cur.child.prev = cur;
            cur.child = null;
            if (tail.next == null) return tail;
            return flattentail(tail.next);
        }
        if (cur.next == null) {
            return cur;
        }
        return flattentail(cur.next);
    }
```

递归版本写法 stack: (偷的, 需要复习)

遇到current.child != null的时候, 先把current.next入栈, 然后current.next = current.child, 然后继续, 当current.next == null的时候说明current走到child尽头了(tail), 需要把之前入栈的元素放到后面.

```
    public Node flatten(Node head) {
        if(head == null) return null;
        Node current = head;
        Stack<Node> stack = new Stack();
        while(current != null){
            if(current.child != null){
                if(current.next != null)
                    stack.push(current.next);
                current.next = current.child;
                current.next.prev = current;
                current.child = null;
            }else if(current.next == null && !stack.isEmpty()){
                current.next = stack.pop();
                current.next.prev = current;

            }
            current = current.next;
        }
        return head;
    }
/*

*/
Initial State
 1---2---3---4---5---6--NULL
         |
         7---8---9---10--NULL
             |
             11--12--NULL
Stack[]

Step 1:
 1---2---3
         |
         7---8---9---10--NULL
             |
             11--12--NULL
Stack[4---5---6--NULL]

Step 2:
 1---2---3
         |
         7---8
             |
             11--12--NULL
Stack[4---5---6--NULL, 9---10--NULL]

Step 3:
1---2---3---7---8---11---12---9---10---NULL
Stack[4---5---6--NULL]

Step 4:
1---2---3---7---8---11---12---9---10---4---5---6---NULL
```

Stack[]

这个题也可以一层一层的加

看上去像O(2n) O(1)

```
    public Node flatten(Node head) {
        if( head == null) return head;
    // Pointer
        Node p = head;
        while( p!= null) {
            /* CASE 1: if no child, proceed */
            if( p.child == null ) {
                p = p.next;
                continue;
            }
            /* CASE 2: got child, find the tail of the child and link it to p.next
    */
            Node temp = p.child;
            // Find the tail of the child
            while( temp.next != null )
                temp = temp.next;
            // Connect tail with p.next, if it is not null
            temp.next = p.next;
            if( p.next != null )  p.next.prev = temp;
            // Connect p with p.child, and remove p.child
            p.next = p.child;
            p.child.prev = p;
            p.child = null;
        }
        return head;
    }
```