Palindrome

September 28, 2017

Question

Implement a function to check if a linked list is a palindrome.

Explanation and Algorithm

There are two different approaches to consider with this problem. Let's first consider the most basic: the iterative approach. The usual way to iterate through a linked list is with two pointers, one that is a node (or more) ahead of the other. Let's call them fast and slow. We could use the stack data structure to collect the first half of the list. Then we make the fast pointer reach the end by the time the slow pointer has reached the middle. If the contents of the stack are the same as the second half of the list, it's a palindrome.

Now, let's consider another approach. Think of the definition of a palindrome. In this case it's a string of numbers that would read the same forwards and backwards. So, based on this, we could have a method to reverse and then clone the linked list. Then, compare this clone to the original list in a separate boolean method. If it is the same, it is a palindrome. If not, it is not a palindrome.

Hints

1. For the iterative approach, think of how you'd utilize pointers that are at different nodes in the array. How can we use this to determine if both halves are the same?
2. How would you deal with an odd number of nodes?
3. With the second method of reversing the list, how would you use this in another method to determine palindrome? Think of all the methods you could use so that the reverse and clone method is useful.
4. Of course, think of potential edge cases.

Code

```
/*Answer 1: Iterative */

boolean isPalindrome(LLNode head){
   LLNode fast = head;
    LLNode slow = head;

   Stack<Integer> stack = new Stack<Integer>();

    while(fast != null && fast.next != null){
      stack.push(slow.data);
      slow = slow.next;
        fast = fast.next
    }

    /* odd number of nodes case */
    if(fast != null){
      slow = slow.next;
    }

    while(slow != null){
      int topNum = stack.pop().intValue();

        if(topNum != slow.data){
         return false;
        }
        slow = slow.next;
    }
    return true;
}
```

```
/*Answer 2: Reversing*/

boolean isPalindrome(LLNode head){
   LLNode reversed = reverseClone(head);
    return isEqual(head, reversed);

}

LLNode reverseClone(LLNode node){
   LLNode head = null;
    while(node != null){
      LLNode n = new LLNode(node.data);
        n.next = head;
      head = n;
        n = n.next;
```

```
    }
    return head;
}

boolean isEqual(LLNode L1, LLNode L2){
   while(L1 != null && L2 != null){
      if(L1.data != L2.data){
         return false;
        }
        L1 = L1.next; //keep incrementing pointers and check for equality
        L2 = L2.next;
    }
    return L1 == null && L2 == null;

}
```

Run time analysis

Run time is $O(n)$ for both solutions because for n nodes in the linked list, they are only visited once. Given this, the nodes are visited n times.

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th Edition by Gayle Laakmann McDowell.