Question 2.8 from CTCI: Loop Detection

September 9, 2017

Question

Given a circular linked list, implement an algorithm which returns node at the beginning of the loop.
Input: $D->E->F->G>H->I$ Output: I

Hints

1. Recall what a circular linked list is. It's a linked list where the head points to the last element.
2. Given this, how could you access that last element? What does it mean for something to be the "beginning of the loop"?
3. Consider some edge cases, like if there's one or no elements. The program would return null.
4. What kinds of pointers would you need to use? We're trying to keep track of when the loop for the circular linked list ends. Therefore, one pointer will have to move faster than the other.
5. At some point, when the pointers will collide at the front of the loop. How can we use this information to perform an algorithm?

Code

```
/* Answer 1 */

LLNode beginningOfLoop(LLNode head){
   LLNode slow = head;
    LLNode fast = head;
```

```
   while(fast != null && fast.next != null){ //find meeting point
      slow = slow.next;
       fast = fast.next.next;
       if(slow == fast){ //collision!
        break;
       }
    }

   if(fast == null || fast.next == null){ //one or no elements
     return null;
    }

    /* Move slow to head and keep fast at intersection. Both pointers
        are k spaces away from loop start, so both will eventually meet
        at loop start.*/
    while(slow != fast){
      slow = slow.next;
        fast = fast.next;
    }
    return fast; //both pointers point to loop start
}
```

Big O analysis

O(n), where n is the size of the list. List has to be traversed n times before rear is accessed.

Sources

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