Linked List question

Q1 Remove\find the duplicate element (CIC 2.1)

Ans. Use hash set , if set is not allowed then O(n2) is complexicity , no better solution.

**Also remember to use delete function.**

**----------------------------------------------**

Q find kth to the last element

How to find the 3rd element from end in linked list in Java  
  
Read more: <http://javarevisited.blogspot.com/2016/07/how-to-find-3rd-element-from-end-in-linked-list-java.html#ixzz4nzHDlmG6>

The challenge here is to solve the problem in just one pass, i.e. you can not traverse the linked list again and you cannot traverse backward because it's a singly linked list. And also you don’t know the size.  
  
Ans 1) Fast and slow integrator, maintain two sets of pointer, one is slow and 2nd is fast one.

Slow moves 1 node when fast moves 3\k nodes.

The slow pointer starts when fast pointer is reached to Nth element e.g. In order to find the 3rd element from last, the second pointer will start when the first pointer will reach the 3rd element.

After that, both pointers will keep moving one step at a time until the first pointer points to null, which signals the end of the linked list. At this time, the second pointer is pointing to the 3rd or Nth node from the last

------------------------------------------------------------------------

**Q2** How to check if a given Linked list is paliondrom

Mistake –

Check starting and end node ? how would you get to the end node , that means you have to travel the list ?

a-b –c-d-c-b-a

Solution

1. Reverse the list , that but again I have to travse the list for it, so not good solution.
2. Push the elements in the stack if you know the size of the list, then compare the
3. If you don’t know the size , use slow and fast runner technique

**-------------------------------------------------**

**Q3** CIC 2.7 .If two list intersects or not

<https://stackoverflow.com/questions/2216666/finding-the-intersecting-node-from-two-intersecting-linked-lists>

Suppose there are two singly linked lists both of which intersect at some point and become a single linked list.

The head or start pointers of both the lists are known, but the intersecting node is not known. Also, the number of nodes in each of the list before they intersect are unknown and both list may have it different i.e. List1 may have n nodes before it reaches intersection point and List2 might have m nodes before it reaches intersection point where m and n may be

* m = n,
* m < n or
* m > n

One known or easy solution is to compare every node pointer in the first list with every other node pointer in the second list by which the matching node pointers will lead us to the intersecting node. But, the time complexity in this case will O(n2) which will be high.

What is the most efficient way of finding the intersecting node?

**Ans**

his takes O(M+N) time and O(1) space, where M and N are the total length of the linked lists. Maybe inefficient if the common part is very long (i.e. M,N >> m,n)

1. Traverse the two linked list to find M and N.
2. Get back to the heads, then traverse |M − N| nodes on the longer list.
3. Now walk in lock step and compare the nodes until you found the common ones.

-----------------------

Q4 Partion

<https://stackoverflow.com/questions/37076648/partitioning-a-linked-list>

**Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x. You should preserve the original relative order of the nodes in each of the two partitions.**

**For example,**

**Given 1->4->3->2->5->2 and x = 3, return 1->2->2->4->3->5.**

Ans

* Keep 2 lists, one for lower nodes and other for greater nodes.
* Iterate the list adding the nodes to the corresponding list.
* Concatenate the lower list with greater list

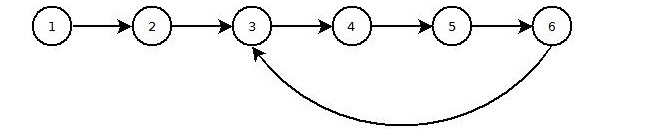
---------------------------------------------------

Q Loop Detection CIC 2.8

What's the best way of writing

boolean hasLoop(Node first)

which would return true if the given Node is the first of a list with a loop, and false otherwise? How could you write so that it takes a constant amount of space and a reasonable amount of time?



idea is to have two references to the list and move them at **different speeds**. Move one forward by 1 node and the other by 2 nodes.

* If the linked list has a loop they will *definitely* meet.
* Else either of the two references(or their next) will become null.

Stregay

Slow and fast runner

Partion