LinkList Problems

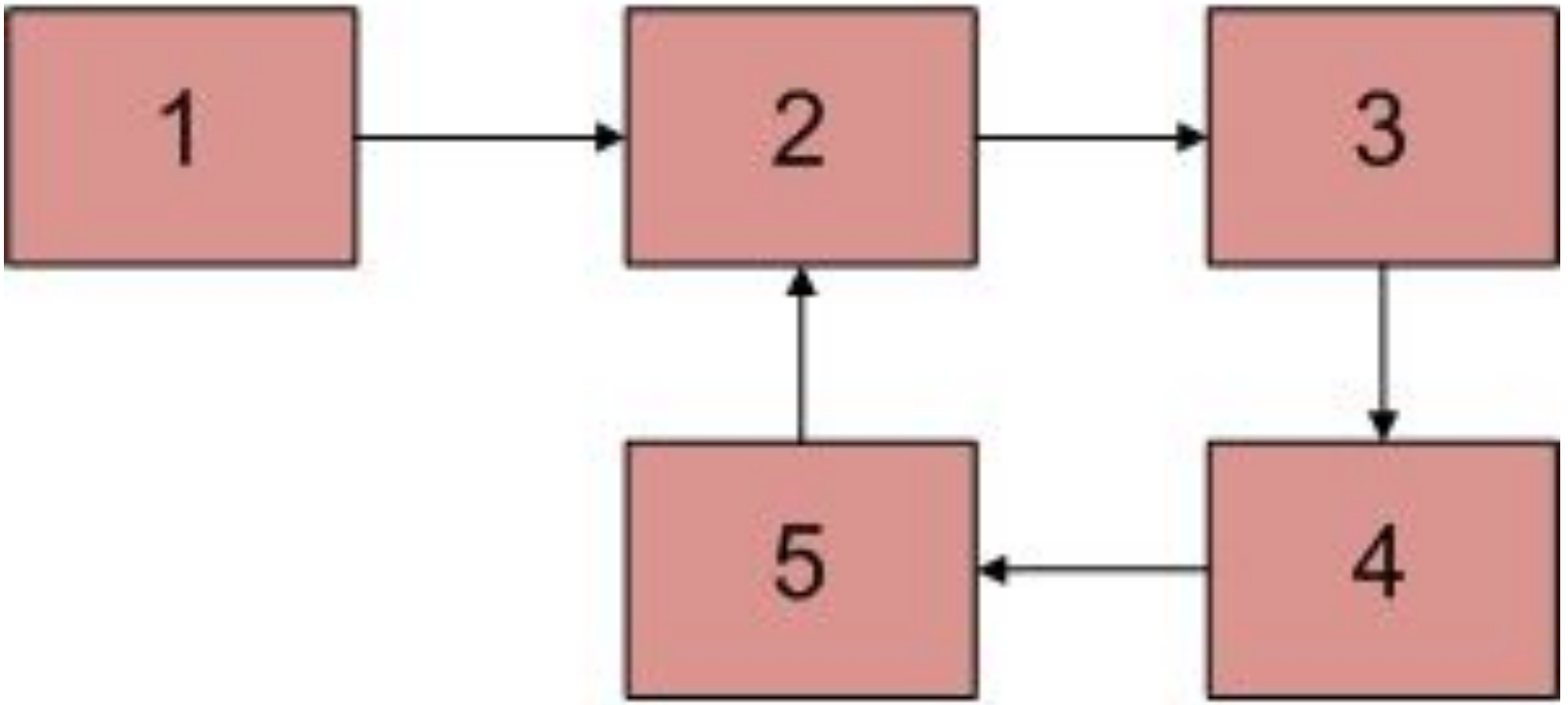
Find the middle of a given linked list

- Given a singly linked list, find middle of the linked list. For example, if given linked list is
- 1->2->3->4->5 then output should be 3.

- If there are even nodes, then there would be two middle nodes, we need to print second middle element. For example, if given linked list is
- 1->2->3->4->5->6 then output should be 4.

- Method 1:
- Traverse the whole linked list and count the no. of nodes. Now traverse the list again till count/2 and return the node at count/2.
- Method 2:
- Traverse linked list using two pointers. Move one pointer by one and other pointer by two. When the fast pointer reaches end slow pointer will reach middle of the linked list.

Detect loop in a linked list



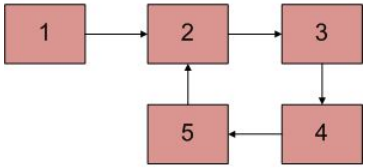
- Use Hashing:
- Traverse the list one by one and keep putting the node addresses in a Hash Table. At any point, if NULL is reached then return false and if next of current node points to any of the previously stored nodes in Hash then return true.

- Mark Visited Nodes:
- This solution requires modifications to basic linked list data structure. Have a visited flag with each node. Traverse the linked list and keep marking visited nodes. If you see a visited node again then there is a loop. This solution works in O(n) but requires additional information with each node.
- A variation of this solution that doesn't require modification to basic data structure can be implemented using hash. Just store the addresses of visited nodes in a hash and if you see an address that already exists in hash then there is a loop.

- Floyd’s Cycle-Finding Algorithm:
- This is the fastest method. Traverse linked list using two pointers. Move one pointer by one and other pointer by two. If these pointers meet at same node then there is a loop. If pointers do not meet then linked list doesn’t have loop.

Find length of loop in linked list

- Floyd's Cycle detection algorithm terminates when fast and slow pointers meet at a common point.
- We also know that this common point is one of the loop nodes (2 or 3 or 4 or 5 in the above diagram). We store the address of this common point in a pointer variable say ptr.
- Then we initialize a counter with 1 and start from the common point and keeps on visiting next node and increasing the counter till we again reach the common point(ptr).
- At that point, the value of the counter will be equal to the length of the loop.



Function to check if a singly linked list is palindrome

Function to check if a singly linked list is palindrome

- METHOD 1 (Use a Stack)
- A simple solution is to use a stack of list nodes. This mainly involves three steps.
- 1) Traverse the given list from head to tail and push every visited node to stack.
- 2) Traverse the list again. For every visited node, pop a node from stack and compare data of popped node with currently visited node.
- 3) If all nodes matched, then return true, else false.
- Time complexity of above method is O(n), but it requires O(n) extra space

- METHOD 2 (By reversing the list)
- This method takes O(n) time and O(1) extra space.
- 1) Get the middle of the linked list.
- 2) Reverse the second half of the linked list.
- 3) Check if the first half and second half are identical.
- 4) Construct the original linked list by reversing the second half again and attaching it back to the first half