Lesson:

Linked Lists One Shot 2

PW SKILLS

Problem 1:Given a linked list and a positive integer k, find the k'th node from the end of the list.
Note: It is guaranteed that k <= length of the linked list.

Code

Steps

1. Create 2 pointers ptr1 and ptr2.
2. Move ptr1 forward till the distance between the two pointers is not equal to k.
3. Make both the pointers move forward till ptr1 reaches the end of the linked list. Since the nodes have a distance of k nodes between them, when ptr1 will be at the end, ptr2 will point to the kth node from the end.

Problem 2: Given the head of a linked list, remove the k-th node from the end of the list and return its head.
Note: It is guaranteed that k <= length of the linked list.

Example 1:
Input:
Number of nodes in list = 5
k = 3
List = 1→2→3→4→5

Output:
1→2→4→5

Example 2:
Input:
Number of nodes in list = 6
k = 2
List = 1→2→3→4→5→6

Output:
1→2→3→4→6

CODE LINK: https://pastebin.com/V96unbZ4

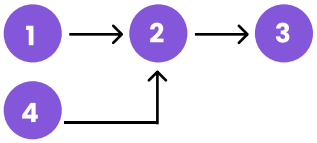
Steps:

1. Use 2 variables to represent 2 nodes that are separated by a distance of k nodes.
2. Make the first variable traverse to the k-th node while keeping the second one at the start of the list.
3. If the first pointer exceeds the end of the list, that means the head is the k-th node from the end. So we return the updated head.

Otherwise traverse the list using both the pointers simultaneously. Whenever the first variable will reach the end, the second variable will point to the node that is just before k-th from the end. So we'll just delete the node after the second variable from the list.

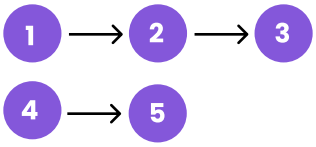
Problem 3: Given the heads of two singly linked-lists headA and headB, return the value of the node at which the two lists intersect. If the two linked lists have no intersection at all, return -1.Note: the values in each node are distinct.

Example 1:
Input:



Output:
2

Example 2:
Input:



Output:
-1
Sol1
CODE LINK: https://pastebin.com/xSPGGW07

Steps:

1. Traverse through the first list.
2. For each node in the first list, traverse through the second list to find if any of the nodes of the second list matches the node from the first list.
   a. If it matches, we have found the intersection point. We can return it.
3. If no common node is found, return -1.

Sol2

CODE LINK: https://pastebin.com/zsbUEzSk

Steps:

1. If either of the nodes is null then there is no intersection.
2. Traverse both the lists simultaneously until either becomes NULL, or both start pointing to the same value.
   a. If node1 reaches the end of the first list, make it the head of the second list.
   b. If node2 reaches the end of the second list, make it the head of the first list.
   c. The traversal will end if we reach the intersection point or both the pointers simultaneously point to NULL.
      i. If node1 is null, return -1.
      ii. Else return the value stored in node1.

Note: This is going to happen in the next iteration of the list itself, because whatever extra nodes were traversed by the pointer of the longer list, will now be traversed by the pointer of the shorter list, and hence, they will be able to get to the common node.

Problem 4: Find the middle element of the given linked list.
Example 1:
Input:
Number of nodes in list = 5

List 1 = 1→3→5→6→2

Output:

5

CODE LINK : https://pastebin.com/E05NCdWx

Steps:

1. Make the fast and slow pointers point to the head of the list.
2. At each iteration make the fast pointer traverse twice the distance as the slow pointer. This will ensure that when the fast pointer reaches the end of the list, the slow pointer will be at the middle of the list.
3. The traversal will stop when the fast pointer either points to the last element of the list or it traverses all the elements of the list.

Problem 5: Delete the middle element of the given linked list.

Example 1:

Input:

Number of nodes in the list = 3

List = 1->2->3

Output

1->3

Code:

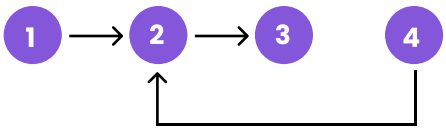
Steps

1. If there is only one node in the list, return null.
2. If there are only 2 nodes in the list, remove the second node of the list.
3. Make a new pointer 'x' to represent the node just before the slow pointer.
4. Normally find the middle element.
5. If fast points to null, i.e. the list contains an odd number of nodes, remove the node that slow points to.
6. Else remove the node next to the slow pointer.

Problem 6: Given head, the head of a linked list. Return true if the linked list has a cycle in it, otherwise return false.

Example 1:

Input:



Output:

true

Example 2:

Input:

Output:
false
CODE LINK: https://pastebin.com/0qUMcfRR

Steps:

1. Make 2 pointers slow and fast. The fast pointer moves through the list twice as fast as the slow pointer.
2. If the pointers meet at some point, there is a loop in the list, otherwise there is not.

Intuition- let's say 2 people are running in a circular track, one person is running slowly and another person is running faster(2 times the speed of first person)
After a certain period of time person 2 again meet or overtake person 1,
In that case we can conclude that the track is circular (replace running track with our Linked list)

Problem 7: Given 2 sorted linked lists, merge them into 1 singly linked list such that the resulting list is also sorted.
Example 1:
Input:
Number of nodes in list 1 = 4
Number of nodes in list 2 = 3
List 1 = 1→3→5→6
List 2 = 2→4→7

Output:
1→2→3→4→5→6→7

CODE LINK : https://pastebin.com/Jwi48SEV

Steps:

1. We start by making a dummy node. This node represents the head of the merged list.
2. We make 2 new variables and assign them to the respective heads of the 2 lists.
3. We traverse the list using the 2 variables till we reach the end of one of the lists.
   a. Add the node with the smaller value into the merged list and update the variable of that list whose node is chosen to the next node of that list.
4. Add the remaining elements of the list that isn't fully traversed, into the merged list.
5. Return the list, excluding the dummy node.

Problem 8:Given a linked list , split it into two lists such that one contains odd values while other contains even values.

Code
Steps

1. Create 4 nodes to represent the current element and head of both the linked list with odd values and even values with dummy values initially.
2. Traverse through the linked list. If the current element is odd, add it to the odd list, else add it to the even list.
3. Return the head of both the linked lists as an array list after removing the dummy nodes.

Problem 9: Given the head of a singly linked list. Print the reversed list.
Example 1:

Input:
Number of nodes = 5
List = 1→2→3→4→5

Output:
5 4 3 2 1

CODE LINK: https://pastebin.com/Fn5ku5wy

Steps:
To traverse a linked list in reverse order, use recursion.

1. Call the function for the head of the list linked.
2. Traverse the list till you reach the end.
3. Print the values while backtracking.

Problem 10: Given the head of a singly linked list, reverse the list, and return the reversed list.
Example 1:
Input:
Number of nodes = 5
List = 1→2→3→4→5

Output:
5→4→3→2→1

CODE LINK: https://pastebin.com/GcWUkkqN

Steps:

1. Make 3 variables to represent the current node, the node before it and the node after it. Initialize the current node to head and previous node to NULL.
2. Loop till you don't reach the end of the loop i.e. head is not pointing to NULL.
3. At each iteration-
   a. Make next node point to the node after the current node.
   b. Make the current node point to the node before it.
   c. Make your previous node and current node move forward by one step.
4. Return the new head of the list, which is pointed by the previous node.

Problem 11: Given head, the head of a linked list, determine if the linked list is a palindrome or not.
Example 1:
Input:
Number of nodes in list = 4
List = 1→3→3→1
Output:
1

CODE LINK: https://pastebin.com/B5sb3w15

Steps:

1. Find the middle part of the linked list.
2. Separate the linked list into 2 linked lists from the middle.
3. Reverse the second half of the linked list.
4. Check if the 2 linked lists are identical or not.
5. In case of an odd length palindrome, we can skip the last value of the longer linked list.

Problem 12 : In a linked list of size n, where n is even, the ith node (0-indexed) of the linked list is known as the twin of the (n-1-i)th node, if 0 <= i <= (n / 2) - 1.
For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4.
The twin sum is defined as the sum of a node and its twin.
Given the head of a linked list with even length, return the maximum twin sum of the linked list.
The first line of input contains n, the size of the list.
The second line of input contains the elements of the linked list.

Input

4
2 4 5 9

Output

11

Solution:

Code

Explanation:

1. Find the middle node of the list.
2. Reverse the second half of the list.
3. Using 2 pointers, find the max value of the sum of corresponding nodes' values.

Note: To prevent the change in the original list, you may make a dummy list first and then apply these moves.

Problem 13: Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list.
Example 1:
Input:
Number of nodes in list = 4
List = 1→2→3→4

Output:
1→3→2→4

CODE LINK : https://pastebin.com/KHrYXNcH

Steps:

1. Make 3 additional points to represent the elements at odd positions, the elements at even positions and the head of the list in which we have even indexed elements.
2. Using the odd and even pointers traverse the list.
   a. Make the odd indexed element point to the next odd indexed element that is 2 positions ahead of it.

1. Do the same with even indexed elements.
2. Update the pointers.
3. Make the last element of the odd-indexed list point to the head of the even-indexed list.

Problem 14: Leetcode 138
A linked list of length n is given such that each node contains an additional random pointer, which could point to any node in the list, or **null**.

Construct a **deep copy** of the list. The deep copy should consist of exactly **n brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the **next** and **random** pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. None of the pointers in the new list should point to nodes in the original list.

For example, if there are two nodes **X** and **Y** in the original list, where **X.random --> Y**, then for the corresponding two nodes x and y in the copied list, **x.random --> y.**

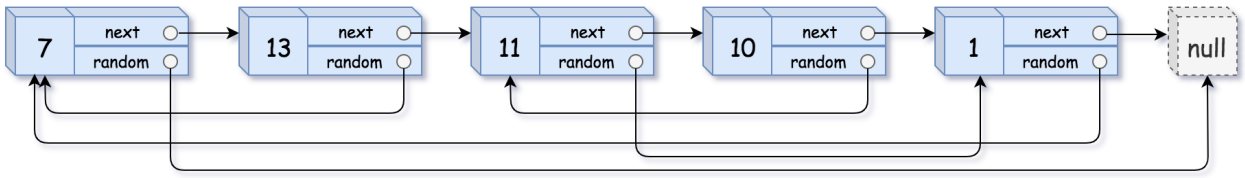
Return *the head of the copied linked list.*

The linked list is represented in the input/output as a list of n nodes. Each node is represented as a pair of **[val, random_index]** where:

- **val:** an integer representing **Node.val**
- **random_index:** the index of the node (range from 0 to n-1) that the **random** pointer points to, or **null** if it does not point to any node.

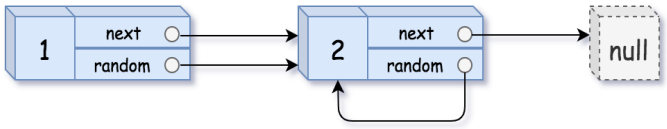
Your code will **only** be given the **head** of the original linked list.

Example 1:



Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]

Example 2:



Input: head = [[1,1],[2,1]]
Output: [[1,1],[2,1]]

Code
Steps

1. Make a copy of each node,and link them together side-by-side in a single list.
2. Assign random pointers for the copy nodes.
3. Restore the original list, and extract the copy list.

Upcoming Class Teasers

- Doubly and Circular Linked List