Lesson:

Patterns of questions based on singly linked list - 2

PW SKILLS

Pre-Requisites

- Singly linked list

List of Concepts Involved

- Patterns of questions based on singly linked list

Pattern: Pointers

Problem 1: Given 2 linked lists, Tell if they are equal or not. Two linked lists are equal if they have the same data and the arrangement of data is also the same.

Example 1:

Input:

Number of nodes in list 1 = 5
Number of nodes in list 2 = 5
List 1 = 1→2→3→4→5
List 2 = 1→2→3→4→5

Output:

1

Example 2:

Input:

Number of nodes in list 1 = 5
Number of nodes in list 2 = 5
List 1 = 1→2→3→4→5
List 2 = 1→2→3→5→5

Output:

0

Code

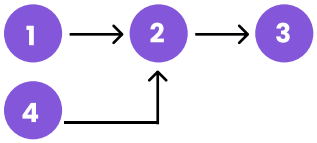
Steps:

1. Use 2 pointers to represent the heads of the respective lists.
2. Loop till either of them points to NULL.
   a. If the values of the pointers are not equal, return false.
   b. Update both the pointers.
3. If both pointers point to NULL then the lists are equal, otherwise not.

Problem 2: Given the heads of two singly linked-lists headA and headB, return the node at which the two lists intersect. If the two linked lists have no intersection at all, return null.

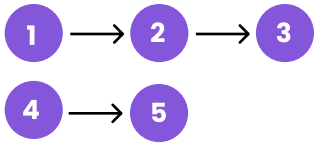
Example 1:

Input:



Output:
2

Example 2:
Input:



Output:
-1

Sol 1

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 NULL.

Sol2:

Code

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.
3. If ptr1 reaches the end of the first list, make it point to the head of the second list.
4. If ptr2 reaches the end of the second list, make it point to the head of the first list.
5. The traversal will end if we reach the intersection point or both the pointers simultaneously point to NULL. in either case return whatever ptr1 is pointing to.

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 3: 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

Code

Steps:

1. Use 2 pointers to represent 2 nodes that are separated by a distance of k nodes.
2. Make the first pointer 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.
4. Otherwise traverse the list using both the pointers simultaneously. Whenever the first node will reach the end, the second node will point to the node that is just before k-th from the end. So we'll just delete the node after the second pointer from the list.

Problem 4: 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

Steps:

1. We start by making a dummy node. This node represents the head of the merged list.
2. We make 2 new pointers and make them point to the respective heads of the 2 lists.
3. We traverse the list using the 2 pointers 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 pointer of that list whose node is chosen.
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.

Pattern: Merging multiple linked list

Problem 1: You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.
Example 1:
Input:
Number of lists = 2
Number of nodes in list 1 = 4
List 1 = 1→3→5→6
Number of nodes in list 2 = 3
List 2 = 2→4→7
Output:
1→2→3→4→5→6→7

Code link: https://pastebin.com/5Uwer7Fx

Steps:

1. Instead of thinking of this problem as merging k linked lists, focus only on merging the first 2 lists.
2. Till the vector of lists has more than 1 list present inside it, take the first 2 lists and apply the merging function on it.

Pattern: Slow fast pointer

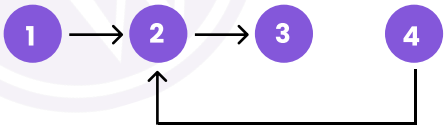
Problem 1: Find the middle element of the given linked list.
Example 1:
Input:
Number of nodes in list = 5
List 1 = 1→3→5→6→2
Output:
5

Code Link: https://pastebin.com/sRXs6RZV

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.
4. it traverses all the elements of the list.

Problem 2: 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: https://pastebin.com/h0Td5Ex0

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 3: 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/x36ewc7Y

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.

Pattern: Rearrangement of nodes in a list

Problem 1: Given the head of a linked list, rotate the list to the right by k places.
Input:
Number of nodes in list = 4
List = 1→3→5→6
K = 2
Output:
5→6→1→3

Input:
Number of nodes in list = 10
List = 1→3→5→6→7→10→13→15→16→17
K = 2
Output:
16→17→1→3→5→6→7→10→13→15

Code Link: https://pastebin.com/1wXFa2Ly

Steps:

1. To calculate the effective traversal of the list, we need to k mod n, the size of the list because after every n-th rotation, the linked list comes back to its original state.
2. If the value of k is 0, then we can return the current head of the list i.e. no rotation needed.
3. Otherwise, traverse the list till you reach the (n - k)th node.
4. Make the (n - k + 1)th node the new head of the list and make the (n - k)th node point to NULL as it will be the new end of the list.
5. Make the end of the second half point to the previous head of the list to complete the linked list.
6. Return the new head.

Problem 2: 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/JcKGhxd6

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

Problem 3: You are given the head of a singly linked-list. The list can be represented as:
L0 → L1 → … → Ln - 1 → Ln
Reorder the list to be on the following form:
L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …
Example 1:
Input:
Number of nodes in list = 4
List = 1→2→3→4
Output:
1→4→2→3

Code Link: https://pastebin.com/GjmZHrhZ

Steps:
1. Find the middle of the list and divide the linked list into 2 parts from the middle.
2. Reverse the latter half of the list.
3. Merge the two lists in the given fashion.

Problem 4: Given a linked list, swap every two adjacent nodes and return its head. You may not modify the values in the list's nodes. Only nodes themselves may be changed.
Input:
Number of nodes in list = 4
List = 1→3→5→6
Output:
3→1→6→5

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

Output:
2->1->4->3->5
Code link: https://pastebin.com/yK8C4yQ1

Steps:

1. Make a recursive function that takes into consideration the first 2 elements of the remaining list.
2. If we are at the end of the list or we have only 1 element to consider, we return.
3. Otherwise
    a. Store the second node in a temporary variable.
    b. Make the first element point to the first element of the remaining list after updating.
    c. Make the second element point to the first element.
    d. Return the second element as the first element of the updated list.

Problem 5: Given the head of a linked list, apply merge sort on it.
Example 1:
Input:
Number of nodes in list = 4
List = 6→3→1→5
Output:
1→3→5→6
Code Link: https://pastebin.com/dm9awPWz

Steps:

1. Just like in the case of arrays, we are going to use a recursive function. This function will take the head of the linked list and return the sorted list.
2. Using the slow-fast pointer, we'll find the middle of the linked list and divide the list into 2 lists.(discussed in patterns of questions related to linked list - 3)
3. Recursively call the merge_sort function for both the lists. It will return sorted lists.
4. Merge the 2 sorted lists with the help of 2 pointers approach. (discussed in patterns of questions related to linked list - 2)

Upcoming Class Teasers

- Doubly and Circular Linked List