Patterns of questions based on singly linked list - 2

Assignment Solutions

PW SKILLS

1. You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and store the sum in the first linked list.
The first line of input contains n and m, the size of the first and the second linked list respectively.
The second line contains the elements of the first linked list.
The third line contains the elements of the second linked list.
Input
3 4
1 2 3
1 2 3 4

Output:
2→4→6→4→NULL

Ans: Code

Explanation:

a. We start by initialising 2 nodes that point to the heads of the 2 lists.
b. Loop through the lists till the first list is empty.
   i. The sum will be equal to the sum of the values in the two linked lists plus the carry. If the second node is pointing to null, then we take the value of the second node to be 0. This will happen when the second list will have less number of nodes than the first list.
   ii. Store the unit digit of the sum in the node of the first list, and update the value of carry.
c. Loop through the second list in case it has nodes unvisited. This will happen if the second list has more nodes than the first list. The steps will be the same as in the first loop. The only difference is that now we will have to create new nodes in the first list to store the additional digits.
d. At the end check if there is any carry value remaining. If yes, add it at the end of the list.

2. Given the head of a sorted linked list, delete all nodes that have duplicates, leaving only distinct numbers from the original list. Return the linked list sorted as well.
The first line of input contains n, the number of elements in the linked list.
The second line of input contains the elements of the linked list.
Input
6
1 1 2 2 3 4

Output:
3→4→NULL

Ans: Code

Explanation:

a. Create a dummy node and attach it before the head of the linked list.
b. Create a node to represent the node before the current node.
c. Traverse the list
   i. If the next node contains the same value as the current node then start deleting the nodes till they in order till have the same value as the current node. Also delete the current node.
   ii. Else, just update the value of previous and current node.

3. Given the head of a linked list and a value x, partition it such that all nodes with value less than or equal to x come before nodes with value greater than x.
You should preserve the original relative order of the nodes in each of the two partitions.
The first line of input contains n, the size of the linked list, and x.
The second line of input contains the elements of the linked list.
Input
5 2
1 5 2 3 1

Output
1→2→1→5→3→NULL

Ans: Code

Explanation:

a. Create 2 new dummy nodes. One will create a linked list of all nodes with value less than or equal to x and the other will create a linked list with the nodes with value greater than x.
b. Traverse the list and depending upon the value at the current position, attach the current node to one of the two lists.
c. Attach the head of the second linked list at the end of the first linked list.

4. Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list.
The first line of input contains n, the size of the linked list.
The second line of input contains the elements of the linked list.
The third line of input contains the value for left and right respectively.
Input
5
1 2 4 10 3
3 5

Output
1→2→3→10→4→NULL

Ans: Code

Explanation:

a. Add a dummy node before at the start of the linked list.
b. Traverse the list to find the element that is just before the part of the list that is to be reversed, let's call it prevToFirst.
c. Reverse the specific part of the list.
d. Make the tail of the reversed part point to the remaining list.
e. Make the next pointer of the prevToFirst node point to the reversed part of the list.

5. Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.
Return the decimal value of the number in the linked list.
The most significant bit is at the head of the linked list.
The first line of input contains n, the size of the linked list.
The second line of input contains the elements of the linked list.
Input
31 0 1

Output
5

Ans: Code

Explanation:
a. Make a global variable to keep track of the current power of 2.
b. Make a head recursive function to traverse the list in reverse order.
c. If we are at the last node of the list then we will simply calculate our answer and return that answer. This will be our base case.To calculate the answer for the current call we multiple the current bit with the corresponding power of 2.
In the base case, the value of power is 1.
d. If it is not the base case then first we call the recursive function to get the result of the remaining list and then add the current result to the ans.

6.Given the head of a linked list, return the node where the cycle begins. If there is no cycle, return null.
Ans: Code

Explanation:
Define two pointers slow and fast. Both start at the head node, fast is twice as fast as slow. If it reaches the end it means there is no cycle, otherwise eventually it will eventually catch up to a slow pointer somewhere in the cycle.
Let the distance from the first node to the node where the cycle begins be A, and let say the slow pointer travels A+B. The fast pointer must travel 2A+2B to catch up. The cycle size is N. Full cycle is also how much faster the pointer has traveled than slow pointer at the meeting point.
A+B+N = 2A+2B
N=A+B
From our calculation, a slow pointer traveled exactly a full cycle when it met a fast pointer, and since originally it traveled A before starting on a cycle, it must travel A to reach the point where the cycle begins! We can start another slow pointer at the head node, and move both pointers until they meet at the beginning of a cycle.

7.Given the head of a singly linked list, sort the list using insertion sort, and return the sorted list's head.
The first line of input contains n, the size of the list.
The second line of input contains the elements of the linked list.
Input5
2 5 3 8 10

Output
2→3→5→8→10->NULL

Ans: Code

Explanation:
Insertion sort in the linked list works the same way as in case of an array. The only difference here is that in a linked list, we have nodes that we need to move.

8.Given the head of a singly linked list and an integer k, split the linked list into k consecutive linked list parts.
The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being null.
The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.
Return an array of the k parts.
The first line of input contains n (the size of the list) and k.
The second line of input contains the elements of the linked list.
Input
5 3
1 2 3 4 5

Output
[1→2→NULL] [3→4→NULL] [5→NULL]

Ans: Code

Explanation:

a. Calculate the length of the linked list and divide it by k. This will give the number of nodes each part will contain if we decide to divide the nodes evenly, let's call it numNodes.
b. Calculate the number of nodes that will be left, let's call it extras. It represents the number of sublists that will have an additional node to account for the remaining nodes.
c. Start the traversal of the linked list. At each iteration-
    i. Generate a sublist to be added into the answer.
    ii. Add the head of the sublist into the answer.
    iii. Find the last node of the sublist with the help of numNodes.
    iv. If we can add an extra element into the list, then add it and reduce the value of extras by one.
    v. Make the last node point to null.
d. If the number of nodes is less than k then while the len remains less than k, we can add NULL to our answer vector.

9.You are given the head of a linked list, which contains a series of integers separated by 0's. The beginning and end of the linked list will have Node.val == 0.
For every two consecutive 0's, merge all the nodes lying in between them into a single node whose value is the sum of all the merged nodes. The modified list should not contain any 0's.
Return the head of the modified 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
6
0 3 1 0 2 0

Output
4->2->NULL

Code

Explanation:

a. Traverse the list starting from the second element i.e. the first non-zero element. At each iteration-
   i. Calculate the sum of the sublist that has non-zero elements.
   ii. Replace the value of the current node with the sum.
   iii. Move onto the next non-zero sublist.

10.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

Ans: Code

Explanation:

a. Find the middle node of the list.
b. Reverse the second half of the list.
c. 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.