Linked List

What is a Linked List and What are its types?

A linked list is a linear data structure (like arrays) where each element is a separate object. Each element (that is node) of a list is comprising of two items – the data and a reference to the next node.

Types of Linked List :

Singly Linked List: In this type of linked list, every node stores address or reference of next node in list and the last node has next address or reference as NULL. For example 1->2->3->4->NULL

Doubly Linked List: Here, here are two references associated with each node, One of the reference points to the next node and one to the previous node. Eg. NULL<-1<->2<->3->NULL

Circular Linked List: A Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list. Eg. 1->2->3->1 [The next pointer of last node is pointing to the first]

The main difference between the singly linked list and the doubly linked list is the ability to traverse. In a single linked list, the node only points towards the next node, and there is no pointer to the previous node, which means you can not traverse back on a singly linked list. On the other hand, the doubly linked list maintains two pointers, towards the next and previous node, which allows you to navigate in both directions in any linked list.

How do you detect a loop in a linked list?

Traverse the list one by one and keep putting the node addresses in a HashTable. 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.

Can Binary Search be used for linked lists?

Since random access is not allowed in a linked list, we cannot reach the middle element in O(1) time. Therefore Binary Search is not possible for linked lists.

In order to find length of linked list we need to first traverse through linked list till we find last node, which is pointing to null, and then in second pass we can find middle element by traversing only half of length. In order to find middle element of linked list in one pass, you need to maintain two-pointer, one increment at each node while other increments after two nodes at a time, by having this arrangement, when first pointer reaches end, second pointer will point to middle element of linked list.

How to find if linked list has a loop?

We can use two pointer approach to solve this problem. If we maintain two pointers, and we increment one pointer after processing two nodes and other after processing every node, we are likely to find a situation where both the pointers will be pointing to same node. This will only happen if linked list has loop.

STEP 1: Take 2 pointers ptr1 and ptr2, both pointing at the start node initially.
STEP 2: Move ptr1 forward one node at a time and move ptr2 forward two nodes at same time.
STEP 3: ptr2 is running at double speed, so definitely it will be ahead of ptr1,
If ptr2 encounters NULL, it means there is no loop in a Linked list and stop execution.
If linked list contains loop, then ptr2 at some point will enter in the loop. some time later ptr1 will also enter in the loop.

(From our jogging track example, we have already seen that)

If they both meet, it means Linked list contains loop and stop execution.

How to find 3rd element from end in a linked list in one pass?

If we apply same trick of maintaining two pointers and increment other pointer, when first has moved up to 3rd element, than when first pointer reaches to the end of linked list, second pointer will be pointing to the 3rd element from last in a linked list.

How do you find the middle element of a singly linked list in one pass?

In the case of singly LinkedList, each node of Linked List contains data and pointer, which is the address of next Linked List and the last element of Singly Linked List points towards the null. Since in order to find middle element of Linked List you need to find the length of LinkedList, which is counting elements till end i.e. until you find the last element of Linked List.

By using two pointers, incrementing one at each iteration and other at every second iteration. When first pointer will point at end of Linked List, second pointer will be pointing at middle node of Linked List.

This is the simple way where two loops are used. The outer loop is used to pick the elements one by one and inner loop compares the picked element with rest of the elements.

How do you find the length of a singly linked list?

In the singly linked lists the last element will point to “null” element, So you will use a counter and increment till it reaches the end of the element. Once you reach at the end of linked list, the value of counter would be equal to the total number of elements encountered i.e. length of the elements.

How do you find the third node from the end in a singly linked list?

The algorithm is also known as tortoise and hare algorithm because of the speed of two pointers used by the algorithm to traverse the singly linked list. If you remember the algorithm uses two pointers, fast and slow. The slow pointer starts when the fast pointer is

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. You have solved the problem, you can either print the value of node or return reference to the caller based upon your requirement.

How do you check if a given linked list contains a cycle? How do you find the starting node of the cycle?

A singly list can only move in one direction, towards end. Now, let's come back to this question. Good thing about this question is that, it can also be solved by using two pointer approach discussed in How to find middle element of linked list in single pass. If a linked list contains a loop or cycle it is known as circular or cyclic linked list. As I said we can use two pointer approach to check if a linked list is circular or not. Two pointers, fast and slow is used while iterating over linked list. Fast pointer moves two nodes in each iteration, while slow pointer moves to one node. If linked list contains loop or cycle than both fast and slow pointer will meet at some point during iteration. If they don't meet and fast or slow will point to null, then linked list is not cyclic and it doesn't contain any loop.

How do you reverse a linked list?

which then becomes the new head of linked list.

With Recursion & Loop: A method reverses linked list using the three-pointers approach and using loops, that's why it is called iterative solution. It traverses through the linked list and adding nodes at the beginning of the singly linked list in each iteration. It uses three reference variables (pointers) to keep track of previous, current, and next nodes.

How do you find the sum of two linked lists using Stack?

1) Calculate sizes of given two linked lists.
2) If sizes are same, then calculate sum using recursion. Hold all nodes in recursion call stack till the rightmost node, calculate sum of rightmost nodes and forward carry to left side.
3) If size is not same, then follow below steps:
….a) Calculate difference of sizes of two linked lists. Let the difference be diff
….b) Move diff nodes ahead in the bigger linked list. Now use step 2 to calculate sum of smaller list and right sub-list (of same size) of larger list. Also, store the carry of this sum.
….c) Calculate sum of the carry (calculated in previous step) with the remaining left sub-list of larger list. Nodes of this sum are added at the beginning of sum list obtained previous step.