15/07/21

Date: Page:

Q25. Reverse a linked list

→ given linked list, need to reverse it

→ create dummy node assigning it to null

→ create next node pointing to next of head node

→ head.next = dummy.

afterwards force each iteration.

→ dummy = head, head = next, next = head → next

→ head → next = dummy

→ have will this end, while head != NULL.

→ d = head.

TC = O(N)

SC = O(1)

code.

```
ListNode * reverse (ListNode* head){
    ListNode * newh = NULL;
    while (head != NULL) {
        ListNode* next = head -> next;
        head -> next = newh;
        ~~head = next~~
        newh = head;
        head = next;
    }
    return newh;
```

Q26. Middle of the linked list.

Approach 1 count number of nodes

$n = \frac{5}{2} = 2+1 = 3$ → number of iterations

and return linked list node

T = O(N) + O(N/2)

S = O(1)

approach 2 : Tortorse Method
use of 2 nodes
⇒ initial → s = head f = head
→ move slow by 1 and fast by 2
→ when fast completes iterations, we return slow tortoise

node

→ reduced TC
TC → O(N/2)
SC → O(1)

next

Q27 Merge 2 linked list

Approach 1 external space approach
$l_1 = l_1$ - head $l_2 = l_2$ - head
dm = 0
TC (O($n_1 + n_2$))
SC = O($n_1 + n_2$)

Approach 2 Inplace.
→ dont create new nodes
→ 2 variables l_1 and l_2
→ smaller value will be pointed at l_1
→ and larger one l_2
→ take res = l_1 first
~~→ create~~

Q29 Delete Node in Linked list
→ given, only node My approach
→ assign next val values to current node
→ do this untill next's next of ~~current~~ node is not null
→ afterwards assign val and next = null.

Steiner's approach
node → val = node → next → val.
node → next = node → next → next.

Q28. Remove Nth node from end of linked list

my approach

~~find size of linked list~~

~~remove $(n+1-i)^{th}$ node. by node~~

if (size == 1), return NULL

if (size == n), return head → next

find d = size − n − 1.

go to d^{th} node and then noded → next = noded → next → next

TC = O(2N) S = O(1)

Approach2 → optimized.

→ create dummy node

→ take fast & slow

→ move fast ~~by~~ to n th ~~step~~ position

→ ~~as~~ move both by 1 step until fast → next = null.

TC = O(N)

SC = O(1)

Q30. Add 2 numbers given as a linked list Amazon

→ create dummy node

→ create temp node pointing to dummy

→ iterate over both linked list as well as carry

→ if sum of 2 nodes exceeds 10, carry = 1.

→ and ~~temp node val~~ sum is sum % 10.

→ create a new list node with val = sum

→ temp shifts to temp → next

TC = O(max(n1, n2))

SC = O(n)

Q31. Intersection of 2 given linked list
→ find node at which intersection of both ~~the linked list exist~~
→ else return null.

Approach 1 Extreme brute force
linearly Iterate over list 1 and at each iteration for list 1, also linearly iterate over list 2. Do this until we find the intersection of linked list or list 1 gets null.
→ if list 1 gets to null, we are returning null.

TC → O(M×n) ☹

Approach 2. use hashing
→ traverse ll 1 → hash the node address
→ traverse ll2 → ~~moment you search the~~ until you get the situation when address is already hashed.

TC → O(N+M)
SC → O(N)

Approach 3
→ find length of both linked list
→ find diff $l_1 - l_2$.
→ move shorter node ll by 1 and longer node ll by $(l_1 - l_2)$
→ when both collide, we have the node
~~→ if they don~~

TC → O(M) + O(M−N)
O(N)
= O(1)

Not Preffered because of long code

Approach 4
→ $d_1 = l_1.head$ → $d_2 = l_2.head$.
→ start make them move simultaneously.
→ moment any dummy node reaches null, make them head of another ll.

→ if there's no intersection, then both are null at same time

Q22. Detect a cycle in a linked list

Approach 1. (hashing: Naive approach)
create hashtable and ~~go~~ return address ~~until~~ that you were about to add but was already present in hashtable

Approach 2. (tortoise method)
initial: fast = head and slow = head.
now check whether fast == slow where fast takes 2 jumps & slow 1 jump, if it does then we have cycle
→ if it doesn't then return NULL

Q33 Check whether given linked list ~~would~~ is palindrome

Approach 1 Naive approach.
→ putting val in array of string
→ checking whether the array is palindrom or not.

$TC = O(N)$

$SC = O(N) + O(N)$

Approach 2.
→ find middle of linked list
→ reverse linked list from slow ptr + 1 to last
→ take another ptr at begin and linearly travel until slow is null and check if slow and another ptr are same
→ moment they are unequal return false

$TC = O(\frac{n}{2}) + O(\frac{n}{2}) + O(\frac{n}{2}) + O(\frac{n}{2}) + 1$

$SC = O(1)$

334. Reverse nodes in kth group.

→ Reverse linked list in groups of k.

① ~~dea~~ solve problems group wise

② reverse k-1 links.

③ count the number of elements in linked list

④ we will has n/k groups

⑤ create dummy node, and assign pre to it
dummy → next = head.

⑥ cur = ~~dummy~~ next ⑦ next = cur → next

⑧ reverse link
→ reverse link cur → next, and cur.next = cur.next.next
→ next.next = cur ⑨ pre → next = next
→ next = cur → next.

⑨ repeat ⑧ for k-1 steps #IMP

#code

```
~~while~~ *pre = dummy
while (count >= k) {
    cur = pre → next;
    nex = cur → next;
    for (int i = 1; i < k; i++) {
        cur → next = nex → next;
        nex → next = ~~cur~~ pre → next.
        pre → next = nex
        nex = cur → next
    }
    pre = cur
    count -= k;
}
```

Q35 Find the starting point of cycle

Approach 1 My approach. (Brute force)

→ save addresses of nodes in a hashtable and check which inserting node is already present

→ the address to be pushed if already present in linked list then ~~we~~ the node is starting point of cycle of linked.

→ if we reach null, then we conclude there is no cycle

TC ⇒ O(N)

SC → O(N) ~~optimize~~

Approach 2

① find out collision pt

- take slow & fast ptr. move 2ptr by 1 & fast by slow
- they are bound to collide

② Find starting pt of cycle

- take entry ptr ~~and~~ & head, move entry & slow simultaneously until they collide
- where they collide, ~~they~~ point will be beginning of cycle

TC = O(N)

SC = O(1)

Q36. Flatening a linked list.

→ every node represents a linked list

→ contains two pointers of its type

① next ptr to next node ② bottom pointer

Approach 1.

① create dummy node and assign it to temp and res.

② Keep 2 ptrs one at head and other at head → next
③ whichever is smaller, we point temp → next ~~at smaller~~
④ now we move down in sub linked list
⑤ copy last remaining nodes from sublinked list
⑥ we will flatten all the linked list in pairs.
⑦ use recursion.

TC: O(Sum(N))
SC: O(1)

C++ solution

```
Node* mergelist (Node* a, Node* b) {
  Node *temp = New Node(0);
  Node *res = temp;
  while (a != NULL && b != NULL) {
    if (a->data < b->data) {
      temp -> bottom = a;
      temp = temp -> bottom
      a = a -> bottom.
    }
    else {
      temp -> bottom = b
      temp = temp -> bottom;
      b = b -> bottom;
    }
  }
  if (a != NULL) temp -> bottom = a;
  else temp -> bottom = b;
  return res -> bottom.

Node * flatten (Node *root)
{
  if (root == NULL || root -> next == NULL)
    return root.
  root -> next = flatten(root -> next);
  root = mergeTwolist (root, root -> next);
  return root
```

37. Rotate a linked list by K places.. ** IMP
Adobe, Amazon, MS, G

if K > len(linked list), we move.

Approach 1 Naive approach.
→ rotate linked list by K iterations
find last and second last node in each iteration, assign second last node → next as null and last node → next as head
→ last node will be new head to linked list
TC ⇒ O(n) + O((K) * n))
SC ⇒ O(1)

Approach 2 optimal.
if K == n, no rotation required
→ compute length of linked list
and K = K % n
find K_2 = len − K
→ make list circular by assigning last node's head to first node
→ now find link to be removed
TC = O(N)
SC = O(N)

Q38. Clone a linked list with next and random pointer.

Approach 1 Brute force (extra space)
→ create hashmap storing <node, node>
→ ~~create~~ store org. node as well their copy
→ traverse linked list and assign as per copy
TC ⇒ O(N) + O(N)
SC ⇒ O(N)

Approach 2.

① Create ~~node~~ copy nodes of all nodes and ~~insert them right after org node~~

② break next nodes of org nodes
Point random pointers.
start with variable iter initially pointing head.
iter → next → random = iter → random → next
and move iter by 2

③ get deep copy linked list

TC → O(N) + O(N) + O(N)
SC → O(1)

Pseudo code.

① iter = head
front = head
while (iter != NULL) {
front = iter → next
copy = new Node(item.val)
iter → next = copy
copy → next = front
~~iter = iter → next~~ → next iter = front.
~~front =~~

② iter = head
while (iter != ~~head~~ NULL) {
if (iter → random → != NULL)
iter → next → random = iter → random → next
iter = iter → next → next.

③ iter = head
ph = new Node (0)
copy = ph.
while (iter != NULL) {
front = iter → next → next
copy → next = iter → next.
iter → next = front.
copy = copy → next iter = iter → next
return ph → next.