**Lab 5**

Solve these problems for the linked list.

Deadline: Week 7

1. <https://leetcode.com/problems/delete-node-in-a-linked-list/>

Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the head of the list, instead you will be given access to **the node to be deleted** directly. It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

**Example 1**

**Input:** head = [4,5,1,9], node = 5

**Output:** [4,1,9]

**Explanation:** You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

**Code**:

class Solution {

public:

void deleteNode(ListNode\* node) {

ListNode \*temp = node->next;

node->val = node->next->val;

node->next = node->next->next;

delete temp;

}

};

Explanation:

In this task I delete node in singly linked list. How I delete, the current node value set to the next node and then delete the next node. Store the next node to delete, list temp equal node and direct to the next and so so. Then delete temp;

1. <https://leetcode.com/problems/reverse-linked-list/>

Reverse a singly linked list.

**Example:**

**Input:** 1->2->3->4->5->NULL

**Output:** 5->4->3->2->1->NULL

**Code:**

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

return reverseRec(NULL, head);

}

ListNode\* reverseRec(ListNode\* rev, ListNode\* cur){

if(cur == NULL){

return rev;

}

ListNode\* newCur = cur->next;

cur->next = rev;

return reverseRec(cur, newCur);

}

};

**Explanation:**

In this task I must to reverse linked list, I use method recursively. As well as I do the reverse with an already reversed list and the current node.

1. <https://leetcode.com/problems/merge-two-sorted-lists/>

Merge two sorted linked lists and return it as a new **sorted** list. The new list should be made by splicing together the nodes of the first two lists.

**Input:** l1 = [], l2 = []

**Output:** []

**Constraints:**

* The number of nodes in both lists is in the range [0, 50].
* -100 <= Node.val <= 100
* Both l1 and l2 are sorted in **non-decreasing** order.

**Code:**

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

if(l1==NULL && l2==NULL)

return NULL;

else if(l1==NULL)

{

return l2;

}

else if(l2==NULL)

{

return l1;

}

else

{

if(l1->val<l2->val)

{

l1->next=mergeTwoLists(l1->next,l2);

return l1;

}

else

{

l2->next=mergeTwoLists(l1,l2->next);

return l2;

}

}

return NULL;

}

};

**Explanation:**

Base Case:

* If both lists are NULL then return NULL.
* If one list is null return the other list.

Reursive approach

1. Compare head element of both linked lists.
2. The smaller node from both list will be the current element.
3. The rest elements of both lists will appear after current element.
4. Now run a recursive function with parameters, the next node of the smaller element and the other list head.
5. The recursive function will return the next smaller element from both list. Now point the next of current element to that, i.e **curr\_element->next=recursivefunc()**
6. <https://leetcode.com/problems/linked-list-cycle/>

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter**.

Return true*if there is a cycle in the linked list*. Otherwise, return false.

**Follow up:**

Can you solve it using O(1) (i.e. constant) memory?

**Example 1:**

**Input:** head = [3,2,0,-4], pos = 1

**Output:** true

**Explanation:** There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

**Code:**

bool hasCycle(ListNode \*head) {

if (!head) return false;

if (!head->next) return false;

ListNode\* current = head; // Slow

ListNode\* runner = head->next; // Fast

int cnt{};

while (runner->next) {

if (cnt % 2 == 0)

current = current->next;

runner = runner->next;

} else {

runner = runner->next;

}

cnt++;

if (runner == current) return true

}

return false;

}

**Explanation:** in this task firstly I check if there no list, return false, then I check if there is only node in the list, return also false, after this with while cycle, a cycle is null terminated, then use condition if in circular it will eventually break and return false. But if it is circular, eventually the fast and slow.

1. <https://leetcode.com/problems/partition-list/>

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

**Example:**

**Input:** head = 1->4->3->2->5->2, x = 3

**Output:** 1->2->2->4->3->5

**Code:**

ListNode\* partition(ListNode\* head, int x) {

ListNode\* cur = new ListNode(0);

cur->next = head;

head = cur;

while (cur->next && cur->next->val < x)

cur = cur->next;

ListNode\* insPos = cur;

ListNode\* tmp;

while (cur->next) {

if (cur->next->val < x) {

tmp = cur->next;

cur->next = tmp->next;

tmp->next = insPos->next;

insPos->next = tmp;

insPos = insPos->next;

} else {

cur = cur->next;

}

}

return head->next;

}

**Explanation:** in this task I should preserve the original relative order of the nodes in each of the two partitions. I find insert position for elements that < x. use while cycle, keep traversing list, insert lower value elements, after Inspos and skip x elements.

1. <https://leetcode.com/problems/intersection-of-two-linked-lists/>

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists: begin to intersect at node c1.

**Input:** intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3

**Output:** Reference of the node with value = 8

**Input Explanation:** The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B.

Code:

class Solution { public:

int length(ListNode\* node){

int len = 0;

while(node != nullptr){ node = node->next; len++;

}

return len;

}

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) { int list1 = length(headA);

int list2 = length(headB);

while(list1 > list2){ headA = headA->next; list1--;

}

while(list1 < list2){ headB = headB->next; list2--;

}

while(headA != headB){ headA = headA->next; headB = headB->next;

}

return headA;

}

};

**Explanation:**

There are you should find intersection of two Linked List,

Firstly, will get the length of the two lists then align them to the same start point and move them together until finding the intersection point, or the end null.

Time Complexity O(n)

Space Complexity O(1)

1. <https://leetcode.com/problems/palindrome-linked-list/>

Given a singly linked list, determine if it is a palindrome.

**Example 1:**

**Input:** 1->2

**Output:** false

**Example 2:**

**Input:** 1->2->2->1

**Output:** true

**Code:** class Solution {

public:

int calc(ListNode\* head,int k,int len){

ListNode\* temp = head;

if(head->next==NULL && k==1) return head->val;

int f = len-k+1;

while(--f>0){

temp=temp->next;

//f-=1;

}

return temp->val;

}

bool isPalindrome(ListNode\* head) {

if(head==NULL || head->next==NULL) return true;

ListNode\* temp = head;

int len = 1;

while(temp->next!=NULL){

len+=1;

temp= temp->next;

}

int n = len/2;

stack<int> s;

while(n > 0){

s.push(head->val);

head=head->next;

n-=1;

}

//cout<<len;

if(len%2==0){

while(head!=NULL){

if(head->val == s.top()) {

s.pop();

head=head->next;

}else return false;

}

return true;

}

else {

head=head->next;

while(head!=NULL){

if(head->val == s.top()) {

s.pop();

head=head->next;

}else return false;

}

return true;

}

}

};

**Explanation: in this task** first of all calculate the length of the linked list. Then if the length is even then I take a stack and input half elements in it and then for the rest half of the elements start pop from the stack if at the end stack is empty return true else return false, as well as if the length is odd, same approach as in case of even length but now after pushing half elements we need to skip the half+1 eleme from pushing into the stack and the rest is fine.

1. <https://leetcode.com/problems/sort-list/>

Given the head of a linked list, return *the list after sorting it in****ascending order***.

**Follow up:** Can you sort the linked list in O(n logn) time and O(1) memory (i.e. constant space)?

**Example 1:**

**Input:** head = [4,2,1,3]

**Output:** [1,2,3,4]

**Example 2:**

**Input:** head = [-1,5,3,4,0]

**Output:** [-1,0,3,4,5]

**Code:**

class Solution {

public:

ListNode\* sortList(ListNode\* head) {

vector<int> v;

ListNode\* temp = head;

while(temp!=NULL){

v.push\_back(temp->val);

temp = temp->next;

}

sort(v.begin(), v.end());

temp = head;

for (auto val:v) {

temp->val = val;

temp = temp->next;

}

return head;

}

};

Explanation: in this task I use the sorting with vector, and I want to describe my algorithm of the code: first pf all I storing all the values in the vector, then I sort the vector. And last of my step is the change the value of the ListNodes.