Student: Pazyl Yerbolat

Group: IS-1808K

**Lab 5**

Solve these problems for the linked list.

Deadline: Week 7

**Problem – 1**

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

**237. Delete Node in a Linked List**

**Description**

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.

**Solution**

**C++ Code**

class Solution {

public:

void deleteNode(ListNode\* node) {

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

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

}

};

**Problem – 2**

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

**206. Reverse Linked List**

**Description**

Reverse a singly linked list.

**Example:**

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

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

**Follow up:**

A linked list can be reversed either iteratively or recursively. Could you implement both?

**Solution**

**C++ Code**

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

if(head == NULL)

return head;

ListNode \* current = head;

ListNode \* prev;

ListNode \* next = NULL;

while(true) {

prev = current->next;

current->next = next;

next = current;

if(prev != NULL)

current = prev;

else

break;

}

return current;

}

};

**Problem – 3**

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

**21. Merge Two Sorted Lists**

**Description**

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.

**Example 1:**

**Input:** l1 = [1,2,4], l2 = [1,3,4]

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

**Solution**

**C++ Code**

class Solution {

public:

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

ListNode \*i = l1;

ListNode \*j = l2;

ListNode \*head = NULL;

ListNode \*current =NULL;

while(i != NULL || j != NULL) {

if(i != NULL && j != NULL) {

if (i->val < j->val) {

if(head == NULL)

head = i;

else

current->next = i;

current = i;

i = i->next;

} else if (i->val > j->val) {

if(head == NULL)

head = j;

else

current->next = j;

current = j;

j = j->next;

} else {

if(head == NULL)

head = i;

else

current->next = i;

current = i;

i = i->next;

}

} else if (i == NULL) {

if (head == NULL)

head = j;

else

current->next = j;

break;

} else {

if (head == NULL)

head = i;

else

current->next = i;

break;

}

}

return head;

}

};

**Problem – 4**

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

**141. Linked List Cycle**

**Description**

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).

**Solution**

**C++ Code**

class Solution {

public:

bool hasCycle(ListNode \*head) {

ListNode \*slow = head;

ListNode \*fast = head;

while(slow != NULL && fast != NULL) {

slow = slow->next;

fast = fast->next;

if(fast == NULL)

return false;

fast = fast->next;

if(slow == NULL || fast == NULL)

return false;

if(slow->val == fast->val)

return true;

}

return false;

}

};

**Problem – 5**

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

**86. Partition List**

**Description**

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

**Solution**

**C++ Code**

class Solution {

public:

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

if(head == NULL)

return NULL;

ListNode \*left = NULL;

ListNode \*leftHead = NULL;

ListNode \*right = NULL;

ListNode \*rightHead = NULL;

ListNode \*current = head;

while(true) {

if (current == NULL)

break;

else if(current->val < x) {

if(left == NULL) {

left = current;

leftHead = current;

} else {

left->next = current;

left = current;

}

current = current->next;

} else {

if(right == NULL) {

rightHead = current;

right = current;

} else {

right->next = current;

right = current;

}

current = current->next;

}

}

if(leftHead == NULL || rightHead == NULL)

return head;

else

{

left->next = rightHead;

right->next = NULL;

return leftHead;

}

}

};

**Problem – 6**

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

**160. Intersection of Two Linked Lists**

**Description**

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

For example, the following two linked lists:

[](https://assets.leetcode.com/uploads/2018/12/13/160_statement.png)

begin to intersect at node c1.

**Example 1:**

[](https://assets.leetcode.com/uploads/2020/06/29/160_example_1_1.png)

**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.

**Solution**

**C++ Code**

class Solution {

public:

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

if(headA == NULL || headB == NULL)

return NULL;

ListNode \*pa = headA;

ListNode \*lastA;

ListNode \*pb = headB;

ListNode \*lastB;

while(pa->next != NULL)

pa = pa->next;

lastA = pa;

while(pb->next != NULL)

pb = pb->next;

lastB = pb;

if(lastB == lastA){

pa = headA;

pb = headB;

while(pa != pb) {

if (pa->next != NULL)

pa = pa->next;

else

pa = headB;

if (pb->next != NULL)

pb = pb->next;

else

pb = headA;

}

return pa;

}

return NULL;

}

};

**Problem – 7**

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

**234. Palindrome Linked List**

**Description**

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

**Solution**

**C++ Code**

class Solution {

public:

ListNode \*reverseLinkedList(ListNode \*head) {

ListNode \*current = head, \*prev = NULL, \*next = NULL;

while(true) {

prev = current->next;

current->next = next;

next = current;

if(prev != NULL)

current = prev;

else

break;

}

return current;

}

int getLength(ListNode \*head) {

int count = 0;

while (head != NULL) {

count++;

head = head->next;

}

return count;

}

bool isPalindrome(ListNode\* head) {

if(head == NULL || head->next == NULL)

return true;

int count = getLength(head);

ListNode \*current = head;

int i = 1;

while(i < (count + 1) / 2 + 1) {

i++;

current = current->next;

}

ListNode \*l = head;

ListNode \*r = reverseLinkedList(current);

i = 1;

while(i <= count / 2) {

if (l->val != r->val)

return false;

l = l->next;

r = r->next;

i++;

}

return true;

}

};

**Problem – 8**

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

**148. Sort List**

**Description**

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]

**Solution**

**C++ Code**

class Solution {

public:

ListNode\* sortList(ListNode\* head) {

for(ListNode \*i = head; i != NULL; i = i->next) {

for (ListNode \*j = i->next; j != NULL; j = j->next) {

if (i->val > j->val) {

swap(i->val, j->val);

}

}

}

return head;

}

};