***Week – 3 (14.04.2021 – 18.04.2021)***

***CODES IN PDF***

1. ***Remove Nth Node From End of List:***

class Solution {

public:

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

ListNode \*l = head;

ListNode \*l2 = head;

int i=0;

while(l!=NULL)

{

i++;

l=l->next;

}

if(i == n)

{

head = l2->next;

return head;

}

int j = i;

while(l2!=NULL)

{

j--;

if(j == n) l2->next = l2->next->next;

else l2 = l2->next;

}

return head;

}

};

1. ***Reverse Linked List:***

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode \*temp = NULL;

ListNode \*prev = NULL;

ListNode \*curr = head;

while(curr!=NULL)

{

temp = curr->next;

curr->next = prev;

prev = curr;

curr = temp;

}

return prev;

}

};

1. ***Reverse Linked List using Recursive Function:***

ListNode\* reverseList(ListNode\* head) {

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

ListNode \*temp = reverseList(head->next);

head->next->next = head;

head->next = NULL;

return temp;

}

};

1. ***Add Two Numbers II:***

class Solution {

public:

ListNode\* reverse(ListNode\* l)

{

if(l == NULL || l->next == NULL) return l;

ListNode\* prev = NULL;

ListNode\* temp = NULL;

while(l != NULL)

{

temp = l->next;

l->next = prev;

prev = l;

l = temp;

}

return prev;

}

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

ListNode\* revl1 = reverse(l1);

ListNode\* revl2 = reverse(l2);

int carry = 0, sum = 0;

ListNode\* add = NULL;

while(revl1 != NULL || revl2 != NULL)

{

sum = carry;

if(revl1 != NULL)

{

sum = sum + revl1->val;

revl1 = revl1->next;

}

if(revl2 != NULL)

{

sum = sum + revl2->val;

revl2 = revl2->next;

}

carry = sum/10;

sum = sum%10;

if(add == NULL)

{

add = new ListNode(sum);

}

else

{

add = new ListNode(sum, add);

}

}

if(carry != 0)

{

while(carry > 0)

{

int c = carry%10;

add = new ListNode(carry, add);

carry = carry/10;

}

}

return add;

}

};

1. ***Palindrome Linked List:***

class Solution {

public:

bool isPalindrome(ListNode\* head) {

ListNode\* fast = head;

ListNode\* slow = head;

while(fast!=NULL && fast->next!=NULL)

{

slow = slow->next;

fast = fast->next->next;

}

ListNode\* curr = slow;

ListNode\* prev = NULL;

ListNode\* temp = NULL;

while(curr != NULL)

{

temp = curr->next;

curr->next = prev;

prev = curr;

curr = temp;

}

ListNode\* first = head;

ListNode\* second = prev;

while(second!=NULL)

{

if(first->val != second->val) return false;

first = first->next;

second = second->next;

}

return true;

}

};

1. ***Merge Two Sorted Lists:***

class Solution {

public:

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

if(l1 == NULL) return l2;

if(l2 == NULL) return l1;

ListNode\* first = NULL;

ListNode\* second = NULL;

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

{

first = second = l1;

l1 = l1->next;

}

else

{

first = second = l2;

l2 = l2->next;

}

while(l1!=NULL && l2!=NULL)

{

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

{

second->next = l1;

second = l1;

l1 = l1->next;

}

else

{

second->next = l2;

second = l2;

l2 = l2->next;

}

}

if(l1 == NULL) second->next = l2;

if(l2 == NULL) second->next = l1;

return first;

}

};

1. ***Middle of the Linked List:***

class Solution {

public:

ListNode\* middleNode(ListNode\* head) {

ListNode\* slow = head;

ListNode\* fast = head;

while(fast!=NULL && fast->next!=NULL)

{

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

};

1. ***Odd Even Linked List:***

class Solution {

public:

ListNode\* oddEvenList(ListNode\* head) {

if(head == NULL) return head;

ListNode \*l1 = head;

ListNode \*l2 = head->next;

ListNode \*l3 = head->next;

while(l1->next!=NULL && l2->next!=NULL)

{

l1->next = l1->next->next;

l2->next = l2->next->next;

l1 = l1->next;

l2 = l2->next;

}

l1->next = l3;

return head;

}

};

1. ***Linked List Cycle (Detect Loop):***

class Solution {

public:

bool hasCycle(ListNode \*head) {

unordered\_set<ListNode\*> s;

while(head!=NULL)

{

if(s.find(head) != s.end()) return true;

s.insert(head);

head = head->next;

}

return false;

}

};

1. ***Reverse Alternate K Nodes:***

ListNode\* Solution::solve(ListNode\* A, int B) {

ListNode\* curr = A;

ListNode\* l1 = A;

ListNode\* temp = NULL;

ListNode\* prev = NULL;

int i=0;

while(curr!=NULL && i<B)

{

temp = curr->next;

curr->next = prev;

prev = curr;

curr = temp;

i++;

}

if(A!=NULL) A->next = curr;

i=0;

while(i<B-1 && curr!=NULL)

{

curr = curr->next;

i++;

}

if(curr!=NULL)

curr->next = solve(curr->next, B);

return prev;

}

1. ***Intersection point of Two Linked Lists:***

class Solution {

public:

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

ListNode\* a = headA;

ListNode\* b = headB;

while(a!=b)

{

if(a == NULL) a = headB;

else a = a->next;

if(b == NULL) b = headA;

else b = b->next;

}

return a;

}

};

1. ***Reverse Nodes in k-Group:***

class Solution {

public:

ListNode\* reverseKGroup(ListNode\* head, int k) {

ListNode\* node = head;

for(int i=0;i<k;i++) {

if(!node) return head;

node = node->next;

}

ListNode\* cur = reverseKGroup(node, k);

while(head != node) {

ListNode\* next = head->next;

head->next = cur;

cur = head;

head = next;

}

return cur;

}

};

1. ***Swapping kth Node from beginning and end of a Linked List:***

class Solution {

public:

ListNode\* swapNodes(ListNode\* head, int k) {

ListNode\* first = head;

for(int i=0;i<k;i++)

first = first->next;

ListNode\* swap1 = head;

ListNode\* swap2 = head;

while(first!=NULL)

{

swap1=swap1->next;

first=first->next;

}

for(int i=1;i<k;i++)

swap2=swap2->next;

int temp= swap1->val;

swap1->val = swap2->val;

swap2->val = temp;

return head;

}

};