Lecture 25 LinkedList

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

class Solution {

public:

bool hasCycle(ListNode \*head) {

ListNode \*slow = head;

ListNode \*fast = head;

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

{

slow = slow->next;

fast = fast->next->next;

if(slow==fast)

{

return true;

}

}

return false;

}

};

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

class Solution {

public:

ListNode \*detectCycle(ListNode \*head) {

ListNode \*slow = head;

ListNode \*fast = head;

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

{

slow = slow->next;

fast = fast->next->next;

if(slow==fast)

{

slow = head;

while(slow!=fast)

{

slow = slow->next;

fast = fast->next;

}

return slow;

}

}

return NULL;

}

};

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

class Solution {

public:

void deleteNode(ListNode\* node) {

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

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

}

};

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

class Solution {

ListNode\* reverse(ListNode\*head)

{

ListNode \*curr = head;

ListNode \*prev = NULL;

ListNode \*Next;

while(curr!=NULL)

{

Next = curr->next;

curr->next = prev;

prev = curr;

curr = Next;

}

return prev;

}

public:

bool isPalindrome(ListNode\* head) {

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

{

return true;

}

// find mid point

ListNode \*slow = head;

ListNode \*fast = head->next;

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

{

slow = slow->next;

fast = fast->next->next;

}

// reverse the right half

slow->next = reverse(slow->next);

slow = slow->next;

// compare slow and head

while(slow!=NULL)

{

if(slow->val!=head->val)

{

return false;

}

slow = slow->next;

head = head->next;

}

return true;

}

};

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

class Solution {

int count(ListNode\*head)

{

int c = 0;

while(head!=NULL)

{

c++;

head = head->next;

}

return c;

}

public:

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

ListNode\* tempA = headA;

ListNode\* tempB = headB;

int l1 = count(tempA);

int l2 = count(tempB);

int diff = 0;

if(l1<=l2)

{

diff = l2-l1;

while(diff--)

{

tempB = tempB->next;

}

}

else

{

diff = l1-l2;

while(diff--)

{

tempA = tempA->next;

}

}

while(tempA!=NULL)

{

if(tempA==tempB)

{

return tempA;

}

tempA = tempA->next;

tempB = tempB->next;

}

return NULL;

}

};