05/11/2023

* Reverse the linked list - (leetcode-206)

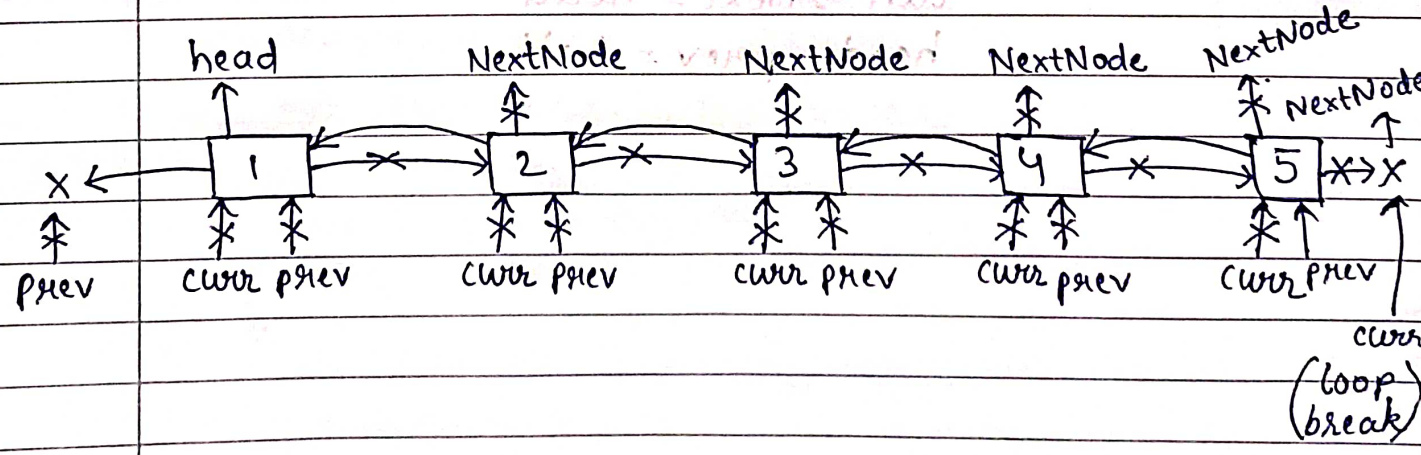
Given the head of singly-linked list, reverse the list and return the reversed list.

I/P - head = [1, 2, 3, 4, 5]

O/P - [5, 4, 3, 2, 1]

- **Iterative approach -**

Step 1: ListNode* prev = NULL.
Step 2: ListNode* curr = head.
Step 3: (i) listNode * NextNode = curr -> next.
(ii) curr -> next = prev.
(iii) prev = curr.
(iv) curr = NextNode.
Step 4: Repeat step3 while (curr != NULL).



Code -

```
class Solution {
    public:
    ListNode* reverseList(ListNode* head){

        ListNode* prev = NULL;
        ListNode* curr = head;
        while (curr != NULL){
            ListNode * NextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = NextNode;
        }
        return prev;
    }
};
```

- Recursive approach -

Base case - if (curr == NULL), return prev

Step1: ListNode* NextNode = curr->next.
Step2: curr->next = prev.
Step3: prev = curr.
Step4: curr = NextNode.
Step5: Make a recursive call.

Code-

```
class Solution {
    public:
    ListNode* reverseUsingRecursion(ListNode *&prev,
                                    ListNode* &curr)
    {
        if(curr == NULL){
            return prev;
        }
        ListNode * NextNode = curr->next;
        curr->next = prev;
        return reverseUsingRecursion(curr, NextNode);
    }

    ListNode reverseList(ListNode* head){

        ListNode * prev = NULL;
        ListNode * curr = head;
        return reverseUsingRecursion(prev, curr);
    }
};
```

__/__/__

* **Middle of the linked list (leetcode - 876)**

Given the head of singly linked list, return the middle node of linked list.
If there are two middle nodes, return the second middle node.

Example 1 - odd no. of nodes

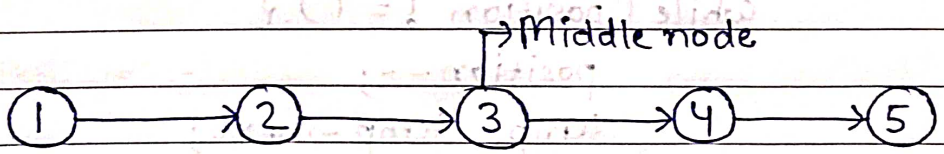
I/P - head = [1, 2, 3, 4, 5]

O/P - [3, 4, 5]

Example 2 - even no. of nodes

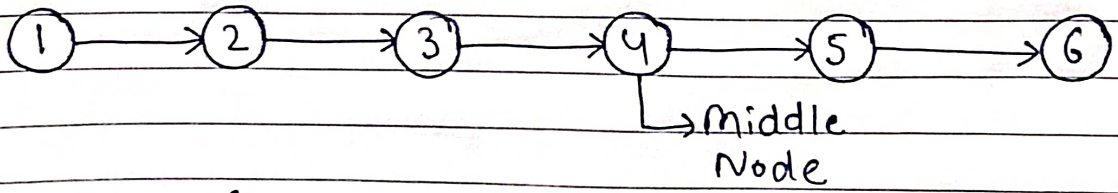
I/P - head = [1, 2, 3, 4, 5, 6]

O/P - [4, 5, 6]



$n = 5$

Middle node $= \frac{n}{2} + 1 = \frac{5}{2} + 1 = 3$



$n = 6$

Middle node $= \frac{n}{2} + 1 = \frac{6}{2} + 1 = 4$

Code -

```
class solution{
    public:
    int findlength (listNode* &head){
        listNode * temp= head;
        int len= 0;
        while (temp != NULL){
        len++;
        temp= temp->next;
        }
        return len;
    }

listNode * MiddleNode (listNode* head) {

    int n= findlength(head);
    int position= n/2 + 1;

    listNode * temp = head;
    while (position != 1) {
        position--;
        temp= temp->next;
        }
    return temp;
    }
};
```

__/__/____

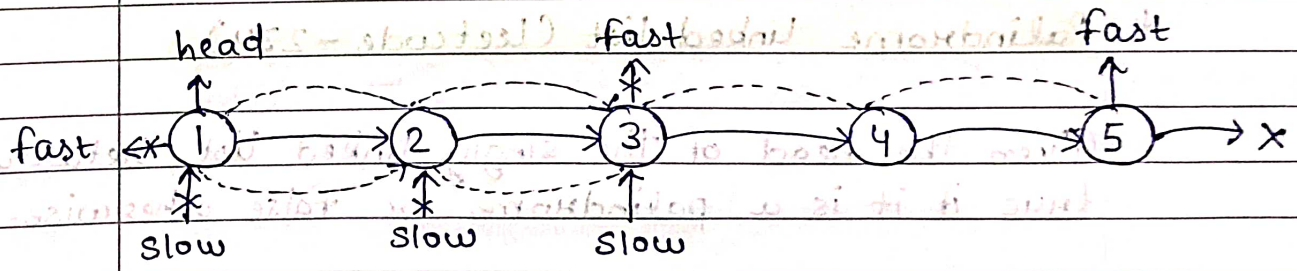
* **Tortoise algorithm** - Middle of linked list

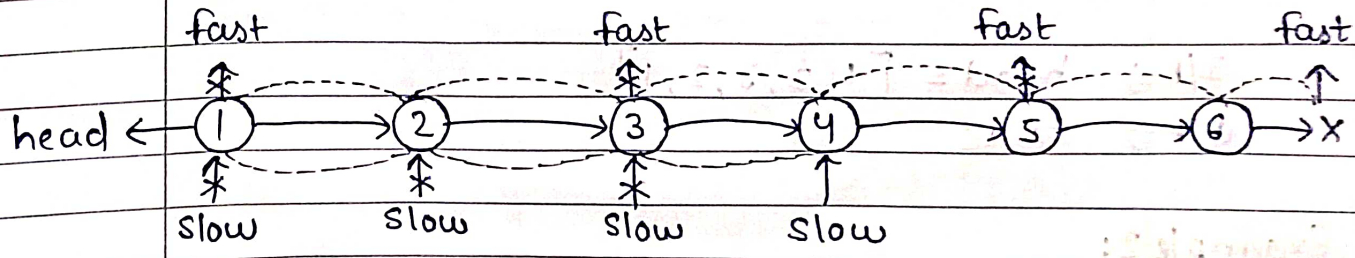
Slow and fast pointers →

Slow - increment by 1 step.
Fast - increment by 2 steps.

If fast pointer 2 step increment kar paya, only then slow pointer 1 step se increment krega.

Step 1: Set slow and fast pointer at head.
Step 2: Increment fast by 2 steps. } until fast approaches
Step 3: Increment slow by 1 step. } NULL.
Step 4: return slow.





Code-

```
class solution {
    public:
    ListNode * middleNode (ListNode * head) {

        ListNode* slow = head;
        ListNode * fast = head;

        while (fast != NULL){
            fast = fast ->next;
            if (fast != NULL){
                fast = fast ->next;
                slow= slow ->next;
            }
        }
        return slow;
    }
};
```

* **Palindrome Linked list (leetcode - 234)**

Given the head of the singly linked list, return true if it is a palindrome or false otherwise.

Example 1:

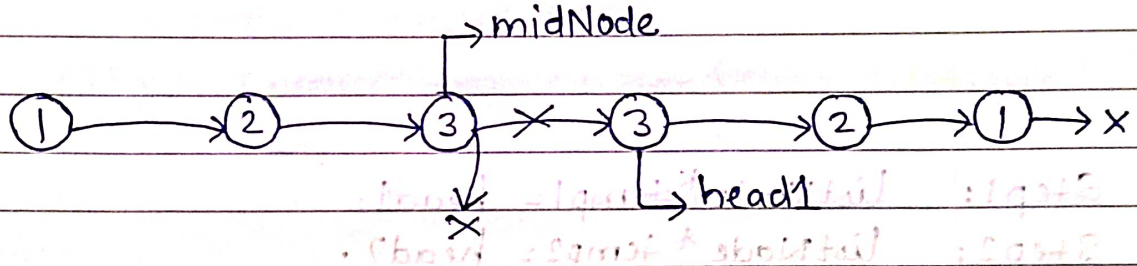
I/P ⇒ head = [1, 2, 3, 2, 1]
O/P ⇒ true

Example 2:

I/P ⇒ head = [1, 2]
O/P ⇒ false

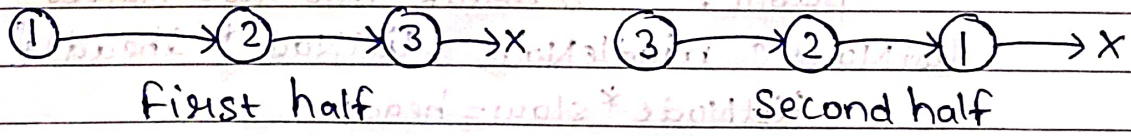
Step 1: Break into two halves.
Step 2: Reverse the second half.
Step 3: Compare the two linked lists.

- **Break into two halves –**

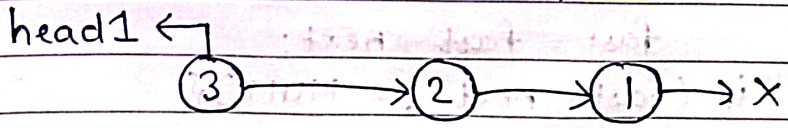


Step1: Find midNode.
Step2: listNode * head1 = midNode->next;
Step3: midNode->next = NULL.

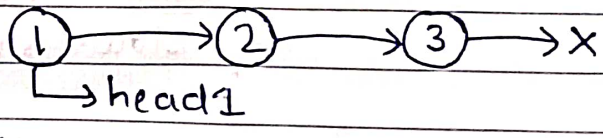
Now, the two halves are →



- **Reverse the second half –**



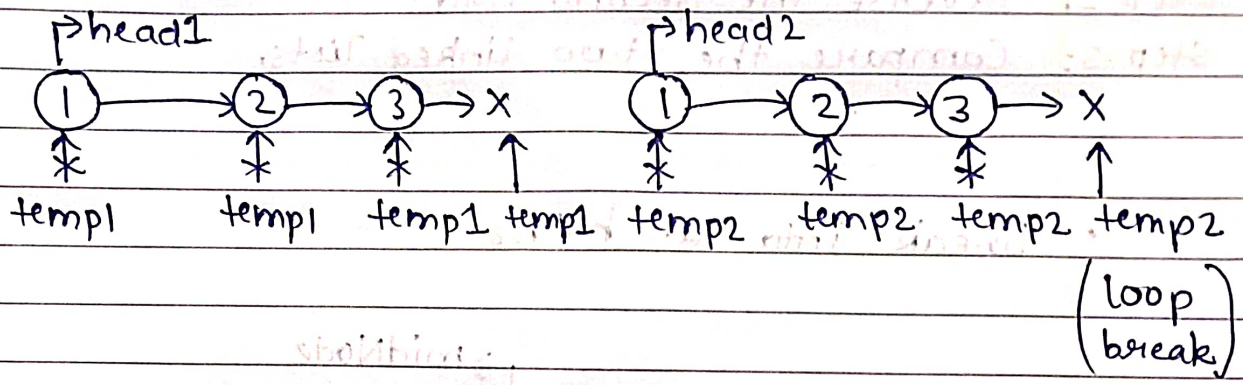
After reversing –



Step1: listNode * prev = NULL,
Step2: listNode * curr = head1.
Step3: head1 is updated now.

__/__/____

• Compare the two linked lists -



Step 1: ListNode* temp1 = head1;
Step 2: ListNode * temp2 = head2.
Step 3: (i) if (temp1 -> val != temp2 -> val) return false
(ii) else , temp1 = temp1 -> next & temp2 = temp2 -> next
Step 4: Repeat the step 3 until (temp2 != NULL).
Step 5: Otherwise return true.

Code-

```
class solution {
    public:        // break into two halves
ListNode * middleNode (ListNode * &head) {
    ListNode *slow = head;
    ListNode * fast = head;

    while (fast -> next != NULL) {
            fast = fast -> next;
        if (fast -> next != NULL) {
            fast = fast -> next;
            slow = slow -> next;
        }
    }
    Return slow;
}
```

```
//reverse the second half
listNode * reverseUsingRecursion (listNode* &prev,
                                  listNode * &curr)
    {
        if (curr == NULL) {
            return prev;
        }
    listNode * NextNode = curr->next;
    curr->next = prev;
    return reverseUsingRecursion (curr, NextNode);
```

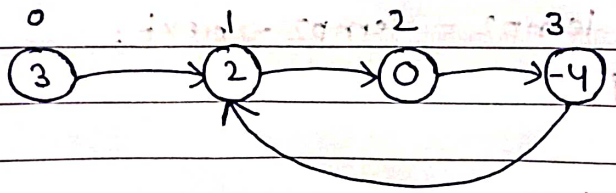
```
//compare the two linked lists

bool compareList (listNode * &head1, listNode * &head2)
    {
      listNode* temp1 = head1;
      listNode* temp2 = head2;

      while (temp2 != NULL){
        if (temp1->val != temp2->val) {
            return false;
        }
        else {
            temp1 = temp1->next;
            temp2 = temp2->next;
        }
    }
  return true;
}
```

```
bool isPalindrome (ListNode * head){

    ListNode * midNode = middleNode(head);
    ListNode * head1 = midNode->next;
    midNode->next = NULL;

    ListNode * prev = NULL;
    ListNode * curr = head1;
    head1 = reverseUsingRecursion(prev, curr);

    return compareList(head, head1);
  }
};
```

* **Linked List Cycle (leetcode-141)**

Given head, the head of the linked list, determine if linked list has cycle in it.
Return true if there is a cycle otherwise return false.

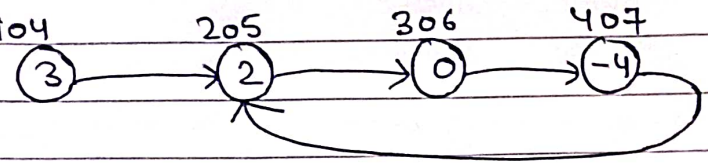
Example 1-



I/P - head = [3,2,0,-4], pos = 1

O/P - true

Logic- Agr linked list traverse krte time koi address repeat ho jaye, it means loop is present.



By default, in map, all addresses are marked false. Mark visited addresses true and if any address is re-visited, it means cycle is present. So, in that case, return true.

Using map

address	status
104	T
205	T
306	T
407	T

→ loop present

Code -

```
class solution{
public:
    bool hasCycle (listNode* head){
    map < listNode* , bool > table ;
    listNode * temp = head;
    while (temp != NULL) {
        if (table[temp] == false) {
            table[temp] = true;
        }
        else {
            return true; //cycle is present
        }
        temp= temp->next;
    }
    return false; // Agr loop se bahr aa gaye, it
                  // means cycle is absent.
} };
```