All Contests > ABCS 18: Linked Lists > Print the Elements of a Linked List

Print the Elements of a Linked List

locked

by harsha_s

Problem Submissions Leaderboard Discussions Editorial

To print the elements of a linked list, we need to traverse the entire list and print the value of every node.The basic idea behind traversing the linked list is to follow the next pointers untill NULL is encountered.

Statistics

Difficulty: Easy
Publish Date: Aug 08 2014

Pseudocode:

```
Initialize ptr to head of the linked list

while ptr is not NULL
    print (*ptr).value
    ptr=(*ptr).next //Move ptr to the next node in the list
```

Insert a node at a specific position in a linked list

locked

by harsha_s

Problem Submissions Leaderboard Discussions Editorial

Editorial by rishi_07

In this problem, we are given the pointer to the $head$ node of a linked list, an integer $data$, to add to the list and the $position$ at which the integer must be inserted.
To insert the node at the desired position, we need to find the node at the previous position. This can be done using a simple loop.
There is one special case where the head of our linked list changes is when $position = 0$ as there is no previous node. In all other cases, our $head$ remains the same.

Refer to the code below for more details:

Statistics

Difficulty: Easy
Time Complexity: O(n)
Required Knowledge: Loops, Linked lists
Publish Date: Dec 29 2015

```
SinglyLinkedListNode* insertNodeAtPosition(SinglyLinkedListNode* head, int
data, int position)
{
    SinglyLinkedListNode* temp = head;
    SinglyLinkedListNode* aux = new SinglyLinkedListNode(data);
    if(position == 0)
    {
        aux->next = temp;
        head = aux;
        return aux;
    }
    int idx = 0;
    while(idx != position - 1)
    {
        idx++;
        temp = temp->next;
    }
    aux->next = temp->next;
    temp->next = aux;
    return head;
}
```

Tested by rishi_07

Problem Tester's code :

```
def insertNodeAtPosition(head, data, position):
    node = head
    if position == 0:
        newNode = SinglyLinkedListNode(data)
        newNode.next = head
        return newNode
    count = 1
    while count < position and node:
        count += 1
        node = node.next
    newNode = SinglyLinkedListNode(data)
    newNode.next = node.next
    node.next = newNode
    return head
```

All Contests > ABCS 18: Linked Lists > Find Merge Point of Two Lists

Find Merge Point of Two Lists

locked

by harsha_s

Problem Submissions Leaderboard Discussions Editorial

Editorial by vatsalchanana

To calculate the merge point, first calculate the difference in the sizes of the linked lists. Move the pointer of the smaller linked list by this difference. Increment both pointers till you reach the merge point.

Problem Setter's code :

```
int getCount(Node* head)
{
  Node* current = head;
  int count = 0;

  while (current != NULL)
  {
    count++;
    current = current->next;
  }
```

Statistics

Difficulty: Easy

Time Complexity:	O(N) Required

Knowledge: Linked List

Publish Date: Mar 08 2015

```
  return count;
}

int getNode(int d, Node* head1, Node* head2)
{
  int i;
  Node* current1 = head1;
  Node* current2 = head2;

  for(i = 0; i < d; i++)
  {
    if(current1 == NULL)
    {  return -1; }
    current1 = current1->next;
  }

  while(current1 !=  NULL && current2 != NULL)
  {
    if(current1 == current2)
      return current1->data;
    current1= current1->next;
    current2= current2->next;
  }

  return -1;
}

int FindMergeNode(Node *headA, Node *headB)
{
    // Complete this function
    // Do not write the main method.
    int c1 = getCount(headA);
  int c2 = getCount(headB);
  int d;

  if(c1 > c2)
  {
    d = c1 - c2;
    return getNode(d, headA, headB);
  }
  else
```

```
    {
      d = c2 - c1;
      return getNode(d, headB, headA);
    }
  }
```

Contest Calendar | Interview Prep | Blog | Scoring | Environment | FAQ | About Us | Support | Careers | Terms Of Service | Privacy Policy | Request a Feature

Delete duplicate-value nodes from a sorted linked list

locked

by harsha_s

Problem Submissions Leaderboard Discussions Editorial

Editorial by vatsalchanana

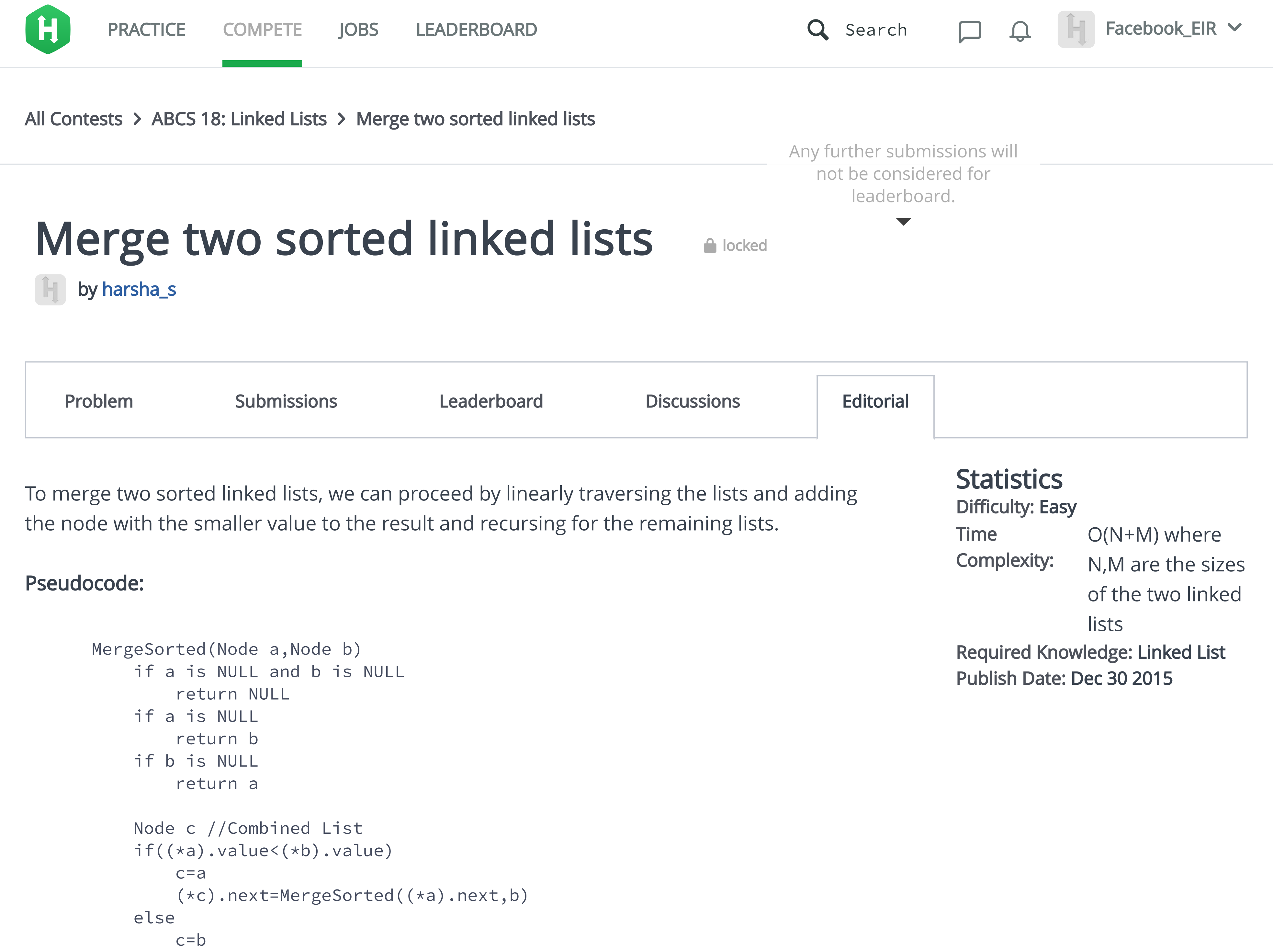
To remove duplicates from the linked list, we can traverse the list and check whether the current node and the next node have the same data. If they have the same data, we can delete the next node.

Statistics

Difficulty: Easy
Time Complexity: O(N)
Required Knowledge: Linked Lists
Publish Date: Jan 08 2016

Problem Setter's code :

```
/*
  Remove all duplicate elements from a sorted linked list
  Node is defined as
  struct Node
  {
     int data;
     struct Node *next;
  }
*/
Node* RemoveDuplicates(Node *head)
{
  // This is a "method-only" submission.
  // You only need to complete this method.

    Node * temp=head;
    while(temp->next!=NULL)
    {
        if(temp->data==temp->next->data)
        {
            Node * t=temp->next;
            temp->next=t->next;
            delete(t);
        }
        else
        {
            temp=temp->next;
        }
    }
    return head;
}
```


PRACTICE
COMPETE
JOBS
LEADERBOARD
Search
Facebook_EIR
All Contests > ABCS 18: Linked Lists > Merge two sorted linked lists
Any further submissions will not be considered for leaderboard.
locked
Problem
Submissions
Leaderboard
Discussions
Editorial

Merge two sorted linked lists

by harsha_s

To merge two sorted linked lists, we can proceed by linearly traversing the lists and adding the node with the smaller value to the result and recursing for the remaining lists.

Pseudocode:

```
MergeSorted(Node a,Node b)
    if a is NULL and b is NULL
        return NULL
    if a is NULL
        return b
    if b is NULL
        return a

    Node c //Combined List
    if((*a).value<(*b).value)
        c=a
        (*c).next=MergeSorted((*a).next,b)
    else
        c=b
```

Statistics

Difficulty: Easy

Time Complexity: O(N+M) where N,M are the sizes of the two linked lists

Required Knowledge: Linked List

Publish Date: Dec 30 2015

```
    (*c).next=MergeSorted(a,(*b).next)
return c
```

Reverse a doubly linked list

locked

by harsha_s

Problem Submissions Leaderboard Discussions Editorial

Editorial by vatsalchanana

All we need to do is swap prev and next pointers for all nodes, change prev of the head (or start) and change the head pointer in the end.

Problem Setter's code :

```
Node* Reverse(Node* head)
{
     Node *temp = NULL;
     Node *current = head;


     while (current !=  NULL)
     {
       temp = current->prev;
       current->prev = current->next;
       current->next = temp;
```

Statistics

Difficulty: Easy

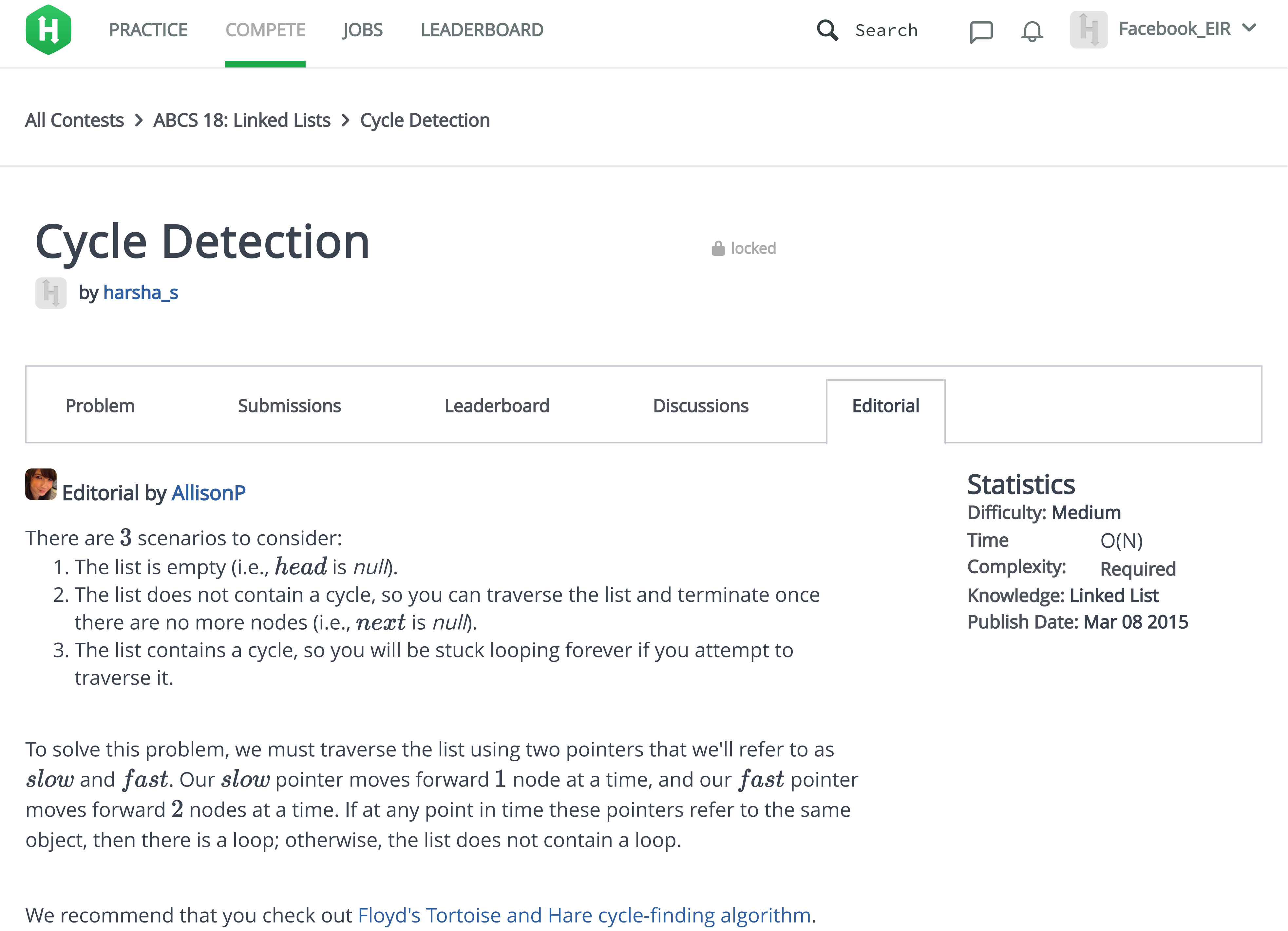
Time Complexity:	O(N)
Required Knowledge:	Linked List
Publish Date:	Jan 08 2016

```
       current = current->prev;
     }
    if(temp != NULL )
        head = temp->prev;

    return head;

}
```

Tested by John Pierce

Problem Tester's code :

```
# Python 3
def reverse(head):
    while head.next:
        head.prev, head.next, head = head.next, head.prev, head.next
    head.next, head.prev = head.prev, None
    return head
```



Cycle Detection

by harsha_s

Editorial by AllisonP

There are 3 scenarios to consider:

1. The list is empty (i.e., $head$ is *null*).
2. The list does not contain a cycle, so you can traverse the list and terminate once there are no more nodes (i.e., $next$ is *null*).
3. The list contains a cycle, so you will be stuck looping forever if you attempt to traverse it.

To solve this problem, we must traverse the list using two pointers that we'll refer to as $slow$ and $fast$. Our $slow$ pointer moves forward 1 node at a time, and our $fast$ pointer moves forward 2 nodes at a time. If at any point in time these pointers refer to the same object, then there is a loop; otherwise, the list does not contain a loop.

We recommend that you check out Floyd's Tortoise and Hare cycle-finding algorithm.

Statistics

Difficulty: Medium

Time Complexity:	O(N) Required

Knowledge: Linked List

Publish Date: Mar 08 2015

Set by vatsalchanana

Problem Setter's code :

C++

```cpp
bool has_cycle(Node* head) {
    Node* fast = head;
    Node* slow = head;
    while(fast != NULL && slow != NULL && fast->next) {
        fast = fast->next->next;
        slow = slow->next;
        if(fast == slow) {
            return 1;
        }
    }

    return 0;
}
```

Tested by AllisonP

Problem Tester's code :

Java

```java
boolean hasCycle(Node head) {
    Node fast = head;

    while(fast != null && fast.next != null) {
        fast = fast.next.next;
        head = head.next;

        if(head.equals(fast)) {
            return true;
        }
    }
    return false;
}
```

Python

```
def has_cycle(head):
    fast = head;

    while(fast != None and fast.next != None):
        fast = fast.next.next;
        head = head.next;

        if(head == fast):
            return True;

    return False;
```