Cycle Detection ☆

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.

Set by vatsalchanana

Problem Setter's code:

C++

```
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

```
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;
```

STATISTICS

Difficulty:	Medium
Time Complexity:	O(N)
Required Knowledge:	Linked List
Publish Date:	Mar 09 2015

This is a Practice Challenge

NEED HELP?

View discussions

View top submissions