Linked List Cycle

Given a linked list, determine if it has a cycle in it.

Follow up:
Can you solve it without using extra space?

Solution 1

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
 /**
 use faster and lower runner solution. (2 pointers)
 the faster one move 2 steps, and slower one move only one step.
 if there's a circle, the faster one will finally "catch" the slower one.
 (the distance between these 2 pointers will decrease one every time.)

 if there's no circle, the faster runner will reach the end of linked list. (NULL
)
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if(head == NULL || head -> next == NULL)
            return false;

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

        while(fast -> next && fast -> next -> next){
            fast = fast -> next -> next;
            slow = slow -> next;
            if(fast == slow)
                return true;
        }

        return false;
    }
};
```

written by autekroy original link here

Solution 2

```
    public boolean hasCycle(ListNode head) {
        if(head==null) return false;
        ListNode walker = head;
        ListNode runner = head;
        while(runner.next!=null && runner.next.next!=null) {
            walker = walker.next;
            runner = runner.next.next;
            if(walker==runner) return true;
        }
        return false;
    }
```

1. Use two pointers, **walker** and **runner**.
2. **walker** moves step by step. **runner** moves two steps at time.
3. if the Linked List has a cycle**walker** and **runner** will meet at some point.

written by fabrizio3 original link here

Solution 3

I cannot give a solution to make it possible. I can only do it in O(1) space using the two runner solution, which I think is the best one.

```
// set two runners
ListNode slow = head;
ListNode fast = head;

// fast runner move 2 steps at one time while slow runner move 1 step,
// if traverse to a null, there must be no loop
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow == fast) {
        return true;
    }
}
return false;
```

written by Ethanlugc original link here