Tu Vo

186 Followers About Following

The Tortoise and the Hare (Floyd's Algorithm)

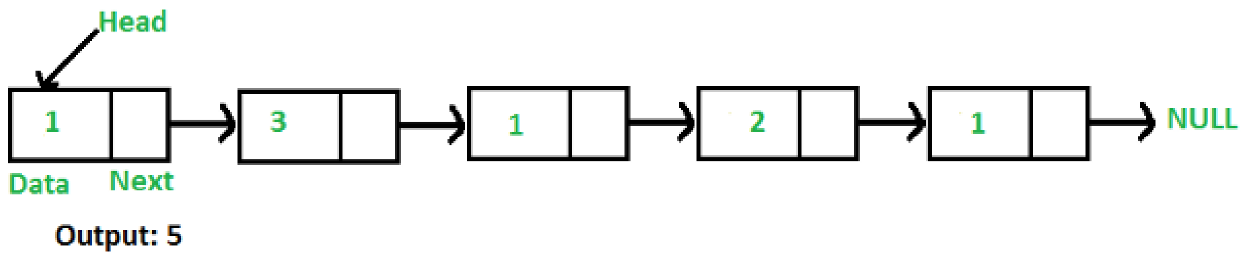
Tu Vo Mar 27, 2019 · 3 min read

Photo by Andy Brunner on Unsplash

Today we are going to write a function that determines if a singly linked list is circular and if it is, we are going to return the node where the cycle begins. This solution is based off of Robert Floyd's algorithm, first discovered in the 1970's.

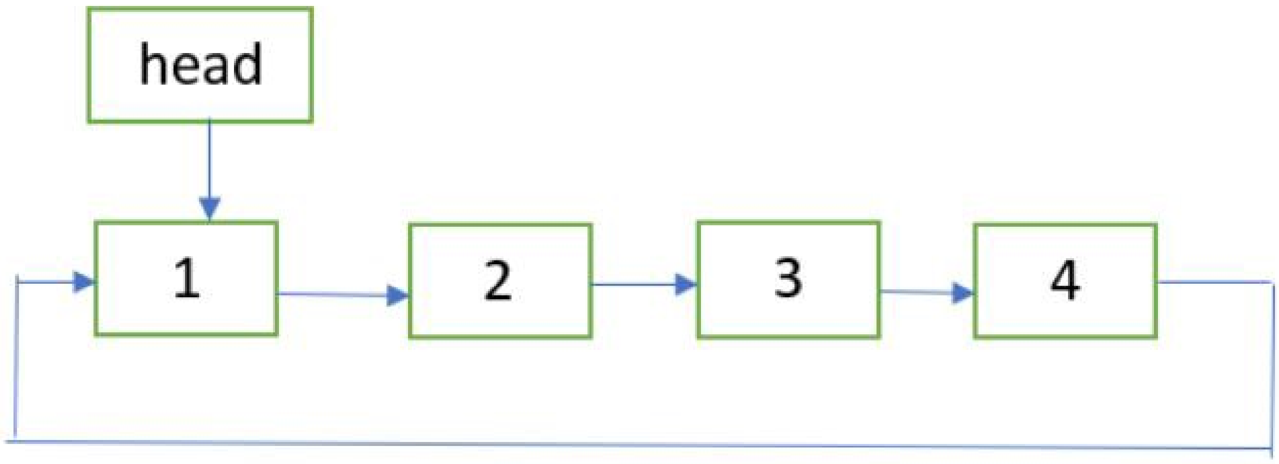
What is a Singly Linked List?

A singly linked list in C is a linear data structure where each element is a separate struct (or class/object in other languages). Linked list elements are not stored at contiguous to null.



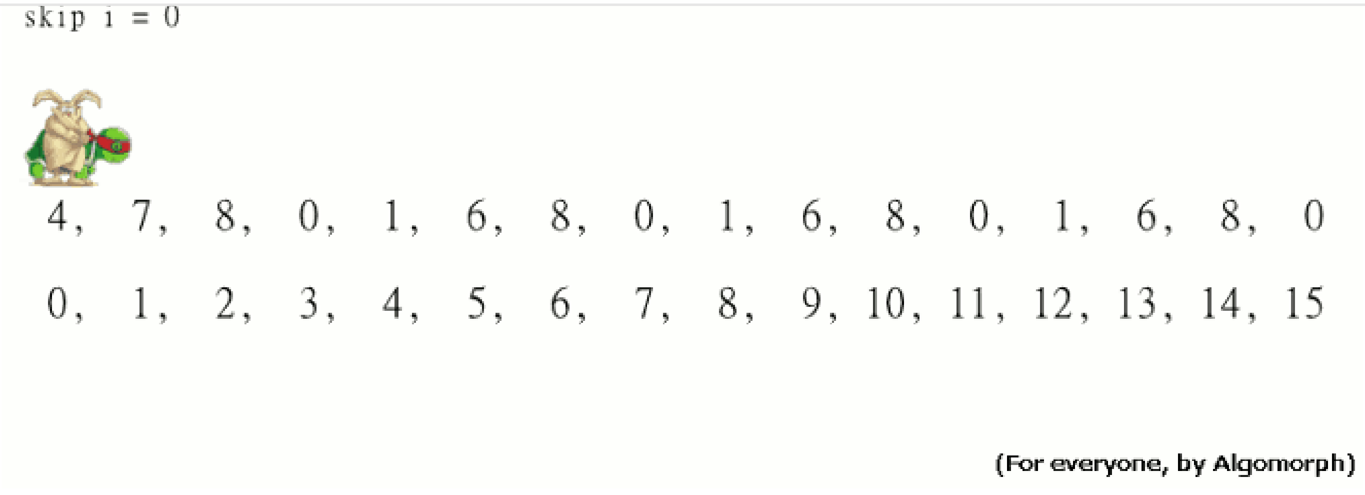
Circular Linked Lists

Circular lists differ in that there is no last node that points to null. Instead, there is a node somewhere that points back to a previous node, therefore making its traversal infinite.



In order to detect cycles in any given singly linked list, we must set two pointers that traverse the data structure at different speeds. If they meet, we can determine that the list was circular. Then if we set the first pointer back to the head and slow them both down to the same speed, the next time they will meet will be the point where the node started pointing backward. The pseudo-code is as follows:

```
1. Initialize two pointers (tortoise and hare) that both point to the
head of the linked list
2. Loop as long as the hare does not reach null
3. Set tortoise to next node
4. Set hare to next, next node
5. If they are at the same node, reset the tortoise back to the head.
6. Have both tortoise and hare both move one node at a time until
they meet again
7. Return the node in which they meet
8. Else, if the hare reaches null, then return null
```



Let’s translate that into C:

```c
node *find_loop(node *head)
{
        node *p1;
        node *p2;

        if (head == NULL)
                return (NULL);
        p1 = head;
        p2 = head;
        while (p2->next != NULL && p2->next->next != NULL)
        {
                p1 = p1->next;
                p2 = p2->next->next;
                if (p1 == p2)
                {
                        p1 = head;
                        while (p1 != p2)
                        {
                                p1 = p1->next;
                                p2 = p2->next;
                        }
                        return (p2);
                }
        }
        return (NULL);
}
```

floyd.c hosted with ♡ by GitHub view raw

Big O

The time complexity of this algorithm is linear: O(n).

Thank you for reading!

Photo by Cédric Frixon on Unsplash

Data Structures Algorithms Programming