Daily Coding Problem Blog

Daily Coding Problem #20

Problem

This problem was asked by Google.

Given two singly linked lists that intersect at some point, find the intersecting node. The lists are non-cyclical.

For example, given A = 3 -> 7 -> 8 -> 10 and B = 99 -> 1 -> 8 -> 10, return the node with value 8.

In this example, assume nodes with the same value are the exact same node objects.

Do this in O(M + N) time (where M and N are the lengths of the lists) and constant space.

Solution

We might start this problem by first ignoring the time and space constraints, in order to get a better grasp of the problem.

Naively, we could iterate through one of the lists and add each node to a set or dictionary, then we could iterate over the other list and check each node we're looking at to see if it's in the set. Then we'd return the first node that is present in the set. This takes O(M + N) time but also O(max(M, N)) space (since we don't know initially which list is longer). How can we reduce the amount of space we need?

We can get around the space constraint with the following trick: first, get the length of both lists. Find the difference between the two, and then keep two pointers at the head of each list. Move the pointer of the larger list up by the difference, and then move the pointers forward in conjunction and check if they match.

```
def length(head):
    if not head:
        return 0
    return 1 + length(head.next)

def intersection(a, b):
    m, n = length(a), length(b)
    cur_a, cur_b = a, b

    if m > n:
        for _ in range(m - n):
            cur_a = cur_a.next
    else:
        for _ in range(n - m):
            cur_b = cur_b.next

    while cur_a != cur_b:
        cur_a = cur_a.next
        cur_b = cur_b.next
    return cur_a
```

© Daily Coding Problem 2019

Privacy Policy

Terms of Service

Press