Floyd's Cycle Detection Extension Proof

Minh Le

August 2022

1 Introduction

The slow-fast pointer algorithm from Floyd tells us whether there is a cycle in our list. We can further extend this by also showing how to find the entrance of the cycle. The implementation is simple, we use Floyd's to detect whether there is a meeting point between the two pointers, and then have two slow pointers approaching the entrance of the cycle. However, one might question how this works exactly, why is the distance from the meeting point of both pointers to the entrance of the cycle **equivalent** to distance from the starting point of two pointers to the entrance of the cycle?

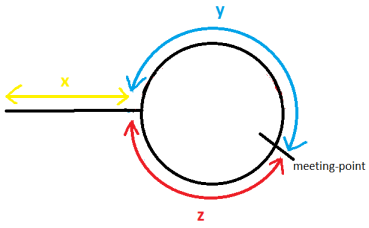

Figure 1: Visual Representation

In the figure above, we represent x as the distance from where both pointers starts to the entrance of the cycle, y as the distance from the entrance to the meeting point, and z as the remainder distance of the cycle. Our extension algorithm utilizes this fact.

2 Implementation

Pseudo code:

```
Initialize both slow and fast pointers at the head of list
Begin Loop with condition to traverse slow and fast pointers:
    Traverse by single node for slow pointer
```

```
    Traverse by double nodes for fast pointer
    If slow and fast converges:
        Break
Reassign slow to the head of list
Begin Loop if previously a meeting point was detected:
    Traverse by single node for slow pointer
    Traverse by single node for fast pointer
    If slow and fast converges:
        Return slow or fast
```

3 Formal Proof

Let us represent the variables the same way as defined in the figure above.

$$\begin{aligned} x &= D_{start-entrance} \\ y &= D_{entrance-meet} \\ z &= D_{meet-entrance} \end{aligned} \tag{1}$$

The distance covered by the slow pointer is the starting point to the meeting point. We also note that by definition, the fast pointer travel twice the distance of the slow pointer. Let us define another variable for these information.

$$\begin{aligned} M = D_{slow} &= x + y \\ D_{fast} &= 2D_{slow} = 2(x + y) \end{aligned} \tag{2}$$

The key here is defining our fast pointer by the distance from the start to the meeting point along with some multiples of the cycle length. We know that after the fast pointer reaches the meeting point, after traversing some cycles it will end at the meeting point again. Thus,

$$\begin{aligned} 2(x + y) &= 2M \\ &= M + kL \end{aligned} \tag{3}$$

where L is the length of the cycle and k is some integer. Now we can also say that:

$$\begin{aligned} 2M &= M + kL \\ M &= kL \\ x + y &= kL \end{aligned} \tag{4}$$

But this means that the distance from the start to the meeting point is **a multiple of the length of the cycle**. Is it always the case? Yes, because we define our meeting point to be dependent on the length of the cycle in the first place, thus arbitrarily works for all cases of our variables.

Let's take a step back to have a sanity check, what are we trying to show? Basically this formal proof is telling us why the **second loop** of our pseudo-code

works. Spend some time to understand what the code is actually doing and use the figure above to visualize the situation better.

We can finally reach the conclusion, isolating x, we find that:

$$\begin{aligned} x + y &= kL \\ x &= kL - y \end{aligned} \tag{5}$$

So the distance from the starting to the entrance is some multiple of the cycle subtract the distance from the entrance to the meeting point. If k is 1, we have indeed, that the distance from the starting point to the entrance is **the same as** the remainder distance from the meeting point to the entry of the cycle again. Or x = z. Then formerly for any k, we have:

$$\begin{aligned} x &= kL - y \\ x &= (k-1)L + L - y \\ x &= (k-1)L + z \end{aligned} \tag{6}$$

Contradictory to some beliefs, we also have to consider cases where the cycle length differs significantly from the distance from the starting point to the entrance. The fast pointer might have to traverse multiple times before the slow pointer finally meet up. Thus, we can finally come to the conclusion that the distance from the meeting point to the entry of the cycle is **congruently equivalent** to the distance from the starting point to the entrance modulo the length of the cycle. That is,

$$z \equiv x \; modulo \; L \tag{7}$$

4 Conclusion

Our final derivation matters not to the implementation of our extended algorithm. If we fix a slow pointer at the meeting point and re-initialize one of our pointers to the starting point, we can guarantee that these two pointers will meetup at the entrance. By definition, z = kL + x, this means that by the time our slow pointer from the starting point traverses distance x, our other pointer at the meeting point will travel some distance z that is also just some distance x plus some amount of cycles. Altogether, if both ends up at some distance x from where it is, both pointers will meet up at a distance we conveniently defined to be the entrance of the cycle.