Analysis and Generalization of the Tortoise and Hare Algorithm for LinkedList Cycle Detection

June 12, 2024

Abstract

This document examines the application of the Tortoise and Hare algorithm, traditionally used to detect cycles in linked lists. We first explore, code, and prove the termination and correctness for the case where the fast pointer moves at 2 units. Subsequently, we extend our analysis to consider the fast pointer advancing by any n units, with $n \geq 2$. This generalization hinges on the fact that the relative speed difference between the two pointers, $n - 1$, and the cycle length, C, is coprime. We present a counterexample to illustrate the impossibility of universal generalization for the algorithm. However, we also acknowledge that alternative assumptions may enable a broader validation of this general case.

Linked List Cycle Detection Problem

Consider a linked list where each node contains a generic data value and a reference to the next node. Without loss of generality, we define the generic ListNode structure in Java as follows:

```
class ListNode<T> {
    T val;
    ListNode<T> next;

    ListNode(T x) {
        val = x;
        next = null;
    }
}
```

Problem Statement

Determine if a linked list has a cycle in it. This is to say, whether any node in the list can be reached again by continuously following the next pointer.

Optimal Solution: Tortoise and Hare Algorithm

The Tortoise and Hare algorithm uses two pointers, the slow pointer (Tortoise) moves one step at a time, and the fast pointer (Hare) moves n=2 (two) steps at a time. The algorithm is as follows:

```
public boolean hasCycle(ListNode head) {
    if (head == null) return false;

    ListNode slow = head; // Tortoise
    ListNode fast = head; // Hare

    while (fast != null && fast.next != null) {
        slow = slow.next;       // move slow by 1
        fast = fast.next.next;  // move fast by 2

        if (slow == fast) {
            return true; // Cycle detected
        }
    }

    return false; // No cycle
}
```

Introduction

Given a LinkedList of length M, we will prove that our algorithm will terminate by the Nth node, where $N = L + C$. Here, L represents the length of the initial non-cyclic segment, and C represents the length of the first cycle. If the LinkedList starts with a cycle, we set $L = 0$. Likewise, if there does not exist a cycle, we set $C = 0$. Given the ListNode structure, it is impossible for M to not equal N. This is because nothing can come after a cycle. We will first prove the primary algorithm where the fast pointer moves $n = 2$ steps at a time. Then, we will generalize the proof for all $n \in Z$, where $n \geq 2$ under certain specified assumptions. It will be shown, however, that the general case does not universally hold beyond the initial assumption of $n = 2$.

Proof of Termination and Correctness for n=2

1. If there is no cycle ($C = 0$), the fast pointer reaches the end of the list, a null node, terminating the algorithm and having the pointers never meet.

2. If there is a cycle, both pointers will eventually enter the cycle after L steps. They will start in the cycle if $L = 0$.

3. Once in the cycle, the distance between the slow and fast pointers is at most C and decreases by one with each iteration of the loop until they

meet. This is because the fast pointer closes the gap by one additional node for every cycle iteration. There is no possibility for the fast pointer to jump over the slow pointer with n=2.

4. Given the cycle's finite size C, the maximum number of steps required for the fast pointer to catch the slow pointer within the cycle does not exceed C iterations. Note that if L=0, the two pointers will require exactly C iterations.

5. Thus, the algorithm will terminate within $N = L + C$ steps since the pointers will meet within the cycle if a cycle exists.

6. The asymptotic time complexity of this solution is $O(N)$, with a space complexity of $O(1)$, as we only store two pointers regardless of the size of the list.

General Proof for Any Step Size n

1 Setup

Consider a linked list with a cycle of length C. Let:

- $s = 1$ be the step size of the slow pointer (Tortoise).
- $f = n$ be the step size of the fast pointer (Hare), where $n \geq 2$.

Assume the slow pointer starts at position b and the fast pointer starts at position a within the cycle.

2 Conditions for Pointer Convergence

The pointers will meet if and only if there exists a positive integer t, such that:

$$a + n \cdot t \equiv b + t \pmod{C}.$$

Simplifying, we have:

$$n \cdot t - t \equiv b - a \pmod{C},$$
$$(n-1) \cdot t \equiv b - a \pmod{C}.$$

3 Mathematical Proof

3.1 Lemma: Existence of t

If $\gcd(n-1, C) = 1$, there exists a positive integer t such that $(n-1) \cdot t \equiv b - a \pmod{C}$.

Assuming $\gcd(n-1, C) = 1$, we know that $n-1$ and C are coprime. Thus, $n-1$ acts as a generator of the unit group of integers modulo C. For any integer

k, the multiples $(n-1)\cdot t$, for $t = 0, 1, 2, \ldots, C-1$, will produce distinct residues modulo C because there are no divisors other than 1 common to $n-1$ and C.

This property ensures that as t varies from 0 to $C-1$, the expression $(n-1)\cdot t$ modulo C will cover all possible residues from 0 to $C-1$. Hence, there must exist some t such that:

$$(n-1)\cdot t \equiv b-a \pmod{C}.$$

This t is the exact number of steps needed for the fast pointer to catch up to the slow pointer, starting from different positions within the cycle.

Failure to Generalize

Consider a scenario where C is the clock size 12, the fast pointer starts at 9 o'clock moving 5 units, and the slow pointer starts at 12 o'clock moving 1 unit:

- The fast pointer's position after t steps is $(9+5t) \mod 12$.
- The slow pointer's position after t steps is $(12+t) \mod 12$.
- They would only collide if $(9+5t) \mod 12 = (12+t) \mod 12$, simplifying to $(4t+9) \mod 12 = 0$.
- Solving $(4t+9) \mod 12 = 0$ shows that t must be a multiple of 3 plus an offset of 0.75 (not an integer), so they never meet at the same time.

This is a counterexample to the prospect of universally generalizing the Tortoise and Hare algorithm for any step size n. We proved that this algorithm generalizes under the assumption that the cycle length C and the step size $n-1$ of the fast pointer are coprime. However, there may be more relaxed or different assumptions that can still allow the algorithm to function correctly. Further investigation into these conditions could yield additional configurations under which the algorithm successfully detects cycles, expanding the viability of a generalized algorithm. Under the condition we proved, the initial positions of the pointers do not influence the algorithm's capacity to detect a cycle. However, if $n-1$ and C are not coprime, the algorithm we cannot guarantee that a cycle will be detected. Naturally, for any n, there exists valid starting positions to detect if it is a cycle, but without the coprime assumption, we cannot guarantee it for any starting points.

3.2 Theorem: Bounded Time Complexity

Under the condition $\gcd(n-1, C) = 1$, the pointers will meet within at most C steps, leading to a time complexity of $O(N)$ for the algorithm, where N is the length of the list.

Given the lemma, since $(n-1)\cdot t$ covers all possible residues modulo C and matches $b-a$ within C cycles, the fast pointer will meet the slow pointer within C steps after they both enter the cycle. If there is no cycle, or a non-cyclic

section before the cycle, the fast pointer will reach the end of the list in fewer steps than the size of the non-cyclic portion. Therefore, the time complexity for detecting a cycle is $O(N)$, with $N = L + C$, where L is the length of the non-cyclic part of the list.

4 Conclusion

This formal proof confirms the conditions under which the Tortoise and Hare algorithm successfully detects cycles in a linked list. The proof rigorously demonstrates that the algorithm's performance is optimal when $n - 1$ and C are relatively prime, ensuring efficient cycle detection within the theoretical bounds.