CS 284 B: Quiz 2
Spring 2022

Exercise 1 (*100 points*)

We say a linked list has a cycle if there is a node that points "back" to a previous one. Here is a picture of such a linked list:

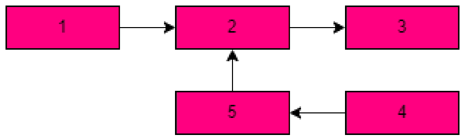

Figure 1: Cycle in a LinkedList

The aim of this exercise is to implement a method `public Boolean hasCycle()` that determines whether the recipient list has cycles. There are various ways this can be done.

One might traverse the list "marking" the nodes that have been seen and making sure that already seen nodes are not visited twice. Another approach is to store the visited nodes in a table and then check that we don't visit the same node twice. However, both of these approaches require consuming additional space for the marks.

Yet another approach is to follow so called Floyd's Algorithm. One starts with two references to head, say `slow` and `fast`. We then advance `slow` one node at a time and `fast` two nodes at a time. If there is no cycle, `fast` will eventually be null. If there is a cycle, then `slow` and fast will definitely meet due to the following observations:

1. When `slow` enters the cycle, `fast` must be inside the cycle already. Let us assume that `fast` is at a distance `k` from slow when this happens.

2. Next notice that the distance between `slow` and `fast` is initially 0 and then increases by one after every iteration. After an iteration following `slow` entering the cycle, the distance between `slow` and `fast` becomes `k + 1`, after two iterations it becomes `k + 2`, and so on. If the cycle has length n, then they will meet in at most `k + n` iterations.

Examples

1. `Example 1`

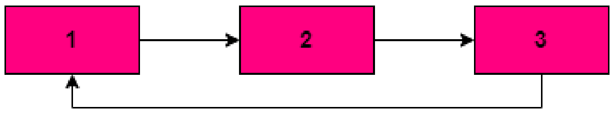

Figure 2: Example 1

Output: `true`

Reasoning: The next node of 3 points back to 1, creating a cycle.

2. `Example 2`

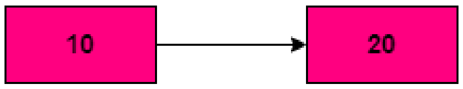

Figure 3: Example 2

Output: `false`

Reasoning: There is no cycle in the list.