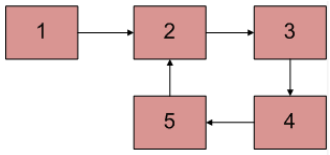
CS 284: Ungraded Exercise for Lists

These exercises are not graded, nor do they have answers, you can practice by yourself, if you have questions, come to my office hour.

Exercise 1

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:



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 a cycle, then they 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.

Here is an sample test where your code should return true:

```
// build a list
SingleLL<Integer> l = new SingleLL<Integer>();
for (int i=0;i<5;i++) {
        l.add(i);
}
// create a loop
l.head.next.next.next.next = l.head;
// test your code
System.out.println(l.detectLoop());
```

Exercise 2

Write a method `public boolean repetitions()` that determines whether the recipient list has repeated elements or not. For example, the result of applying this operation to `[2;9;5;4]` should be false, but true for `[2;4;6;8;2]`.

Exercise 3

Write a method `public void stutter()` that modifies the recipient list by duplicating each of its elements. For example, the result of applying this operation to `[2;9;5;4]` should be `[2;2;9;9;5;5;4;4]`. If the recipient list is empty, then the operation has no effect.

Exercise 4

Write a method `public boolean sorted()` that determines whether the list that starts at the node that receives this method is sorted in ascending order or not. For example, the result of applying this operation to `[2,9,5,4]` should be false, but true for `[2,4,6,8]`.