Data Structures and Algorithms.
Universidad Carlos III de Madrid

Lab Case. COURSE 2022-2023

Lab case statement.

The lab case consists of two phases. This document describes the statement of the first phase. Statement for phase 2 will be published in the coming weeks.

Phase 1 – Linear ADT

Given the SList class, implementation of the List ADT using a single reference to the first node of the list (_head), you are asked to implement a subclass of the SList class, which you will call SList2, and that will include the following methods:

1. **delLargestSeq(self)**, the calling list (self) is an object of the class SList2. This method deletes all the elements of the largest sequence of equal numbers from the calling list. The method should find the position of the first element of the largest sequence and then delete all the elements of that sequence. If there is not the largest sequence of equal numbers, then the method should remove the last sequence in the list. The method doesn't return anything, it just modifies the calling list. Let's see some cases:

Input:	*self: 3->3->3->4->5->6->6->6->**7->7->7->7**->2*
Output:	*self: 3->3->3->4->5->6->6->6->2*
Explanation:	*Since 7->7->7->7 is the largest sequence of equal numbers.*

Input:	*self: 8->8->8->8->4->5->6->6->6->**7->7->7->7**->2*
Output:	*self: 8->8->8->8->4->5->6->6->6->2*
Explanation:	*Since 7->7->7->7 is the last largest sequence of equal numbers in the list.*

Input:	*self: 6->6->8->8->4->4->**12->12***
Output:	*self: 6->6->8->8->4->4*
Explanation:	*Since there is no largest sequence of equal numbers. Removes the last sequence in the list*

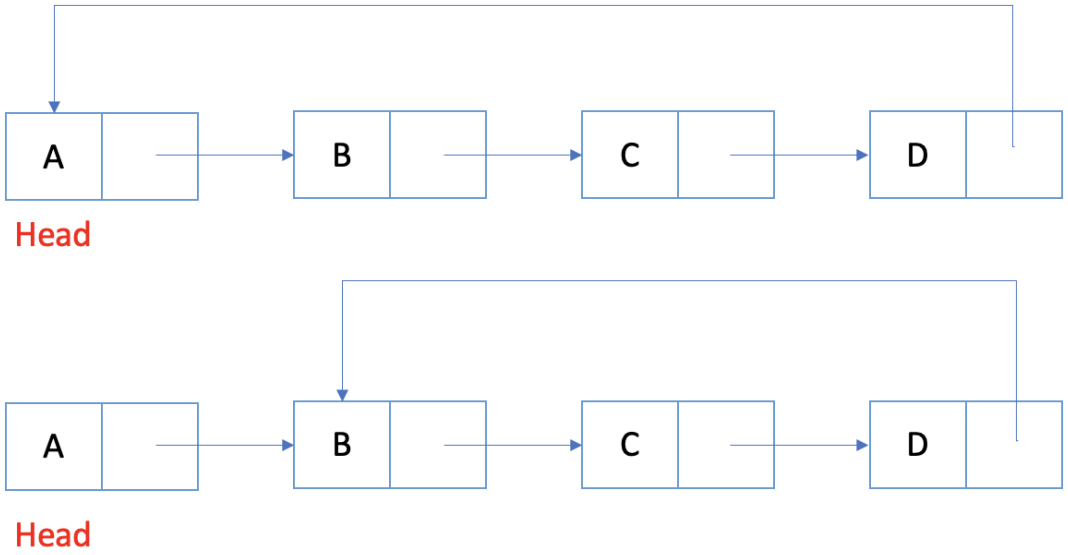
Input:	*self: 1->2->3->4->**5***

Output:	*self: 1->2->3->4*
Explanation:	*Since an element is a valid sequence in the list*

Input:	*self:* ***10***
Output:	*self: empty*
Explanation:	*Since there is only one element in the list. Removes that element*

Input:	*self: empty*
Output:	*self: empty*
Explanation:	*Since the list is empty*

2. **fix _loop(self).** The purpose of this method is to detect if the calling list contains a loop and, if so, to break it. A list contains a loop if the next reference of its last node points to another node in the list instead of None. The following figure shows two examples of SList containing a loop (in the first case the list is completely circular; in the second only part of it has a loop - nodes B, C and D -). In both cases node A is the head of the list.

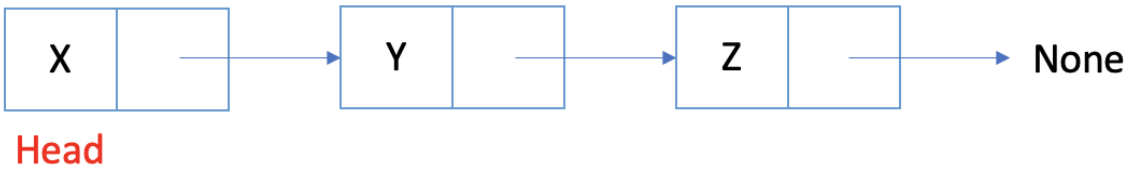


If the loop is detected, the method must break the loop and return True, or return False and not modify the list otherwise. Note that a SList can have a single loop or none at all, since each node has only a single reference to its next node.

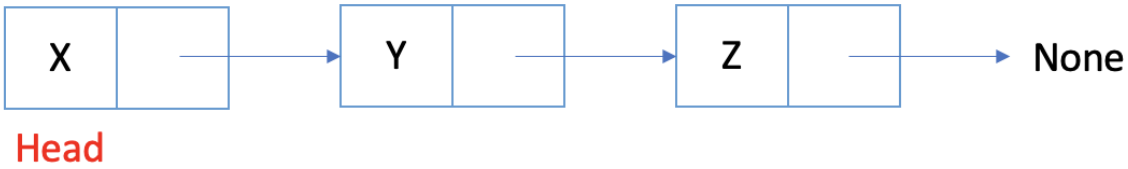
The following examples show the expected result in each case. If the list is empty, the method should return False.

Note: To solve this exercise you are NOT allowed to make use of the size of the list, i.e., you cannot use the size attribute or the len function to determine the final node of the list directly.

***Input**:*

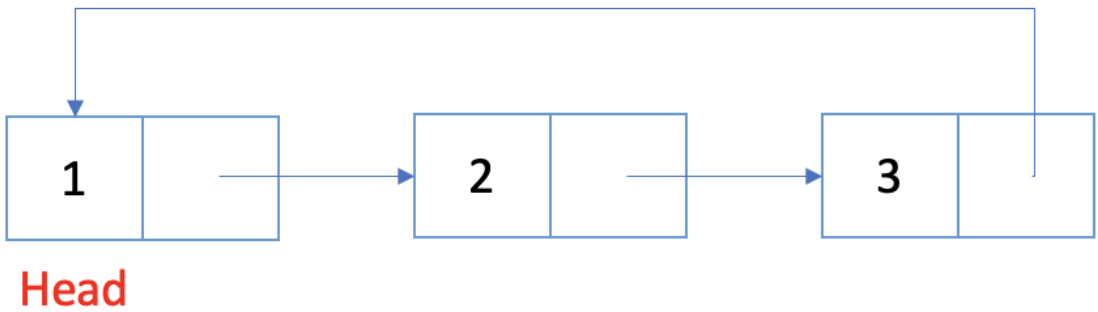


***Result**:*

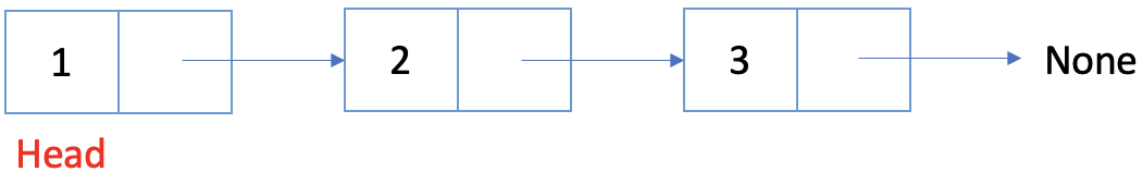


Output: ***False***

***Input**:*

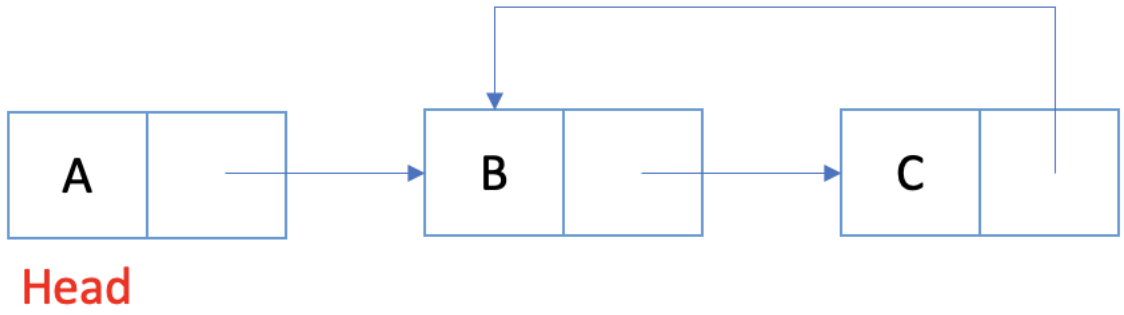


***Result**:*

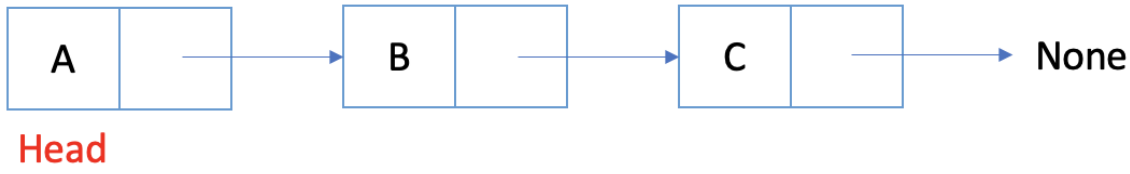


Output: ***True***

Input:



Result:



Output: ***True***

3. **leftrightShift(self, left, n),** the calling list (self) is an object of the class SList2, left a Boolean value and n is a positive integer greater than 0. This method left shifts (left =True) or right shifts (left =False) the single linked list by n nodes when n is smaller than or equal to the length of the linked list. The method doesn't return anything; it just modifies the calling list. Let's see some examples:

Input:	*self: 1->2->3->4->5->6->7 left = True, n=2*
Output:	*self: 3->4->5->6->7->1->2*

Input:	*self: 1->2->3->4->5->6->7 left = True, n=1*
Output:	*self: 2->3->4->5->6->7->1*

Input:	*self: 1->2->3->4->5->6->7 left = True, n=4*
Output:	*self: 5->6->7->1->2->3->4*

Input:	*self: 1->2->3->4->5->6->7 left = False, n=2*
Output:	*self: 6->7->1->2->3->4->5*

Input:	*self: 1->2->3->4->5->6->7 left = False, n=1*
Output:	*self: 7->1->2->3->4->5->6*

Input:	*self: 1->2->3->4->5->6->7 left = False, n=4*
Output:	*self: 4->5->6->7->1->2->3*

Input:	*self: 1->2->3->4->5->6->7 left = True or False, n=7*
Output:	*self: 1->2->3->4->5->6->7*
Explanation:	*Since n is equal to the size of the list, n left shift returns the same list*

Input:	*self: 1->2->3->4->5->6->7 left = True or False, n=10*
Output:	*self: 1->2->3->4->5->6->7*
Explanation:	*Since n is greater than the size of the list, the list is not modified.*

Input:	*self: empty*
Output:	*self: empty*

4. **class Test (unittest.TestCase):,** implement the **class Test** which consists of a set of test cases that are used to test the implementation of the previous methods. A test case is the individual unit of testing. It checks for a specific response to a particular set of inputs. unittest provides a base class, TestCase, which may be used to create new test cases. In the implementation of this class, you can use python lists to create your test cases.

Rules:

1. A solution is considered correct if it is robust (has no errors for any input), correct (solves the problem), and efficient (there may be several solutions, you should always try to find the most efficient solution)

2. In the implementation of the lab case, it is not allowed to use Python structures such as Lists, Queues or Dictionaries, etc.

3. The Lab case must be carried out by a group of two members (both must belong to the same reduced group). In no case will groups with more than two members be allowed. Individual groups are not allowed either, as one of the skills to be assessed will be teamwork. If you don't have a partner, please send an email to your lab teacher.

4. Along with the statement of this phase, the student is given the following files:

a. The slist.py file that contains the implementation of the SList (Single Linked List) and Node (Simple Linked Node) classes. The SList class only stores the reference to the first node of the list and the size of the list. **This file cannot be modified in any case. If you detect any errors, please notify your teacher.**
b. The phase1.py file that you will have to complete to elaborate the solution to this first phase. The file already contains some code that will help you get started.
c. The unittest-phase1.py file that you will have to elaborate the solution to this first phase. This file must contain the necessary tests to evaluate each of the methods proposed in this phase.

5. Delivery method: For each reduced group, a task entitled **'Phase 1 Delivery**' will be posted on the corresponding Aula Global course. The deadline for this first phase is in the 9th week.

6. Delivery format: a zip file whose name will be formed by the two NIAS of the students that make up the group, separated by a hyphen. For example, 10001234-10005678.zip. Only one of the two members will be responsible for uploading the group solution. The zip file must contain the files: slist.py, unitest_phase1.py and phase1.py.

7. **Defense**: it will be held during the face-to-face class on the 14th week, in the schedule of the small groups. The defence of the practical case is an oral exam. Attendance is mandatory. If any student does not attend, their grade in the practical case will be NP. During this defence, the teacher will ask each member of the team a series of questions that they must answer individually. Each member of the team must be able to discuss any of the decisions made in the design and implementation of any of the functionalities described in the case study. The final mark of the practical case will be conditioned by the mark obtained in the defence. If a student is not able to discuss and defend certain decisions, they will not be graded with respect to these, even if they have been correctly implemented.

8. It is recommended to follow the recommendations described in Zen of Python (https://www.python.org/dev/peps/pep-0020/) and the style guide (https://www.python.org/dev/ peps / pep-0008 /) published on the official Python website.