Due: Oct. 16, 2022 at 11:55pm

Overview

Using singly-linked lists, create a working implementation of the stack and queue data structures. With these two data structures, solve the simple problem of palindrome detection.

Data Structures

The elementary structure we will use in this exercise is a node. The particular node is capable of storing a “data” element and a “pointer” element. The “data” element can be used to store information relevant to our problem and the ”pointer” element is used to point to the next node in our stack or queue. Using the provided skeleton code in **lab1.py**, fill in the following methods for the class **Node**:

Node

setData(self, data) : Store input data in the node’s “data” attribute. setNext(self,
next_node):Store input next_node in the node’s “pointer” attribute. getData(self) : Fetch
the data stored in the node.
getNext(self) : Fetch the pointer stored in the node.

Using the node structure as a building block, implement the stack and queue structures. Each should be implemented as a class using the following methods:

Stack

str(self) : return a formatted string for printing the stack. push(self,
newData): push a node containing newData onto the stack. pop(self) : return
data from top of the stack and remove top node. isEmpty(self) : Check if stack
is empty.

Queue

str(self) : return a formatted string for printing the queue.
enqueue(self, newData) : Create new node containing newData and add it to the queue.
dequeue(self) : return data stored at the head of the queue and update the head.
isEmpty(self) : check if queue is empty.

For both stack and queue, the method_str_(self) should leave the contents of the data structure unchanged. You may NOT use Python built-in data abstractions like lists or dictionaries. You may add additional methods but should NOT add additional public data to Stack or Queue. You also should not change the name of any of the classes, files, or existing methods. Arguments and return types for public methods should also be left unchanged.

Palindrome Detection

A palindrome is a sequence of characters which reads the same backward or forward (e.g. ‘radar’). Define a function, isPalindrome(s), which processes a string s using **Stack** and **Queue**. You should think about how to use these two data structures to determine if a string is a palindrome.

isPalindrome(s) should return True if s is a palindrome and False otherwise. Addi- tionally, your program should be able to handle input strings which contain capital letters and spaces. Below are some input/output examples for isPalindrome():

```
>>> isPalindrome('Hello') False
>>> isPalindrome('ni t I n') True
>>> isPalindrome('&$(^^)$&') False
>>> isPalindrome('My gym') True
>>> isPalindrome('12TENET12') False
>>> isPalindrome('63488436') True
```

We strongly suggest that while you create each individual class, you also write a robust set of tests. That is, if you write the **Stack** class, then immediately afterwards you should write tests for **Stack**. This makes it easier to isolate edge cases and helps prevent problems later on. **test_lab1.py** has some test cases to start you off but you should expand on the test file. Some things to keep in mind while making tests:

— dequeue() / pop()on an empty queue/stack should raise an exception.

— Capital letters and spaces are valid inputs.

At a minimum your code should be tested with the following minimal test set:

— pushing one item to an empty stack

— popping one item from an empty stack (should raise exception for AttributeError)

— popping one item from a stack with one item

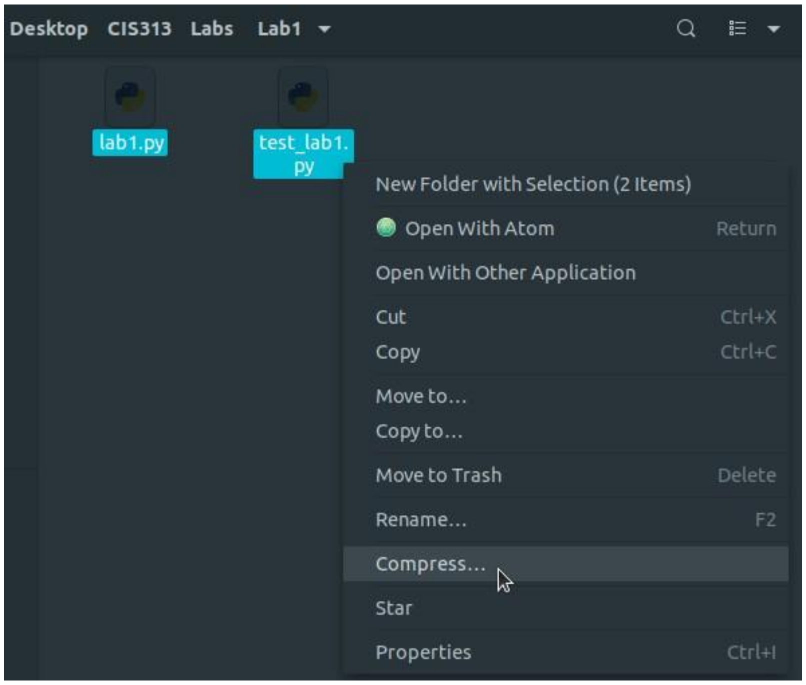
— enqueueing one item to an empty queue

— dequeueing one item from an empty queue (should raise exception for AttributeError)

— isPalindrome('TaCo CaT') returnsTrue

Files to Submit

You should submit the **lab1.py** (with the filled in methods for all the classes) along with the **test_lab1.py** file with all the test cases that you had written to test your code. These two files should be compressed and uploaded in **.zip** format using the filename **submission.zip**. To do this properly, make sure you select both files together then compress them. Do not compress the folder containing these two files otherwise the **.zip** file will contain a subfolder.



Grading

The assignment will be graded as follows:

— AutoGrader - 70pts

— Style - 30pts

  - To earn points for style, your code must be clear enough for us to understand and classes, methods, and functions have appropriate docstrings.

- You may not use any data structures from the python library including lists and dictionaries.
- If you do not attempt the Extra Credit problem, you may only see a maximum of 70/120 on GradeScope. In that case, you might also see some grading test cases (append with '_EC') fail which are meant for the extra credit questions.

Extra Credit - 20pts

To receive points for the extra credit you must implement a queue using two stacks:

- Create a class called **TwoStackQueue**, which has two stacks as member variables.
- The signature of this class should be the same as the **Queue** class.
  - It should have the same public methods.
  - Each public method should match in argument types (argument number, etc), and return type.
  - Note that this is not the same as having the same methods.
    - In particular, you may modify the bodies of public methods, and change add or remove private methods
    - Use the **TwoStackQueue** instead of a **Queue** in the isPalindromeEC() func- tion.

Testing Extra Credit

In order to test your TwoStackQueue class, simply test it in all the same ways you tested your original program.