CS/IT 200

Lab 3: Stacks

Recommended Due Date: Thursday, October 1, 11:59pm

Submission

- Submit all code in a single zip file to the Lab 3 assignment folder on Kodiak.

Goal: Implement a fixed-capacity stack that allows old elements to "leak" out when adding to a full stack.

Part I – Leaky Array Stack

One common use of stacks is to allow users to "undo" actions in various applications. While this ability can certainly be implemented using a stack with unlimited capacity, many applications provide limited support for undo operations with a fixed-capacity stack. In such applications, you can only undo the last *K* operations, where *K* is the capacity of the stack. In such stacks, when the `push()` method is invoked while the stack is full, the typical strategy is to "leak" the oldest element from the bottom of the stack in order to make room.

In `leaky_stack.py`, define the class `LeakyArrayStack` with the following requirements:

- The class must use an array-based list to store the data, just as we did in `ArrayStack`.
- In addition to the list, there must be an instance variable for the maximum capacity of the stack. The constructor should have a parameter for this capacity, which must be at least 1. Your constructor should raise a `ValueError` when given an invalid capacity.
- When you initially create the list, instead of creating an empty list, create a list whose size is equal to the given capacity. (For example, if given a capacity of 3, create your list as `[None, None, None]`.) By doing this, there will be no reason for the size of the underlying list to change. As a result, any operation that changes the size of the stack must never cause the size of the list to change (thereby preventing resizing from impacting running times).
- Your stack must have the following functions:
    - `push` (as described above)
    - `pop`
    - `top`
    - `is_empty`
    - `__len__`
- Each function must have a comment describing its purpose.
- You may also wish to have a `__str__` function for testing purposes and to help you demonstrate that the leaked item has fallen out. (This will not be graded.)
- The `push()` function should take, at worst, $O(n)$ time. However, a $O(1)$ solution exists, and I encourage you to find it. The other functions must run in $O(1)$ time.

Create a `main()` function that demonstrates the features of the `LeakyArrayStack`. You must demonstrate the consequences of pushing on to a full stack.

Part II – Leaky Linked Stack

Repeat Part I, but use Nodes instead of an array-based list to implement a `LeakyLinkedStack` class.

In `leaky_stack.py`, define the class `LeakyLinkedStack` with the following requirements:

- The class must use nodes to store the data, just as we did in `LinkedStack`.
- In addition to the list, there must be an instance variable for the maximum capacity of the stack. The constructor should have a parameter for this capacity, which must be at least 1. Your constructor should raise a `ValueError` when given an invalid capacity.
- You may determine for yourself what extra linked list features (e.g. sentinels, circular, etc.) would be beneficial for this task.
- Your stack must have the following functions:
    - `push` (as described above)
    - `pop`
    - `top`
    - `is_empty`
    - `__len__`
- Each function must have a comment describing its purpose.
- You may also wish to have a `__str__` function for testing purposes and to help you demonstrate that the leaked item has fallen out. (This will not be graded.)
- The `push()` function should take, at worst, $O(n)$ time. However, a $O(1)$ solution exists, and I encourage you to find it. The other functions must run in $O(1)$ time.

Add code to your `main()` that demonstrates the features of the `LeakyLinkedStack`. You must demonstrate the consequences of pushing on to a full stack.

What to Submit

- Submit a zip file containing all code: `leaky_stack.py`, and any other modules that your code relies upon.

Rubric

Grade	Overall	Part I	Part II
4	At least a 3 in both parts and a 4 in one part.	All functions are correct and meet expected time requirements.	All functions are correct and meet expected time requirements.
3	At least a 2 in both parts and at least a 3 in one part.	All functions behave as expected, but one function fails to meet	All functions behave as expected, but one function fails to meet

		the expected time requirement. -OR- Expected behavior is present in all functions, but push() and/or pop() changes the size of the underlying list.	the expected time requirement.
2	At least a 2 in both parts.	Multiple functions fail to meet the expected time requirement. -OR- Push() and/or Pop() has a significant bug.	Multiple functions fail to meet the expected time requirement. -OR- Push() has a significant bug.
1	At least a 1 in both parts.	The push() function does not demonstrate “leaking” behavior and/or no capacity is defined. All functions are present.	The push() function does not demonstrate “leaking” behavior and/or no capacity is defined. All functions are present.
0		Not implemented using an array-based list, or fails to meet the standards for a 1.	Not implemented using Nodes, or fails to meet the standards for a 1.