Lab 4 – CSCI 112

Due Date: Monday, Sep 21 at 5:00pm EST

Information

- This lab is intended to be completed **individually.**
- The files must be submitted with the exact file name provided in this file. If the file names do not match you will receive **zero** points for that file.
- Before you submit, make sure that your code runs. Any code which does not run without errors will receive **zero** points.
- Do not share your work with anyone other than Professor Khan or the TAs. You may discuss algorithms, approaches, ideas, but **NOT** exact code.
- If you submit work after a second past the due date **WILL** be locked out from submission.

Review

Bags

Bags are a container which can contain duplicate values but does not enforce a certain order to the items. Think of it like an actual bag in which you place unique marbles inside. You can have more than one blue marble inside the bag, but it doesn't matter when you placed that marble inside. Unlike our marble example, however, the order of the items is not required to be **random**. That is, the only rule on the order is that there is no guaranteed order. (Unless it is specifically a sorted bag…)

Naming Conventions

When defining a class in Python, a programmer may opt to use a specific format to indicate to others who might use the class that these variables are **private**. Private means that only the object instance itself should be directly accessing it, and nothing outside should use it. For example, if your class looks like the following:

```
class A(object):
    def __init__(self):
        self._x = 10
```

Then outside access of that object should not directly reference `_x`. It needs to use a **`getter`** method, or accessor:

```
class A(object):
      def __init__(self):
            self._x = 10
      def getX(self):
            return self._x

aObj = A()
aObj.getX()

# Do not use aObj._x!
```

Assignment

Unzip the Lab_4 archive and put your names in each file. Open the file **bagInterface.py** and review the interface for the bag collections. Note that this code can be loaded into the shell, but the methods won't do anything.

Open the file **testBag.py** and examine the code for testing a bag type. Note that the test function creates some bag objects and exercises the methods on them. Note that this function can be called with any bag type as an argument.

The other files contain code provided for you from the book. Familiarize yourself with the methods provided and how they work.

Task 1 – Array Bag [3 points]

Program the missing methods from the bag interface for an **ArrayBag**, inside the file named **arrayBag.py.** The method **count** needs to be completed. Make a general **resize** method which copies over the values in **_items** into a new array resized by a **sizeFactor** parameter. Do not let the array size shrink to less than the default capacity.

Add the behaviors for **growing** and **shrinking** during **add** and **remove**: grow twice as large when the physical size is the same as the logical size, shrink to half as much when the logical size is one fourth the physical size. (Think: why not shrink when half full?) Test the class with the test function in **testBag.py.**

Task 2 – Linked Bag [3 points]

Program the missing methods from the bag interface for an **LinkedBag**, inside the file named **linkedBag.py**. The methods **count**, **add** and **clear** need to be programmed by you. Test the class with the test function in **testBag.py**.

Task 3 – Sorted Bags **[9 points]**

A sorted bag behaves just like a regular bag, but allows the user to visit its items in ascending order with a for loop. Therefore, the items added to this type of bag must have a natural ordering and recognize the comparison operators. Some examples of such items are strings and integers.

Create two new files, **`arraySortedBag.py`** and **`linkedSortedBag.py`**. Inside each, define a new class for both an **`ArraySortedBag`** and **`LinkedSortedBag`**. Most of the code for these two bags can be copied over wholesale from the unsorted bag variants, so it is a good place to start by copying over your working code from the previous two parts.

In order to make a bag sorted, the method which needs to change is the **`add`** method. Normal bags simply add the new item at the most convenient spot (at the end for Arrays, and at the front for Linked). Sorted containers must **`insert`** the item in the correct location. Change the **`add`** method in both sorted bags and test their behavior with **`testBag.py`**. (Think: What's the easiest way to insert **one** item in a **sorted** array? A sorted linked structure?)

Aside from presenting things in sorted order, the **`LinkedSortedBag`** cannot take much advantage of the sorted nature of the bag. The array implementation, however, can take the divide and conquer approach due to its random access ability. Add a binary search method for **`ArraySortedBag`** to utilize in **`__contains__`**. Change **`__eq__`** to use the linear **O(N)** approach for equality in both **Linked** and **Array** variants.

What To Turn In

Create a zip file named **Lab4_<your W&L ID>.zip**. Inside this zip archive should submit all the files or this lab including **`arraySortedBag.py`** and **`linkedSortedBag.py`**. Make sure your files have the correct names (including case), and extensions.