Richard Hayes Crowley

07/01/2021

CSC\_242\_Lab\_07

**SOURCE CODE:**

**ShoppingBag.py**

*from* arrayBag *import* ArrayBag

*from* linkedBag *import* LinkedBag

*from* simple\_term\_menu *import* TerminalMenu

groceriesDict = {"Fruits": 1.50, "Eggs": 2.50, "Cereal": 4, "Milk": 4.00,

"Bread": 3.95, "Meats": 8.25, "Cheeses": 6.75, "Vegetables": 3.25}

groceriesList = list(groceriesDict.keys())

groceriesInventory = [

f"{value[0]}: ${value[1]:.02f}" *for* value *in* groceriesDict.items()]

def main():

global groceriesDict

'''

Shopping Bag uses two different "bag" data structures (a linkedBag and a dynamic arrayBag)

The user can choose between using a linkedBag or an arrayBag

The array Bag is dynamically sized, e.g., a new array of a greater or lesser length is created whenever a certain threshold (or, "load factor") is reached. This is described in the ArrayBag class code.

In this program, the user can:

• Open a bag to be utilized

• Add contents to the bag

• Remove an item from the bag

• Check if an item is contained within the bag

• Clear their bag

• Check out (run a sum function to find price of all items in bag)

'''

print("Welcome to the Shopping bag app! Let's get some groceries!")

bag = None

entryPoint = TerminalMenu(["Array Bag", "Linked Bag", "Exit"],

*title*="What kind of shopping bag would you like to use?").show()

*if* entryPoint == 0:

print("You chose Array Bag")

bag = ArrayBag()

goShopping(bag)

*elif* entryPoint == 1:

print("You chose Linked Bag")

bag = LinkedBag()

goShopping(bag)

*else*:

print("Goodbye!")

exit()

def goShopping(*bag*):

global groceriesDict

global runningTotal

print(

f"Let's go shopping!\nHere are the groceries and their prices: {[i *for* i *in* groceriesInventory]}")

*while* True:

option = TerminalMenu(["View bag", "Add items to bag", "Remove items from bag",

"Check if item is in bag", "Empty Bag", "Checkout", "Exit"], *title*="What would you like to do?").show()

*if*(option == 0):

print([f"{i}" *for* i *in* *bag*])

*elif*(option == 1):

item\_to\_add = TerminalMenu(

groceriesList, *title*="What item would you like to add to your bag?").show()

*bag*.add(groceriesList[item\_to\_add])

print(*bag*)

*elif*(option == 2):

*if* *bag*.isEmpty():

print("Nothing to remove!")

*else*:

item\_to\_remove = TerminalMenu(

[item *for* item *in* *bag*], *title*="Which item would you like to remove?").show()

item\_to\_remove = [item *for* item *in* *bag*][item\_to\_remove]

*bag*.remove(item\_to\_remove)

print(*bag*)

*elif*(option == 3):

item\_to\_check = TerminalMenu(

groceriesList, *title*="What item would you like to check in your bag?").show()

checkItem = groceriesList[item\_to\_check] in *bag*

*if* checkItem:

print(f"{groceriesList[item\_to\_check]} is in your bag!")

*else*:

print(f"{groceriesList[item\_to\_check]} is NOT in your bag!")

*elif*(option == 4):

*bag*.clear()

print(f"Bag cleared!")

*elif*(option == 5):

print(f"Checking out! Items in bag: {[f'{i}' *for* i *in* *bag*]}")

runningTotal = sum([groceriesDict[i] *for* i *in* *bag*])

print(f"Your total is: ${runningTotal:0.2f}")

print("Goodbye!")

exit()

*else*:

print("Goodbye!")

exit()

*if* \_\_name\_\_ == "\_\_main\_\_":

main()

**arrayBag.py**

*from* arrays *import* Array

class ArrayBag(object):

''' An array bag is an array-based implementation of a bag collection'''

*# class variable*

DEFAULT\_CAPACITY = 10

def \_\_init\_\_(*self*, *sourceCollection*=None) -> None:

super().\_\_init\_\_()

*self*.items = Array(ArrayBag.DEFAULT\_CAPACITY)

*self*.size = 0

*if* *sourceCollection*:

*for* item *in* *sourceCollection*:

*self*.add(item)

*# Accessor methods*

def isEmpty(*self*):

*return* len(*self*) == 0

def \_\_len\_\_(*self*):

*return* *self*.size

def \_\_str\_\_(*self*):

*return* "{" + f"{', '.join(map(str, *self*))}" + "}"

def \_\_iter\_\_(*self*):

''' supports iteration over a view of self, importatnt for other methods to work!'''

cursor = 0

*while* cursor < len(*self*):

*# yield sends each item to the caller, the for loop*

*yield* *self*.items[cursor]

cursor += 1

def \_\_add\_\_(*self*, *other*):

result = ArrayBag(*self*)

*for* item *in* *other*:

result.add(item)

*return* result

def \_\_eq\_\_(*self*, *other*):

*if* *self* is *other*:

*return* True

*if* type(*self*) != type(*other*) or len(*self*) != len(*other*):

*return* False

*for* item *in* *self*:

*if* *self*.count(item) != *other*.count(item):

*return* False

*return* True

def count(*self*, *items*):

*return* 0

*# Mutator methods*

def clear(*self*):

*self*.size = 0

*self*.items = Array(ArrayBag.DEFAULT\_CAPACITY)

def add(*self*, *item*):

*# check array memory / load factor here, increase size if necessary*

*# increasing size of array*

*if* *self*.size == len(*self*.items):

*# create new array and copy data from old array*

*# double the size of the array instead of adding one new cell each time the array needs to be resized to ensure better performance*

temp = Array(len(*self*.items)\*2)

*for* i *in* range(*self*.size):

temp[i] = *self*.items[i]

*# reset old array variable to new array, old arrays memory is left out for the garbage collector*

*self*.items = temp

*self*.items[len(*self*)] = *item*

*self*.size += 1

def remove(*self*, *item*):

*if* not *item* in *self*:

*raise* KeyError(str(*item*) + " is not in bag")

targetIndex = 0

*for* targetItem *in* *self*:

*if* targetItem == *item*:

*break*

targetIndex += 1

*# shift items to the right of target index by one*

*for* i *in* range(targetIndex, len(*self*) - 1):

*self*.items[i] = *self*.items[i + 1]

*# decrement logical size*

*self*.size -= 1

*# check array memory / load factor here, decrease size if necessary*

*# performant choice is, if logical size is less than or equal to the 1/4 the length of b and the length of b is greater than 2x the default capacity*

*if* *self*.size <= len(*self*.items) // 4 and len(*self*.items) >= ArrayBag.DEFAULT\_CAPACITY \* 2:

temp = Array(len(*self*.items) // 2)

*for* i *in* range(*self*.size):

temp[i] = *self*.items[i]

*self* = temp

*# aBag = ArrayBag(["a", "b", "c", "d", "e", "f", "g", "h"])*

**Arrays.py (dependency of ArrayBag)**

'''

Array data structure

An array is like a list, but the client can use only [], len, iter and str

'''

class Array(object):

'''Represents an array'''

def \_\_init\_\_(*self*, *capacity*, *fillValue*=None) -> None:

''' capacity is the static size of the array'''

super().\_\_init\_\_()

*self*.items = list()

*for* \_ *in* range(*capacity*):

*self*.items.append(*fillValue*)

def \_\_len\_\_(*self*):

*return* len(*self*.items)

def \_\_str\_\_(*self*):

*return* str(*self*.items)

def \_\_iter\_\_(*self*):

*return* iter(*self*.items)

def \_\_getitem\_\_(*self*, *index*):

*return* *self*.items[*index*]

def \_\_setitem\_\_(*self*, *index*, *newItem*):

*self*.items[*index*] = *newItem*

**LinkedBag.py**

*from* node *import* Node

class LinkedBag(object):

'''A linkedBag is a linkedList implementation of a bag collection'''

*# constructor*

def \_\_init\_\_(*self*, *sourceCollection*=None) -> None:

super().\_\_init\_\_()

*self*.items = None

*self*.size = 0

*if* *sourceCollection*:

*for* item *in* *sourceCollection*:

*self*.add(item)

*# Accessor methods*

def isEmpty(*self*):

*return* len(*self*) == 0

def \_\_len\_\_(*self*):

*return* *self*.size

def \_\_str\_\_(*self*):

*return* "{" + f"{', '.join(map(str, *self*))}" + "}"

def \_\_eq\_\_(*self*, *other*):

*if* *self* is *other*:

*return* True

*if* type(*self*) != type(*other*) or len(*self*) != len(*other*):

*return* False

*for* item *in* *self*:

*if* *self*.count(item) != *other*.count(item):

*return* False

*return* True

def \_\_iter\_\_(*self*):

*# cursor in linked struct is a reference pointer to nodes in the linked structure*

*# cursor initially set to the external pointer, self.items, and stops the loop when it reaches the end (None)*

cursor = *self*.items

*while* cursor != None:

*# yeild to the caller*

*yield* cursor.data

cursor = cursor.next

def \_\_add\_\_(*self*, *other*):

result = LinkedBag(*self*)

*for* item *in* *other*:

result.add(item)

*return* result

def count(*self*, *items*):

*return* 0

*# Mutator methods*

def clear(*self*):

*self*.size = 0

*self*.items = None

def add(*self*, *item*):

*self*.items = Node(*item*, *self*.items)

*self*.size += 1

def remove(*self*, *item*):

*if* not *item* in *self*:

*raise* KeyError(str(*item*) + " not found")

probe = *self*.items

trailer = None

*for* targetItem *in* *self*:

*if* targetItem == *item*:

*break*

trailer = probe

probe = probe.next

*# unhook node to be deleted, either first one or one thereafter*

*if* probe == *self*.items:

*self*.items = probe.next

*else*:

trailer.next = probe.next

*self*.size -= 1

*# linkedBag = LinkedBag(["1", "2", "3"])*

**Node.py (dependency of linkedBag.py)**

class Node(object):

'''represents singly linked node'''

def \_\_init\_\_(*self*, *data*, *next*=None) -> None:

super().\_\_init\_\_()

*self*.data = *data*

*self*.next = *next*

**OUTPUT:  
  
Please see attached MP4 video for demo**