CS2006 Python Project 1 -
Classes and Iterators in Python

Matriculation Numbers: 160001362, 160014817, 160013384 (Group 14)

06/03/2018

Contents

1 Summary of Functionality

This practical specified the development of a mathematical system to explore discrete mathematical structures using the programming language Python.

The provided README file gives detailed instructions on what the solution can do and how to run each command.

The following functionality has been implemented:

1.1 Basic Specification:

All requirements from the basic specification have been implemented. They are as follows:

- An implementation of the twisted integers data structure which supports addition and multiplication given by the following rules:
  - a $\bigoplus$ b = (a + b) mod n
  - a $\bigotimes$ b = (a + b + a b) mod n
- Exceptions that are used to check that user input is valid.
- Unit tests to ensure the correctness of the solution.

1.2 Additional Requirements:

From the suggested additional requirements, all of the easy, medium and hard requirements have been implemented:

1.2.1 Easy

- **Easy Requirement 1 -** The function 'mulEqualToOne(n)' has been developed in the checker.py file which calculates for a given n all elements x ϵ Zn such that x $\bigotimes$ x = 1, where 1 ϵ Zn
- **Easy Requirement 2 -** For a given n functions (in checker.py) have been developed to check whether the following properties hold for all x, y, z ϵ Zn (each of which returns a boolean signifying whether the properties hold):
  - x $\bigoplus$ y = y $\bigoplus$ x - 'isCommutativeAdd(n)'
  - x $\bigotimes$ y = y $\bigotimes$ x - 'isCommutativeMul(n)'
  - (x $\bigoplus$ y) $\bigoplus$ z = x $\bigoplus$ (y $\bigoplus$ z) - 'isAssociativeAdd(n)'
  - (x $\bigotimes$ y) $\bigotimes$ z = x $\bigotimes$ (y $\bigotimes$ z) - 'isAssociativeMul(n)'
  - (x $\bigoplus$ y) $\bigotimes$ z = (x $\bigotimes$ y) $\bigoplus$ (y $\bigotimes$ z) - 'isDistributive(n)'

- **Easy Requirement 3 -** The file twisted_integers.py contains a class TwistedIntegers which implements a data structure representing Zn with respect to the operations a $\bigoplus$ b = (a $\bigoplus$ b) mod n and a $\bigotimes$ b = (a $\bigotimes$ b) mod n and contains the methods __init__, __str__ and size where size returns the number of elements in Zn.

1.2.2 Medium

- **Medium Requirement 1 -** In the twisted_integers.py file (along with the aforementioned TwistedIntegers class) the IteratorOfTwistedIntegers class has been implemented to iterate over instances of TwistedIntegers and contains an __init__ function (to initialise it with a given instance of TwistedIntegers), a hasNext() boolean function (to indicate if the iterator has a next value) and a next() function (to return the next value in the iterator).

- **Medium Requirement 2 -** The function 'findValAdd(n)' in the twisted_integers.py file finds for a given n all elements τ of Zn such that $\tau \bigoplus$ x = x for all x ϵ Zn.

- **Medium Requirement 3 -** The function 'findValMul(n)' in the twisted_integers.py file finds for a given n all elements ε of Zn such that $\varepsilon \bigotimes$ x = x for all x ϵ Zn.

1.2.3 Hard

- **Hard Requirement 1 -** In the twisted_int_matrix.py file the class TwistedIntMatrix has been implemented which represents a matrix of TwistedInt objects. The __init__ function of the class takes an x and y dimension as well as a list of TwistedInt objects and creates a matrix containing the values in the list. The __mul__ function implements matrix multiplication (between two given matrices) using the row by column rule and the $\bigoplus$ and $\bigotimes$ operators. This is achieved using the calcDotProduct function which calculates the dot product between a row and a column of the two matrices (which itself uses the getCol function to access a column of the second matrix).

  For the same requirement, an algorithm has been implemented using the function getPossibleMatrices(matrices) which takes a list of M matrices and calculates all of the possible unique matrices that may be obtained by multiplying these matrices, i.e. the set {g1 g2 . . . gk | k ϵ N, gi ϵ M }.

- **Hard Requirement 2 -** The level of testing and documentation has been enhanced by including docstrings for every function and class, as well as example doctests within these docstrings that can be extracted and run to verify the correctness of the functions which they document.

1.3 Further Features:

The follow additional features have also been implemented:

- An iterator class, IteratorOfTwistedIntMatrix, has been implemented in twisted_int_matrix.py file which iterates over an instance of TwistedIntMatrix from top left to bottom right across the TwistedInt elements. As with the IteratorOfTwistedIntegers iterator, this iterator has a hasNext(), next() and __init__ functions.
-

2 Design and Implementation

- **TwistedInt Data Structure** - The TwistedInt data structure is implemented in twisted_int.py. It is initialised by passing an integer value 'val' and mod value 'n' into __init__. The constructor of the TwistedInt class will raise a TypeError exception if 'val' or 'n' are not integers and will raise the custom InvalidModError or InvalidValError if the value of n passed in is less than 0 or if val is not in Zn, 0 to (n - 1). These exceptions can be handled by any implementing code to validate the inputs of the user.

  The printing of TwistedInt objects has be modified in the function __str__ to printed TwistedInt instances in the format "<object:n>".

  TwistedInt multiplication is implemented by the __mul__ function which takes two TwistedInts and compares their mod n values - raising a custom MismatchedModError exception if they are not equal - and otherwise performs the operation (a + b + a . b) mod n and assigns the result to the value of a new TwistedInt (as well as the same mod n value) and returns this object.

  Similarly the __add__ function performs the operation (a + b) mod n with two given TwistedInt instances and returns a TwistedInt instance with the result as the integer value.