Homework Assignment 5

The assignment is due by 11:55PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student’s own work. Assistance should be sought or accepted only from the course instructor. Any violation of this rule will be dealt with harshly.

This assignment requires you to implement a class for polynomial equations and a class for quadratic equations as a subclass of polynomials. As usual, you are graded not only on the correctness of the code, but also on clarity and readability. I will deduct points for not following the guidelines for your class design, poor indentation, poor choice of object names, and lack of documentation.

Please read the submission guidelines at the end of this document before you start your work.

Problem 1 [40 points] Bags. You are given a class called Bag in a module called bag.py, which is available on the Sakai page for this homework assignment. A Bag object is simply a container that stores an unordered collection of items. An item may occur several times in a bag. Read the implementation and the documentation carefully to understand all the methods for this class.

In this problem, you are asked to write some functions that have Bag objects as parameters or return values. The point of this exercise is to use a class that is given to you. Insert these functions in the bagfunctions.py file. Make sure you import the bag module when implementing bagfunctions.py. In each of the following functions, you are required to use the Bag class methods to implement the function. You may not manipulate Bag instance attributes directly (in any case, you will not be able to, as the attribute is private).

1. Implement a function called remove_item with two parameters: a Bag object B and an item item. The function removes all occurrences of item from B. The actual Bag parameter will be modified by this function.

2. Implement a function called remove_repeats with a single parameter, which is a Bag object B. The function removes all repeating items from B and retains exactly one copy of each item. The actual parameter will be modified by this function.

3. Implement a function called mode with a single parameter, which is a Bag object B. The function returns a list containing the most frequently occurring item(s) in the bag. The actual bag parameter should not be modified by this function. Note: The order of items in the returned list is not relevant.

4. Implement a function called union with two parameters, a Bag object B1 and another Bag object B2. The function returns a Bag object containing the union of B1 and B2. The union of two bags B1 and B2 is a bag containing all the items from B1 and from B2.
Note that since a bag may have repeating items, the count of an item in the union is equal to the sum of the counts of that item in each of B1 and B2. The actual parameters should not be modified by this function.

5. Implement a function called intersection with two parameters, a Bag object B1 and another Bag object B2. The function returns a Bag object containing the intersection of B1 and B2. The intersection of two bags B1 and B2 is bag containing items that are common to B1 and B2. Note that the count of an item in the intersection is equal to the minimum of the counts of that item in each of B1 and B2. The actual parameters should not be modified by this function.

For example, let bone be a Bag object containing the items 1, 1, 'hello', 'hello', 2, 2, 'there', 3 and let btwo be a Bag object containing the items 2, 2, 2, 'hello', 'there', 'there', 'there', 3. Then,

- After the function call remove_item(bone, 'hello'), bone will contain the items 1, 1, 2, 2, 'there', 3.
- Assuming the original contents of bone, after the function call remove_repeats(bone), bone will contain the items 1, 'hello', 2, 'there', 3.
- Assuming the original contents of bone, the function call mode(bone) returns the list [1, 2, 'hello'] and mode(btwo) returns the list [2].
- Assuming the original contents of bone, the function call union(bone, btwo) returns a Bag whose contents are 1, 1, 'hello', 'hello', 'hello', 'there', 'there', 'there', 2, 2, 2, 2, 3, 3, and the function call intersection(bone, btwo) returns a Bag whose contents are 'hello', 2, 'there', 3.

Problem 2 [60 points] Polynomial objects. In this problem, you are asked to implement a class for polynomials. Please read the description carefully, as it will provide details about class methods and instance attributes.

A polynomial is an expression of the form

\[ c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_n x^n \]

where \( c_0, c_1, \ldots, c_n \) are real numbers. Each of \( c_i x^i \) is called a term of the polynomial, and \( c_i \) is called the coefficient of the term with exponent \( i \). Note that \( c_0 \) is simply the coefficient of the term with exponent 0. The maximum exponent with a non-zero coefficient is called the degree of the polynomial. For example, \( 6x^{14} + 9x^{11} - 12x^3 + 42 \) is a polynomial of degree 14. The polynomial \(-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9\) has degree 6. The following are standard operations on polynomials:

Scaling a polynomial: Given a polynomial \( p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0 \) and a real value \( s \), scaling \( p(x) \) by \( s \) gives the polynomial \( s \cdot p(x) \) obtained by scaling the coefficient of every term in \( p(x) \) by the factor \( s \). For example, if \( p(x) = 6x^{14} + 9x^{11} - 12x^3 + 42 \), then \( 2p(x) = 12x^{14} + 18x^{11} - 24x^3 + 84 \).

Sum of two polynomials: The sum of two polynomials \( p_1(x) \) and \( p_2(x) \), denoted by \( p_1(x) + p_2(x) \), is the polynomial obtained by adding the terms of \( p_1(x) \) and \( p_2(x) \). For example, if \( p_1(x) = 6x^5 - 4x^3 + 10x^2 + 4 \) and \( p_2(x) = 3x^9 - 5x^7 - 3x^2 + 8x \), then \( p_1(x) + p_2(x) = 3x^9 - 5x^7 + 6x^5 - 4x^3 + 7x^2 + 8x + 4 \).
Difference of two polynomials: The difference of two polynomials \( p_1(x) \) and \( p_2(x) \) is the polynomial obtained by subtracting one from the other. Therefore, \( p_1(x) - p_2(x) \) is the polynomial obtained by subtracting the terms of \( p_2(x) \) from \( p_1(x) \). For example, if \( p_1(x) = 6x^5 - 4x^3 + 10x^2 + 4 \) and \( p_2(x) = 3x^7 - 3x^2 + 8x \), then \( p_1(x) - p_2(x) = -3x^9 + 5x^7 + 6x^5 - 4x^3 + 13x^2 - 8x + 4 \).

Product of two polynomials: The product of two polynomials \( p_1(x) \) and \( p_2(x) \), denoted by \( p_1(x) \cdot p_2(x) \), is the polynomial obtained by the pair-wise multiplication of terms in \( p_1(x) \) and \( p_2(x) \). For example, if \( p_1(x) = x^4 + 4x^3 + 4x^2 \) and \( p_2(x) = 2x - 1 \), then \( p_1(x) \cdot p_2(x) = 2x^5 + 7x^4 + 4x^3 - 4x^2 \). This is obtained as follows: \((x^4 + 4x^3 + 4x^2)(2x) + (x^4 + 4x^3 + 4x^2)(-1) = 2x^5 + 8x^4 + 8x^3 - x^4 - 4x^3 - 4x^2 = 2x^5 + 7x^4 + 4x^3 - 4x^2\).

In this program, you are asked to implement a `Polynomial` class. The main instance attribute for an object of this class is a list that stores the coefficients of the polynomial. A polynomial of degree \( d \) is stored in a list of length \( d + 1 \). For each term \( c_ix^i \), coefficient \( c_i \) is stored at index \( i \) in the list. Note that if a term of exponent \( i \) does not exist in the polynomial, its coefficient is 0 and hence the element at index \( i \) in that case will be 0.

Implement the following methods for the `Polynomial` class. **Your implementation should appear in a module called** `poly.py`. A test module called `test_poly.py` is also provided, which imports your `poly.py` module. When you are ready, test your implementation of the `Polynomial` class by typing `python3 test_poly.py`.

1. `__init__`: The parameters to the constructor are a number of term pairs, where each term pair is a 2-tuple of the form (coefficient, exponent). Note that the number of such pairs is unspecified, since a polynomial may have any number of terms. Hence, the `def` statement for the constructor will be as follows:

   ```python
   def __init__(self, *termpairs):
   ```

   For example, to create a `Polynomial` instance for the polynomial \( 6x^{14} + 9x^{11} - 12x^3 + 42 \), we would do the following:

   ```python
   P = Polynomial((6, 14), (9, 11), (-12, 3), (42, 0))
   ```

   Note that the term pairs can be specified in any order. Hence, `Polynomial((6, 14), (9, 11), (-12, 3), (42, 0))` and `Polynomial((42, 0), (6, 14), (-12, 3), (9, 11))` are the same polynomial. The constructor should create an instance attribute called `coeffs`, which is a list that stores coefficients of terms in the polynomial as described above.

2. `__str__`: This method returns a printable version of a polynomial, which is a “nice” string representation of the polynomial. This means that only terms with non-zero coefficients must appear in the string, and the terms must appear from largest exponent to smallest. Furthermore, if the coefficient of a term is negative, the - (minus sign) must be embedded in the string.

3. `__repr__`: This method returns a meaningful string representation of the polynomial. For example, this could be the same string returned by the `__str__` method.

4. `degree`: This method returns the degree of the polynomial.

5. `evaluate`: This method returns the result of evaluating the polynomial at a real value \( x = X \). For example, if \( P \) is the polynomial \(-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9\), then
P.evaluate(2) should return -911 (which we obtain by plugging in 2 as the value of x). This method has two parameters: self and X (the real value at which the polynomial is to be evaluated).

6. addterm: This method adds a term with coefficient coeff and exponent exp to the polynomial. If a term with this exponent already exists, the new term is simply added on to it. For example, if P is the polynomial \(-4x^3 + 13x^2 + 4\), then P.addterm(6, 5) adds the term \(6x^5\) to P, so that P is now \(6x^5 - 4x^3 + 13x^2 + 4\). Subsequently, P.addterm(-3, 2) causes P to become \(6x^5 - 4x^3 + 10x^2 + 4\). \textbf{Note} the order of the parameters: we specify the coefficient first and then the exponent.

7. removeterm: This method removes the term with exponent exp from the polynomial. If such a term does not exist, the polynomial remains unchanged. For example, if P is \(6x^5 - 4x^3 + 10x^2 + 4\), P.removeterm(3) causes P to become \(6x^5 + 10x^2 + 4\).

8. scale: This method returns the result of scaling the polynomial by a factor s. \textbf{Note that a polynomial object is returned.} Moreover, the activating polynomial itself should remain unchanged. For example, if P is \(6x^5 + 10x^2 + 4\), then P.scale(2.5) returns the polynomial \(15x^5 + 25x^2 + 10\). P itself is unchanged.

9. __add__: This method overloads the + operator and returns the sum of two polynomials, which is also a polynomial (as explained above). Note that __add__ has two parameters, self and other, where other is also a polynomial. Implement this function by \textit{reusing} code already written, i.e., by calling methods already implemented. \textbf{Important:} Neither self nor other should be modified by this method.

10. __sub__: This method overloads the - operator and returns the difference of two polynomials, which is also a polynomial (as explained above). Comments on the __add__ method apply here as well.

11. __mul__: This method overloads the * operator and returns the product of two polynomials, which is also a polynomial (as explained above). Again, neither self nor other should be modified by this method.

12. __getitem__: This method overloads the [ ] (indexing) operator. If idx is the parameter to the index operator, __getitem__ must return the coefficient of the term with exponent idx. For example, if P is \(6x^5 + 10x^2 + 4\), P[5] should return 6, P[0] should return 4, and P[3] should return 0.

13. __setitem__: This method implements the index assignment operation. If idx and value are the two parameters to this method, then value should become the coefficient of the term with exponent idx. If a term with this exponent already exists, its coefficient is changed to value. If a term with this exponent does not exist, it should be created.

In addition to the above Polynomial class methods, you must implement the following two functions (these are regular functions, not methods):

- A function called read_polynomial with a single parameter, a string polyfilename, which is the name of a file that contains the terms of a polynomial in the following format: The file has as many lines as there are terms, and each line of the file contains a pair of numbers, the first being the coefficient of the term and the second being the exponent. This function opens the file for reading, creates a Polynomial instance from the data in the file, and then returns this instance.
Two files called poly1.txt and poly2.txt containing polynomial data have been created for you. They are available on the Sakai site for this homework assignment. You may use these to test your implementation. Try creating some of your own as well for further testing.

- A function called `arith_ops_polys` with two parameters `P` and `Q`, both of which are polynomial instances. This function simply prints (on separate lines) the degree of each polynomial, and the polynomials resulting from adding, subtracting, and multiplying the polynomials `P` and `Q`.

**SUBMISSION GUIDELINES**

Implement the first problem in a module called `bagfunctions.py` and the second one in a module called `poly.py`. Your name and RUID should appear as a comment at the very top of each module. Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Sakai as follows:

Submit your homework files via Sakai as follows:

1. Use your web browser to go to the website `https://sakai.rutgers.edu`.
2. Log in by using your Rutgers login id and password, and click on the `OBJECT-ORIENTED PROG S19` tab.
3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #5' to find the homework file (`hw5.pdf`), and the modules `bag.py` (for Problem 1) and `test_poly.py` (for Problem 2).
4. Use this same link to upload your homework files (`bagfunctions.py` and `poly.py`) when you are ready to submit.

You must submit your assignment at or before 11:55PM on April 19, 2019.