Problem-Based Intro. to Computer Science (2010-1)
Polynomials Week 7 - Homework

1 Problem

Introduction

Polynomials are mathematical expressions of the form $a_n x^n + \cdots + a_1 x + a_0$, where x is a variable, and each a_i is a number. Here are some examples: $x^2 + 3x + 1$, $4x^2 + 1$, $3x$, 5, and 0. Polynomials can also be defined recursively as follows.

A *polynomial* is one of the following.

- 0
- $p \times x + c$, where p is a polynomial, x is the variable, and c is a number.

Like numbers, we can add and multiply polynomials. Here are some examples.

$$(x + 1) + (x + 2) = 2x + 3$$
$$(x + 1) \times (x + 2) = x^2 + 3x + 2$$

We can represent polynomials in Python using lists. For example, we can represent $3x + 2$ as `[2, 3]`. Notice the numbers got reversed. In general, polynomials are represented as follows.

- The Python empty list `[]` represents the zero polynomial.
- The Python list `[`a_0`,` a_1`,` $\ldots a_n$`]` represents the polynomial $a_n x^n + \cdots + a_1 x + a_0$.

Unfortunately, there are several representations that refer to the same polynomial. For example, both `[2, 3]` and `[2, 3, 0]` represent the polynomial $3x+2$, since $0x^2+3x+2 = 3x+2$. Therefore, it is useful to speak of a *standard* representation, which does not allow trailing zeros.

Task and Examples

You will write Python code to add and multiply polynomial representations.

```
>>> P = [2, 3, 0]
>>> standardize(P)
>>> P
[2, 3]
>>> isZeroPoly([])
True
>>> isZeroPoly([2, 1])
False
>>> addPoly([1, 1], [2, 1])
[3, 2]
>>> addPoly([3, 2], [6, 5, 4])
[9, 7, 4]
>>> scalePoly(3, [1, 2, 3])
[3, 6, 9]
```

```
>>> constCoef([7, 2, 5])
7
>>> shiftLeft([7, 2, 5])
[0, 7, 2, 5]
>>> shiftRight([7, 2, 5])
[2, 5]
>>> mulPoly([1, 1], [2, 1])
[2, 3, 1]
```

Details

You will define the following functions. You should assume that polynomial representation inputs are in standard form; polynomial representation return values should always be in standard form. Note: Although there are many required functions, their definitions are often quite short.

- `standardize`, which is a function that takes a polynomial representation (i.e. a list of numbers), and destructively removes all trailing zeros. For example, when adding $[1, 2, 5]$ and $[1, 1, -5]$, the pairwise result is $[2, 3, 0]$. The `standardize` function should change the list so that the trailing zero is deleted. See the example in the Examples section.
- `isZeroPoly`, which is a function that takes a polynomial representation (i.e. a list of numbers), and returns a Boolean that indicates whether or not the polynomial representation represents zero.
- `addPoly`, which is a function that takes two polynomial representations (i.e. lists of numbers), and returns a polynomial representation that represents their sum. For the most part, addition involves adding together numbers that are in the same positions in the corresponding lists. One complication is that you will have to allow for lists that are different lengths. Another complication is that numbers can cancel out, leading to non-standard results; use `standardize` to deal with that one.
- `scalePoly`, which is a function that takes a number, c, and a polynomial representation, ℓ, (i.e. a list of numbers), and returns a polynomial representation such that all the numbers in ℓ have been scaled by c. Be careful; there is one case when merely multiplying can result in a non-standard representation.
- `constCoef`, which is a function that takes a polynomial representation (i.e. a list of numbers), and returns a number. It implements the operator $\mathsf{constCoef}$ which is defined by $\mathsf{constCoef}(a_n x^n + \cdots + a_1 x + a_0) = a_0$. Be careful to handle all polynomials.
- `shiftLeft`, which is a function that takes a polynomial representation (i.e. a list of numbers), and returns a polynomial representation. It implements the operator $\mathsf{shiftLeft}$ which is defined by $\mathsf{shiftLeft}(a_n x^n + \cdots + a_1 x + a_0) = a_n x^{n+1} + \cdots + a_1 x^2 + a_0 x$. Be careful; there is one case that can result in a non-standard representation.
- `shiftRight`, which is a function that takes a polynomial representation (i.e. a list of numbers), and returns a polynomial representation. It implements the operator $\mathsf{shiftRight}$ which is defined by $\mathsf{shiftRight}(a_n x^n + \cdots + a_1 x + a_0) = a_n x^{n-1} + \cdots + a_1$. Be careful to handle all polynomials.
- `mulPoly`, which is a function that takes two polynomial representations (i.e. lists of numbers), and returns a polynomial representation that represents their product. A recursive algorithm

to compute the product can be derived as follows. Let $p_1 = a_n x^n + \cdots + a_1 x + a_0$.

$$\begin{aligned} p_1 \times p_2 &= (a_n x^n + \cdots + a_1 x + a_0) \times p_2 \\ &= (a_n x^n + \cdots + a_1 x) \times p_2 + a_0 \times p_2 \\ &= (a_n x^{n-1} + \cdots + a_1) \times (p_2 \times x) + a_0 \times p_2 \\ &= \mathsf{shiftRight}(p_1) \times \mathsf{shiftLeft}(p_2) + \mathsf{constCoef}(p_1) \times p_2 \end{aligned}$$

Repeatedly shifting right will eventually result in zero, and the answer when $p_1 \equiv 0$ is straight forward.

Submission

You need to submit a file `poly.py` containing the code detailed above.