Polynomial Arithmetic Console Application: Project Report

Introduction

Polynomial operations can be tedious and error-prone if done by hand. The goal of this project was to create an interactive console application that allows users to perform polynomial arithmetic operations such as addition, subtraction, multiplication, and division, as well as evaluate polynomials at given values. The application prioritizes clarity, guided user input, and robust mathematical computation using object-oriented programming and custom data structures.

Data Structures Used

The application leverages three main data structures: dictionaries, a stack, and a queue.

1. **Dictionaries** are used to store polynomial terms. Each polynomial is represented as a dictionary where the keys are exponents and values are coefficients. This provides O(1) average-time complexity for lookups, insertions, and updates, making polynomial arithmetic efficient.

2. **Stack** is implemented, using a Python list, to maintain a history of user-entered polynomials and calculation results. This is particularly helpful for reuse and backtracking without requiring complex undo-redo logic. The stack supports basic push, pop, and peek operations and has a fixed maximum size to prevent memory overload. The time complexity is O(1) for push, pop, and peek.

3. **Queue** is used during polynomial evaluation to store a series of x-values entered by the user. This First-In-First-Out (FIFO) structure, implemented using a Python list, ensures that the values are processed in the order they were received. The time complexity is O(1) for enqueue, dequeue, and peek.

Unused Data Structures

Several data structures were deliberately excluded from this project, such as:

- **Binary Search Trees (BSTs):** Although a BST could offer sorted term insertion, its overhead for rebalancing and traversal complexity was unnecessary for our relatively small datasets.

- **Linked Lists:** These are traditionally helpful for representing sparse polynomials. However, dictionaries in Python already provide sparse behavior with direct indexing and are easier to use, so linked list was not selected.

Challenges

Implementing polynomial division was complex due to the iterative nature of long division. Special care had to be taken to ensure terms were subtracted correctly. Another challenge was handling polynomial representations conforming to the standard form, especially the various polynomial identities. Ensuring correct representation of terms like 1x^1 as x or 0x^0 as 0 using regex can be a bit tricky. It took a lot of tests, especially edge cases, to get various pattern matchings work correctly.

Impact Statement

This project could serve as a valuable learning aid for students learning to work with polynomials. It provides a hands-on way to visualize and manipulate polynomials. However, expanding and integrating into a graphical interface will significantly improve its user interface, making entering polynomials much easier.

Conclusions

This project demonstrates the effective use of object-oriented design and several data structures to serve as a practical learning tool. While limited in scope, the system provides a solid foundation for more advanced symbolic algebra systems. The challenges faced underscored the importance of robust input validation and thoughtful user interface design, even in text-based programs. With further development, this project has the potential to evolve into a full-fledged algebraic tool for both educational and analytical use.