Polynomials Operations using LinkedList

Name : Sai Maruthi Lakshman Kumar Sreepathi

Section : K22UN

# Introduction:

The problem involves adding two polynomials represented as linked lists.

The algorithm aims to efficiently add polynomials of potentially different degrees.

Algorithm Overview:

The algorithm handles cases where the powers of terms in the input polynomials may differ or be the same.

Detailed Algorithm:

When adding two polynomials, we iterate through their linked list representations.

Three cases are considered:

If the power of the term in p1 is greater than the power in p2, we create a new node in the result with p1's power and coefficient and move p1 to the next node.

If the power of the term in p2 is greater than the power in p1, we create a new node in the result with p2's power and coefficient and move p2 to the next node.

If both p1 and p2 have the same power, we add their coefficients. If the result is not zero, we create a new node with the same power and the new coefficient. We then move both p1 and p2 to the next nodes.

We continue this process until we finish calculations on all nodes from both polynomials.

# Node Structure:

Each linked list node represents a term in a polynomial.

The node stores the coefficient and exponent.

# Append Function:

The append function creates a new linked list node based on input power and coefficient.

It appends the new node to the tail of the list and returns the new tail node.

Time Complexity Analysis:

The algorithm traverses each linked list node only once.

The overall running time is O(n + m), where n and m are the numbers of terms in the two input polynomials.

# Implementation Details:

The Java implementation utilizes the Polynomial and Polynomial Term classes.

These classes work together to perform polynomial addition using linked lists.

while pl ‡ null and p2 ‡ null do if pl.power > p2.power then

tail = append (tail, pl.power, pl.coefficient);

p1 = pl.next;

end

else if p2. power > pl.power then

tail = append (tail, p2.power, p2.coefficient);

p2 = p2.next;

end else

coefficient = pl.coefficient + p2.coefficient;

if coefficient + 0 then

tail = append (tail, pl.power, coefficient);

end

pl = pl.next;

p2 = p2.next;

end

# Multiplication Process:

To multiply two polynomials, we iterate through each term of one polynomial and multiply it by each term of the other polynomial.This results in a new linked list with n \* m terms, where n and m are the numbers of terms in the input polynomials.

For example, multiplying (2 - 4x + 5x^2) and (1 + 2x - 3x^3) produces a linked list of terms representing the products of all possible term pairs.

# Sorting Linked List:

The linked list generated in the previous step is not sorted by the powers of the terms.To generate the final result, the linked list is sorted using a merge-sort algorithm based on each term's power.Sorting ensures that terms with the same power are grouped together.

# Merging Like Terms:

After sorting, like term nodes (terms with the same power) are grouped together.A two-pointer approach is used to efficiently merge the like terms.One pointer represents the start of a like term group, and the other traverses the like terms within the same group.When a like term is found, its coefficient is added to the start-like term node.

This process continues until all like terms are merged. Or just use a basic approach to solve this using two while loops as shown in below

# Algorithm Analysis:

The multiplication process takes O(nm) time, where n and m are the numbers of terms in the input polynomials.

Sorting the linked list takes O(nm \* log(nm)) time due to merge-sort.

Merging like terms using a two-pointer approach also takes O(nm) time.

Overall, the algorithm's time complexity is O(nm \* log(nm)).

This algorithm efficiently multiplies two polynomials represented as linked lists, sorting the result and merging like terms to obtain the final polynomial product.

while (ptr1 != null) {

while (ptr2 != null) {

int coeff, power;

coeff = ptr1.coeff \* ptr2.coeff;

power = ptr1.power + ptr2.power;

poly3 = addnode(poly3, coeff, power);

ptr2 = ptr2.next;

}

ptr2 = poly2;

ptr1 = ptr1.next;

The time complexity of polynomial multiplication using linked list is O(n\*m), where n is the total number of nodes in the first polynomial and m is the number of nodes in the second polynomial.

The space complexity of polynomial multiplication using linked list is O(n+m), we need to store all the multiplied values in the node.

# Subtraction Algorithm for Polynomials using Linked Lists:

## Initialization:

Create an empty linked list to represent the result polynomial.

## Iterating through Polynomial Terms:

Start by iterating through each term of the first polynomial (p1).

For each term in p1, add a corresponding term to the result polynomial with the same coefficient and exponent. This effectively copies terms from p1 to the result polynomial.

# Subtracting Terms from the Second Polynomial:

Next, iterate through each term of the second polynomial (p2).

For each term in p2, subtract its coefficient from the coefficient of the corresponding term in the result polynomial. This step is performed by adding the negation of p2's coefficient to the result polynomial's coefficient.If a term with the same exponent does not exist in the result polynomial, a new term with the negation of p2's coefficient is added to represent the subtraction.

# Final Result:

The result polynomial obtained after these steps represents the subtraction of p1 and p2.

Example:

Suppose we want to subtract (2x^2 - 3x + 4) from (5x^2 + 2x - 1). The algorithm proceeds as follows:

Initialize an empty result polynomial.

Start with (5x^2 + 2x - 1) (representing p1).

Iterate through p1 and add each term to the result polynomial: 5x^2 + 2x - 1.

Iterate through p2 (terms of (2x^2 - 3x + 4) representing p2).

Subtract 2x^2 from 5x^2, resulting in 3x^2.

Subtract -3x from 2x, resulting in 5x.

Subtract 4 from -1, resulting in -5.

The final result is 3x^2 + 5x - 5.

This algorithm effectively performs polynomial subtraction using linked lists and ensures that terms are combined correctly, considering their exponents and coefficients.