1634. Add Two Polynomials Represented as Linked Lists Medium Linked List Math **Two Pointers**

Problem Description

points about the linked list structure and the problem requirements: Each node contains a coefficient (an integer representing the multiplier) and a power (an integer representing the exponent).

The problem defines a special type of linked list where each node represents a term in a polynomial expression. Here are the key

Leetcode Link

- The linked list represents a polynomial in which the terms are in strictly descending order by their power. Terms with a coefficient of 0 are not included in the list. • The goal is to add two such polynomial linked lists and return the resulting polynomial linked list that maintains the same
- properties.

these polynomials to get a new linked list representing the polynomial $8x^3 + 4x + 1$.

For example, two polynomials $5x^3 + 4x - 7$ and $3x^3 - 2x + 8$ are represented by their respective linked lists. The task is to sum

To solve the addition of two polynomials represented by linked lists, we follow these steps: 1. Create a dummy head for the result polynomial linked list, which helps in easily returning the result.

input polynomials.

Solution Approach

Intuition

3. If one polynomial has a higher power term than the other, we add that term to the result list and move to the next term in that polynomial. 4. If both have terms with the same power, we add the coefficients. If the sum is not zero, we create a new node with this sum and

the corresponding power, then add it to the result list. 5. Continue this process until we reach the end of one or both polynomials.

2. Traverse both linked lists simultaneously while comparing the powers of the current nodes.

6. If there are remaining terms in one of the polynomials, append them to the result list.

To implement the solution for adding two polynomial linked lists, here's the step-by-step approach used:

7. Finally, return the next node of the dummy head as it points to the actual head of the result linked list.

By ensuring terms are added in descending order of power, we construct a valid polynomial linked list representing the sum of the

PolyNode which acts as a placeholder for the head of the new list, and a curr pointer which is used to add new nodes to the list. 1 dummy = curr = PolyNode()

2. Traversing Both Lists: We use a while loop to iterate through both poly1 and poly2 as long as there are nodes in both lists.

1. Initial Setup: We use a dummy node approach to simplify the process of building a new linked list. We start with a dummy

7 else:

1 curr = curr.next

10

11

1 while poly1 and poly2:

curr.next = poly2

poly2 = poly2.next

poly1 = poly1.next

poly2 = poly2.next

1 curr.next = poly1 or poly2

• poly1 represents 5x^2 + 3x + 1

2. We begin to iterate through both lists:

 \circ Add coefficients (3 + 1) = 4.

No more comparison is needed.

 \circ We add the coefficients (5 + 2) = 7.

poly2 represents 2x^2 + x

if c := poly1.coefficient + poly2.coefficient:

We need to update curr to the next node in the list after adding a node:

curr.next = PolyNode(c, poly1.power)

length. We append the remaining nodes to curr.

represents the head of the resultant polynomial linked list.

We will walk through a small example to illustrate the solution approach.

Suppose we have two polynomials represented by linked lists:

Represented as linked lists, they would look something like this:

poly1: $5 \rightarrow 3 \rightarrow 1$ (each node has terms decreasing in power from left to right) poly2: $2 \rightarrow 1$

Comparing the powers, 2 and 2 have the same power for the first nodes.

Create a new node with coefficient 7 and power 2 and link it to curr.

Move both poly1 and poly2 to the next nodes (3 and 1 respectively).

4. Now, poly2 has been completely traversed, but poly1 still has one term left:

5. The final polynomial linked list, represented by dummy next, is $7x^2 + 4x + 1$.

The constructed linked list will have the following structure, representing the sum of the input polynomials: $7 \rightarrow 4 \rightarrow 1$

3. Power Comparison and Node Addition: Inside the loop, we compare the powers of the current nodes from poly1 and poly2. • If poly1 has a higher power term, we link the current term from poly1 to curr and advance the poly1 pointer.

• If poly2 has a higher power term, we do the equivalent by linking poly2 to curr and advancing poly2.

is not zero. We then advance both pointers. 1 if poly1.power > poly2.power: curr.next = poly1 poly1 = poly1.next 4 elif poly1.power < poly2.power:</pre>

• If both terms have the same power, we sum their coefficients and only create and link a new node if the resultant coefficient

since each list is only traversed once and nodes are added to the result list in a single pass. Example Walkthrough

The overall complexity of the algorithm is linear O(n + m), where n and m are the number of nodes in poly1 and poly2 respectively

4. Handling Remaining Terms: After the loop, there might be remaining nodes in either poly1 or poly2 if they were not of equal

5. Returning the Result: Since dummy is a placeholder with an unused 0 coefficient and 0 power, we return dummy next, which

Steps: 1. We initiate a dummy node, which will help us in easily building and returning the result.

poly1 has power 1 and poly2 also has power 1.

class PolyNode:

class Solution:

10

11

12

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

45

3. The next comparison:

 Create a new node with coefficient 4 and power 1 and link it to curr. Move both pointers forward.

Append the last node of poly1 (which is 1) to curr.

def __init__(self, coefficient=0, power=0, next_node=None):

Python Solution # Definition for polynomial singly-linked list.

self.coefficient = coefficient

dummy = current = PolyNode()

if poly1.power > poly2.power:

current.next = poly1

current.next = poly2

poly2 = poly2.next

poly1 = poly1.next

poly2 = poly2.next

current = current.next

* Definition for a polynomial singly-linked list node.

PolyNode(int x, int y) { this.coefficient = x; this.power = y; }

* @param poly1 First polynomial represented as a linked list.

* @param poly2 Second polynomial represented as a linked list.

* @return The result of the polynomial addition, also as a linked list.

* Adds two polynomials represented as linked lists.

// dummy node to simplify list operations

// Iterate over both polynomial linked lists

if (poly1->power > poly2->power) {

} else if (poly1->power < poly2->power) {

current->next = poly1;

current->next = poly2;

if (sumCoefficient != 0) {

current = current->next;

current->next = (poly1 != nullptr) ? poly1 : poly2;

// Move forward in both lists

poly2 = poly2->next;

poly1 = poly1->next;

poly2 = poly2->next;

current = current->next;

PolyNode* headOfSum = dummyHead->next;

* @param poly1 The first polynomial singly-linked list.

* @param poly2 The second polynomial singly-linked list.

if (current->next) {

return headOfSum;

// Define a class for a polynomial node.

this.coefficient = x;

const dummy = new PolyNode();

current = current.next;

current.next = poly2;

poly1 = poly1.next;

poly2 = poly2.next;

current.next = poly1 || poly2;

Time and Space Complexity

return dummy.next;

current = current.next;

} else if (poly1.power < poly2.power) {</pre>

if (sumOfCoefficients != 0) {

current = current.next;

let current = dummy;

Typescript Solution

power: number;

coefficient: number;

next: PolyNode | null;

this.power = y;

this.next = next;

class PolyNode {

} else {

poly1 = poly1->next;

while (poly1 != null && poly2 != null) {

if (poly1.power > poly2.power) {

PolyNode dummyNode = new PolyNode();

// iterator for the resultant list

PolyNode current = dummyNode;

public PolyNode addPoly(PolyNode poly1, PolyNode poly2) {

if current.next:

elif poly1.power < poly2.power:</pre>

if sum_coefficient != 0:

Move to the next nodes in both lists

poly1 = poly1.next

self.power = power

else:

self.next = next_node

13 # Iterate while neither of the polynomial lists is exhausted 14 15 while poly1 and poly2: # Compare powers of both poly nodes 16

If poly1 has the higher power, append it to the result list

If poly2 has the higher power, append it to the result list

current.next = PolyNode(sum_coefficient, poly1.power)

Append the remaining nodes of the polynomial list that is not yet exhausted

PolyNode(int x, int y, PolyNode next) { this.coefficient = x; this.power = y; this.next = next; }

Please note that the main method signature of addPoly cannot be changed as it's probably a constraint given by the problem statemer

sum_coefficient = poly1.coefficient + poly2.coefficient

Move to the next node in the result list if we appended a node

Both powers are equal, add coefficients if they don't cancel out

def addPoly(self, poly1: "PolyNode", poly2: "PolyNode") -> "PolyNode":

Create a dummy node to serve as the head of the result list

39 current.next = poly1 or poly2 40 # Skip the dummy node and return the result list 41 42 return dummy.next 43

Java Solution

class PolyNode {

class Solution {

/**

int coefficient, power;

PolyNode next = null;

// Constructors

PolyNode() {}

/**

*/

8

9

10

11

13

14

15

16

18

19

20

21

22

23

24

25

26

27

28

29

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

61

62

64

10

11

13

14

22

23

24

25

31

32

33

34

35

36

37

38

39

40

41

43

44

45

46

47

48

49

50

51

52

53

54

55

56

58

57 };

12 }

/**

63 };

12 }

```
30
                   // The term from poly1 should come next in the result list
31
                   current.next = poly1;
32
                    poly1 = poly1.next;
33
               } else if (poly1.power < poly2.power) {</pre>
34
                   // The term from poly2 should come next in the result list
                    current.next = poly2;
                   poly2 = poly2.next;
36
37
                } else {
38
                   // Powers are equal, sum the coefficients
39
                   int sumCoefficient = poly1.coefficient + poly2.coefficient;
                   if (sumCoefficient != 0) { // Only add non-zero terms to the result
40
                        current.next = new PolyNode(sumCoefficient, poly1.power);
43
                   // Move forward in both lists
                   poly1 = poly1.next;
44
                   poly2 = poly2.next;
45
46
47
               // Don't move current in the case where sumCoefficient is zero to avoid zero coefficient terms in the list
               if (current.next != null) {
                    current = current.next;
49
50
51
52
53
           // If there's remaining terms in poly1 or poly2, append them to the result
           current.next = (poly1 != null) ? poly1 : poly2;
54
55
56
           // The first node is dummy, so return the next node which is the head of the actual result
57
           return dummyNode.next;
58
59 }
60
C++ Solution
  1 /**
     * Definition for polynomial singly-linked list.
     * This structure is used for representing terms in a polynomial expression
      * where 'coefficient' represents the term's coefficient and 'power' represents
      * the term's exponent. The linked list is sorted in decreasing order of powers.
      */
    struct PolyNode {
         int coefficient, power;
         PolyNode *next;
         PolyNode(): coefficient(0), power(0), next(nullptr) {};
         PolyNode(int x, int y) : coefficient(x), power(y), next(nullptr) {};
         PolyNode(int x, int y, PolyNode* next) : coefficient(x), power(y), next(next) {};
 12
 13 };
 14
    class Solution {
     public:
 16
 17
         /**
 18
          * Adds two polynomial expressions represented by linked lists.
 19
          * @param poly1 First polynomial linked list
 20
          * @param poly2 Second polynomial linked list
 21
          * @return A new polynomial linked list representing the sum of poly1 and poly2.
 22
 23
         PolyNode* addPoly(PolyNode* poly1, PolyNode* poly2) {
 24
             // Dummy node to simplify edge cases and to keep a reference to list start
             PolyNode* dummyHead = new PolyNode();
 25
 26
             PolyNode* current = dummyHead;
 27
 28
             while (poly1 && poly2) {
```

// If poly1 has a term with greater exponent, append it next in the result

// If poly2 has a term with greater exponent, append it next in the result

// If both have the same exponent, sum the coefficients

// Append the new term if the sum is non-zero

// Move forward in the result list if we appended a term

// Append any remaining terms from either polynomial to the result

delete dummyHead; // Clear the memory occupied by the dummy node

constructor(x: number = 0, y: number = 0, next: PolyNode | null = null) {

* @return The sum of poly1 and poly2 as a new polynomial singly-linked list.

// Create a dummy node to serve as the anchor for the result list.

// This will be used to iterate over the new polynomial linked list.

poly1 = poly1.next; // Move to the next term in poly1.

poly2 = poly2.next; // Move to the next term in poly2.

// Move to the next terms in both poly1 and poly2.

// The first node in our resulting linked list is a dummy node,

higher power, or both if the powers are equal and need to be summed.

// so we return the next node which starts the resulting polynomial.

const sumOfCoefficients = poly1.coefficient + poly2.coefficient;

current.next = new PolyNode(sumOfCoefficients, poly1.power);

// Attaching the remaining elements of poly1 or poly2 in case one is longer than the other.

const addPoly = (poly1: PolyNode | null, poly2: PolyNode | null): PolyNode | null => {

* Adds two polynomial singly-linked lists and returns the sum as a new polynomial singly-linked list.

// If the current term of poly2 has a greater power, include it in the result.

// If the powers are equal, add the coefficients if non-zero and include in the result.

// Return the next of dummyHead which points to the start of the sum list

int sumCoefficient = poly1->coefficient + poly2->coefficient;

current->next = new PolyNode(sumCoefficient, poly1->power);

26 // Iterate over both polynomials as long as there are nodes in each list. 27 while (poly1 && poly2) { 28 if (poly1.power > poly2.power) { 29 // If the current term of poly1 has a greater power, include it in the result. current.next = poly1; 30

} else {

The given code defines a function addPoly that takes two polynomial singly-linked lists as input, where each node represents a term in the polynomial (with a coefficient and power), and returns a new polynomial linked list representing the sum of the two input polynomials. **Time Complexity**

The time complexity of this function is 0(n + m), where n is the number of terms in poly1 and m is the number of terms in poly2. This

is because the function iterates over both lists at most once, advancing one node at a time in whichever list has the term with the

Each operation inside the while loop—comparing powers, creating a new node (when coefficients sum to a non-zero value), and

linking nodes—occurs in constant time, 0(1). Since these operations are not nested and we only iterate through each list once

without revisiting any nodes, the time complexity is linear with respect to the total number of nodes in both input lists. Space Complexity

The space complexity of the function is $0(\max(n, m))$. In the worst-case scenario every term from both polynomials needs to be included in the resulting polynomial. However, since the input polynomials' nodes are reused when possible (i.e., when one term's power is greater than the other or when the sum of coefficients is non-zero), additional space is only required when two terms have the same power and their coefficients sum to a non-zero value, which requires allocating a new node. Therefore, the space complexity depends on the number of such term pairs, which is at most the length of the shorter list, resulting in a space complexity that is the maximum of n or m, excluding any space that may be required for the output.