Name: LAKKIREDDY MYTHILI

ROLL: BU21CSEN0101924

PROBLEM STATEMENT Consider an unknown polynomial of degree m. You would require m+1 roots of the polynomial to solve for the coefficients, represented as k = m + 1.

An unknown polynomial of degree m can be represented as:

f(x) = a\_m x^m + a\_{m-1} x^{m-1} + ... + a\_1 x + c

Where:

* f(x) is the polynomial function
* m is the degree of the polynomial
* a\_m, a\_{m-1}, ..., a\_1, c are coefficients (real numbers)
* a\_m ≠ 0 (since it's the highest degree term, ensuring the polynomial is of degree m)

This representation shows that a polynomial of degree m is a sum of terms, where each term is a coefficient multiplied by a power of x. The highest power of x is m, and the powers decrease by 1 for each subsequent term until we reach the constant term c, which has no x.

The task is to find the constant term i.e, ‘c’ of the polynomial with the given roots. However, the points are not provided directly but in a specific format.

You need to read the input from the test cases provided in JSON format.

SOLUTION:

To implement the simplified version of Shamir's Secret Sharing algorithm in Java, we need to solve for the constant term c of the polynomial given the points in a specific format (such as roots or values of the polynomial). The challenge is to find the constant term c by evaluating the polynomial using the given points.

Below is a simplified version of how this can be implemented in Java, following the structure of Shamir's Secret Sharing, using interpolation to reconstruct the polynomial and determine the constant term c.

import java.util.\*;

public class ShamirSecretSharing {

// A helper class to represent a point (x, f(x))

static class Point {

int x;

int y;

Point(int x, int y) {

this.x = x;

this.y = y;

}

}

// Function to compute the constant term 'c' using Lagrange interpolation

// Given a set of points (x\_i, y\_i), find f(0), which is the constant term

public static int findConstantTerm(List<Point> points) {

int constantTerm = 0;

// Lagrange interpolation formula to calculate f(0)

for (int i = 0; i < points.size(); i++) {

int xi = points.get(i).x;

int yi = points.get(i).y;

// Compute the Lagrange basis polynomial for the i-th point

int li = 1;

for (int j = 0; j < points.size(); j++) {

if (i != j) {

int xj = points.get(j).x;

li \*= (0 - xj) / (xi - xj); // Calculating the Lagrange basis li(0)

}

}

// Add the contribution of the i-th point to the constant term

constantTerm += yi \* li;

}

return constantTerm;

}

// Main method to read input and calculate the constant term

public static void main(String[] args) {

// Example input: List of points (x, f(x)) in JSON format

// You can replace this with a real JSON parser or any input format as needed

List<Point> points = new ArrayList<>();

points.add(new Point(1, 6));

points.add(new Point(2, 11));

points.add(new Point(3, 18));

// Calculate the constant term 'c'

int constantTerm = findConstantTerm(points);

// Output the result

System.out.println("The constant term 'c' of the polynomial is: " + constantTerm);

}

}

**Explanation:**

1. **Input:**
   * The input consists of a list of points (x, y) where x is the input to the polynomial, and y = f(x) is the corresponding output of the polynomial at x. The input could be represented in JSON format, but for simplicity, this example initializes the points directly.
2. **Lagrange Interpolation:**
   * The algorithm uses **Lagrange interpolation** to reconstruct the polynomial. We specifically compute f(0), which gives the constant term c of the polynomial. This method ensures that we only need k points to recover the polynomial of degree m = k - 1.
3. **findConstantTerm Function:**
   * This function uses the Lagrange basis polynomials to compute the contribution of each point (x, y) to the constant term c.
4. **Output:**
   * Once the constant term is computed, it is printed out.

**How It Works:**

1. The points (x, y) are used to reconstruct the polynomial.
2. For each point, the Lagrange basis polynomial is computed, and the value f(0) is derived by adding all the contributions.
3. The final result is the constant term c.

**Example:**

For the points:

* (1, 6)
* (2, 11)
* (3, 18)

This will output the constant term c of the polynomial.

**Important Notes:**

* This implementation assumes integer coefficients and points.
* The findConstantTerm function will compute f(0) using Lagrange interpolation. For more complex or floating-point coefficients, appropriate handling is required.
* If the input is provided in JSON format, you can parse it using a library like org.json or Jackson to populate the list of points.

You can now run this Java program, and it will output the constant term c based on the input points representing the polynomial.

**Step 1: Read the JSON Input**

1. **JSON Structure**: The input contains the number of roots n, the minimum number of roots k, and each root has a base and a value. The JSON input file is named input<dependency>
2. <groupId>com.fasterxml.jackson.core</groupId>
3. <artifactId>jackson-databind</artifactId>
4. <version>2.12.1</version>
5. </dependency>.json.
6. **Read the JSON**: You will use the Jackson or Gson library to parse the JSON input in Java.
7. **Install the Library**: If you haven't installed Jackson or Gson, you can add it as a dependency:

import java.util.ArrayList;

public class PolynomialSolver {

public static void main(String [] args) {

try {

ObjectMapper mapper = new ObjectMapper();

InputData data = mapper.readValue(new File("input.json"), InputData.class);

// Step 2: Decode the values and collect points

List<int []> points = new ArrayList<>();

for (Root root : data.roots) {

int x = root.x;

int y = Integer.parseInt(root.value, root.base); // Decoding value

points.add(new int[]{x, y});

}

// Print decoded points

for (int[] point : points) {

System.out.println("Point: (" + point[0] + ", " + point[1] + ")");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Step 3: Collect Decoded Points**

1. **Points Representation**: After decoding, you will have a set of points (x, y), where x is the original key and y is the decoded value.
2. The points will look like:
   * (2, 7)
   * (10, 4)
   * (4, 27)
   * (12, 12)

The list of points will be stored in a 2D list or array.

**Step 4: Calculate the Polynomial Coefficients**

1. **Lagrange Interpolation: Use Lagrange interpolation to calculate the polynomial. The formula for Lagrange interpolation is:**
2. f(x)=i=0∑n​yi​⋅Li​(x)
3. where Li(x)L\_i(x)Li​(x) is the Lagrange basis polynomial:

**Li​(x)=j=0j=i​∏n​xi​−xj​x−xj​​**

**public class PolynomialSolver {**

**// Method to compute Lagrange interpolation and find the constant term**

public static double lagrangeInterpolation(List<int[]> points) {

int n = points.size();

double constantTerm = 0;

for (int i = 0; i < n; i++) {

double li = 1.0;

for (int j = 0; j < n; j++) {

if (i != j) {

li \*= (0 - points.get(j)[0]) / (double) (points.get(i)[0] - points.get(j)[0]);

}

}

constantTerm += li \* points.get(i)[1];

}

return constantTerm;

}

public static void main(String[] args) {

try {

ObjectMapper mapper = new ObjectMapper();

InputData data = mapper.readValue(new File("input.json"), InputData.class);

// Step 2: Decode the values and collect points

List<int[]> points = new ArrayList<>();

for (Root root : data.roots) {

int x = root.x;

int y = Integer.parseInt(root.value, root.base); // Decoding value

points.add(new int[]{x, y});

}

// Step 4: Calculate the constant term using Lagrange interpolation

double constantTerm = lagrangeInterpolation(points);

System.out.println("Constant term (c) = " + constantTerm);

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Step 5: Find the Constant Term (c)**

1. **Constant Term: The constant term is the value of the polynomial when x = 0. In Lagrange interpolation, this term is obtained after calculating the polynomial.**
2. **Output: The lagrangeInterpolation() function will return the constant term after calculating the polynomial using the given points.**

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.File;

import java.util.ArrayList;

import java.util.List;

class Root {

public int x;

public int base;

public String value;

}

class InputData {

public int n;

public int k;

public List<Root> roots;

}

public class PolynomialSolver {

// Method to compute Lagrange interpolation and find the constant term

public static double lagrangeInterpolation(List<int[]> points) {

int n = points.size();

double constantTerm = 0;

for (int i = 0; i < n; i++) {

double li = 1.0;

for (int j = 0; j < n; j++) {

if (i != j) {

li \*= (0 - points.get(j)[0]) / (double) (points.get(i)[0] - points.get(j)[0]);

}

}

constantTerm += li \* points.get(i)[1];

}

return constantTerm;

}

public static void main(String[] args) {

try {

// Step 1: Read the JSON input

ObjectMapper mapper = new ObjectMapper();

InputData data = mapper.readValue(new File("input.json"), InputData.class);

// Step 2: Decode the values and collect points

List<int[]> points = new ArrayList<>();

for (Root root : data.roots) {

int x = root.x;

int y = Integer.parseInt(root.value, root.base); // Decoding value

points.add(new int[]{x, y});

}

// Step 4: Calculate the constant term using Lagrange interpolation

double constantTerm = lagrangeInterpolation(points);

System.out.println("Constant term (c) = " + constantTerm);

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Summary of Steps:**

1. **Step 1: Read the JSON input using Jackson library.**
2. **Step 2: Decode the value from the specified base for each root.**
3. **Step 3: Collect the decoded points as (x, y) pairs.**
4. **Step 4: Use Lagrange interpolation to calculate the polynomial coefficients.**
5. **Step 5: Find the constant term by evaluating the polynomial at x = 0.**

**This complete Java solution reads the input, decodes the roots, performs Lagrange interpolation, and prints the constant term of the polynomial.**