DOCUMENTATION

Big Rational Arithmetic and Robust Geometric Primitives

Introduction

The project has the following cpp files along with their respective header files –

1. BigInteger
2. BigRational
3. Number
4. RobustGeometricPrimitives2D

Source.cpp file includes all the header files and tests implementation of the operators implemented.

Features

BigInteger provides operators for the following operations –

1. Mathematical operations
    - Summation (operator +)
    - Subtraction (operator -)
    - Division (operator /)
    - Multiplication (operator *)
    - Modulus (operator %)
2. Logical operators
    - <
    - >
    - <=
    - >=
    - ==
    - !=
3. GCD
4. LCM
5. Power operator (Pow)
6. Absolute

Implementation

BigInteger

All the operations in BigInteger have been implemented based on the schoolbook implementation of the following operations operators –

1. For summation of two BigIntegers x and y, the addition takes place depending on the signs of the two BigIntegers. For an instance, if the two BigIntegers are x and –y, the resulting addition would be the subtraction of x and y. If the two BigIntegers are –x and –y, the resulting addition would be the addition of x and y and then adding the negative sign bit to the result.

   After the addition on the absolute values of the BigIntegers, the sign bit is implemented according to the inputs given. The result would have the same sign as that of the number with the higher absolute value.

2. For subtraction of two BigIntegers x and y, the subtraction takes place depending on the signs of the two BigIntegers. For an instance, if the two BigIntegers are x and –y, the resulting subtraction would be the addition of x and y since x-(-y) = x + y

   After the subtraction on the absolute values of the BigIntegers, the sign bit is added according to the inputs given. The result would have the same sign as that of the number with the higher absolute value.

3. For multiplication of two BigIntegers x and y, the multiplication takes place on the absolute values of the two BigIntegers based on the schoolbook paper implementation.

   The sign bit is implemented according to the inputs given. If one of the numbers is a negative number then the result would have the negative sign bit otherwise the result would have a positive sign bit.

4. For division of two BigIntegers x and y, the division takes place on the absolute values of the two BigIntegers according to the algorithm BasecaseDivRem (from the Integer arithmetic chapter)

   The sign bit is implemented according to the inputs given. If one of the numbers is a negative number then the result would have the negative sign bit otherwise the result would have a positive sign bit.

5. For modulus of two BigIntegers x and y, if one of the numbers is a negative number, then the modulus is calculated based on the following logic on the absolute values of the numbers -

   Quotient = dividend/divisor
   Modulus = ((quotient + BigInteger(1)) * divisor) – dividend

   And the sign bit is added based on the sign of y, if its negative the result would be negative otherwise it would be positive.

   In case both the numbers are positive or negative, the modulus is calculated based on the following logic on the absolute values of the numbers –

   Quotient = dividend/divisor
   Modulus = dividend - (quotient * divisor)

   And the sign bit is added based on the signs of the two numbers.

BigRational

BigRational makes use of BigIntegers to implement all the operations. A BigRational object is basically represented in a p/q form, i.e.

BigRational = BigInteger/BigInteger

Following are the operations implemented on the BigRational –

1. For summation of two BigRationals a and b, following is their representation as BigIntegers,

   a = BigInteger p /BigInteger q

   b = BigInteger r /BigInteger s

   So for addition of a and b,

   p/q + r/s = (p*s + r*q)/ (q*s)

   And the result we obtain is also in p/q form which is then further reduced by dividing the numerator and the denominator with the highest common factor (HCF) of the numerator and the denominator. The resulting rational number is then represented in the decimal form.

2. For subtraction of two BigRationals a and b –

   p/q – r/s = (p*s - r*q)/ (q*s)

   And the result we obtain is also in p/q form which is then further reduced by dividing the numerator and the denominator with the highest common factor (HCF) of the numerator and the denominator. The resulting rational number is then represented in the decimal form.

3. For multiplication of two BigRationals a and b –

   p/q * r/s = (p *r)/(q*s)

   And the result we obtain is also in p/q form which is then further reduced by dividing the numerator and the denominator with the highest common factor (HCF) of the numerator and the denominator. The resulting rational number is then represented in the decimal form.

4. For division of two BigRationals a and b –

(p/q) / (r/s) = (p*s)/(r*q)

And the result we obtain is also in p/q form which is then further reduced by dividing the numerator and the denominator with the highest common factor (HCF) of the numerator and the denominator. The resulting rational number is then represented in the decimal form.

Number:

The number is implemented using the BigRational as its internal value. All the calculations on Number value are based on BigRational calculations. The result is stored in a number in the form of p/q (BigRational) and then displayed to the user as a decimal value.

The Number is input by a user in string form. To initialize a number as 25.26, for example, the user calls the constructor as Number(“25.26”). The constructor is programmed to extract the numerator and denominator from the string and call the BigRational constructor with respective numerator and denominator.

For output. We have a **toDecimalString(int numberOfDigits)** function in the BigRational class which takes the number of Decimals the user wants to display and displays those integers. The algorithm works as follows:

```
calculate quotient and the remainder.
the quotient.toString() gets appended to the resulting string output.
if the remainder is zero, done.
else push remainder to a vector<BigInteger> to keep track of the remainders
Enter a loop: from zero to the number of digits.
    multiply the remainder by 10
    append(remainder / denominator).toString() to the resulting string
    update the remainder : remainder%denominator
    if the remainder exists in the vector, append brackets("|") to the resulting string
    output
    to show repeating part, and break the loop.
    else, push the remainder in the vector and continue the loop.
```

The default output is displaying ten digits.

Robust Geometric Primitives Documentation

List of Spatial Datatypes

Poi2D

The point datatype consists of two instances of type 'Number', which represents the x and the y coordinate respectively.

Data Members

Number x: The x coordinate of the point.

Number y: The y coordinate of the point.

Logical Operations

The following operations have been overloaded and can be used as a Boolean operation between two points <, >, >=, <=, ==,!= .

Initits

Seg2D

The Seg2D datatype is composed of two points in which the point with the lesser x-coordinate is stored in p1 and the other point in p2. If the x coordinates are equal, the point with the lesser y coordinate is stored in p1 and the other point is stored in p2.

Data Members

Poi2D p1: The left endpoint of the segment.

Poi2D p2: The right endpoint of the segment.

Logical Operations

The following operations have been overloaded and can be used as a Boolean operation between two points <, >, >=, <=, ==,!=.

HalfSeg2D

The HalfSeg2D datatype is composed of a segment 'seg' and a Boolean flag called 'isLeft' which indicates which point should be the dominating point.

Data Members

Seg2D seg: The segment which the Halfsegment is based on.

Bool isLeft: The flag indicating the dominatingPoint.

Member Functions

Poi2D dominatingPoint: Returns the dominating point corresponding to the given halfsegment.

Logical Operations

The following operations have been overloaded and can be used as a Boolean operation between two points <, >, >=, <=, ==, !=.

AttrHalfSeg2D

The AttrHalfSeg2D datatype is composed of a halfsegment and a Boolean flag, called 'insideAbove'.

Data Members

HalfSeg2D hseg

Bool insideAbove

Logical Operations

The following operations have been overloaded and can be used as a Boolean operation between two points <, >, >=, <=, ==,!=.

SimplePolygon2D

The SimplePolygon2D datatype is composed of a vector of type Poi2D.

Data Members

std::vector<Poi2D>: This is a vector of points that represent the vertices of the polygon.

Logical Operations

The following operations have been overloaded and can be used as a Boolean operation between two points <, >, >=, <=, ==,!=.

List of Geometric primitives

Point Functions

1. bool **PointLiesOnSegment**(Poi2D& poi, Seg2D& seg)

   This function checks to see whether the given point lies anywhere on the segment, including the endpoints.

2. Poi2D **getPointLiesOnSegmentAndNotEndpoints**(Poi2D& poi, Seg2D& seg)

   This function returns the point that lies on the segment and does not lie on one of the endpoints.

3. bool **PointLiesOnSegmentAndNotEndpoints**(Poi2D& poi, Seg2D& seg)

   This function returns true if the point obtained as input lies on the segment and not on the endpoints.

4. bool **PointLiesAboveSegment**(Poi2D& poi, Seg2D& seg)

   This function returns true if the point lies above the segment. It uses the anticlockwise test to determine whether or not the point lies above the given segment.

5. bool **PointLiesBelowSegment**(Poi2D& poi, Seg2D& seg)
   This function returns true if the point lies above the segment.

6. bool **PointLiesAboveOrOnSegment**(Poi2D& poi,Seg2D& seg)

   This function returns true if the point lies above or on the given segment and false otherwise.

7. bool **PointLiesBelowOrOnSegment**(Poi2D& poi, Seg2D& seg)

   This function returns true if given point lies below or on the given segment and false otherwise.

8. bool **PointLiesOnLeftEndPointOfSegment**(Poi2D& poi, Seg2D& seg)

   This function returns true if given point lies on the left endpoint of the given segment and false otherwise.

9. bool **PointLiesOnRightEndPointOfSegment**(Poi2D& poi,Seg2D& seg)

   This function returns true if given point lies on the left endpoint of the given segment and false otherwise.

10. bool **PointIsCollinearToSegment**(Poi2D& poi, Seg2D& seg)

    This function returns true if the given point is collinear to the given segment and false otherwise.

11. bool **PointLiesLeftOFSegmentAndIsCollinear**(Poi2D& poi, Seg2D& seg)

    This function returns true if the given point is collinear and lies to the left of the given segment false otherwise.

12. bool **PointLiesRightOfSegmentAndIsCollinear**(Poi2D& poi, Seg2D& seg);

    This function returns true if the given point is collinear and lies to the left of the given segment false otherwise.

Segment Functions

1. bool **SegmentLiesAboveSegment**(Seg2D& seg, Seg2D& seg1)

   This function returns true if the segment 'seg' lies above the segment 'seg1' and returns false otherwise.

2. bool **SegmentLiesBelowSegment**(Seg2D& seg, Seg2D& seg1)

   This function returns true if the segment 'seg' lies below the segment 'seg1' and returns false otherwise.

3. bool **SegmentLiesLeftOFSegment**(Seg2D& seg, Seg2D& seg1)

   This function returns true if the segment 'seg' lies to the left of the segment 'seg1' and returns false otherwise.

4. bool **SegmentLiesRightOfSegment**(Seg2D& seg, Seg2D& seg1)

This function returns true if the segment ‘seg’ lies to the right of the segment ‘seg1’ and returns false otherwise.

5. bool **SegmentIsCollinear**(Seg2D& seg, Seg2D& seg1)

   This function returns true if the segment ‘seg’ is collinear to the segment ‘seg1’ and returns false otherwise.

6. bool **SegmentLiesLeftOFSegmentAndIsCollinear**(Seg2D& seg, Seg2D& seg1)

   This function returns true if the segment ‘seg’ lies to the left and is collinear to the segment ‘seg1’ and returns false otherwise.

7. bool **SegmentLiesRightOfSegmentAndIsCollinear**(Seg2D& seg, Seg2D& seg1)

   This function returns true if the segment ‘seg’ lies to the right and is collinear to the segment ‘seg1’ and returns false otherwise.

8. bool **SegmentIsCollinearAndMeetsLeftEndpoint**(Seg2D& seg, Seg2D& seg1)

   This function returns true if the segment ‘seg’ is collinear and meets the left endpoint to the segment ‘seg1’ and returns false otherwise.

9. bool **SegmentIsCollinearAndMeetsRightEndpoint**(Seg2D& seg,Seg2D& seg1)

   This function returns true if the segment ‘seg’ is collinear and meets the right endpoint to the segment ‘seg1’ and returns false otherwise.

10. bool **SegmentIsCollinearAndCrossesLeftEndpoint**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is collinear to the segment ‘seg1’ and extends past the left endpoint and returns false otherwise.

11. bool **SegmentIsCollinearAndCrossesRightEndpoint**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is collinear to the segment ‘seg1’ and extends past the right endpoint and returns false otherwise.

12. bool **SegmentIsCollinearAndMeetsBothEndpoint**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is collinear to the segment ‘seg1’ and extends past the right endpoint and returns false otherwise.

13. bool **SegmentIsParallel**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is parallel to the segment ‘seg1’ and false otherwise.

14. bool **SegmentIsParallelAndAbove**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is parallel and lies above the segment ‘seg1’ and false otherwise.

15. bool **SegmentIsParallelAndBelow**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is parallel and lies below the segment ‘seg1’ and false otherwise.

16. bool **SegmentIsParallelAndLiesLeft**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is parallel and lies to the left of the segment ‘seg1’ and false otherwise.

17. bool **SegmentIsParallelAndLiesRight**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ is parallel and lies to the right of the segment ‘seg1’ and false otherwise.

18. bool **Intersects**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ intersects the segment ‘seg1’ and false otherwise. Segments that meet at the endpoints are not considered to intersect.

19. bool **SegmentIsLesserThanSegment**(Seg2D& seg1, Seg2D& seg2)

    This function checks the length of the segments and returns true if the segment ‘seg1’ is lesser than the segment ‘seg2’.

20. Poi2D **IntersectionPoint**(Seg2D& seg, Seg2D& seg1)

    This function returns the point of intersection of ‘seg’ and ‘seg1’. This must be used in conjunction with the ‘Intersects’ function, which first determines whether a point of intersection exists.

21. bool **Meet**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ meets the segment ‘seg1’ at one its endpoints.

22. Poi2D **MeetingPoint**(Seg2D& seg, Seg2D& seg1)

    This function returns true if the segment ‘seg’ meets the segment ‘seg1’ at one its endpoints.

23. Poi2D **MidPoint**(Seg2D& seg1)

    This function returns the midpoint of of the given segment, ‘seg1’.

24. bool **BasicPointInBoundingBox**(Poi2D& poi, SimplePolygon2D& polygon)

This function constructs a bounding box around the given simple polygon and returns true if the point lies outside the constructed bounding box. This is a basic first step that we take to check whether or not a point lies inside a polygon.

25. bool **simplePointInsideSimplePolygon**(Poi2D& poi, SimplePolygon2D& simplepolygon)

    This is a function that determines whether a given point lies within a polygon.

26. bool **simplePointOnBoundaryOfSimplePoly**(Poi2D& poi, SimplePolygon2D& simplepolygon)

    This is a function that determines whether a given point lies on the boundary of the simple polygon.

27. bool **segInsideSimplePolygon**(Seg2D& seg, SimplePolygon2D& simplepolygon)

    This function returns true if the given segment is inside the given polygon and false otherwise.

28. bool **segOnBoundaryOfSimplePolygon**(Seg2D& seg, SimplePolygon2D& simplepolygon)

    This function returns true if the given segment lies on the boundary of the given polygon and false otherwise.