Object-Oriented Programming
Homework 1 Documentation

Iova Daniel-Alexandru
CEN 2.2B

Table of contents

Problem statement

4. Design and implement a module for polygons. The module must allow to solve the following problems related to polygons:
- Creating of a polygon;
- Printing of a polygon;
- Determining the rectangle with a minimum area, which contains inside the polygon;
- Determining the area of a polygon;
- Determining the perimeter of a polygon;
- Determining the gravity center;
- Determining if the polygon is convex;
- Comparison of polygons according to their area;
- Concatenation of two polygons; the concatenation of two polygons is a convex polygon obtained as following:
- Determines the reunion of the two sets containing the vertices of the two polygons;
- Determines the convex-hull of the set of vertices above determined;
- The vertices of the convex-hull represent the vertices of the new polygon;

Things to note

- All reading is done from stdin.
- All writing is done to stdout.
- Input is considered to be correct by default. The program does not check if the given polygon self-intersects or that it's not suited for computing.
- The terms "Polygon" and "Point List" are used interchangeably.
- When an operation fails, the testing function will output a message to stdout that lets the user know.
- For the minimum area rectangle we have considered a rectangle with segments parallel to the Ox and Oy axees. No rotation was done to minimize the rectangle area. Also, if the polygon point only has collinear points the bounding box function will not return a correct answer.

Functionality and Testing

Creating a polygon

Polygon creation is done in the following steps:

1. Start with an empty list. This will be our polygon
2. Read the number of points the polygon will have (we'll call it n)
3. Read the coordinates of the n points. The format of the input will be "{x} {y}"
4. Append a list of [x, y] format to the point list
5. Return the point list

Testing polygon creation

Polygon creation is tested by first assigning the value returned by the polygon creation function to a new list. If that list is empty, we assign the return of the polygon creation function to the said list until the list is not empty anymore. This ensures that the read polygon is valid everytime. In the end we return the polygon.

Printing a polygon

Polygon printing is done through the following steps:

1. Iterate through the polygon's points
2. For each point, output a line of format "Point[{index}]'s coordinates: {x} {y}", where:
    a. index is the index of the current point
    b. x is the first element of the point
    c. y is the second element of the point

Testing polygon printing

To test the polygon printing we first check to see if the polygon is empty. If it is, printing is impossible. If it is not, we call the polygon printing function.

Determining the bounding box

The bounding box is computed as follows:

1. Compute the bottom left point:
    a. The x coordinate will be the minimum of all x coordinates (we'll call it left_x)
    b. The y coordinate will be the minimum of all y coordinates (we'll call it low_y)
2. Compute the top right point:
    a. The x coordinate will be the maximum of all x coordinates (we'll call it right_x)
    b. The y coordinate will be the maximum of all y coordinates (we'll call it high_y)
3. The bounding box will have the following coordinates (or any rotation of these points):
    a. [left_x, high_y]
    b. [right_x, high_y]
    c. [right_x, low_y]
    d. [left_x, low_y]
4. We return the bounding box, which is a list containing the points above

The minimum and maximum are done using python's max, min and list comprehension tools.

Testing polygon bounding box

The bounding box function will always return a non-empty point list.
Knowing this, we will only test if there is only one point in the polygon, and, if there is, we cannot compute the bounding box. If there are two or more, we call the minimum bounding box function and print it using the print-testing function created earlier.

Determining the area of a polygon

The area was calculated using the shoelace method:

$$\text{Area} = \frac{1}{2}\left|\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_{i1})\right|$$

The computation went as follows:

1. Start with area being equal to 0
2. Iterate through the points of the polygon (index i):
    a. Store the next point (index j equal to (i+1)%length to wrap-around at the end)
    b. Compute (polygon[i].x*polygon[j].y)-(polygon[j].x*polygon[i].y)
    c. Add the result of the above computation to the area
3. Make the area absolute then divide by 2
4. Return the area

The returned area will be of float type.

Testing polygon area

We test if the polygon has less than three points, and, if it does, the area cannot be computed. If it has more than two, we print the result of the polygon area function.

Determining the perimeter of a polygon

Determining the perimeter was done using 2 functions:

1. The first one determines the distance between 2 points
2. The second uses the first to calculate the perimeter

The first function:

1. It receives two points as arguments
2. Returns the distance between them using the formula $\sqrt{(x_B - x_A)^2 + (Y_B - Y_A)^2}$

The second function:

1. Initializes the perimeter with 0
    a. Adds the distance between the current and the next points to the perimeter
    b. The index of the next point will wrap-around for the final edge
2. Returns the perimeter

Testing polygon perimeter

We test if the polygon has less than two points, and, if it does, the perimeter cannot be computed (no edges to add to the perimeter). If it has more than one, we print the result of the polygon perimeter function.

Determining the gravity center of a polygon

Determining the gravity center uses the area of the polygon in the following formulas:

$$C_X = \frac{1}{6 \cdot Area} \left| \sum_{i=0}^{n-1} (x_i + x_{i+1}) \cdot (x_i y_{i+1} - x_{i+1} y_{i1}) \right|$$

$$C_Y = \frac{1}{6 \cdot Area} \left| \sum_{i=0}^{n-1} (y_i + y_{i+1}) \cdot (x_i y_{i+1} - x_{i+1} y_{i1}) \right|$$

Knowing this, the computing is done as follows:

1. Initialize Cx and Cy with 0
2. Compute the area of the matrix using the already created function
3. If the area is 0 we return an empty list, because the gravity center can't be computed
4. If the area is not 0:
    a. Calculate the multiplier which is equal to 1/(6*area)
    b. Iterate from 0 to length-2 (index i):
        i. Store the next point (index j equal to i+1)
        ii. Compute (polygon[i].x*polygon[j].y)-(polygon[j].x*polygon[i].y)
        iii. Store the computation in a variable, we'll call it shoelace
        iv. Add (polygon[i].x+polygon[j].x)*shoelace to Cx
        v. Add (polygon[i].y+polygon[j].y)*shoelace to Cy
5. Return a list of format [Cx*multiplier, Cy*multiplier], with the values rounded down to 3 digits for easier readability

The gravity center computation will not work with self-intersecting polygons.

Testing polygon gravity center

Testing the gravity center is done by first seeing if we only have one point in the polygon.
If we do, the gravity center cannot be calculated.
If there are two or more points, we call the polygon gravity center function and check if the returned list is empty. If it is, the centroid computing failed. If it is not, we print the gravity center.

Determining if a polygon is convex

Determining the convexity was done using 2 functions:

1. The first one determines the orientation of three consecutive points
2. The second uses the first to compute the polygon convexity

The orientation function:

1. Receives 3 points as input
2. Uses their slope to compute the orientation:
    a. We start from the comparison of the slopes of lines p-q and q-r
    b. The slopes are:
        i. p-q = (q.y-p.y)/(q.x-p.x)
        ii. q-r = (r.y-q.y)/(r.x-q.x)
    c. Knowing this, we can get the formula:
        i. (q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y)
    d. Who's returned value can be compared against 0 as follows:
        i. Value > 0 : the points are in clockwise order
        ii. Value < 0 : the points are in counterclockwise order
        iii. Value = 0 : the points are collinear
3. Returns the value of the computed formula

The convexity function:

1. Receives the polygon as input
2. Computes the orientation of the first three points
3. It then calculates a multiplier based on the orientation above, assuming clockwise default:
    a. 1 if the orientation is clockwise
    b. -1 if the orientation is counter clockwise
4. After this, it iterates through the points:
    a. Computes the orientation of the current, next and next next point
    b. If the multiplier*orientation is < 0 we return False (because we break the chain of similar orientations)
5. We return true at the end (there was no orientation chain break)

Testing polygon convexity check

A polygon with less than three points is only a segment (or point) and therefore the convexity check cannot be made. If the polygon has three or more points, we call the convexity check function and use it's return value to determine if it's convex (returns true) or concave (returns false).

Comparison of polygons according to their area

Simple function which takes two parameters.
Returns if the area of the first polygon is greater than the area of the second.

Testing polygon area comparison

This testing function only receives the original polygon, so we read the second one using the test reading function from earlier. Once the second polygon is read, the steps are similar to the gravity center testing function. Using the return value of the comparison function, we output who has the greater area.

Concatenation of two polygons

Concatenating two polygons is done as follows:

1. Make a reunion of the two input polygons
2. Return the convex hull of the reunion

Making a reunion is straight forward, we have an empty list to which we simply add the first and the second polygon.

For the convex hull, we use the Jarvis' March Algorithm, described as follows:

1. Automatically return an empty list if the number of points is less than 3

2. Initialize the hull as an empty list
3. Find the index of the leftmost point and initialize p with it
4. Do the next steps while we are not back at the leftmost point:
    a. Store the next element's index after index p in index q (we mod p by the number of points so that it wraps around)
    b. Iterate through the polygon's points (index i):
        i. If the orientation of polygon[p], polygon[i] and polygon[q] is counterclockwise q gets the value of i
    c. At the end of the current iteration we store q in p for the next iteration
5. Return the computed hull

To find the leftmost point we iterate through the list of points and do the minimum of the x values of each point, but only store the index of the minimum (so we'd have polygon[i][0] < polygon [left][0] for each index i). We start with left equal to 0 so we iterate from 1 to length-1.

To calculate the orientation we use the function created for the polygon convexity check function.

Testing polygon concatenation

We read the second polygon using the read testing function. From this we call the concatenation function and check if the returned convex hull is empty. If it is, the concatenation failed. If it isn't, we print the resulting polygon using the test print function.

Menu function

Everything is connected by the menu function present in the main file. This receives the polygon, an empty list, and is populated only by the test read function. The menu works like an infinite switch statement, with the initial option being -1 and then running the menu until the user presses 0. Inside the menu, the options are displayed and user input is requested. For each option their respective test function is run.