Goldy Kumar

ME558- Homework 1

Classes :

1. Main
2. HalfPlanes
3. LineSeg
4. Point
5. Vector

Main

a) Gets input
b) Gets points and form arraylist of ***LineSeg*** (edges): (Detail in LineSeg Class)
c) Gets a, b, c values and forms an array ***HalfPlanes*** (Deatails in HalfPlanes Class)
d) Clips polygon (edges) by every halfplane one by one by calling the method ***ClipPolygon*** and forming new polygon all the time
e) Calculate the area for the polygon

HalfPlane

a) Variables:
   - a, b, c : Constructs an halfplane using a, b, c values
   - LineSeg: Finds two points on the halfplane and forms a line segment passing through it
b) Methods:
   - PMI method (Point membership) which returns 1, -1, 0 depending on whether the point is inside, outside or on.
   - ClipPolygon: Clips the edges passed as argument by the halfplane to get the new polygon contained in the half plane

LineSeg

a) Variables:
   - Points: starting and ending point
   - Vector: from starting to ending point

b) Methods:
   - lineSegHalfPlaneIntersection: takes halfplane as argument and finds the intersection point with the line segment it is called upon

Point

a) Variable:
   - double x, y – coordinates of the point

b) Method:
   - Equals: takes another point as input and returns true if the they lie in proximity to each other (circle of radius 10e-6)

Vector

a) Variable:
   - double x, y – components along X and Y direction

b) Method:
   - CrossProd – takes another vector as input and returns cross-product with that

<u>Pseudo-Code (explaining the important methods)</u>

1. Get the input as a string
2. Read 1st integer as ***n***
3. Read 2nd integer as ***m***
4. Get next ***2n*** double and form ***n*** instances of Point
   a. ArrayList<Point> ***polygon*** = points
5. Taking 2 Points at a time form ***n*** LineSeg
   a. Store the start point ***'a'*** and end point ***'b'*** considering clockwise ordering
   b. Form a Vector ***vL*** from ***'a'*** to ***'b'***
   c. ArrayList<LineSeg> ***lineSeg*** = lines formed
6. Read next 3m doubles to form HalfPlanes
   a. Store a, b, c values considering the equation ax+by <= c
   b. Form a line segment: ***lineSeg*** by finding two points on the halfplane
      i. Let x1 = -1.0, x2 = 1.0

ii. Let y1 = -1.0, y2 = 1.0

iii. If ((***a*** and ***b*** very small) or (***a*** and ***b*** not very small))
y1 = (c-a*x1)/b;
y2 = (c-a*x2)/b;

iv. Else if (***b*** is very small) [half plane is parallel to x axis]
1. x1 = c/a;
2. x2 = x1;

v. Else if (***a*** is very small) [half plane is parallel to y axis]
1. y1 = c/b;
2. y2 = y1;

vi. Point ***p*** = new Point(x1, y1)

vii. Point ***q*** = new Point(x2, y2)

viii. ***lineSeg*** = new LineSeg(p, q)

c. ArrayList<HalfPlanes> ***halfplanes*** = halfplanes read

7. For (every halfplane ***hp***)
   a. lines = Hp.ClipPolygon(lines)
      i. if ***lines*** empty - return lines
      ii. ArrayList<Point> ***clipped*** ; //to store points on clipped polygon
      iii. ArrayList<LineSeg> ***clippedLineS*** ; //line segements of the clipped polygon
      iv. For (every lines = ***ls***)
         1. ***P*** = ***ls.Linesegment_Halfplane_Intersection(Hp)*** //null if no intersection
         2. If (clipped is empty and start point of ***ls*** inside the halfspace)
            a. Clipped.add(***ls*** start point)

         //if vertex is the point of intersection don't include it twice
         3. Else if (last point in clipped is not equal to start point of ***ls***)
            a. If (start point of ***ls*** is inside half plane)
               i. Clipped.add(***ls*** start point)
         4. If (there is a point of intersection)
            a. //check if that point of intersection is not the start vertex of ls
            b. If (***p*** not equal to ***ls*** start point)
            c. Clipped.add(intersection point ***p***)
      v. Form line segments- ***lines*** from the clipped vertex
      vi. Return ***lines***
8. Area of the Resultant clipped polygon (clockwise) using the formula:
   i. 2x***Area*** = Sum (xi+1 * yi - xi * yi+1)

Point Memebership classification of point ‘***p***’ for a half plane(***a, b, c***)

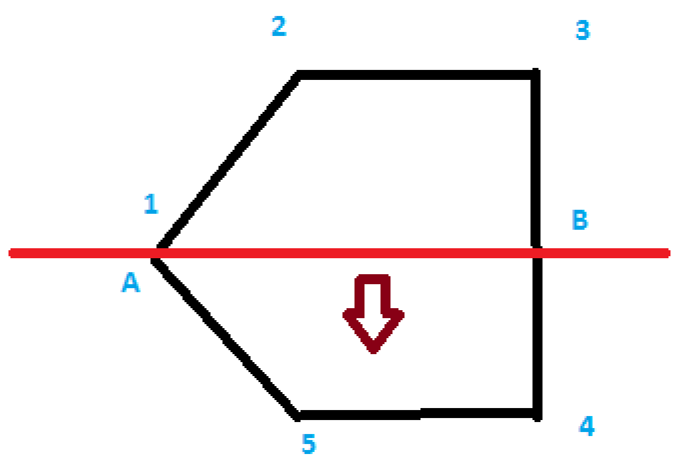
I. If ((a*p_x + b*p_y – c) is very near to 0) → return 0
II. If ((a*p_x + b*p_y – c) < 0) → return 1
III. If ((a*p_x + b*p_y – c) > 0) → return -1

Line Segement- ***ls (a to b)*** and Half Plane – ***hp(x to y)*** Intersection

1. Vector ***xy*** = ***hp***.vector
2. Vector ***ax*** = vector from ***x*** to ***a***
3. Vector ***bx*** = vector from ***x*** to ***b***
4. //there is an intersection between ***hp*** and ***ls*** if endpoints of ***ls*** are in opposite side of ***hp***
5. If (***ab***.CrossProd(***xy***) == 0) //lines are collinear
    a. Return ***a;***
    b. Else If (***ax***_crossprod_***xy*** and ***bx***_crossprod_***xy*** are of opposite sign)
        i. ***s*** = (***xy***.CrossProd(***ax***))/***ab***.CrossProd(***xy***);
        ii. Point of intersection ***p*** = ***a*** + ***s****ab***
        iii. Return ***p***

Special Cases:

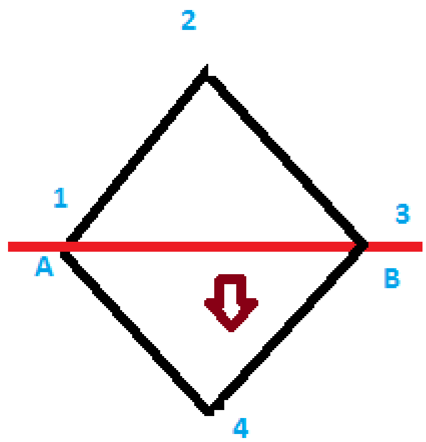
1.



a) Adds vertex 1 to the clipped polygon vertex list (clipped list)
b) Finds intersection of 1-2 with hp (halfplane) – A. Since A == 1. A is not added

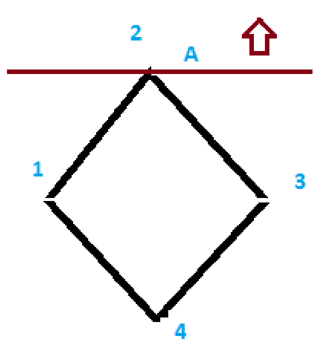
c) 2, 3 rejected
d) B found as intersection; not equal to the last element in the clipped list so added
e) 4 and 5 are added.
f) Final clipped vertex list- 1, B, 4, 5

Case 2:



a) Adds vertex 1 to the clipped polygon vertex list (clipped list)
b) Finds intersection of 1-2 with hp (halfplane) – A. Since A == 1. A is not added
c) 2 rejected; B found as intersection of 2-3; B added
d) 3 == B (last added element); 3 rejected
e) 4 inside; 4 added
f) Final clipped vertex list- 1, B, 4

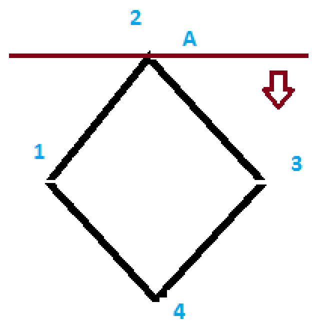
Case 3:



a) 1 outside rejected; 1-2 intersects at A; A added
b) 2 ON; but 2==A (last added element); 2 rejected

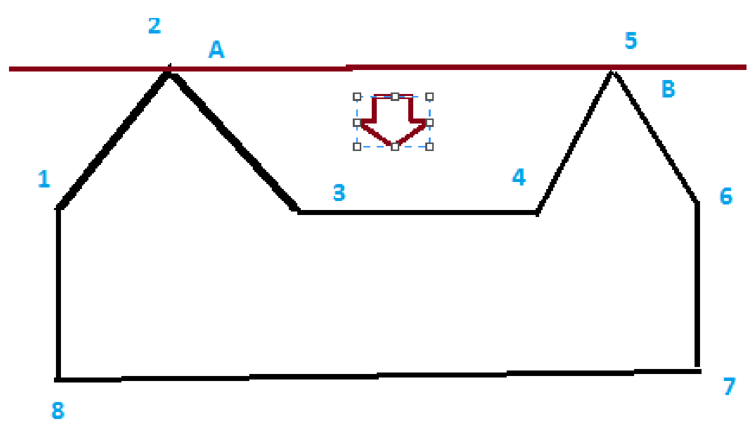
c) 3 and 4 rejected;
d) Final vertex list – A → area = 0

Case4:



a) 1 inside- added; 1-2 intersects at A; A added
b) 2 ON; but 2==A (last added element); 2 rejected
c) 3 and 4 added;
d) Final vertex list – 1, A, 3 ,4

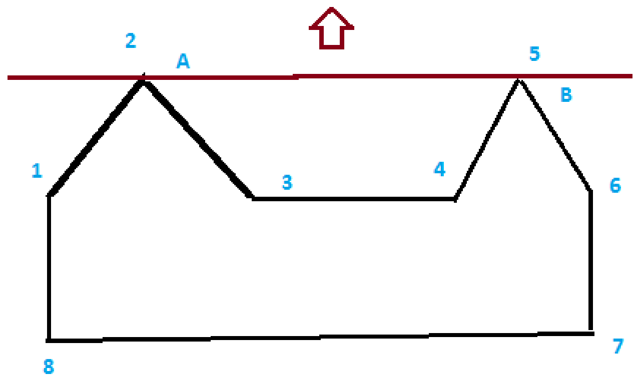
Case5:



a) By above algorithm – Points in the clipped polygon will be

1, A, 3, 4, B, 6, 7, 8 → right polygon

Case6:

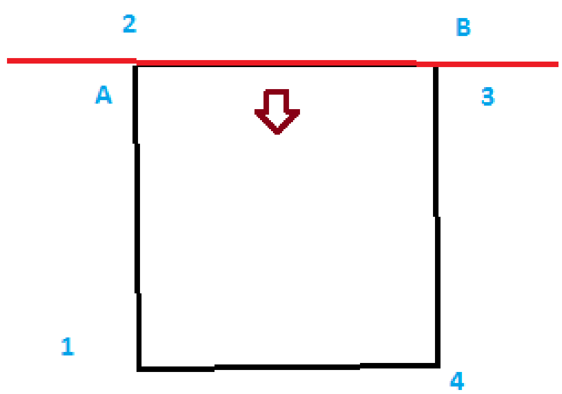


a) By above algorithm – Points in the clipped polygon will be

A, B → which should not happen but the formula for area calculation take cares of that

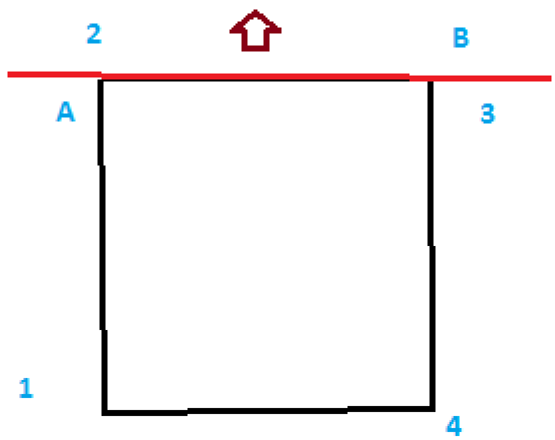
Since two line segements formed are (A, B) and (B, A)→ net area by these two lines is equal to zero

Case7:



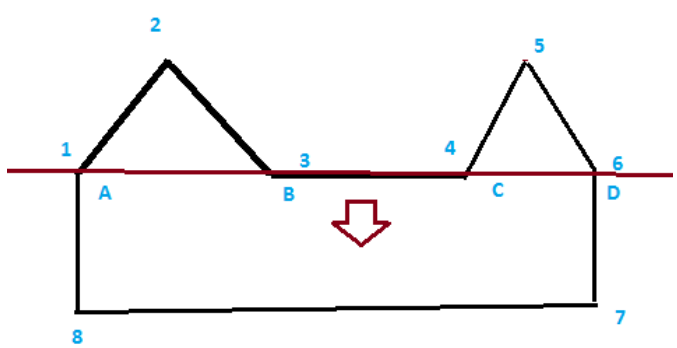
a) 1 inside- added; 1-2 intersects at A; A added
b) 2-3 colinear; 2 is point of intersection; 2==A (last added element); 2 rejected
c) 3 is On; 3 added; 3-4 intersects as B; B==3 (last element added); rejected;
d) 4 is in; 4 added
e) Final vertex list – 1, A, 3 ,4

Case8:



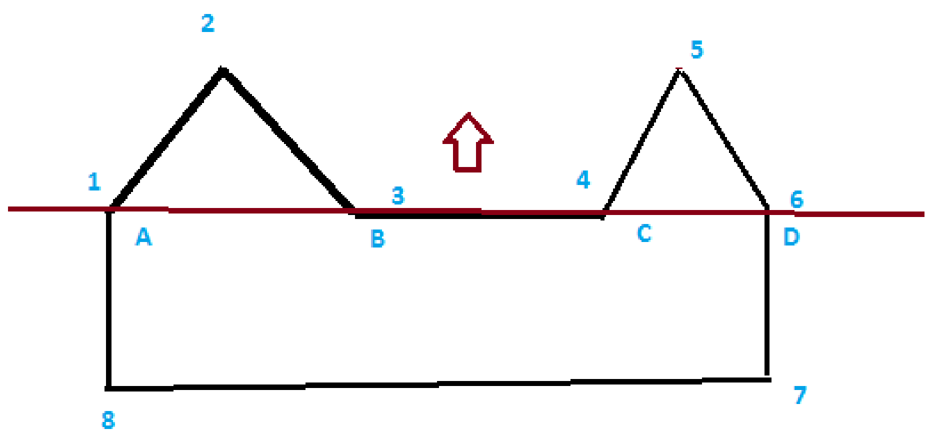
a) 1 outside- rejected; 1-2 intersects at A; A added
b) 2-3 colinear; 2 is point of intersection; 2==A (last added element); 2 rejected
c) 3 is On; 3 added; 3-4 intersects as B; B==3 (last element added); rejected;
d) 4 is out; 4 rejected
e) Final vertex list – A, 3 → segements (A,3) and (3, A)→ area = 0

Case 9:



a) 1 is On; 1 added ; 1-2 intersects at A; A ==1 (last element added); rejected
b) 2 is outside; 2 rejected; 2-3 intersects at B; B is added;
c) 3 is On; 3 == B; 3 rejected; 3-4 intersects as B; already in the list; rejected
d) 4 is on; 4 added; similarly 5 rejected; D is added
e) Final vertex list – 1, B, 4, D, 7, 8

Case10:



a) Final vertex list: 1, 2, B, 4, 5, D → Not a valid polygon but edge D-1 takes care of the area

Complexity of the program:

In worst case each half-plane will be checked for intersection with every other line.

→O(mxn)

Maximum points of intersections (mxn)

Complexity of methods - Point Membership classification, line segment – half plane intersection, area calculation, cross product is constant time for 1 entity

→O (C x m x n) = O (m x n) [for m x n point of intersection]

Order = O(mxn)