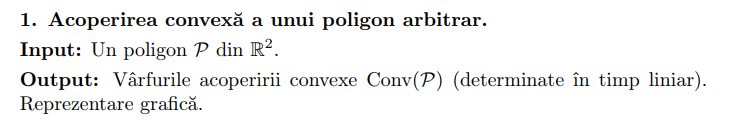


Acoperirea convexă a unui poligon arbitrar

Vârfurile acoperirii convexe (determinate în timp liniar)

Dospra Cristian, Gorneanu Andrei, Lupașcu Marian | Geometrie Computațională | 15-12-17

# Cerință



# Implementare

## Graham’s scan, varianta lui Andrew

Punctele sunt mai înâi sortate (lexicografic, după coordonatele carteziene, prioritar dupa abscisă, iar în cad de abscise egale după ordonată) și renumerotate. Algoritmul determină două liste, reprezentând marginea inferioară și cea superioară a frontierei, pentru a le determina sunt folosite la inițializare punctele P1, P2, respectiv Pn, Pn−1. În final, aceste liste sunt concatenate.

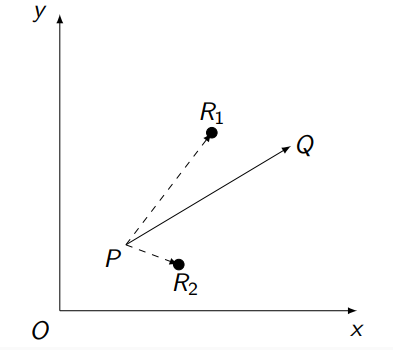
Principiul este asemănător celui de la Graham’s scan: punctele sunt adăugate unul câte unul la listă. Se efectuează testul de orientare pentru ultimele trei puncte și este eliminat penultimul punct, în cazul în care ultimele trei puncte nu generează un viraj la stânga.

## Detalii de implementare

import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.util.ArrayList;  
import java.util.Collections;  
  
import javafx.util.Pair;  
  
//Convex Hull determined using "Graham's scan, Andrew's variant"  
public class ConvexHull {  
  
 private ArrayList<Point> hull; //points on the hull  
 private ArrayList<Pair<Pair<Point, Point>, Boolean>> upperEnvelope; //segmentele ce trebuiesc desenate true = apartine lui hull false altfel  
 private ArrayList<Pair<Pair<Point, Point>, Boolean>> lowerEnvelope;  
  
 public ConvexHull(ArrayList<Point> points) {  
  
 this.hull = new ArrayList<Point>();  
 this.upperEnvelope = new ArrayList<Pair<Pair<Point, Point>, Boolean>>();  
 this.lowerEnvelope = new ArrayList<Pair<Pair<Point, Point>, Boolean>>();  
  
 Collections.*sort*(points, new PointComparator());  
  
 Point[] hull = new Point[2 \* points.size()];  
 Pair<Pair<Point, Point>, Boolean>[] upperEnvelope;  
 Pair<Pair<Point, Point>, Boolean>[] lowerEnvelope;  
 upperEnvelope = new Pair[2 \* points.size()];  
 lowerEnvelope = new Pair[2 \* points.size()];  
  
 //computing upper envelope  
 Integer L = -1;  
 Integer Up = 0, Lo = 0;  
 for (Integer i = 0; i < points.size(); ++i) {  
 while (L > 0 && Point.*determinant*(hull[L - 1], hull[L], points.get(i)) < 0) {  
 upperEnvelope[Up++] = new Pair<Pair<Point, Point>, Boolean>(new Pair<Point, Point>(hull[L - 1], hull[L]), false);  
 L--;  
 }  
  
 hull[++L] = points.get(i);  
 if (L > 0)  
 upperEnvelope[Up++] = new Pair<Pair<Point, Point>, Boolean>(new Pair<Point, Point>(hull[L - 1], hull[L]), true);  
 }  
 upperEnvelope[Up++] = new Pair<Pair<Point, Point>, Boolean>(new Pair<Point, Point>(hull[L], points.get(points.size() - 1)), true);  
 Integer l = L;  
 L--;  
 System.*out*.print(L + " ");  
  
 //computing lower envelope  
 for (Integer i = points.size() - 1; i >= 0; --i) {  
 while (L > l && Point.*determinant*(hull[L - 1], hull[L], points.get(i)) < 0) {  
 lowerEnvelope[Lo++] = new Pair<Pair<Point, Point>, Boolean>(new Pair<Point, Point>(hull[L - 1], hull[L]), false);  
 L--;  
 }  
  
 hull[++L] = points.get(i);  
 if (L > 0)  
 lowerEnvelope[Lo++] = new Pair<Pair<Point, Point>, Boolean>(new Pair<Point, Point>(hull[L - 1], hull[L]), true);  
 }  
  
 lowerEnvelope[Lo++] = new Pair<Pair<Point, Point>, Boolean>(new Pair<Point, Point>(hull[L], points.get(0)), true);  
  
 for (Integer i = 0; i < L; ++i) {  
 this.hull.add(hull[i]);  
 }  
 for (Integer i = 0; i < Up; ++i) {  
 this.upperEnvelope.add(upperEnvelope[i]);  
 }  
 for (Integer i = 0; i < Lo; ++i) {  
 this.lowerEnvelope.add(lowerEnvelope[i]);  
 }  
 }  
  
 public ArrayList<Point> getHull() {  
 return hull;  
 }  
  
 public ArrayList<Pair<Pair<Point, Point>, Boolean>> getUpperEnvelope() {  
 return upperEnvelope;  
 }  
  
 public ArrayList<Pair<Pair<Point, Point>, Boolean>> getLowerEnvelope() {  
 return lowerEnvelope;  
 }  
  
 public void print(String outFile, int offset) throws FileNotFoundException {  
  
 PrintWriter out = new PrintWriter(outFile);  
  
 for (Point point : hull) {  
 out.println(point.forPrint(offset));  
 }  
  
 out.close();  
 }  
  
}

# Chestiuni Matematice

## testul de orientare



Fie P = (p1, p2), Q = (q1, q2) două puncte distincte din planul RxR , fie R = (r1,r2) un punct arbitrar și

Atunci R este situat:

1. pe dreapta PQ ⇔ ∆(P, Q, R) = 0 (”ecuat¸ia dreptei”);
2. în dreapta” segmentului orientat ⇔ ∆(P, Q, R) < 0;
3. în stânga” segmentului orientat ⇔ ∆(P, Q, R) > 0.

Testul de orientare se bazează pe calculul unui polinom de gradul II (∆(P, Q, R)).

# Complexitate

Algoritm specific pentru context 2D. Nu este on-line, fiind nevoie de toate punctele. Complexitatea: O(n log n); spațiu: O(n); complexitate algebrică: polinoame de gradul II. Tratarea cazurilor degenerate: corect. Robustețea: datorită erorilor de rotunjire este posibil ca algoritmul să returneze o listă eronată (dar coerentă) de muchii.