GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: POLÍGONO

1. Introducción

El polígono es una figura geométrica que es una fuente de muchos problemas de geometría (computacionales) en concurso es por eso que vamos a dedicarle una guía a los principales elementos de él que son utilizados como base de diferentes problemas.

2. Conocimientos previos

2.1. Distancia euclidiana

En matemáticas, la distancia euclidiana o euclídea, es la distancia ordinaria entre dos puntos de un espacio euclídeo, la cual se deduce a partir del teorema de Pitágoras.

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

En general, la distancia euclidiana entre los puntos $P = (p_1, p_2, \ldots, p_n)$ y $Q = (q_1, q_2, \ldots, q_n)$, del espacio euclídeo n-dimensional, se define como:

$$d_E(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

2.2. Polígono

Un polígono es una figura plana que está delimitada por un camino cerrado (camino que comienza y termina en el mismo vértice) compuesto por una secuencia finita de segmentos de línea recta. estos segmentos se llaman aristas o lados. El punto donde se unen dos aristas es el vértice o esquina del polígono. Dentro de los elementos que distiguen a un polígono podemos citar:

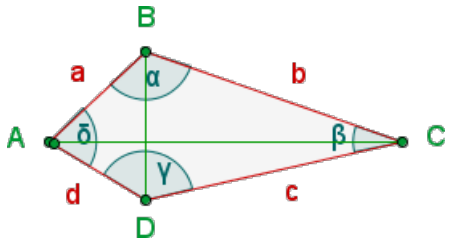


Figura 1: Ejemplo de un polígono

1. **Lados:** Los lados de un polígono son los segmentos que lo limitan.
2. **Vértices:** Los vértices son los puntos donde concurren dos lados. En la figura de arriba, los vértices son los puntos A, B, C, y D.
3. **Ángulos interiores:** Los ángulos interiores son determinados por dos lados consecutivos. En la figura de arriba, tenemos los ángulos α, β, $\bar{o}$, y Y. Para sumar los ángulos interiores de

un polígono,si n es el número de lados, tenemos la siguiente fórmula: Suma de los ángulos interiores de un polígono $= (n-2) \cdot 180$

4. **Diagonales:** Las diagonales son los segmentos que determinan dos vértices no consecutivos. En la figura precedente, tenemos dos diagonales, el segmento que uno los vértices A y C, y el segmento que une los vértices B y D. Para averiguar el número de diagonales de un polígono, si n es el número de lados, podemos usar la siguiente fórmula: Número de diagonales de un polígono $= n \cdot (n-3) \div 2$

3. Desarrollo

3.1. Representación

La forma estándar de representar un polígono es simplemente enumerar los vértices del polígono en sentido horario o antihorario. Pero los vértices como se representa ?. Un vértice nos es mas que un punto en este caso en la plano 2D por tanto se va definir con una valor en la coordenada X y otro valor en la coordenada Y.

Por tanto para representar un polígono primero necesitamos representar un vértice y luego con una colección de vertices podemos representar computacional un polígono.

3.2. Perímetro

El perímetro de un polígono (ya sea convexo o cóncavo) con n vértices dados en algún orden (ya sea en el sentido de las agujas del reloj o en el sentido contrario a las agujas del reloj) se puede calcular a través de la suma de las distancias euclidianas entre todos los pares consecutivos de vértices del polígono.

3.3. Área

Esto es fácil de hacer si pasamos por todos los lados y agregamos áreas trapezoidales limitadas por cada lado y el eje x. El área debe tomarse con señal para que se reduzca el área adicional. Por lo tanto, la fórmula es la siguiente:

$$A = \sum_{(p,q) \in \text{lados}} \frac{(p_x - q_x) \cdot (p_y + q_y)}{2}$$

Otra variante es elegir un punto O arbitrariamente, iterar sobre todos los bordes agregando el área orientada del triángulo formado por el borde y el punto O. Nuevamente, debido al signo de área, se reducirá el área extra. Este método es mejor ya que puede generalizarse a casos más complejos (como cuando algunos lados son arcos en lugar de líneas rectas).

3.4. Comprobar si es convexo

Se dice que un polígono es convexo si cualquier segmento de línea dibujado dentro del polígono no intersecar cualquier lado del polígono. De lo contrario, el polígono se llama cóncavo. En la figura de abajo a la izquierda tenemos un polígono convexo mientras a la derecha tenemos un polígono cóncavo

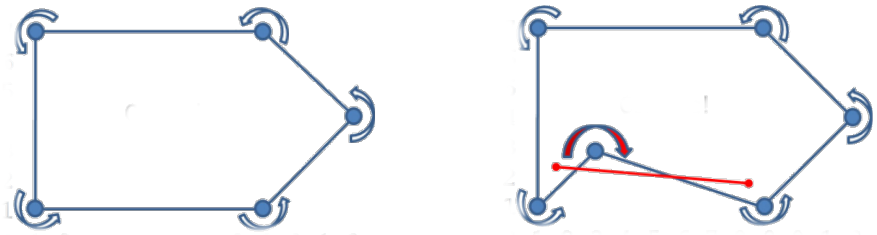
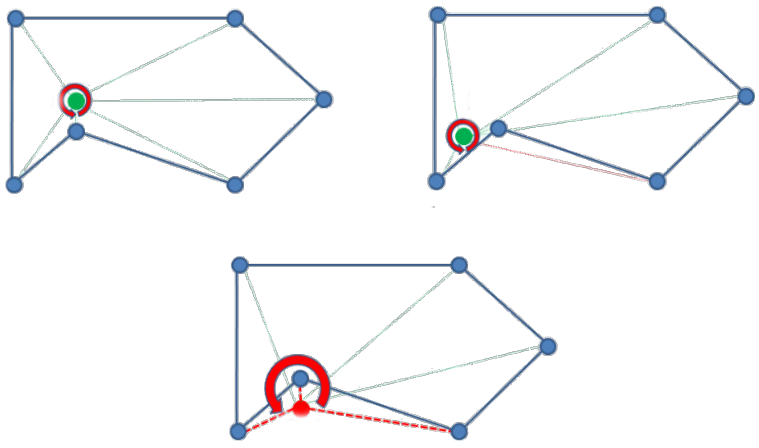


Figura 2:

Sin embargo, para probar si un polígono es convexo, existe un enfoque computacional más sencillo que "tratando de verificar si todos los segmentos de línea se pueden dibujar dentro del polígono". Simplemente podemos comprobar si los tres vértices consecutivos del polígono forman los mismos giros (todos los giros a la izquierda/CCW si los vértices se enumeran en el sentido contrario a las agujas del reloj o todos giran a la derecha/cw si los vértices están enumerados en el sentido de las agujas del reloj). Si podemos encontrar al menos un triple donde esto es falso, entonces el polígono es cóncavo. Es importante notar que los puntos que conforman el polígono nunca debe existir tres vértices consecutivos que sean colineales entre sí.

3.5. Posición de un punto con respecto a un polígono

Otra prueba común que se realiza en un polígono P es verificar si un punto pt está dentro o fuera polígono P. En la imagen se reflejan tres situaciones del problema planteado en las dos primeras el punto esta dentro mientras en el tercer caso esta fuera.



El procedimiento es calculando las sumas de los ángulos entre tres puntos de todas las posibles

tripletas de la forma P[i],pt,P[i+1] donde (P[i]-P[i+1]) son lados consecutivos del polígono P , cuidando los giros a la izquierda (sumar el ángulo) y los giros a la derecha (restar el ángulo) respectivamente. Si la suma final es 2π (360 grados), entonces pt está dentro del polígono P. La otro posición que puede tomar un punto con respecto a un polígono es que este sobre un lado del polígono en este caso va depender de lo que plantee el problema para este caso, en la mayoría de los casos se considera dentro. Para su comprobación basta con cada posible tripleta P[i],pt,P[i+1] comprobar si con colineales si al menos uno cumple el punto está en el borde del polígono.

4. Implementación

4.1. C++

```
#include <math.h>
#define EPS 1e-9
#define LL long long
using namespace std;

struct Point {
   int X, Y;
   Point(int _x = 0, int _y = 0) : X(_x), Y(_y) {}
};

struct Vector {
   double x, y;
   Vector(double _x = 0, double _y = 0) : x(_x), y(_y) {}
};

Vector toVec(Point a, Point b){ return Vector(b.X-a.X, b.Y-a.Y); }
double dist(Point p1, Point p2){ return hypot(p1.X-p2.X, p1.Y-p2.Y); }
double dot(Vector a, Vector b) { return (a.x * b.x + a.y * b.y); }
double norm_sq(Vector v) { return v.x * v.x + v.y * v.y; }
double cross(Vector a, Vector b) { return a.x * b.y - a.y * b.x; }
bool ccw(Point p, Point q, Point r) { return cross(toVec(p, q), toVec(p, r)) >
    0; }

double angle(Point a, Point o, Point b) {
   Vector oa = toVec(o,a), ob =toVec(o, b);
   return acos(dot(oa, ob) / sqrt( norm_sq(oa) * norm_sq(ob) ));
}

struct Polygon {
   vector<Point> points;

   Polygon(vector<Point> _points) { this->points = _points;}

   double perimeter() {
          int sz = (int) points.size();
      double result = 0.0;
```

```
      for(int i=0;i<sz;i++)
         result+=dist(points[i],points[(i+1)%sz]);
      return result;
   }

   double area() {
      double res = 0;
      for (unsigned i = 0; i < points.size(); i++) {
        Point p = i ? points[i - 1] : points.back();
        Point q = points[i];
        res += (p.X - q.X) * (p.Y + q.Y);
      }
      return fabs(res) / 2;
   }

   bool isConvex() {
      int sz = (int) points.size();
      if (sz < 3) return false;
      bool isLeft = ccw(points[0], points[1], points[2]);
      for(int i=1;i<sz;i++)
                if(ccw(points[i],points[(i+1)%sz],points[(i+2)%sz])!=isLeft)
                   return false;
      return true;
   }

   bool inPolygon(Point pt) {
     if ((int) points.size() == 0) return false;
     double sum = 0;
     for (int i = 0; i < (int) points.size() - 1; i++) {
       if(ccw(pt,points[i],points[i+1])) sum+=angle(points[i],pt,points[i+1]);
       else sum-=angle(points[i],pt,points[i+1]);
     }
     return fabs(fabs(sum) - 2 * M_PI) < EPS;
   }
};
```

4.2. Java

```
private class Point {
   int X, Y;
   public Point(int _x, int _y) { X = _x; Y = _y;}
}

private class Vector {
   double x, y;
   public Vector(double _x, double _y) { x = _x; y = _y;}
}
```

```
public Vector toVec(Point a, Point b){return new Vector(b.X-a.X,b.Y-a.Y);}
public double dist(Point p1, Point p2){return Math.hypot(p1.X-p2.X,p1.Y-p2.Y)
    ;}
public double dot(Vector a, Vector b){return (a.x*b.x+a.y*b.y);}
public double norm_sq(Vector v){return v.x*v.x+v.y*v.y;}
public double cross(Vector a, Vector b){return a.x * b.y - a.y * b.x;}
public boolean ccw(Point p, Point q, Point r){return cross(toVec(p,q),toVec(p,
    r))>0;}
public double angle(Point a, Point o, Point b){
   Vector oa = toVec(o, a), ob = toVec(o, b);
   return Math.acos(dot(oa,ob)/Math.sqrt(norm_sq(oa)*norm_sq(ob)));
}

private class Polygon {
   List<Point> points;
   private final double EPS = 1e-9;

   public Polygon(List<Point> _points){this.points = _points;}

   public double perimeter() {
      int sz = (int) points.size(); double result = 0.0;
      for(int i = 0; i < sz; i++)
         result += dist(points.get(i), points.get((i + 1) % sz));
      return result;
   }

   public double area() {
      double res = 0;
      for(int i=0;i<points.size();i++) {
         Point p = i==0 ? points.get(i-1):points.get(points.size()-1);
         Point q = points.get(i);
         res+=(p.X-q.X)*(p.Y+q.Y);
      }
      return Math.abs(res)/2;
   }

   public boolean isConvex() {
      int sz =(int)points.size();
      if (sz < 3) return false;
      boolean isLeft = ccw(points.get(0), points.get(1), points.get(2));
      for(int i = 1; i < sz; i++)
         if(ccw(points.get(i), points.get((i + 1) % sz), points.get((i + 2) %
             sz)) != isLeft)
            return false;
      return true;
   }

   public boolean inPolygon(Point pt) {
      if ((int) points.size() == 0) return false;
          double sum = 0;
```

```
        for (int i = 0; i < (int) points.size() - 1; i++) {
            if(ccw(pt,points.get(i),points.get(i+1)))
                sum+=angle(points.get(i),pt,points.get(i+1));
            else sum-=angle(points.get(i),pt,points.get(i+1));
        }
        return Math.abs(Math.abs(sum) - 2 * Math.PI) < EPS;
    }
}
```

5. Complejidad

Como en cada una de las operaciones vistas anteriormente se debe recorrer todos los vértices que componen el polígono la complejidad de estas implementaciones es O(N) siendo N la cantidad de vértices que conforman el polígono.

6. Aplicaciones

El polígono es una figura geométrica que es una fuente de muchos problemas de geometría (computacionales) como se había mencionado anteriormente no solamente para concursos sino para el desarrollo propio de software. El cálculo de determinadas áreas se basa en la descomposición de poligonos por lo general para dar un resultado bastante a los aproximado.

7. Ejercicios propuestos

A continuación una lista de ejercicios que se pueden resolver aplicando los elementos abordados en esta guía:

- UVA 109 - SCUD Busters
- UVA 634 - Polygon
- UVA 10060 - A hole to catch a man
- UVA 11473 - Campus Roads