GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: REPRESENTACIÓN DE PUNTOS Y INTERSECCIÓN DE SEGMENTOS

1. Introducción

Dentro de las áreas de temáticas que debe dominar un concursante de ICPC y IOI para resolver problemas está la geometría computacional, la cual parte de la geometría clásica impartida en aula pero ahora representada de forma computacional. Es por eso que vamos a partir de dos elementos que son las bases y dan soporte a todos lo que componen la geometría computacional de concursos: puntos y segmentos.

2. Conocimientos previos

2.1. Punto

El punto es la unidad más simple, irreductiblemente mínima, de la comunicación visual es una figura geométrica sin dimensión, tampoco tiene longitud, área, volumen, ni otro ángulo dimensional. No es un objeto físico. Describe una posición en el plano, determinada respecto de un sistema de coordenadas preestablecidas.

2.2. Segmento

Es un fragmento de la recta que está comprendido entre dos puntos, llamados puntos extremos o finales. Así, dado dos puntos A y B, se llama segmento AB a la intersección de la semirrecta de origen A que contiene al punto B con la semirrecta de origen B que contiene al punto A. Los puntos A y B son extremos del segmento y los puntos sobre la recta a la que pertenece el segmento.

2.3. Estructura

Las estructuras (también llamadas **struct**) son una forma de agrupar varias variables relacionadas en un solo lugar. Cada variable en la estructura se conoce como un miembro de la estructura.

3. Desarrollo

3.1. Punto

El punto va estar caracterizado por las coordenadas que componen el espacio en que se encuentran. En el caso de un Punto en un espacio de 2D tiene una coordenada X y otra Y. Si fuera un punto en un espacio de tres dimensiones tendría una tercera coordenada Z.

Para su representación computacional podemos hacer uso de una *struct* o *pair* en el caso de C++ mientras en Java se puede utilizar una clase.

En el caso que sea un punto de 2 dimensiones y sea C++ que utileces tambien puedes apoyarte en la estructura de números complejos que propone C++ para trabajar con estos números en este caso la parte real sería la x mientras la parte compleja sería la y.

3.2. Segmento

El segmento se puede representar utilizando una estructura en caso de C++ y una clase en Java. El segmento puede ser definido por los dos puntos que sirven de extremos.

3.3. Intersección de segmentos

Para analizar este caso vamos a partir que tenemos dos segmentos A y C definidos por los puntos a y b el primero y c y d el segundo. Para que los segmentos A y C se intersecten una de las siguientes condiciones tiene que cumplirse o ser verdadera.

1. Los puntos a, b y c ser colineales y c estar entre a y b.
2. Los puntos a, b y d ser colineales y d estar entre a y b.
3. Los puntos c, d y a ser colineales y a estar entre c y d.
4. Los puntos c, d y b ser colineales y b estar entre c y d.
5. Los puntos c y d estar en lados opuestos con respecto al segmento A y los puntos a y b estar en lados opuestos con respecto al segmento C.

Una vez definida las condiciones que harían que los segmentos A y C se intersecten, nos podemos percatar que que para primera cuatro condiciones son identicas solo cambian los puntos o su orden. Mientras en la quinta se comprueba en un sentido y luego en el otro. Es por eso que vamos centrarnos en como determinar:

- Dados tres puntos saber son colineales.
- Saber si un punto está entre otros dos.
- Dado un punto saber de que lado esta con respecto a un segmento definido por dos puntos.

3.4. Dados tres puntos saber son colineales

Dos puntos siempre van ser colineales porque es la mínima cantidad de puntos para definir un línea. Ahora tres o más puntos van ser colineales si se es capaz de trazar una línea recta entre los dos puntos extremos y que pase por el resto de los puntos.

Para determinar si los puntos e, f y g son colineales vamos construir dos vectores $\overrightarrow{F}$ y $\overrightarrow{G}$ que tengan como punto de origen el punto e y como de destino los puntos f y g respectivamente. Sin importar las posiciones de los puntos el ángulo entre los dos vectores siempre va estar en el rango de 0 a 180 grados. Precisamente con esa amplitudes sería los casos en que los puntos serían colineales. Bueno entonces queda ver como hallar el ángulo entre dos vectores y luego ver si esa amplitud es 0 ó 180 entonces los puntos son colineales.

Con la operación binaria producto vectorial o producto cruz entre dos vectores es posible determinar el valor del seno del ángulo comprendido entre los vectores segun la siguiente expresión:

$$\overrightarrow{F} \mathrm{x} \overrightarrow{G} = (|\overrightarrow{F}||\overrightarrow{G}| \sin\theta)\hat{n}$$

donde $\hat{n}$ es el vector unitario y ortogonal a los vectores $\overrightarrow{F}$ y $\overrightarrow{G}$ y su dirección está dada por la regla de la mano derecha y θ es, como antes, el ángulo entre a y b. A la regla de la mano derecha se la llama a menudo también regla del sacacorchos. El valor de θ para nuestro caso será 0 o 180 (0 o π) y en ambos caso la expresión $\sin\theta$ se hace cero y esto produce que el miembro derecho de la

expresión sea cero.

$\overrightarrow{F}\text{x}\overrightarrow{G} = 0$

Esto conduce a plantaer que tres puntos son colineales si el producto vectorial o cruz entre dos de los vectores que se construir con ellos es igual cero. Desarrollando el miembro izquierdo de la expresión:

$$((Pe_x - Pf_x) * (Pg_y - Pf_y) - (Pe_y - Pf_y) * (Pg_x - Pf_x)) = 0$$

3.5. Saber si un punto está entre otros dos puntos

Determinar si un punto *c* esta entre los puntos *a* y *b* es bastante sencillo. Solo se debe cumplir las siguientes condiciones.

- min(a_x,b_x) $<= c_x$
- min(a_y,b_y) $<= c_y$
- max(a_x,b_x) $>= c_x$
- max(a_y,b_y) $>= c_y$

Donde *min* y *max* son funciones que devuelven el mínimo y máximo respectivamente entre los valores pasado por argumentos.

3.6. Dado un punto saber de que lado esta con respecto a un segmento definido por dos puntos

Como saber que posición ocupa el punto *c* o de que lado está con respecto al segmento delimitado por los puntos *a* y *b*. Anteriormente vimos como saber si tres puntos son colineales, pues bien el análisis para resolver esta nueva problemática es identica a la de los tres puntos colineales. Una vez hecho el mismo análisis nos podemos percatar que cuando se cumpla la siguiente expresión:

$\overrightarrow{F}\text{x}\overrightarrow{G} > 0$

El punto se encuentra a la derecha del segmento.

4. Implementación

4.1. C++

```
#include <stdio.h>
```

```
#include <math.h>
#include <iostream>
#include <algorithm>
#include <complex>
using namespace std;
typedef long long lld;

struct Point{
   double X, Y;
   Point(double x, double y){
      this.X = x;
      this.Y = y;
   }
};

inline double crossProduct(Point a, Point b, Point c){
   return ((b.X - a.X) * (c.Y - a.Y) - (b.Y - a.Y) * (c.X - a.X));
}

inline bool isLeft(Point a, Point b, Point c){
   return (crossProduct(a, b, c) > 0);
}

inline bool isCollinear(Point a, Point b, Point c){
   return (crossProduct(a, b, c) == 0);
}

inline bool oppositeSides(Point a, Point b, Point c, Point d){
   return (isLeft(a, b, c) != isLeft(a, b, d));
}

inline bool isBetween(Point a, Point b, Point c){
   return (min(a.X, b.X) <= c.X && c.X <= max(a.X, b.X) && min(a.Y, b.Y) <= c.
      Y && c.Y <= max(a.Y,b.Y));
}

inline bool intersect(Point a, Point b, Point c, Point d){
   if (isCollinear(a, b, c) && isBetween(a, b, c)) return true;
   if (isCollinear(a, b, d) && isBetween(a, b, d)) return true;
   if (isCollinear(c, d, a) && isBetween(c, d, a)) return true;
   if (isCollinear(c, d, b) && isBetween(c, d, b)) return true;
   return (oppositeSides(a, b, c, d) && oppositeSides(c, d, a, b));
}

double dist(Point A, Point B) {
   return hypot(A.X - B.X, A.Y - B.Y);
}

int main(){
   Point A(0.0, 0.0), B(0.0, 2.0), C(-1.0, 2.0), D(1.0, 2.0);
```

```
    printf(intersect(A,B,C,D) ? "YES" : "NO");
    printf("\n");
    return 0;
}
```

4.2. Java

```
private class Point{
    public double X;
    public double Y;

    public Point(double x, double y){
        this.X = x;
        this.Y = y;
    }
}

public double crossProduct(Point a, Point b, Point c){
    return ((b.X - a.X) * (c.Y - a.Y) - (b.Y - a.Y) * (c.X - a.X));
}

public boolean isLeft(Point a, Point b, Point c){
    return (crossProduct(a, b, c) > 0);
}

public boolean isCollinear(Point a, Point b, Point c){
    return (crossProduct(a, b, c) == 0);
}

public boolean oppositeSides(Point a, Point b, Point c, Point d){
    return (isLeft(a, b, c) != isLeft(a, b, d));
}

public boolean isBetween(Point a, Point b, Point c){
    return (Math.min(a.X, b.X) <= c.X && c.X <= Math.max(a.X, b.X) && Math.min(
        a.Y, b.Y) <= c.Y && c.Y <= Math.max(a.Y,b.Y));
}

public boolean intersect(Point a, Point b, Point c, Point d){
    if (isCollinear(a, b, c) && isBetween(a, b, c)) return true;
    if (isCollinear(a, b, d) && isBetween(a, b, d)) return true;
    if (isCollinear(c, d, a) && isBetween(c, d, a)) return true;
    if (isCollinear(c, d, b) && isBetween(c, d, b)) return true;
    return (oppositeSides(a, b, c, d) && oppositeSides(c, d, a, b));
}

public double dist(Point A, Point B) {
    return Math.hypot(A.X- B.X, A.Y - B.Y);
```

```
}
```

5. Complejidad

Como podemos la implementación cuenta con una estructura o clase de tipo *Point* que servirá para almacenar los puntos que definen los segmentos. Este algoritmo como solo cuenta con operaciones elementales por lo que su complejidad es O(1).

6. Aplicaciones

Con el algoritmo de intersección de segmentos se abordan ciertos elementos que son utilizados en la implementación de otros algoritmos dentro de la geometría computacional como son:

- Posición de un punto con respecto a un segmento.
- Colinealidad de tres puntos

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven utilizando los conceptos anteriormente vistos:

- DMOJ - José el mago
- DMOJ - Juego de Líneas.
- MOG - H - Colors I
- DMOJ - Arroz curioso