

Geometría Computacional



Román Castellarin

Geometría: ¿por qué?

La geometría es una rama divertida de la matemática.

Las competencias de programación son divertidas...

¡bam!, la geometría puede aparecer (quizás oculta) en los problemas.

En lugar de trabajar con números, trabajamos con entidades geométricas...

- puntos
- rectas
- polígonos
- círculos
- etc...

Geometría

Existen muchos tipos de geometría, en la OIA (e IOI) sólo se considera la geometría común en el plano con coordenadas enteras.

¡Esto simplifica mucho los cálculos involucrados!

Algunos problemas geométricos...

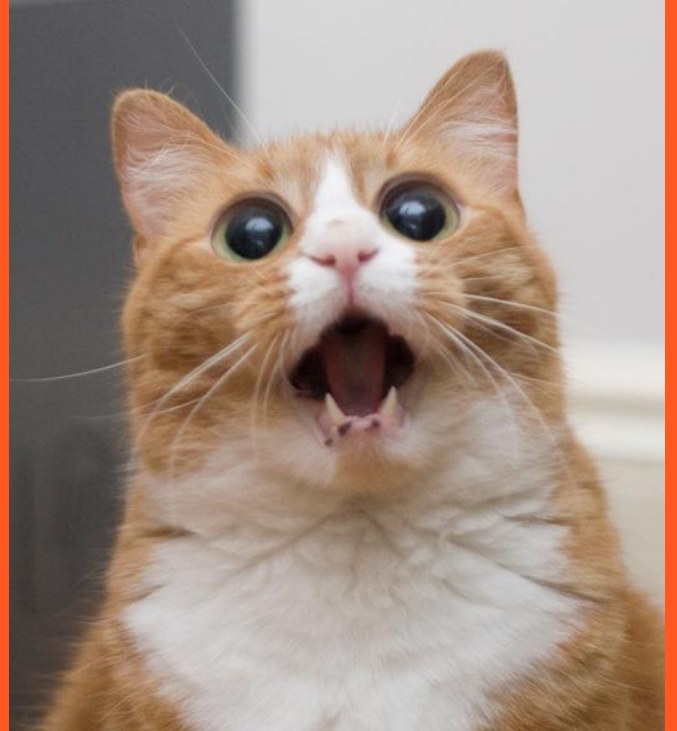
- Calcular áreas/perímetros de polígonos
- Calcular intersecciones de segmentos
- Verificar si un punto es interior a un polígono o círculo
- etc...

Entidades Geométricas

Vector

(& punto)

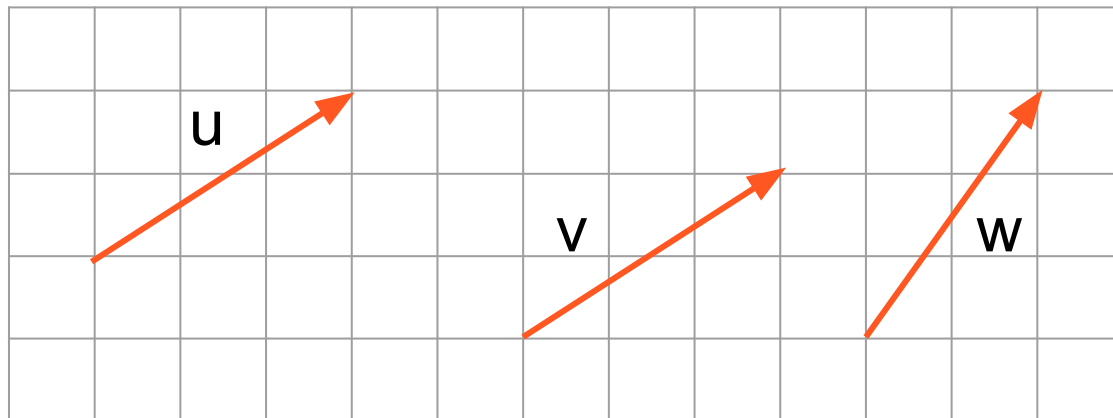
*gato luego de aprender
vectores*



Vector

Un vector es una flechita en el plano, tiene una dirección, un sentido, y un largo, pero no tiene una posición fija.

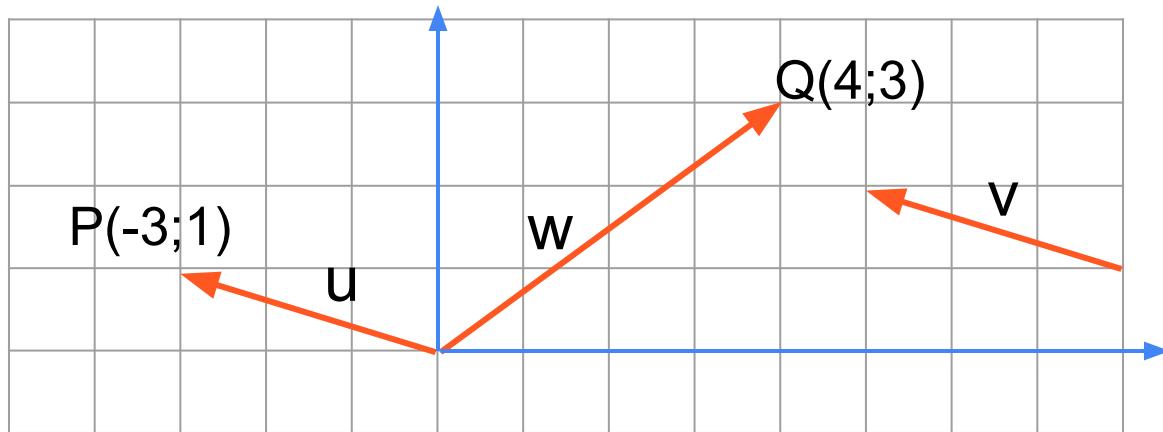
En el ejemplo, $u = v \neq w$.



Punto

Un punto en el plano puede representarse por un vector cuyo origen se dibuja en el centro de coordenadas. *(en el dibujo $P = u$, $Q = w$)*

Dicho esto, representamos los vectores así: $u = v = (-3, 1)$ $w = (4, 3)$

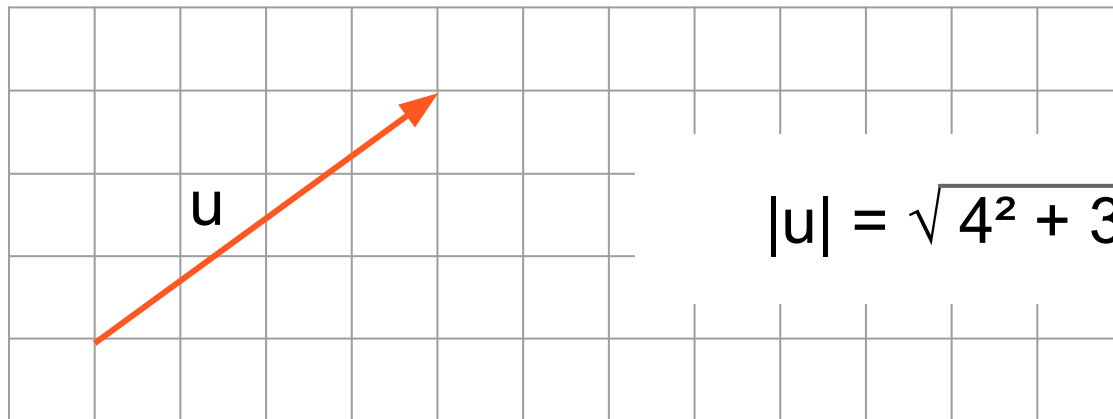


Longitud de un vector

La longitud de un vector (llamada norma) se puede calcular utilizando Pitágoras sobre sus componentes...

Dado $u = (x, y)$, su longitud es $|u| = \sqrt{x^2 + y^2}$

jusualmente no es un número entero!



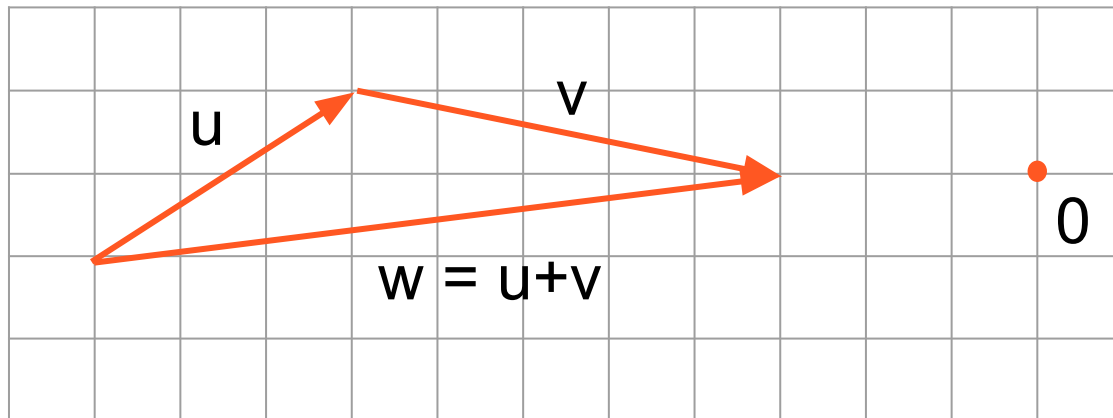
$$|u| = \sqrt{4^2 + 3^2} = 5$$

Suma y resta de vectores

La suma y resta de vectores se hace componente a componente:

$$u = (3, 2) \quad v = (5, -1) \Rightarrow u+v = (3+5, 2+(-1)) = (8, 1)$$

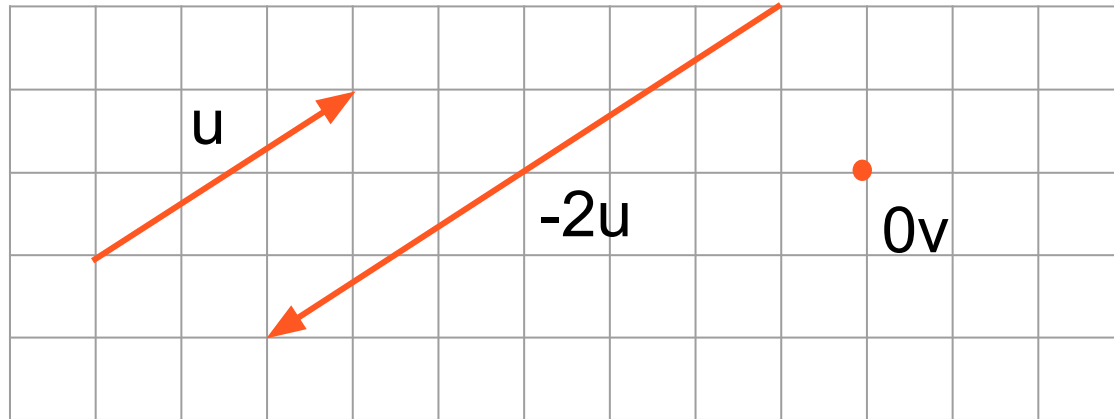
Notemos que como $w = u+v$, entonces $v = w-u$. El vector nulo 0 no tiene largo.



Multiplicación de un vector por un número

Un vector se puede multiplicar por un entero, y su longitud varía en dicho factor, si el entero es negativo, su sentido se invierte.

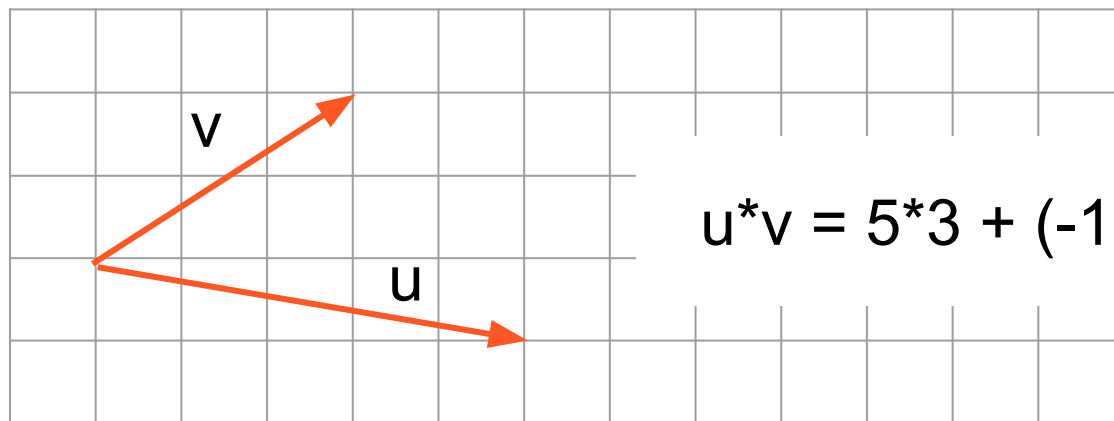
$$u = (x, y) \Rightarrow ku = (kx, ky)$$



Producto escalar

El producto escalar de u con v es un número que se nota $u \cdot v$.

Dados $u = (u_x, u_y)$ y $v = (v_x, v_y) \Rightarrow u \cdot v = u_x v_x + u_y v_y = |u| \cdot |v| \cdot \cos \angle(u, v)$



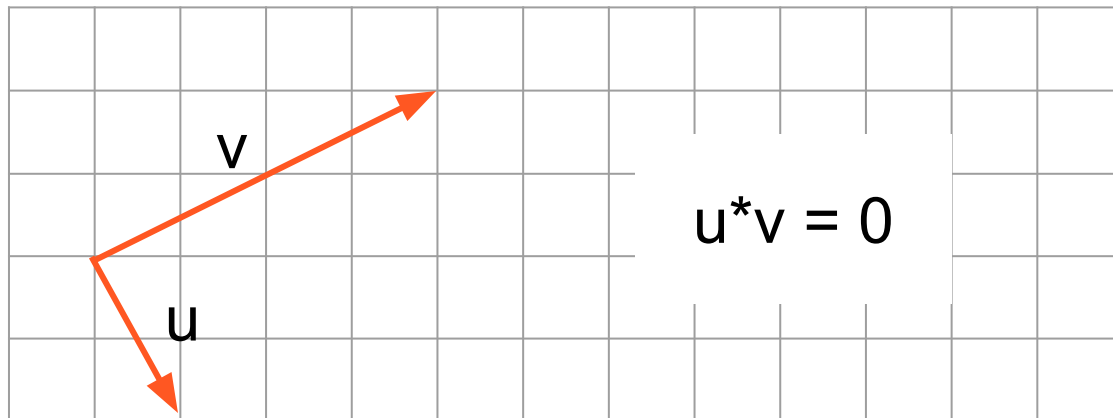
Perpendicularidad de vectores

Si u y v son no nulos, $u^*v = 0 \Leftrightarrow u \perp v$

Adicionalmente,

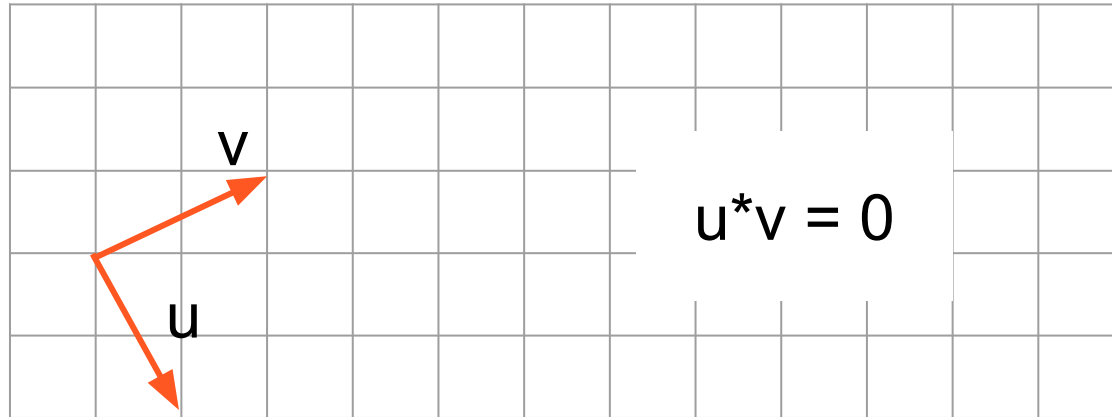
$u^*v < 0 \Rightarrow$ mismo semiplano

$u^*v > 0 \Rightarrow$ distinto semiplano



Rotación 90° antihoraria

Dado $u = (x, y)$, el vector $v = (-y, x)$ se obtiene de rotar u 90° antihorariamente.
Es claro que u y v son perpendiculares ya que $u \cdot v = 0$

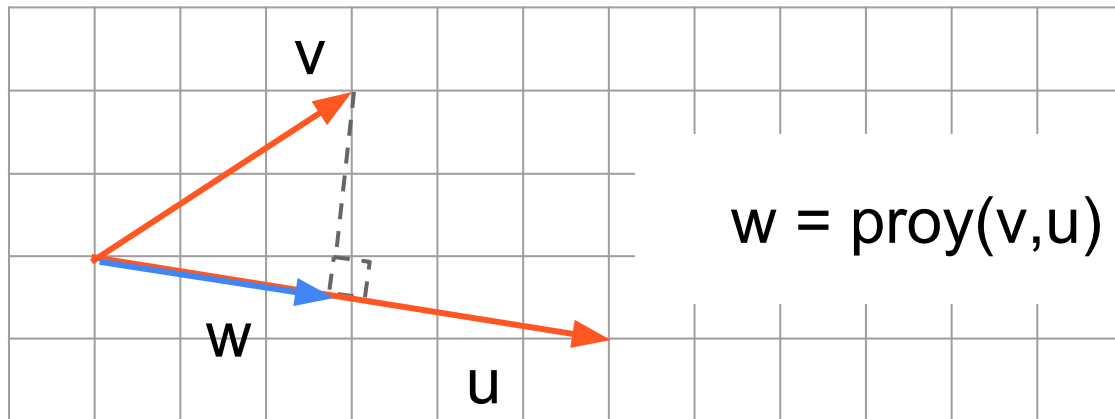


Proyección de un vector sobre otro

Dados u y v , decimos que w es la proyección de v sobre u , si es "la sombra".

Se obtiene $w = \frac{u \cdot v}{|u|^2} u$.

¡Las proyecciones rara vez resultan en componentes enteras!

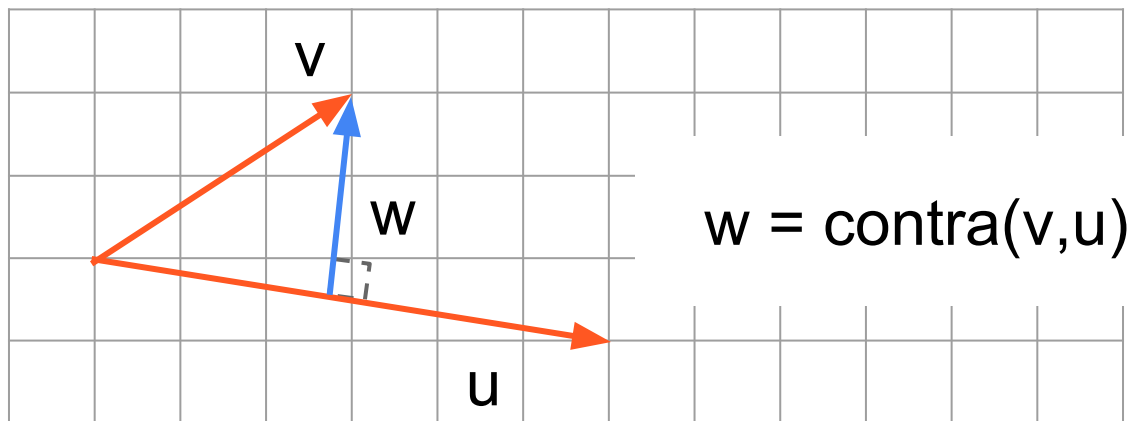


Contraproyección de un vector sobre otro

La contraproyección de v sobre u es la diferencia entre v y $\text{proy}(v,u)$.

$$\text{contra}(v,u) = v - \text{proy}(v,u)$$

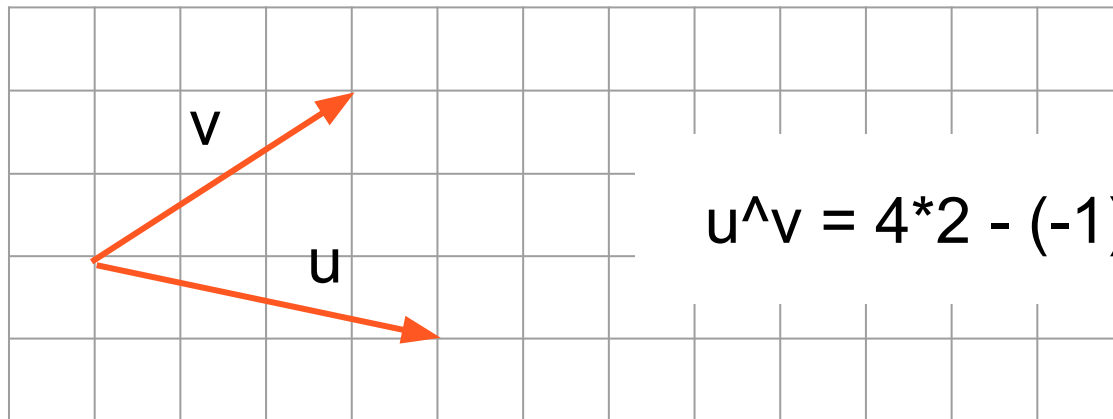
¡Las contraproyecciones rara vez resultan en componentes enteras!



Producto vectorial (o cruzado)

El producto vectorial de u con v es un número que se nota $u^{\wedge}v$.

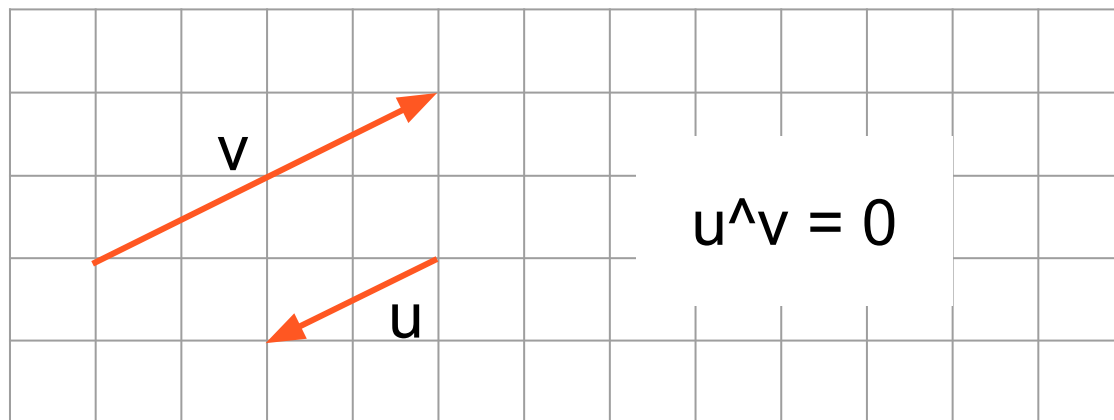
Dados $u = (u_x, u_y)$ y $v = (v_x, v_y) \Rightarrow u^{\wedge}v = u_x v_y - u_y v_x = |u| \cdot |v| \cdot \sin \angle(u,v)$



$$u^{\wedge}v = 4 \cdot 2 - (-1) \cdot 3 = 11$$

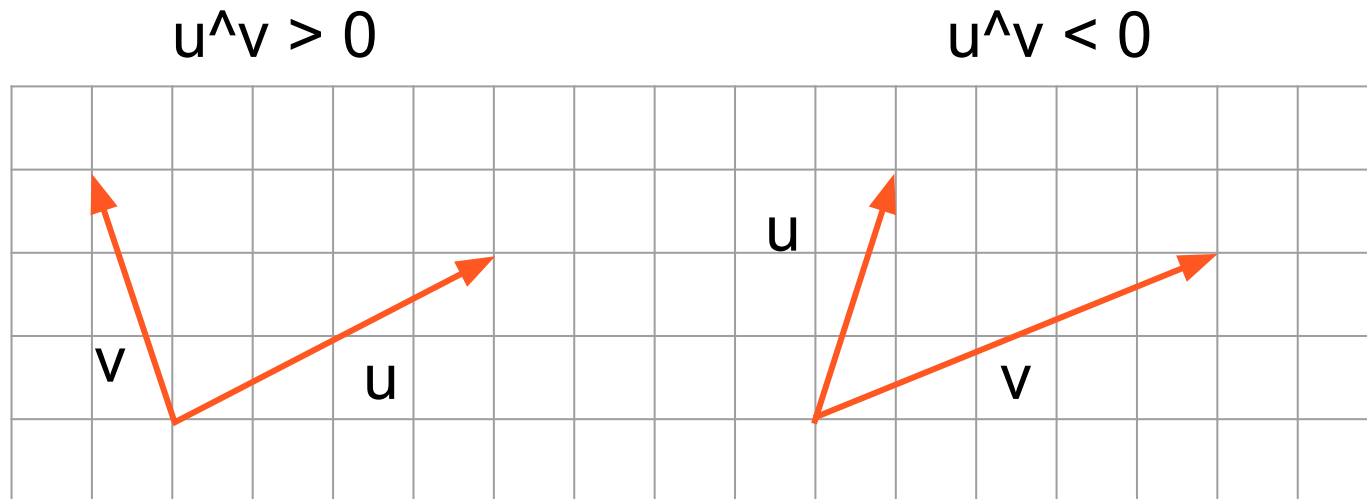
Paralelismo de vectores

Si u y v son no nulos, $u \wedge v = 0 \Leftrightarrow u \parallel v$



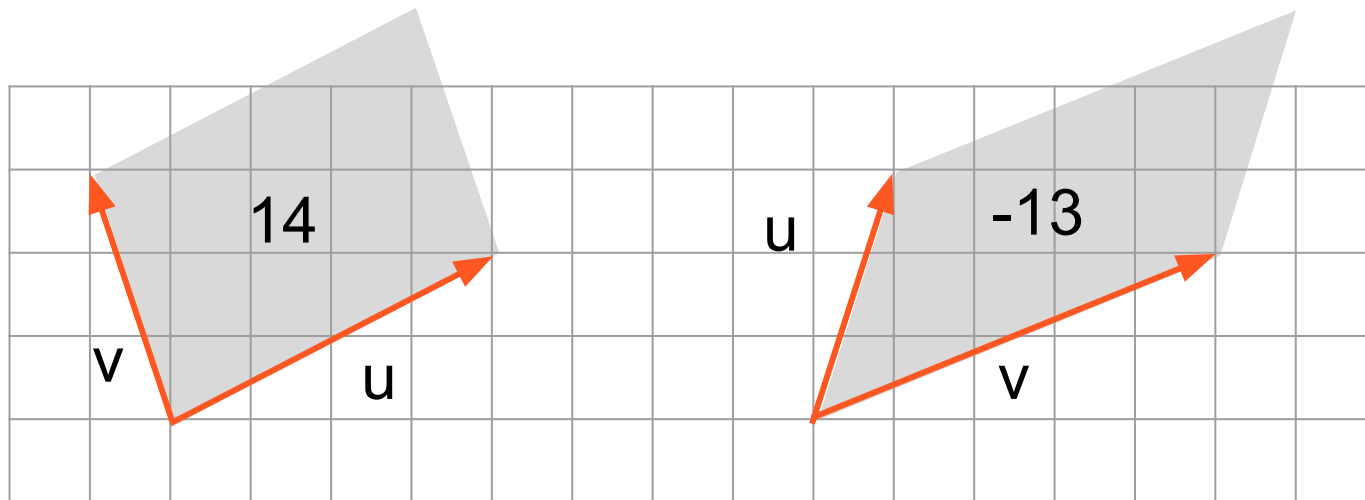
Giro entre vectores

Es más, $u^{\wedge}v < 0 \Leftrightarrow \vec{uv}$ es un giro horario. (Regla de la mano derecha)



Área signada de un paralelogramo

$u \wedge v$ es el área (con signo) del paralelogramo determinado por u y v .
El signo está dado por el tipo de giro.

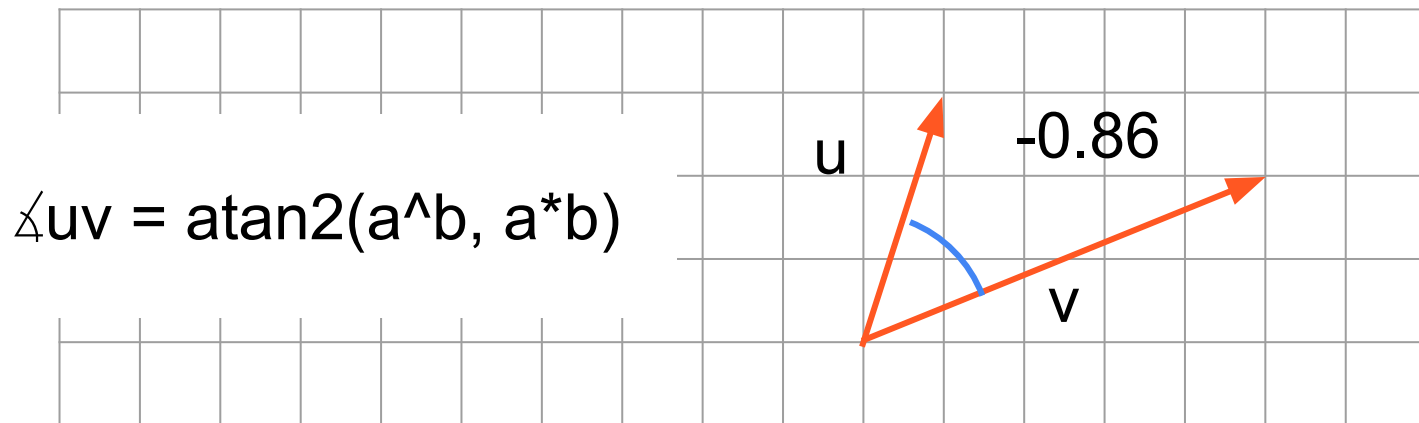


Ángulo signado entre vectores

- Consideramos sólo el ángulo más chico entre dos vectores u y v .
- El sentido positivo es el antihorario.
- Recordemos que trabajamos en radianes ($2\pi = 360^\circ$).
- C++ tiene una función muy precisa (`atan2`) que dado el seno y el coseno de un ángulo, nos devuelve el valor.
 - si multiplicamos el seno y el coseno por el mismo valor, el ángulo no cambia, ya que $\frac{\sin(x)}{\cos(x)} = \frac{k * \sin(x)}{k * \cos(x)}$
- u^v es proporcional al seno. ($k = |u| \cdot |v|$)
- $u \cdot v$ es proporcional al coseno. ($k = |u| \cdot |v|$)

Ángulo signado entre vectores

Finalmente...



Implementación

```
struct punto{  
    ll x, y;  
};
```

¡Usamos long long para evitar overflow!
typedef long long ll;

```
punto operator+(punto a, punto b){  
    return {a.x + b.x, a.y + b.y};  
}
```

Suma

```
punto operator*(ll k, punto a){  
    return {k*a.x, k*a.y};  
};
```

Multiplicación por un
número

Implementación

```
ll operator*(punto a, punto b) {  
    return a.x*b.x + a.y*b.y;  
}  
  
ll operator^(punto a, punto b) {  
    return a.x*b.y - a.y*b.x;  
}  
  
ll norma2(punto a) {  
    return a*a;  
}
```

Producto escalar

Producto vectorial

Longitud al cuadrado
(así siempre es un entero)

Implementación

```
double angulo(punto a, punto b) {  
    return atan2(a^b, a*b);  
}  
  
double norma(punto a) {  
    return hypot(a.x, a.y);  
}
```

Ángulo en radianes

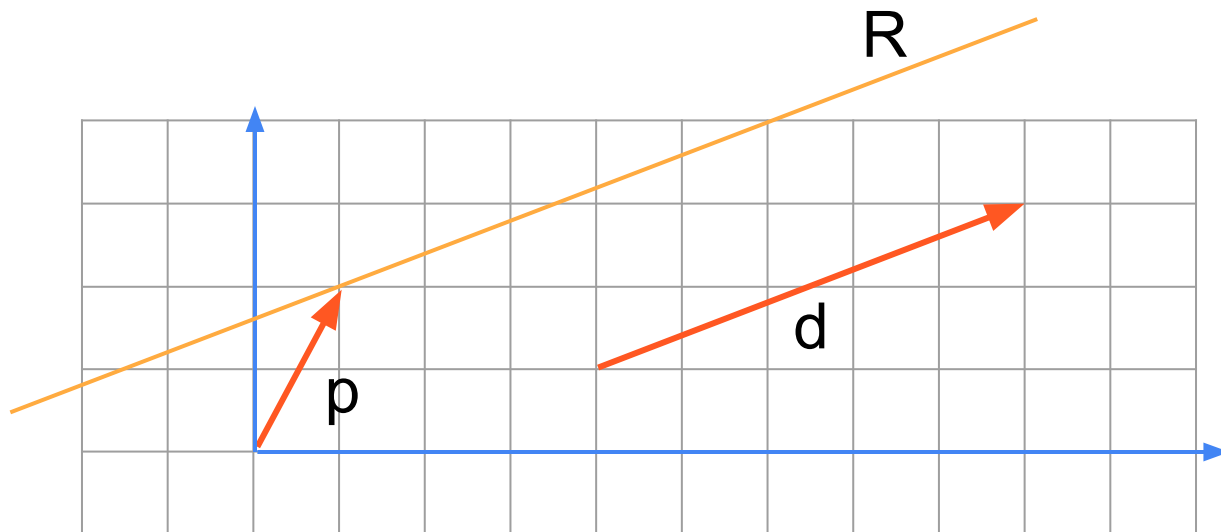
Longitud

Recta

(& Segmento)

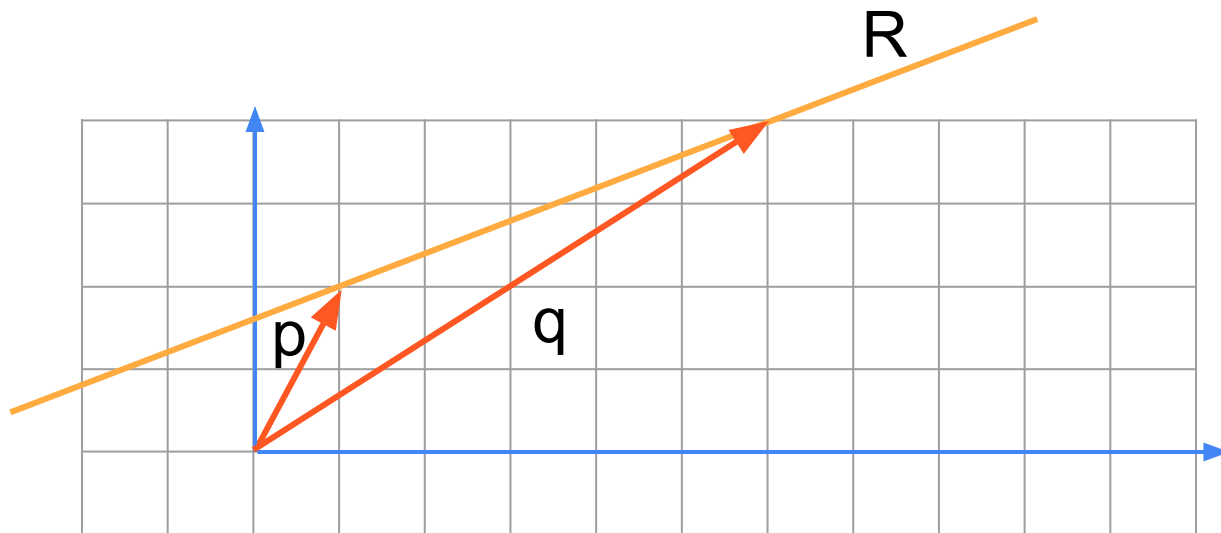
Recta

Una recta R se puede describir mediante un punto de paso p , y una dirección d .



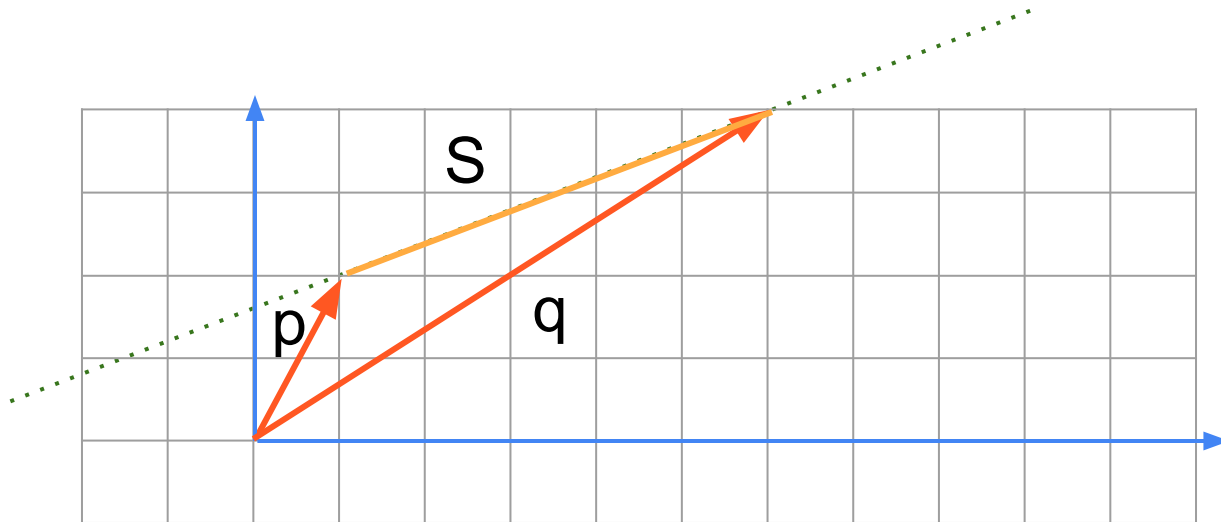
Recta

Dado dos puntos p y q que definen R , podemos tomar p como punto de paso, y $q-p$ como dirección:



Segmento

Dado dos puntos p y q , podemos definir un segmento, como los puntos de la recta pq entre p y q .

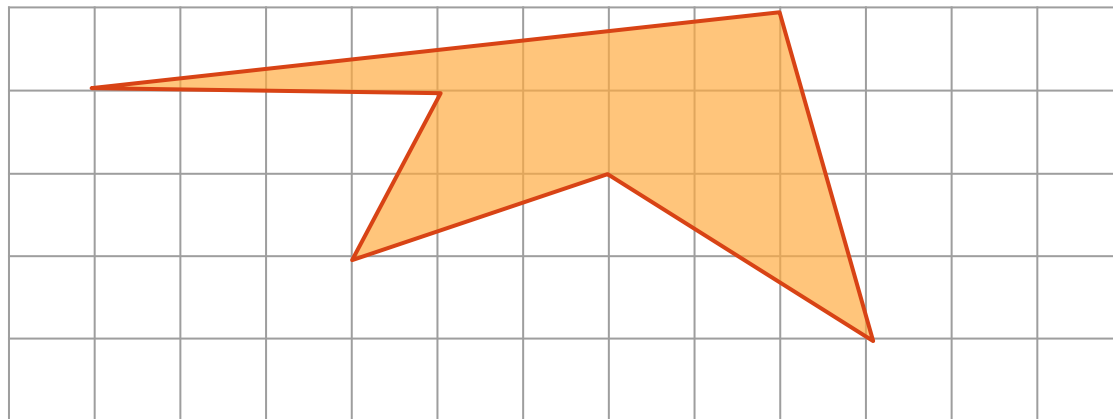


Entidades más complejas

Polígono

Un polígono es una secuencia cerrada de segmentos.

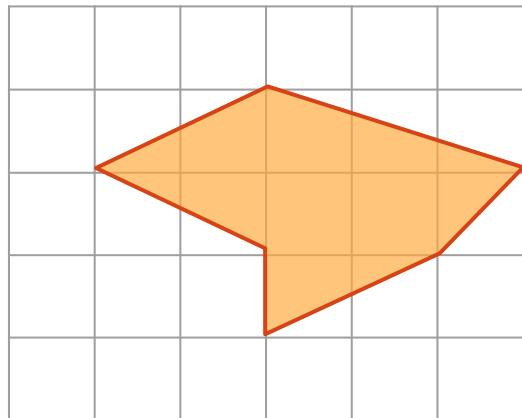
Podemos representarlos por `typedef vector<punto> poligono;`



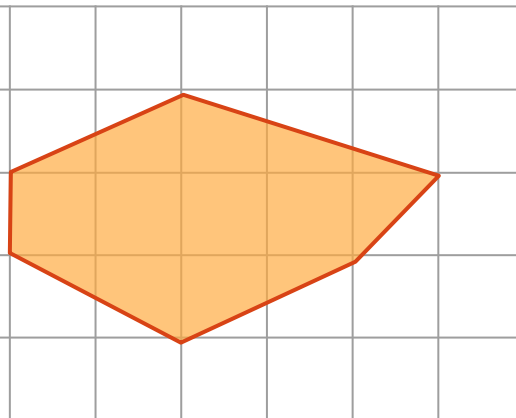
Convexidad

Un polígono es convexo si sus ángulos interiores son no-mayores 180° .
Es estrictamente convexo si sus ángulos interiores son menores a 180° .

no-convexo



convexo



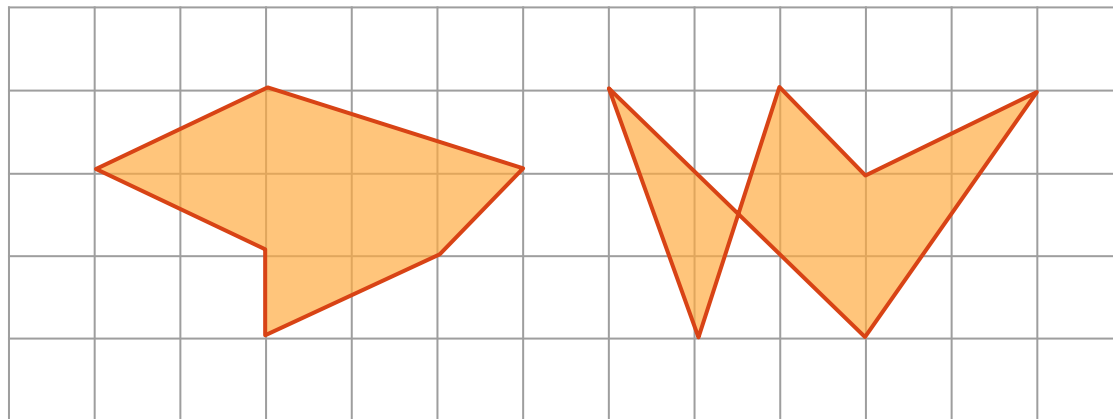
Simplicidad

Un polígono es simple si no se corta a sí mismo.

Si no se aclara lo contrario, siempre los supondremos simples.

simple

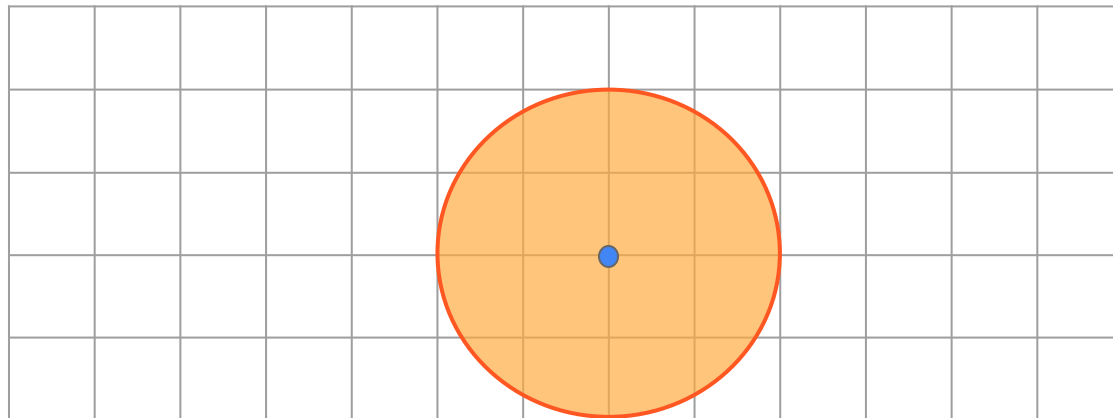
complejo



Círculo

Un círculo puede ser representado por un centro y un radio.

```
struct circulo{  
    punto o; int r;  
};
```

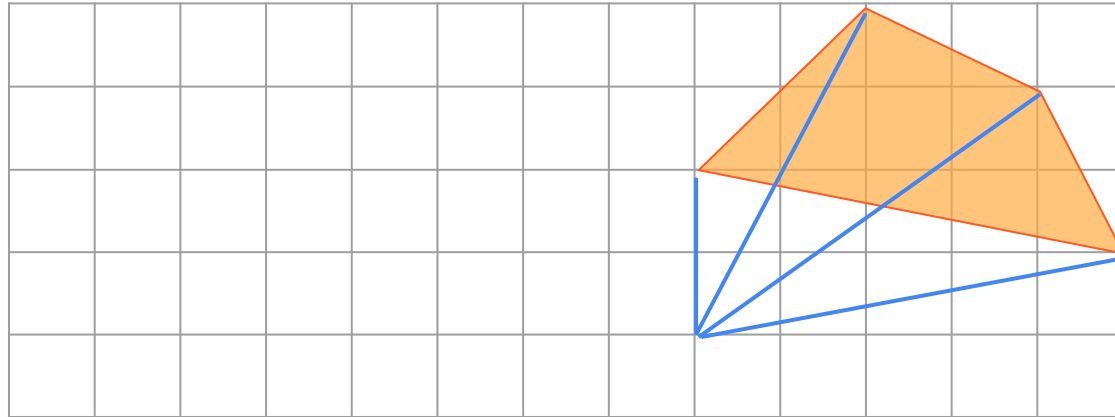


Algoritmos

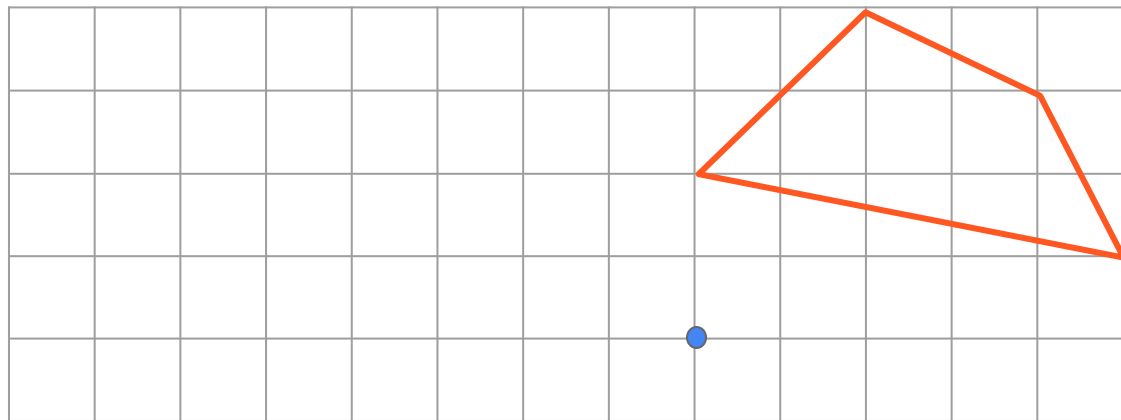
Área & perímetro de un polígono

Área

Medio producto vectorial nos da el área signada del triángulo determinado por el centro de coordenadas y dos vértices consecutivos del polígono.

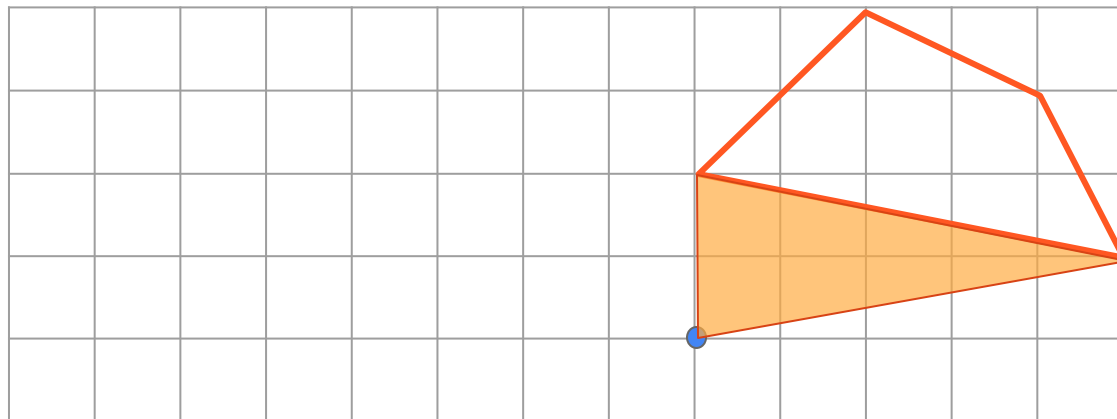
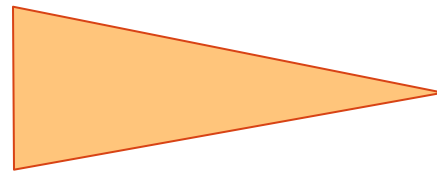


Área



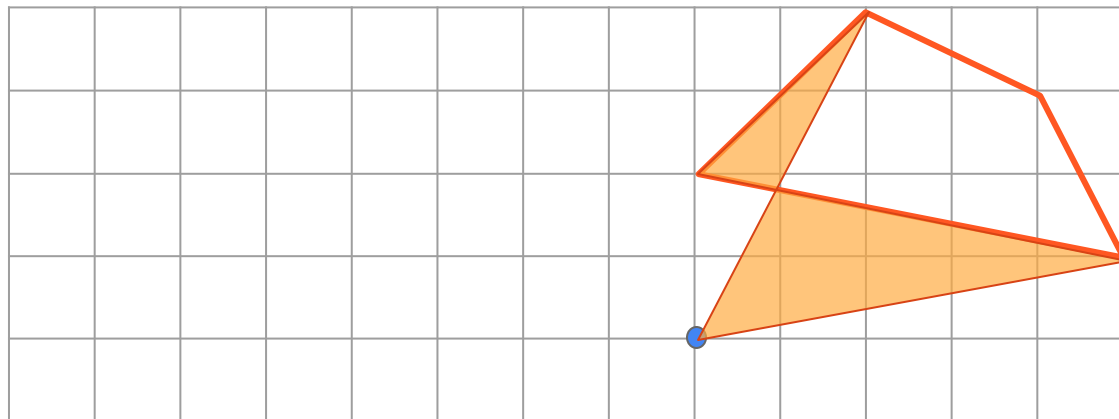
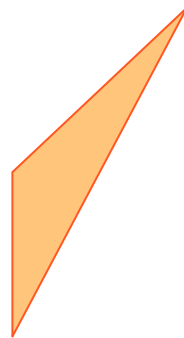
Área

Sumando:
signo (+)



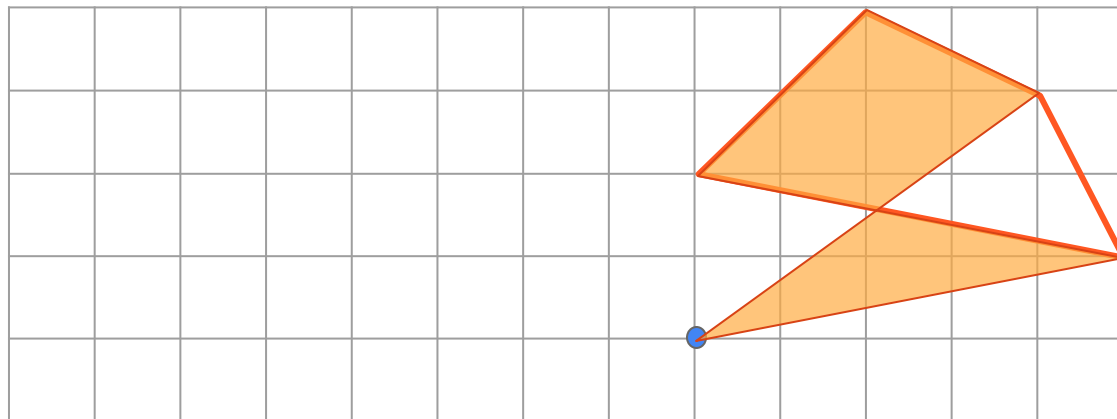
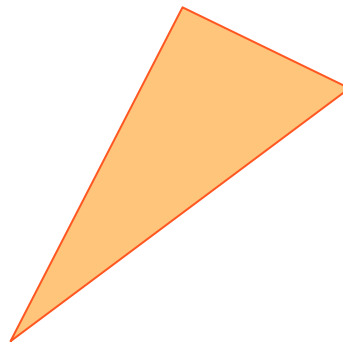
Área

Sumando:
signo (-)



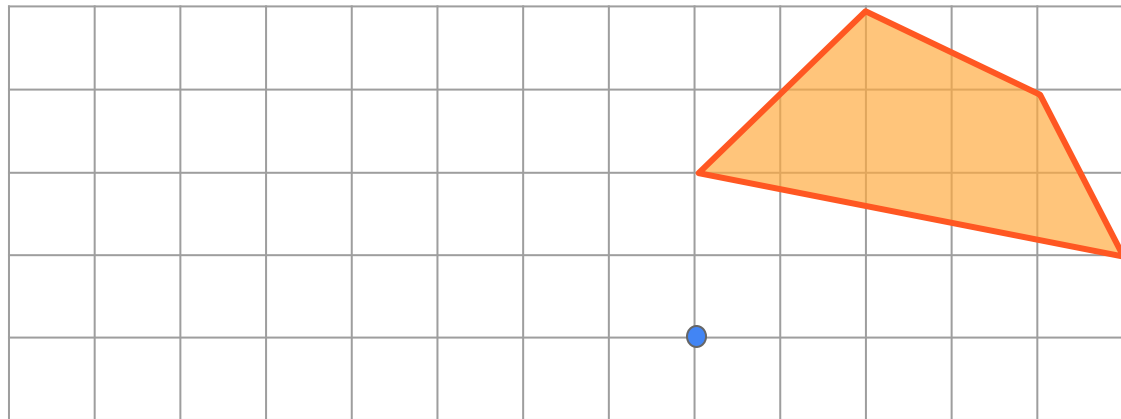
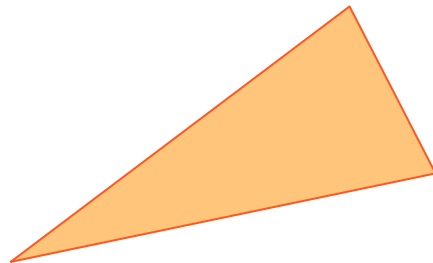
Área

Sumando:
signo (-)



Área

Sumando:
signo (-)



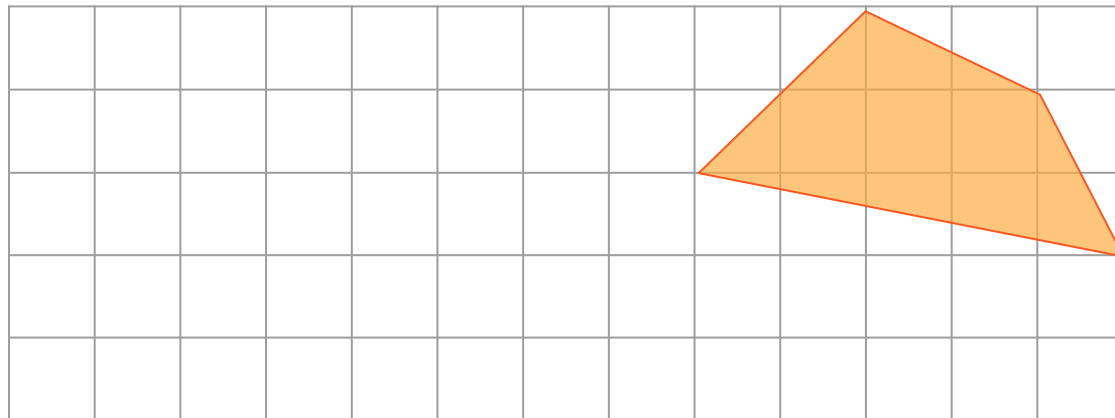
Área

Si queremos, podemos devolver el doble del área para trabajar siempre con enteros. Recordar siempre poner el valor absoluto.

```
ll doble_area(poligono &p) {  
    ll a = 0;  
    forn(i, p.size()) {  
        a += p[i]^p[(i+1)%p.size()];  
    }  
    return a > 0 ? a : -a;  
}
```

Perímetro

El perímetro se obtiene simplemente sumando los largos de los vectores determinados por los lados del polígono.



Perímetro

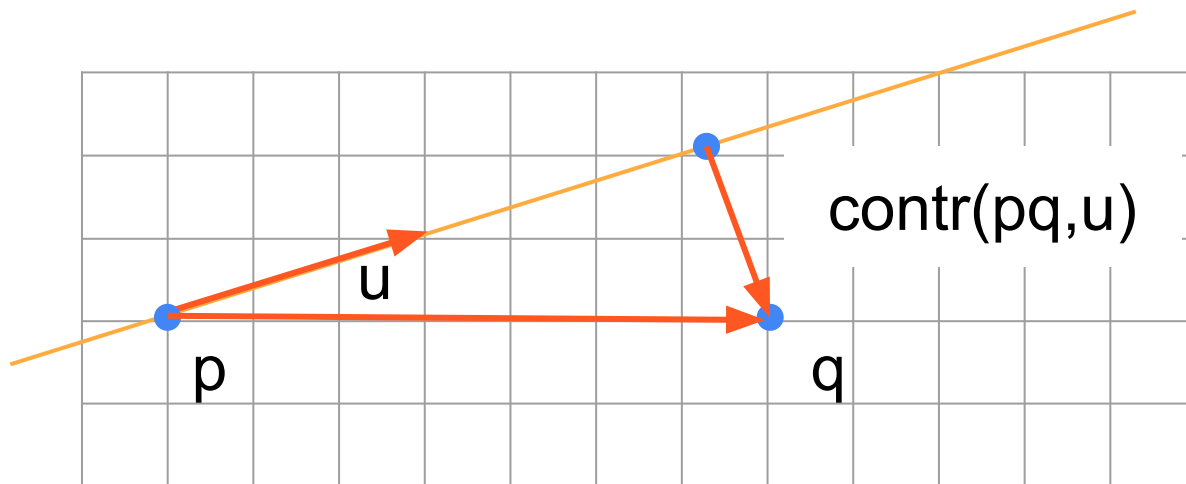
Obligatoriamente tendremos que trabajar con floats...

```
double perimetro(poligono &p) {  
    double l = 0;  
    forn(i, p.size()) {  
        l += norma(p[i] - p[(i+1)%p.size()]);  
    }  
    return l;  
}
```

Distancias

Distancia punto a recta

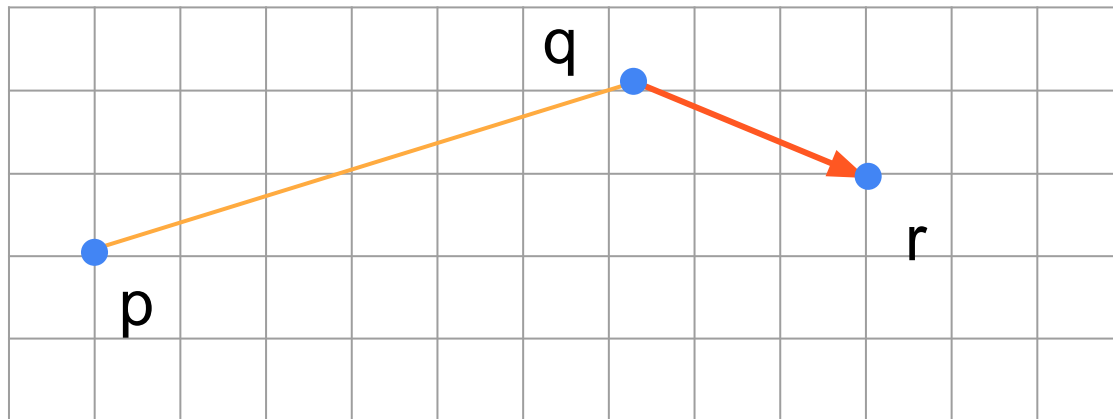
La distancia de un punto q a una recta se puede calcular mediante el largo del vector contraproyección de pq sobre u .



Distancia punto a segmento

Para segmentos hay que tener cuidado con los extremos...

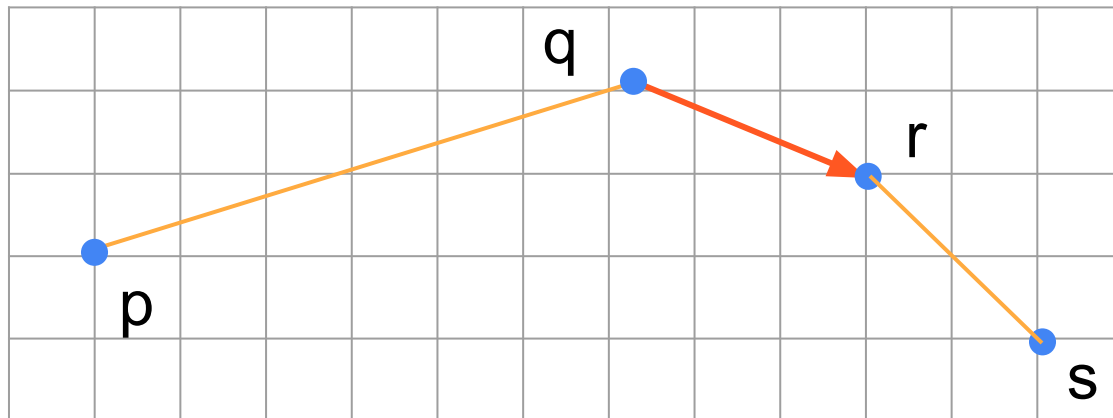
Por ejemplo, en este caso la distancia de r a pq es el vector qr ,



Distancia entre segmentos o rectas

Si dos segmentos se secan, la distancia es cero.

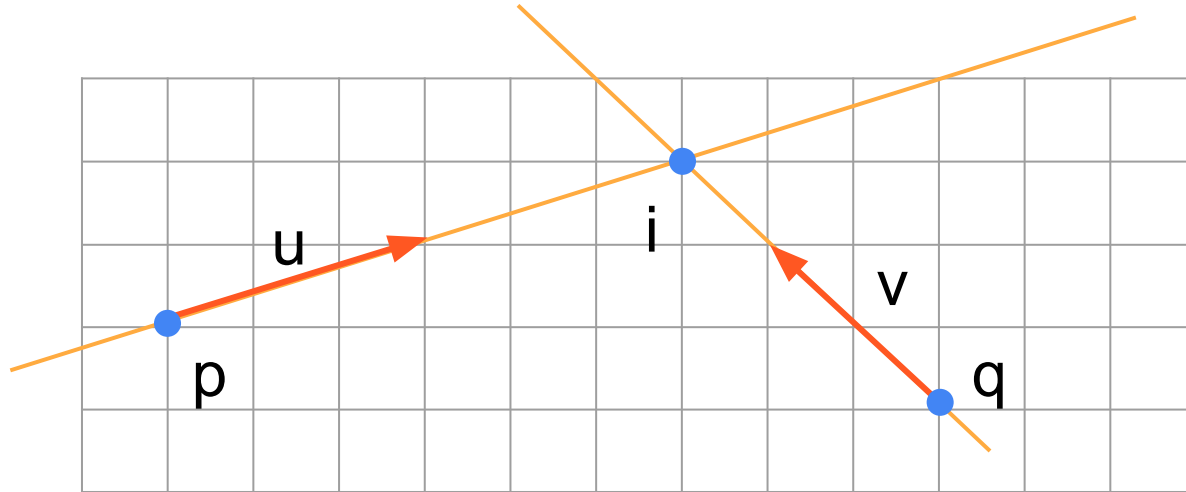
Si no, considerar los extremos.



Intersección de rectas

Intersección

En el plano, dos rectas no paralelas se intersecan en un único punto.



Intersección

Para algún par de números λ, ϕ , resulta:

$$i = p + \lambda u = q + \phi v$$

ϕv es paralelo a v , entonces
 $\phi v \wedge v = 0$

Despejamos λ :

$$\begin{aligned}(p + \lambda u) \wedge v &= (q + \phi v) \wedge v \\ p \wedge v + (\lambda u) \wedge v &= q \wedge v + (\phi v) \wedge v\end{aligned}$$

no son paralelas así
que $u \wedge v \neq 0$

$$\lambda(u \wedge v) = (q - p) \wedge v$$

$$\lambda = (q - p) \wedge v / (u \wedge v)$$

entero / entero

Análogamente para ϕ :

$$\phi = (q - p) \wedge u / (u \wedge v)$$

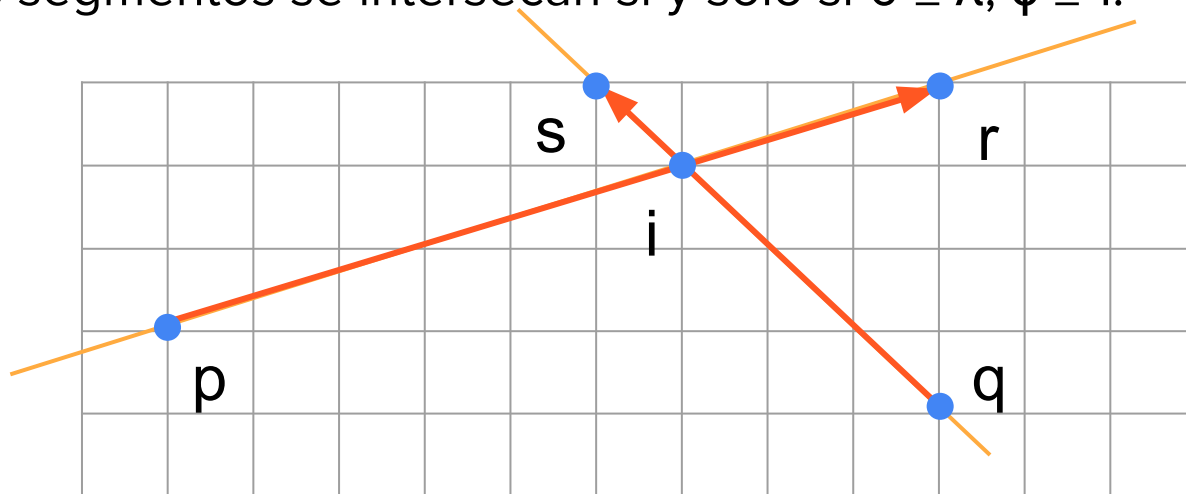
Intersección de segmentos

Dado los segmentos pr y qs , hacemos:

$$u = pr = r - p$$

$$v = qs = s - q$$

Luego, los segmentos se intersecan si y sólo si $0 \leq \lambda, \phi \leq 1$.



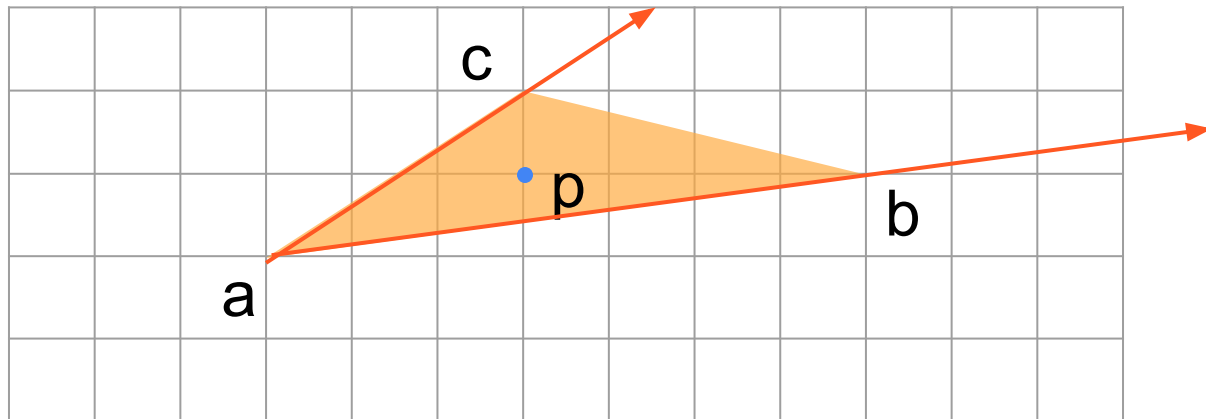
Inclusión

Inclusión ángulo convexo

Dado cuatro puntos a , b , c y p ; determinar si p pertenece al ángulo $\widehat{bâc}$.

solución:

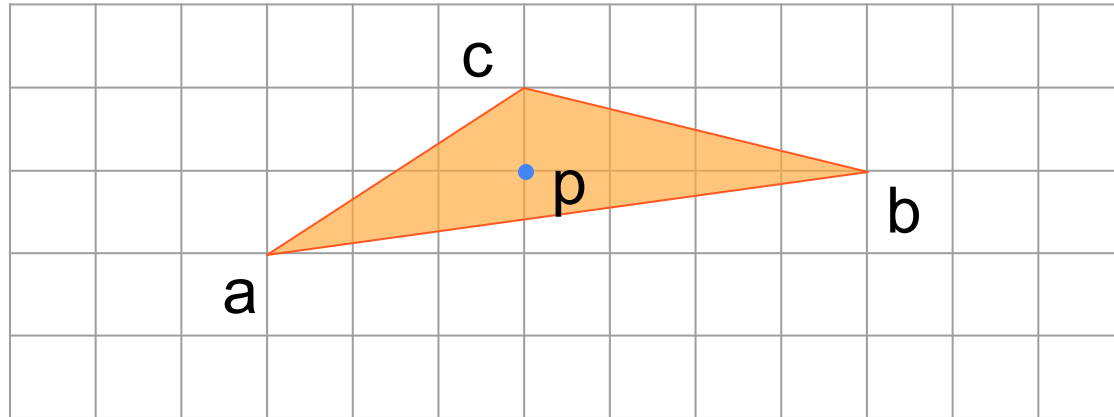
$p \in \widehat{bâc}$ si y sólo si está a la izquierda de ab y a la derecha de ac .



Inclusión triángulo

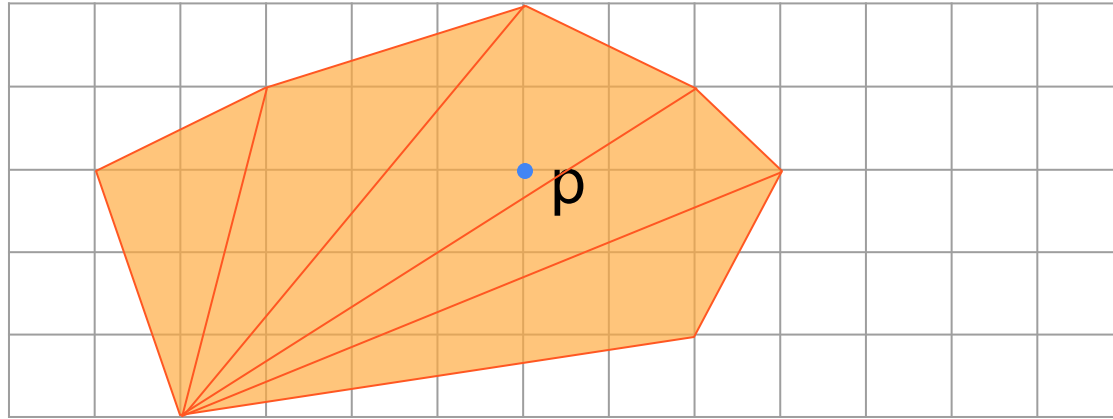
Dado cuatro puntos a , b , c y p ; determinar si p pertenece al triángulo abc .
solución:

$p \in abc$ si y sólo si está del mismo lado del vector ab , bc y ca



Inclusión polígono estrictamente convexo

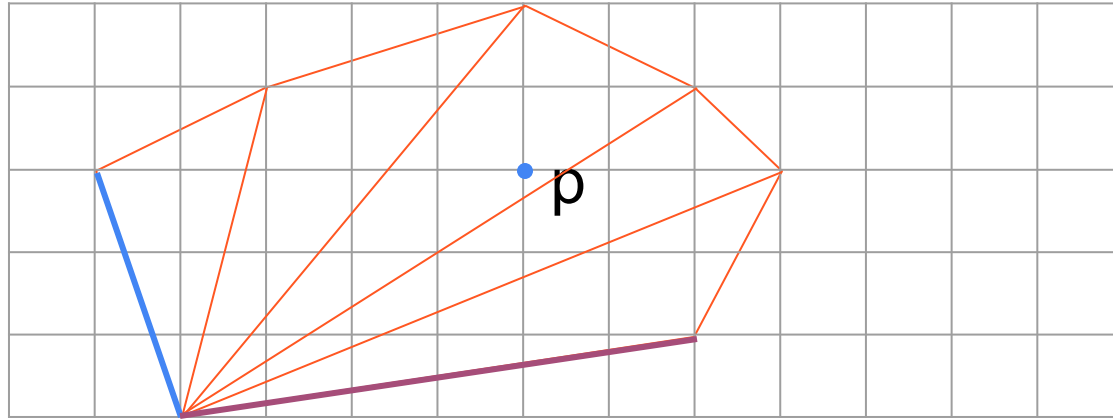
Dado un polígono $P[0..N-1]$ estrictamente convexo, y un punto p ; determinar si p pertenece a P .



Inclusión polígono estrictamente convexo

solución:

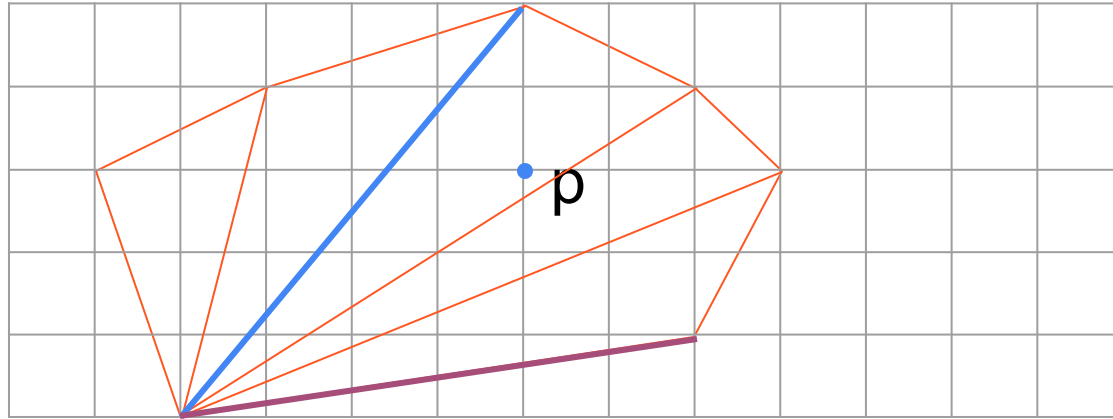
Triangular P, y realizar búsqueda binaria en el ángulo al cual pertenece, luego, chequear inclusión en el triángulo. Complejidad: **$O(\log n)$**



Inclusión polígono estrictamente convexo

solución:

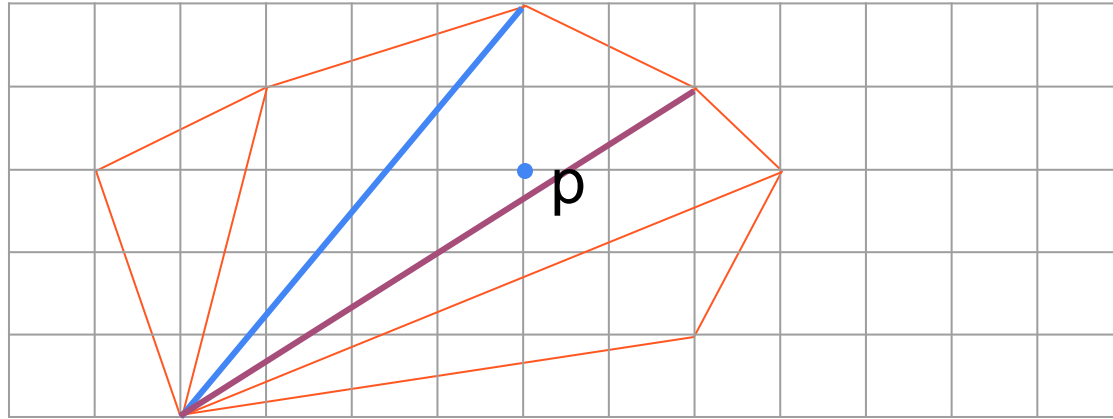
Triangular P, y realizar búsqueda binaria en el ángulo al cual pertenece, luego, chequear inclusión en el triángulo. Complejidad: **$O(\log n)$**



Inclusión polígono estrictamente convexo

solución:

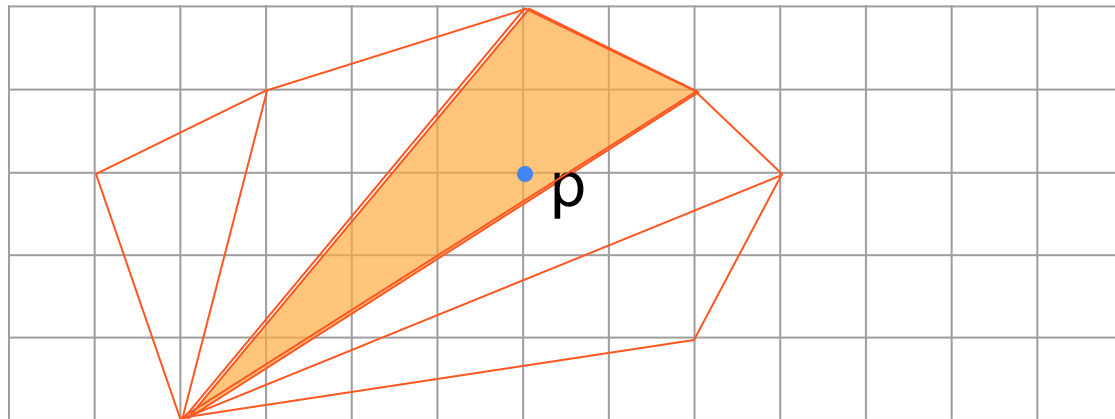
Triangular P, y realizar búsqueda binaria en el ángulo al cual pertenece, luego, chequear inclusión en el triángulo. Complejidad: **$O(\log n)$**



Inclusión polígono estrictamente convexo

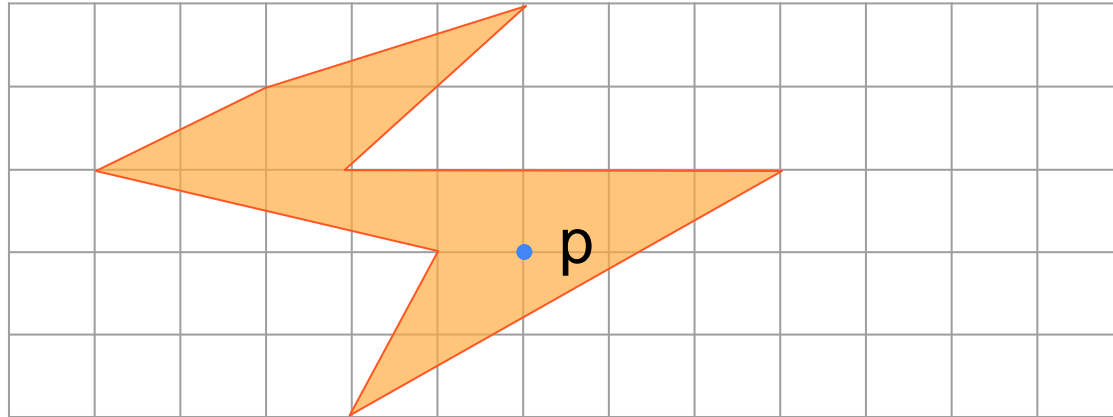
solución:

Triangular P, y realizar búsqueda binaria en el ángulo al cual pertenece, luego, chequear inclusión en el triángulo. Complejidad: **$O(\log n)$**



Inclusión polígono no convexo

Dado un polígono simple $P[0..N-1]$ y un punto p ; determinar si p pertenece a P .

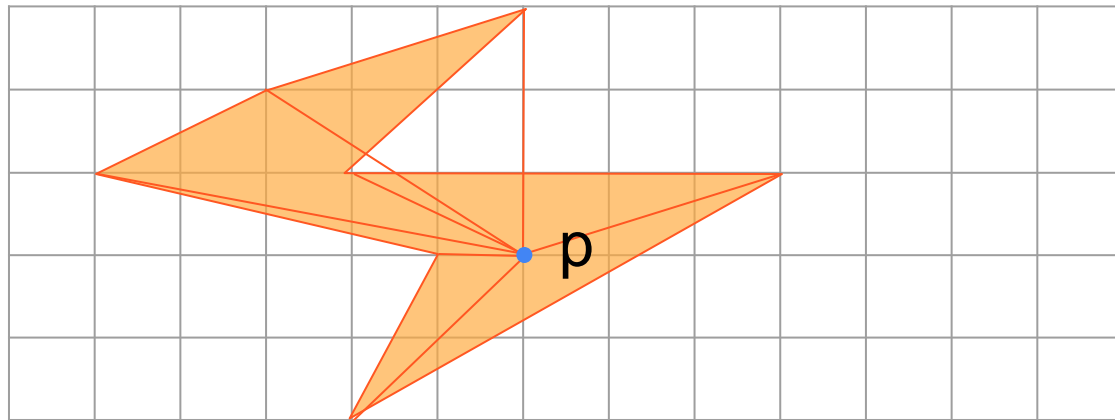


Inclusión polígono no convexo

solución:

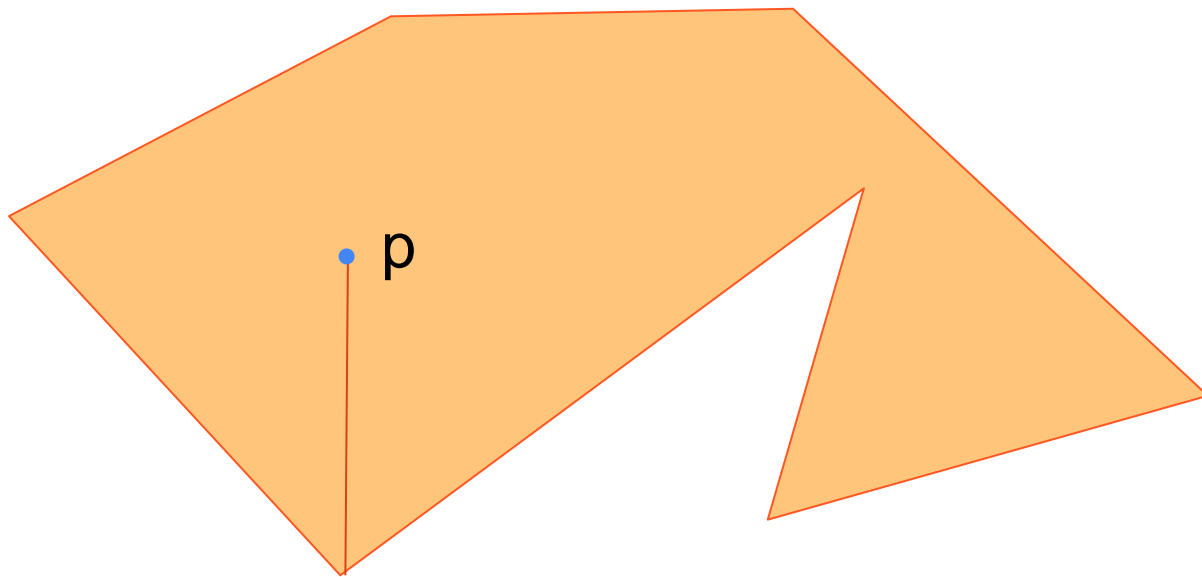
Sumar los ángulos respecto a p . Si la suma da 0, entonces p es exterior.

Si la suma da $\pm 2\pi$, es interior. Complejidad: **$O(n)$**



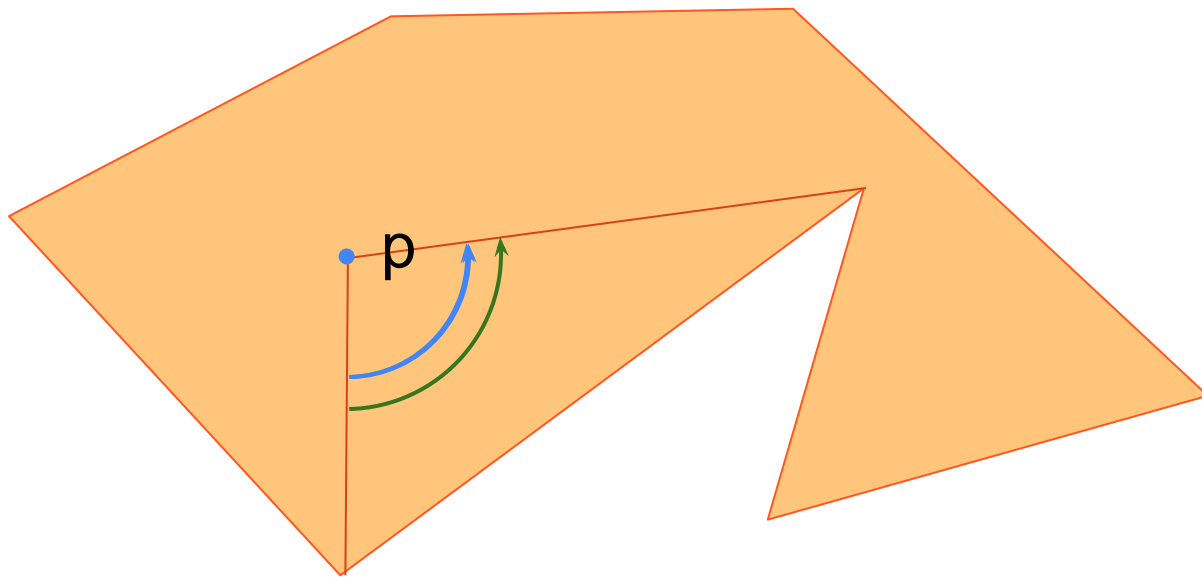
Animación

"Sumar los ángulos respecto a p."



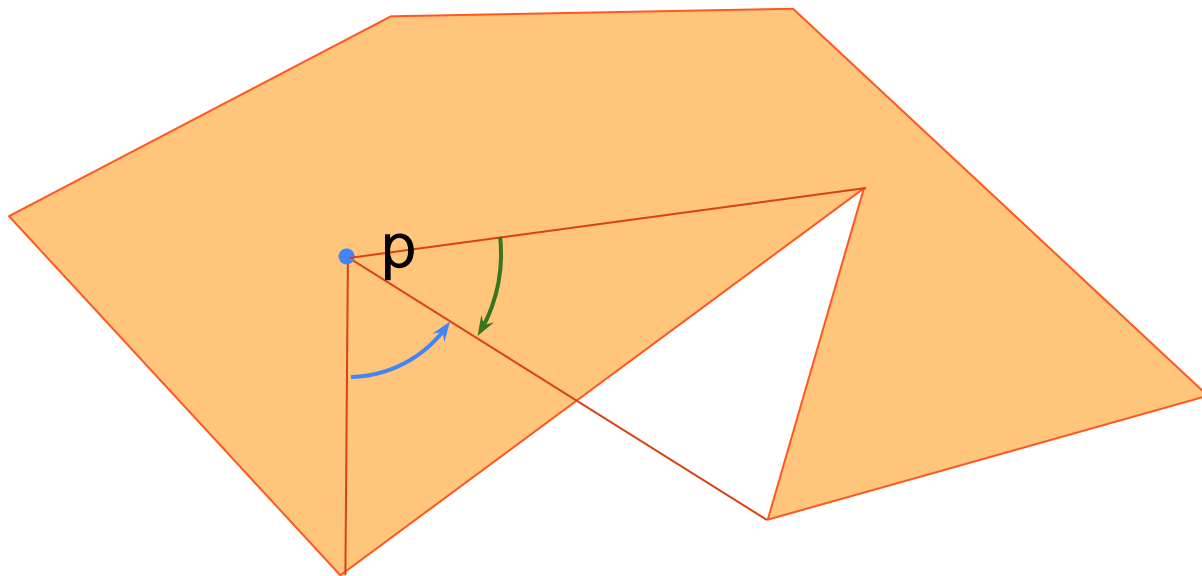
Animación

"Sumar los ángulos respecto a p."



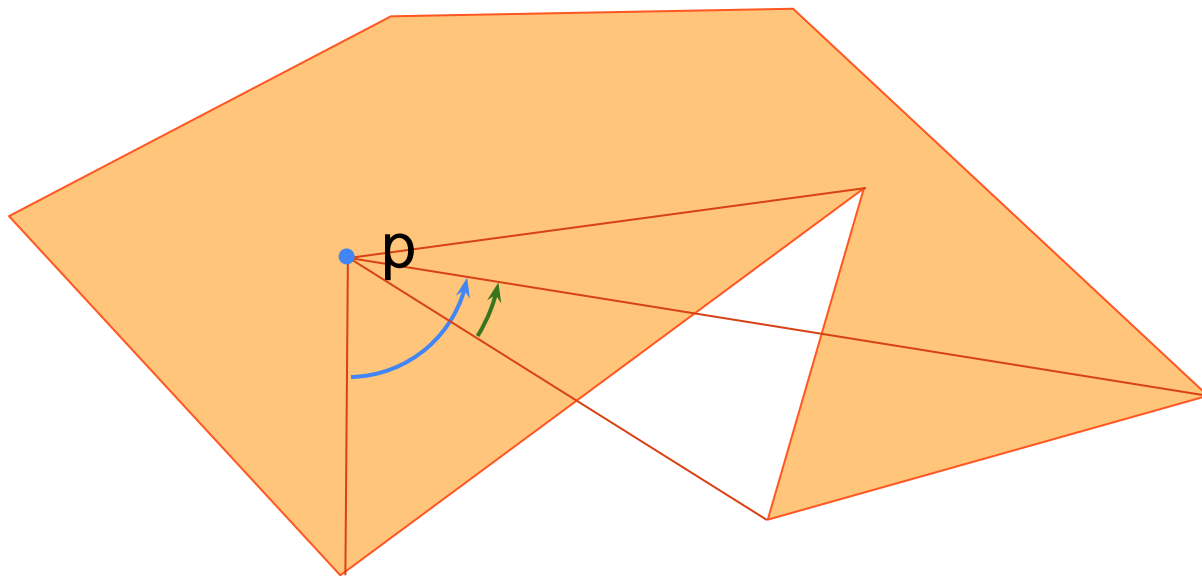
Animación

"Sumar los ángulos respecto a p."



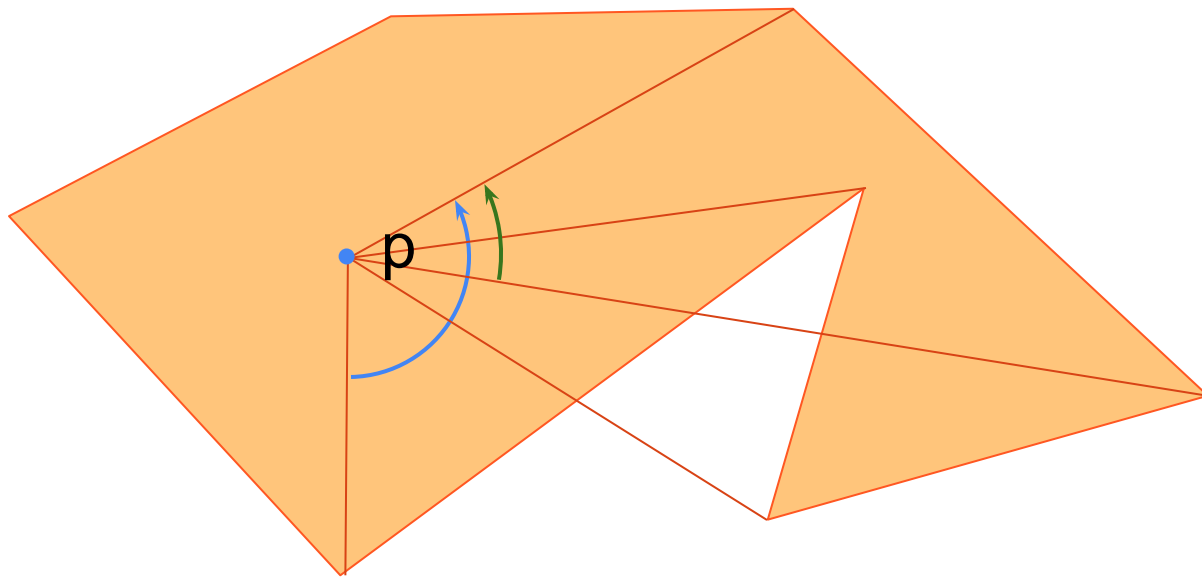
Animación

"Sumar los ángulos respecto a p."



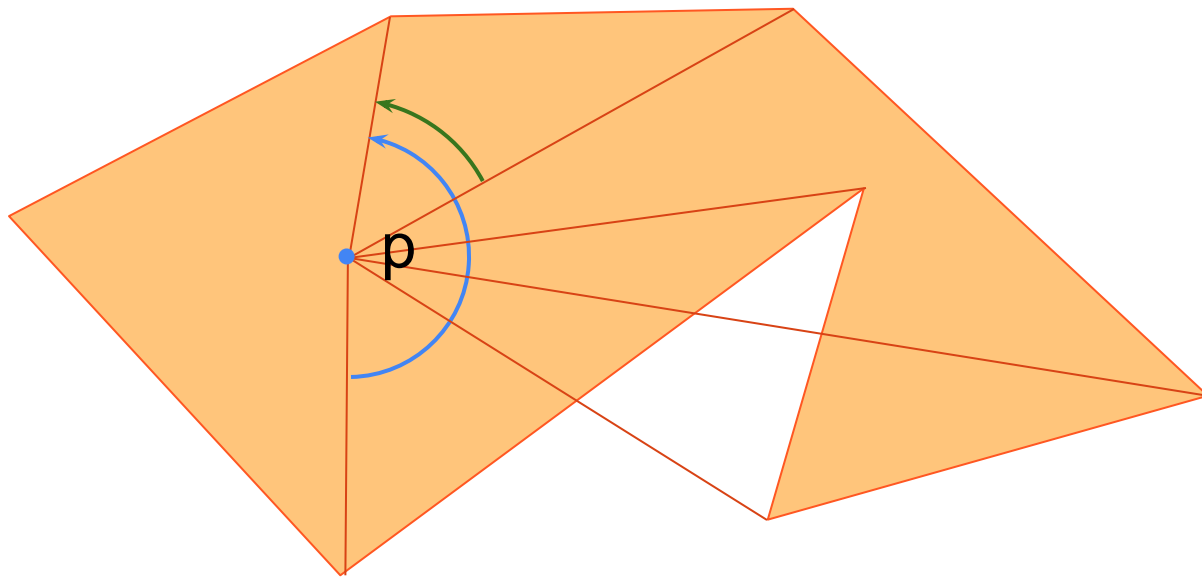
Animación

"Sumar los ángulos respecto a p."



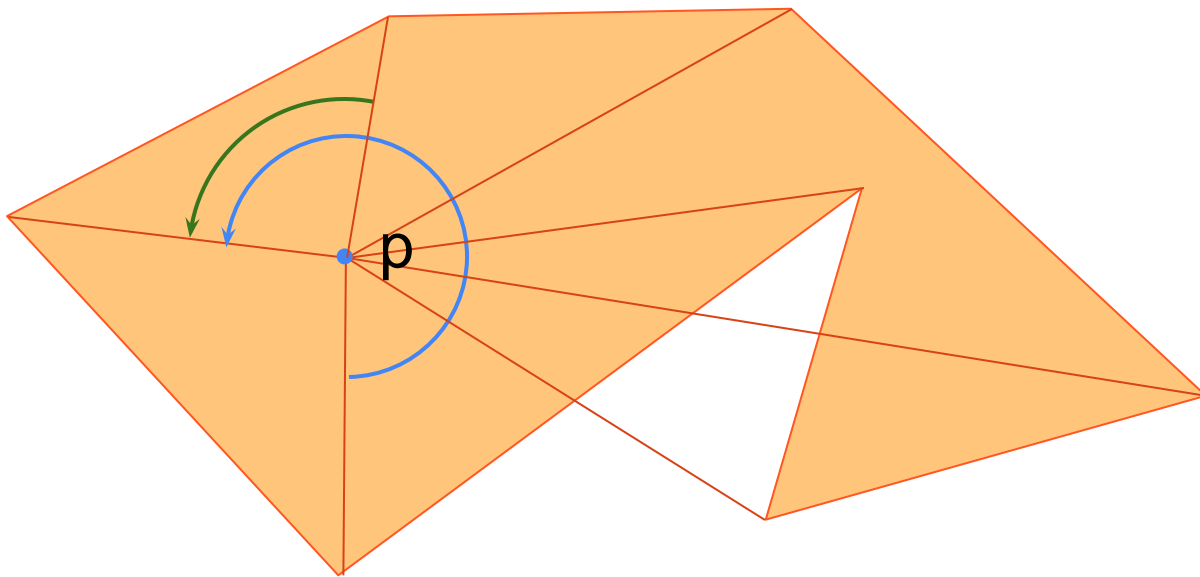
Animación

"Sumar los ángulos respecto a p."



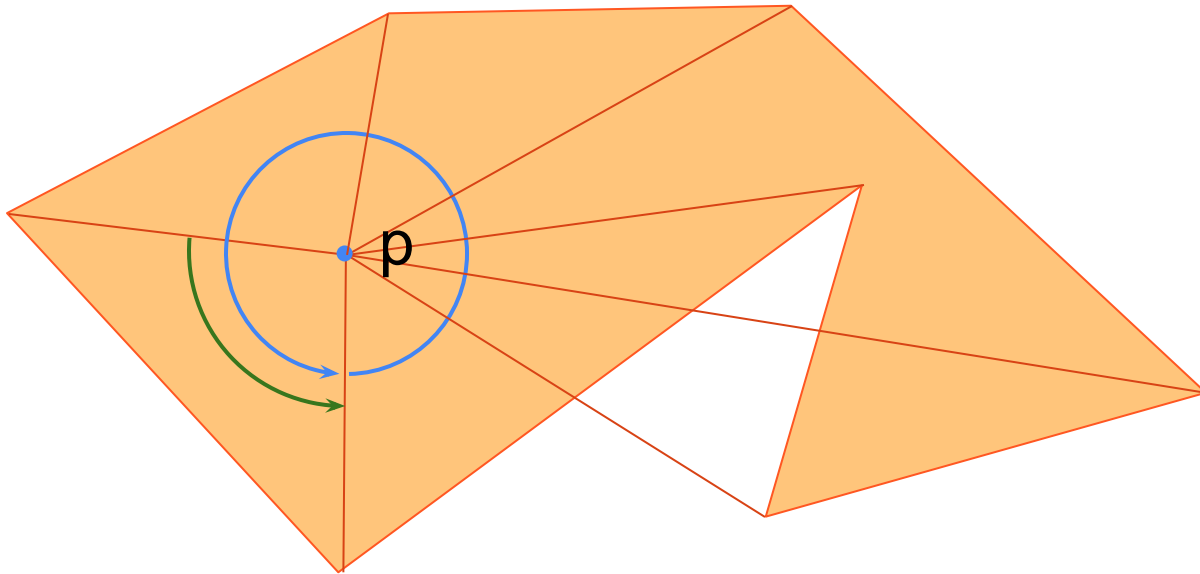
Animación

"Sumar los ángulos respecto a p."



Animación

"Sumar los ángulos respecto a p."



Inclusión polígono no convexo

Obligatoriamente tendremos que trabajar con floats...

```
bool pertenece(poligono &P, punto p){  
    double a = 0;  
    forn(i, P.size()){  
        a += angulo(P[i], p, P[(i+1)%P.size()]);  
    }  
    return abs(abs(1) - 2*acos(-1)) < 0.0001 ;  
}
```

π

*jelegir un épsilon
acorde!*

Sweep Line

Sweep Line

La idea es tener una línea imaginaria que se desplaza barriendo el plano.

Puede ser una recta que recorre tanto verticalmente u horizontalmente.

Quizás una semirrecta que gira (polarmente, por ángulo).

Tal vez no es una recta, es una franja, o una circunferencia...

CHULL

(cápsula convexa)

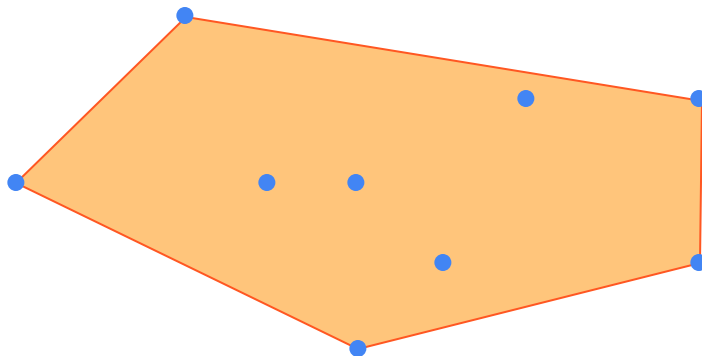
CHULL

Encontrar el polígono más pequeño que contiene un conjunto de puntos.



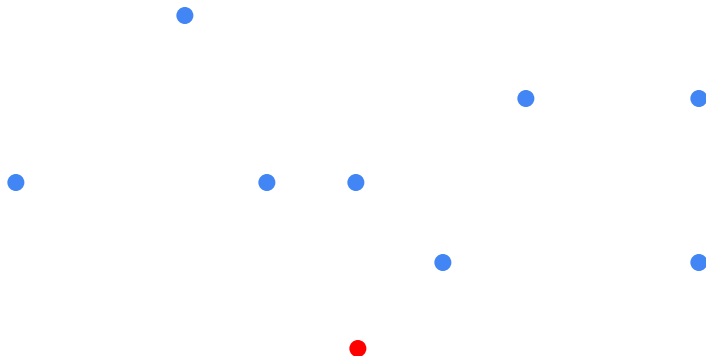
CHULL

¡Se puede resolver mediante un algoritmo de sweep line!



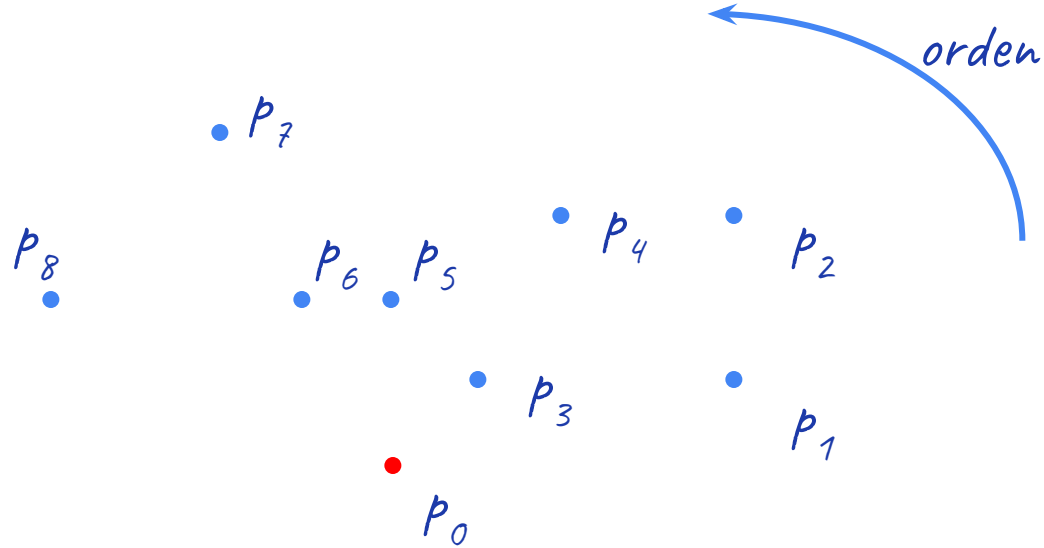
Algoritmo de Graham

Para esto, encontraremos un punto extremo. Por ejemplo, el punto más bajo (desempatando por el más a la izquierda).



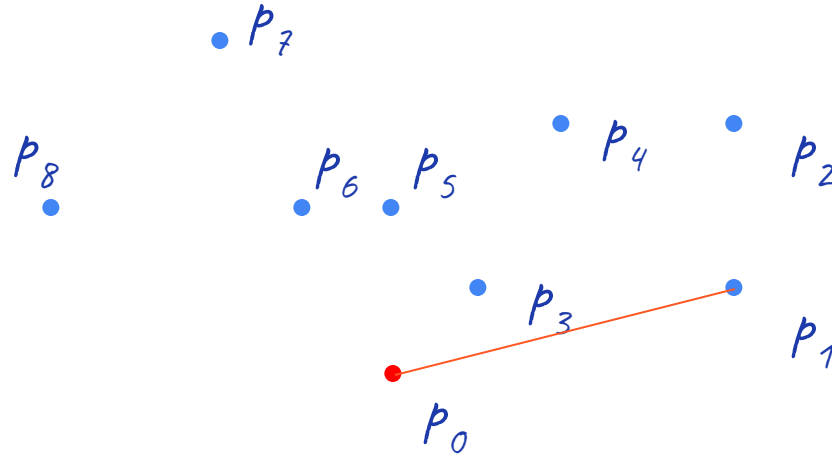
Algoritmo de Graham

Ordenamos los puntos por ángulo respecto a este punto extremo.



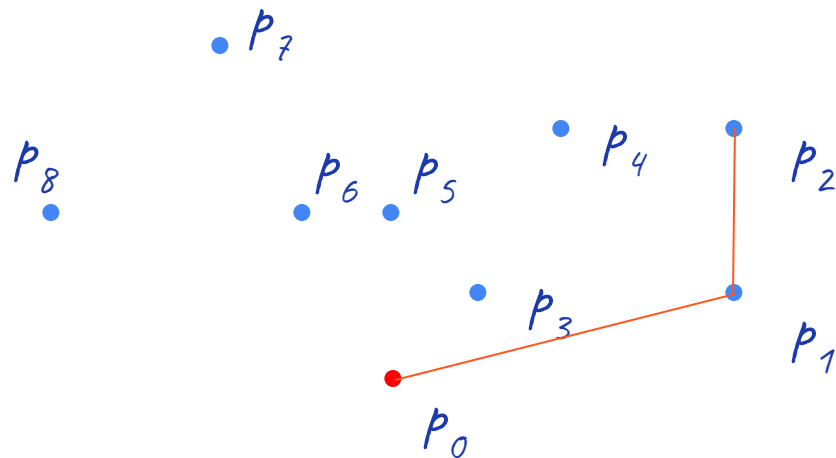
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



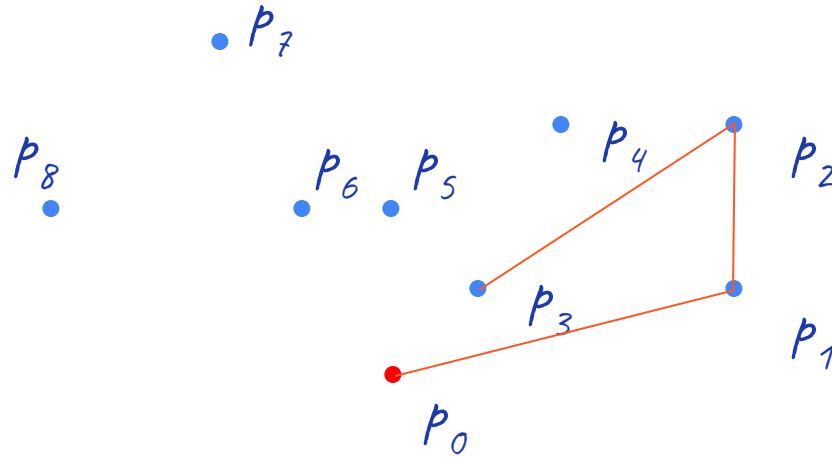
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



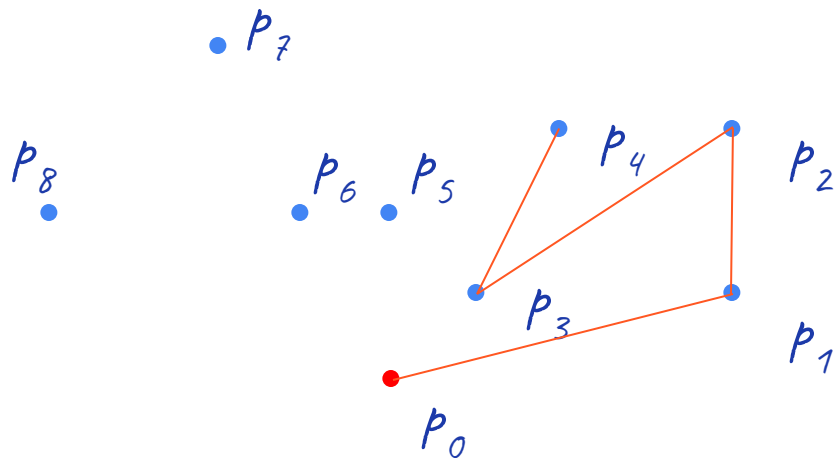
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



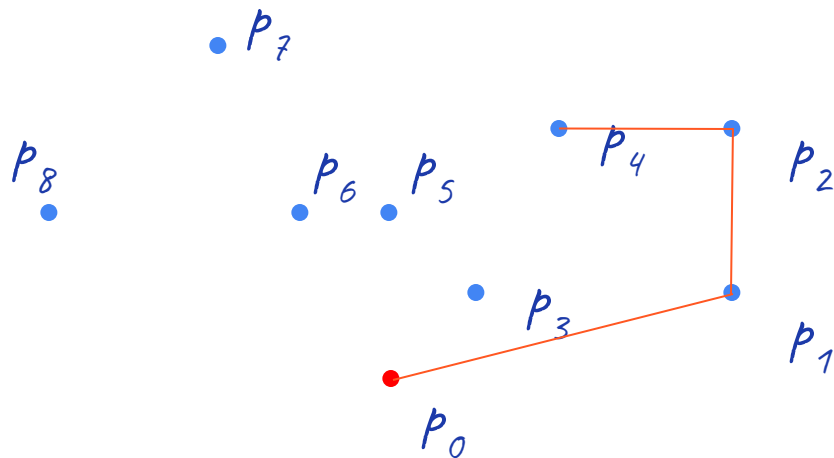
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



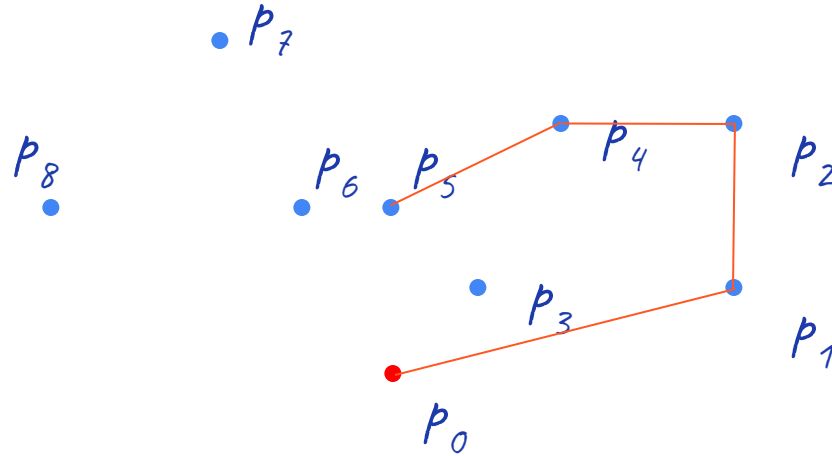
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



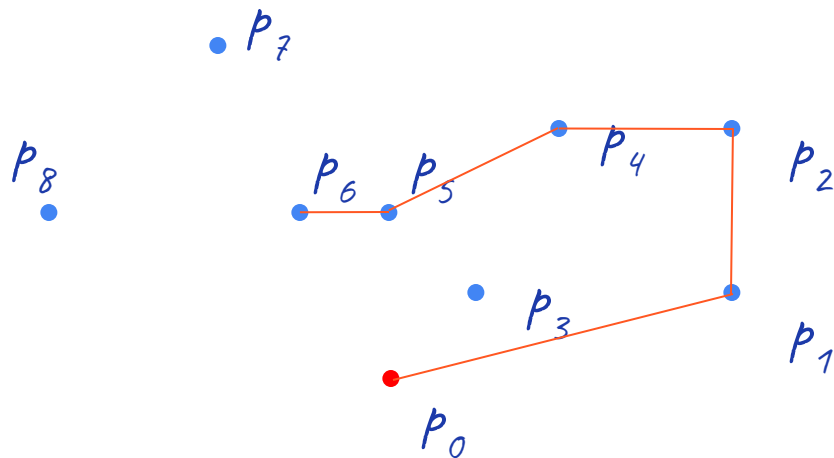
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



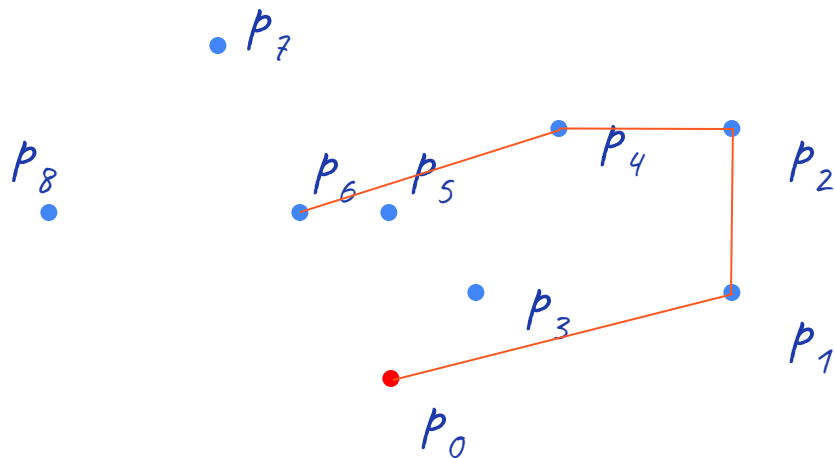
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



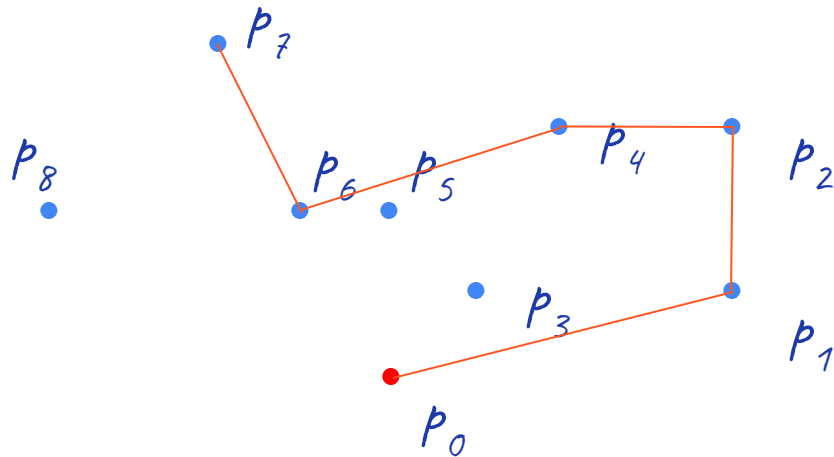
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



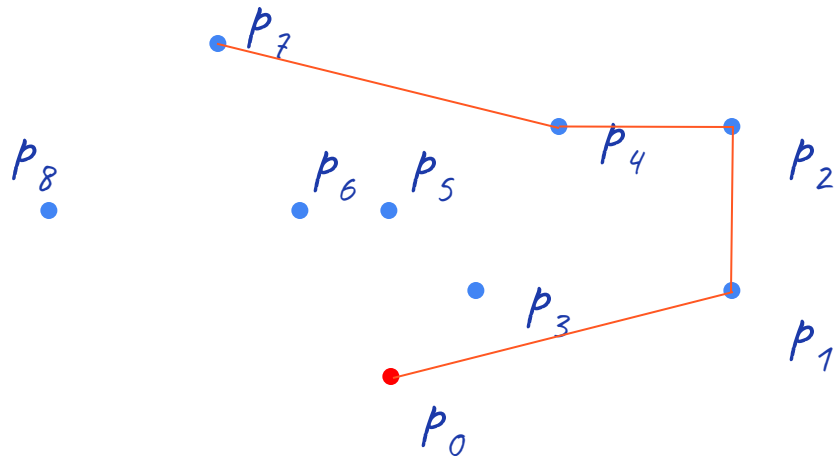
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



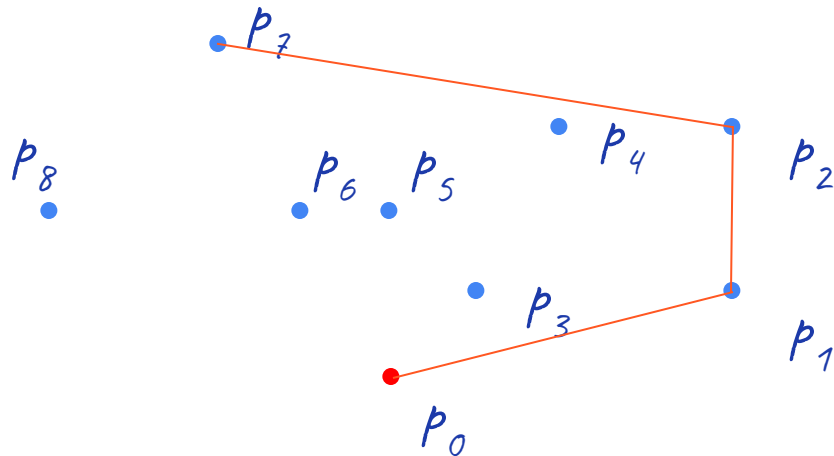
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



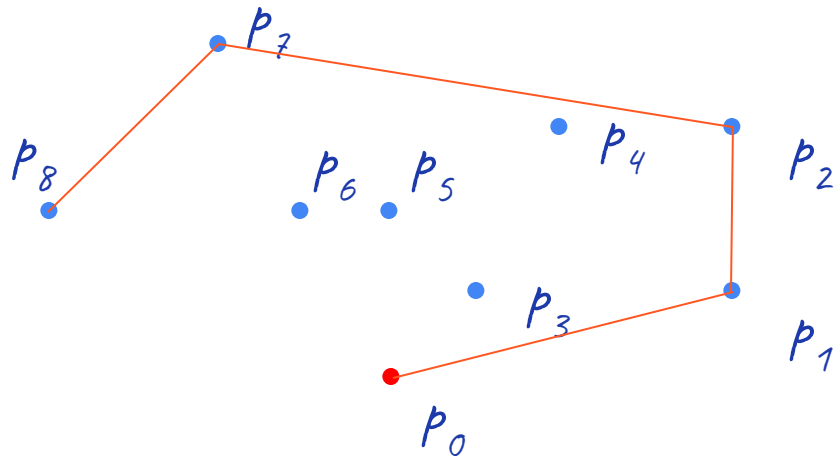
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



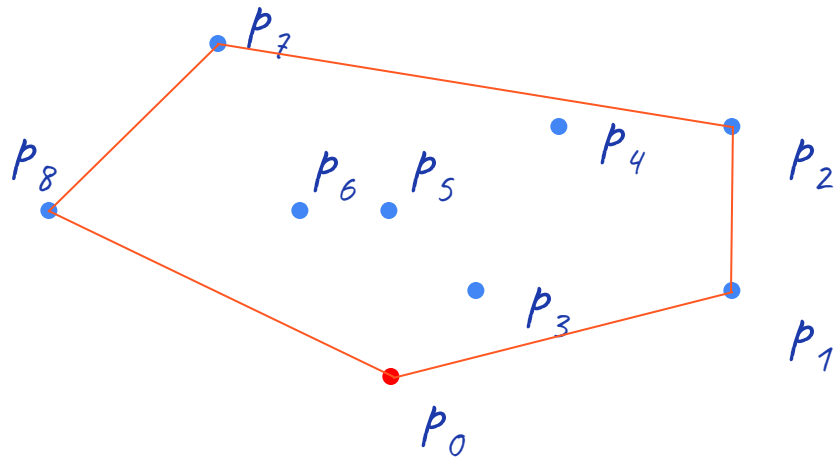
Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



Algoritmo de Graham

Barremos el plano en este orden (sweep line polar) y vamos agregando punto por punto mientras no formemos un giro a izquierda. En dicho caso, removemos el penúltimo punto hasta evitar la violación.



Algoritmo de Graham

```
poligono GS(vector<point> p){
    intercambiar p[0] con un punto extremo
    ordenar p[1..N-1] polarmente (por ángulo)
    poligono chull = vacio
    forn(i, N)
        int m = chull.size()
        while(m >= 2 and chull[m-2],chull[m-1],p[i] es giro a la derecha)
            chull.pop_back() // borro el penultimo considerado
            m--
        chull.push_back(p[i]) // agrego el ultimo
    return chull
}
```

Complejidad $O(N)$ + sort = $O(N \log N)$

PPMA

(par de puntos más alejado)

Par de puntos más alejado

Este problema consiste en, dada una nube de puntos, encontrar dos de ellos cuya distancia sea máxima entre todos los pares.

La solución trivial es probar todos los pares de puntos $O(N^2)$.

Luego de pensarlo un poco, resulta evidente que estos puntos pertenecen a la cápsula convexa de la nube, calcularla es $O(N \log N)$. Pero probar todos los pares de la CHULL también es $O(N^2)$

Par de puntos más alejado

La idea es utilizar un concepto llamado 2-pointers o rotating-calipers que surge de la siguiente observación:

Si p es un punto de la chull, y q es el punto más alejado de p (que también estará en la chull), entonces al recorrer una vuelta completa con p ; q habrá recorrido una vuelta completa también. Por lo que podemos hallar el par más alejado en sólo $O(N)$.



Cuidado

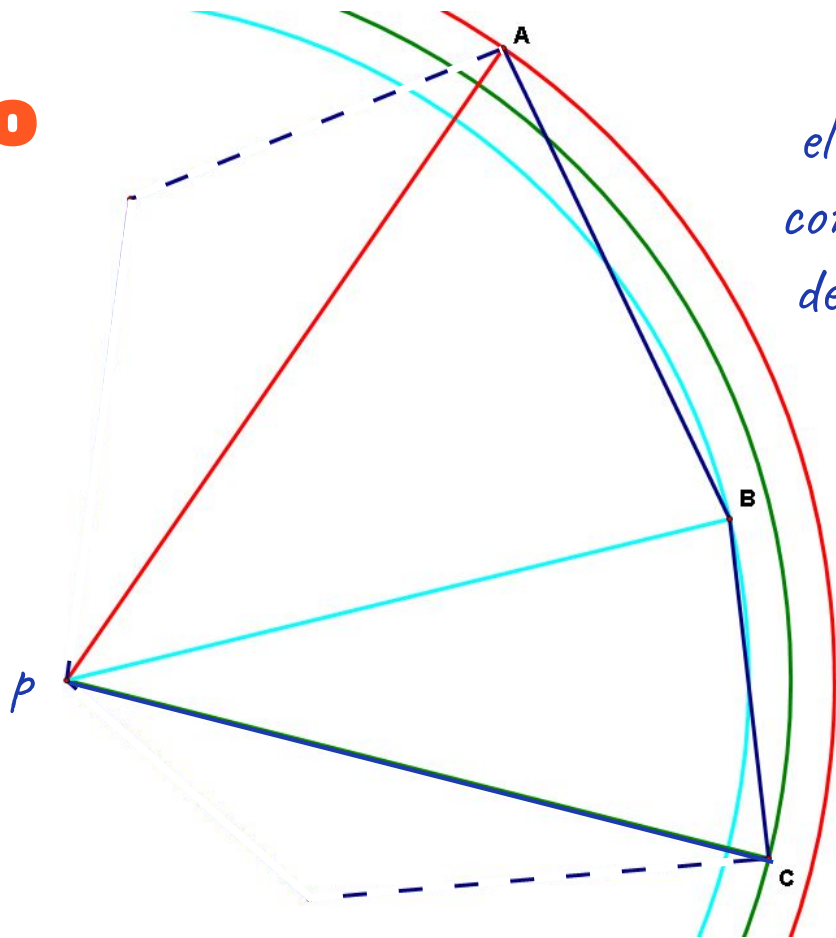
Observación, el siguiente algoritmo puede **fallar** en encontrar el ppma p-q

Sea $P[0..n-1]$ un polígono convexo

```
// hallar q (=P[j]) inicial (aquel correspondiente a p = P[0])
j = 0
for(i = 1 .. n-1)
    if( norma(P[i]-P[0]) > norma(P[j]-P[0]) )
        j = i
// para cada p (=P[i]) hacer
for(i = 0 .. n-1)
    while(j' = j+1 mod n; norma(P[j']-P[i]) ≥ norma(P[j]-P[i]); j'++ mod n)
        j = j'
    considerar par i-j
```

Cuidado

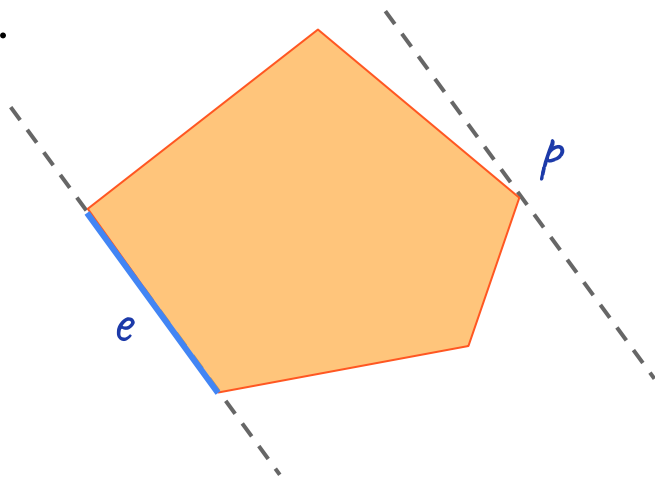
Ejemplo



el punto A nunca se considera como candidato para q , a pesar de encontrarse más alejado de p que C .

Solución

La solución es una pequeña modificación a nuestro algoritmo. Recorreremos el polígono por arista $e=(r,s)$, y para cada una de ellas, encontraremos el punto p más alejado de la recta que contiene a e . Luego, consideraremos $p-r$ y $p-s$. Obligatoriamente en algún momento consideramos el ppma.



Par de puntos más alejado

```
par<punto, punto> PPMA(vector<point> nube){
    hull = GS(nube) // calculamos la hull
    respuesta = (hull[0], hull[1])
    n = hull.size(), j = 0
    for(i = 0 .. n-1)
        while( hull[j+1] no se acerca a la recta  hull[i]hull[i+1] )
            j++; // recordemos j++, j+1, y i+1 siempre en mod n
        if( distancia(hull[i], hull[j]) > distancia(respuesta) )
            respuesta = (hull[i], hull[j])
        if( distancia(hull[i+1], hull[j]) > distancia(respuesta) )
            respuesta = (hull[i+1], hull[j])
    return respuesta
}
```

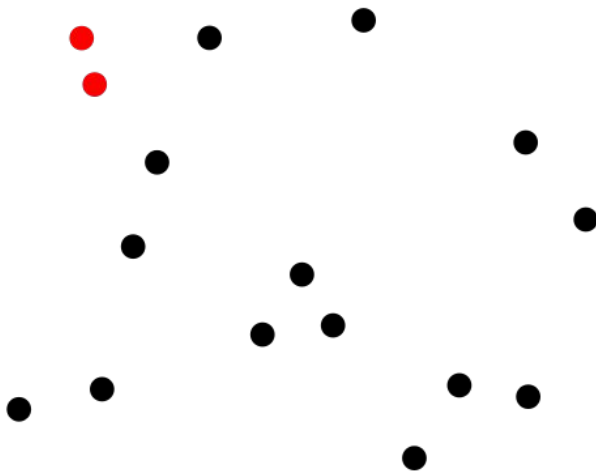
Complejidad $O(N)$ + hull = $O(N \log N)$

PPMC

(par de puntos más cercano)

PPMC

Dado un conjunto de puntos, hallar el par de puntos que minimice la distancia entre ellos.



PPMC

Estos puntos no necesariamente pertenecen a la CHULL del conjunto, sin embargo podemos encontrar un algoritmo del estilo sweep line para resolverlo.

La idea es barrer la nube de puntos horizontalmente, teniendo en cuenta el mejor par visto hasta el momento, e ir considerando nuevos pares a medida se vaya alcanzando los demás puntos.

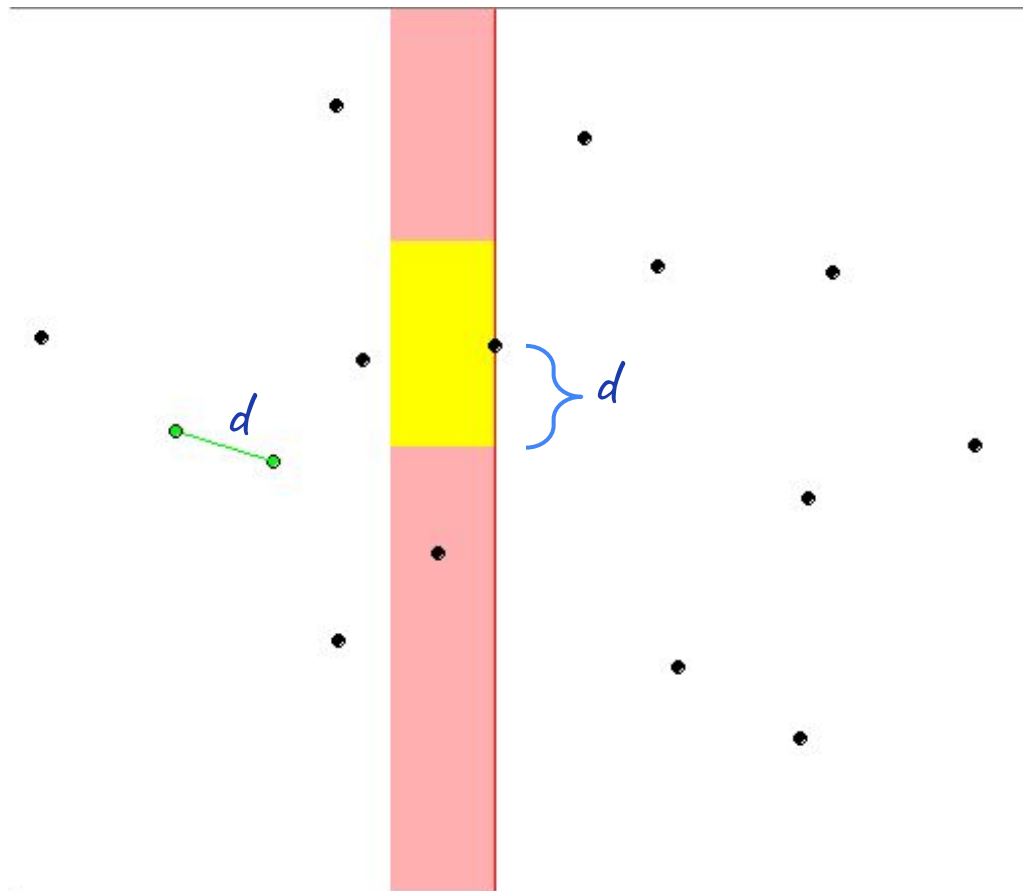
La clave está en no considerar pares de más para mantener el barrido lineal.

PPMC

solución:

- Barrer los puntos de izquierda a derecha.
- Si la menor distancia obtenida hasta el momento es " d ", mantengo una franja de ancho d de todos los puntos a la izquierda de mi sweep line, ordenados por " y ".
- Para cada punto que recorro con mi sweep line, reviso el rango de puntos de mi franja que se encuentran a distancia no mayor que d unidades hacia arriba o hacia abajo.

Solo puede haber 6 puntos en dicha región!



PPMC

```
par<punto, punto> ppmc(vector<point> p){
    respuesta = (p[0], p[1])
    double d = distancia(respuesta)
    set<point> S = vacío (ordenado por coordenada y)
    ordeno p[0..N-1] por x
    int a = 0
    for(int b = 0..N-1)
        while( p[b].x - p[a].x > d )
            S.erase(p[a++])
        for( q = S.lower_bound(p[b].y - d); q.y - p[b].y < d; sig(q))
            if( d > largo(q, p[b]) )
                d = largo(q, p[b])
                respuesta = (q, p[b])
        S.insert(p[b])
    return respuesta
}
```

Problemas

Problemas

Básicos:

- <https://codeforces.com/problemset/problem/598/C>
- <https://www.spoj.com/problems/GOALFR/>

← (contiene círculos)

Pertenencia: punto en polígono convexo:

- <https://www.spoj.com/problems/INOROUT/>

Intersección de segmentos:

- <https://www.spoj.com/problems/ANTTT/>

PPMC:

- <https://www.spoj.com/problems/CLOPPAIR/>

Chulls:

- <https://www.spoj.com/problems/VMILI/>
- <https://www.spoj.com/problems/TFOSS/>

Círculos:

- <https://www.spoj.com/problems/BLMIRINA/>

Uno de nacional de OIA:

- <http://www.oia.unsam.edu.ar/wp-content/uploads/2013/10/c3a15n3p2.pdf>

Uno de selectivo de OIA:

- <http://juez.oia.unsam.edu.ar/#/task/buscandof/statement>