ALGORITMOS GEOMÉTRICOS

Análisis y diseño de algoritmos II- 2009

Algoritmos geométricos

La geometría computacional es una rama de la ciencia de la computación que estudia algoritmos para resolver problemas geométricos.

Aplicaciones

Computación gráfica, CAD (Computer-Aided Design), robótica, diseño de circuitos integrados, GIS (Geographic information System),…

Algoritmos geométricos

Los algoritmos geométricos operan sobre objetos geométricos tales como puntos, segmentos o polígonos.

Analizaremos algoritmos en dos dimensiones en los que cada objeto geométrico es representado como un conjunto de puntos $\{p_i\}$ donde $p_i=(x_i,y_i)$ con $x_i, y_i \in R$

Por ejemplo, un polígono de n vértices es representado por una secuencia $<p_0,p_1,\ldots p_{n-1}>$ de sus vértices.

Algoritmos geométricos

Interesantes problemas de geometría computacional se presentan actualmente en espacios 3D o de orden superior.

En este curso ejemplificaremos técnicas de geometría computacional en 2D que son la base para manipular entidades geométricas de mayores dimensiones.

Algoritmos geométricos

Propiedades de segmentos

Una combinación convexa de dos puntos distintos $p_1=(x_1,y_1)$ y $p_2=(x_2,y_2)$ es algún punto $p_3=(x_3,y_3)$ tal que para algún α en el rango $0 \leq \alpha \leq 1$ resultan

$$x_3 = \alpha\, x_1 + (1-\alpha)\, x_2$$

$$y_3 = \alpha\, y_1 + (1-\alpha)\, y_2$$

Intuitivamente,

$$p_3 = \alpha\, p_1 + (1-\alpha)\, p_2$$

denota a un punto que pertenece a la recta que pasa por p_1 y p_2 y está sobre o entre p_1 y p_2.

Algoritmos geométricos

Segmento

Dados dos puntos p_1 y p_2, el segmento $\overline{p_1p_2}$ es el conjunto de combinaciones convexas de p_1 y p_2.

Vector

Un segmento orientado entre p_1 y p_2 se denota $\overrightarrow{p_1p_2}$. Si p_1 es el origen (0,0) $\overrightarrow{p_1p_2}$ es el vector $\overrightarrow{p_2}$

Algoritmos geométricos

Problemas simples

Dados dos segmentos $\overline{p_0p_1}$ y $\overline{p_0p_2}$, ¿ $\overline{p_0p_1}$ está en sentido horario desde $\overline{p_0p_2}$ con respecto al extremo p_0?

Dados $\overline{p_1p_2}$ y $\overline{p_2p_3}$, si recorremos $\overline{p_1p_2}$ y luego $\overline{p_2p_3}$, giramos a la izquierda en p_2?

Dados dos segmentos $\overline{p_1p_2}$ y $\overline{p_3p_4}$, se intersecan?

Algoritmos geométricos

Los algoritmos geométricos se basan en cálculos simples que usan sumas, restas, multiplicaciones y comparaciones.

No calculan divisiones ni funciones trigonométricas por razones de costo y error de redondeo.

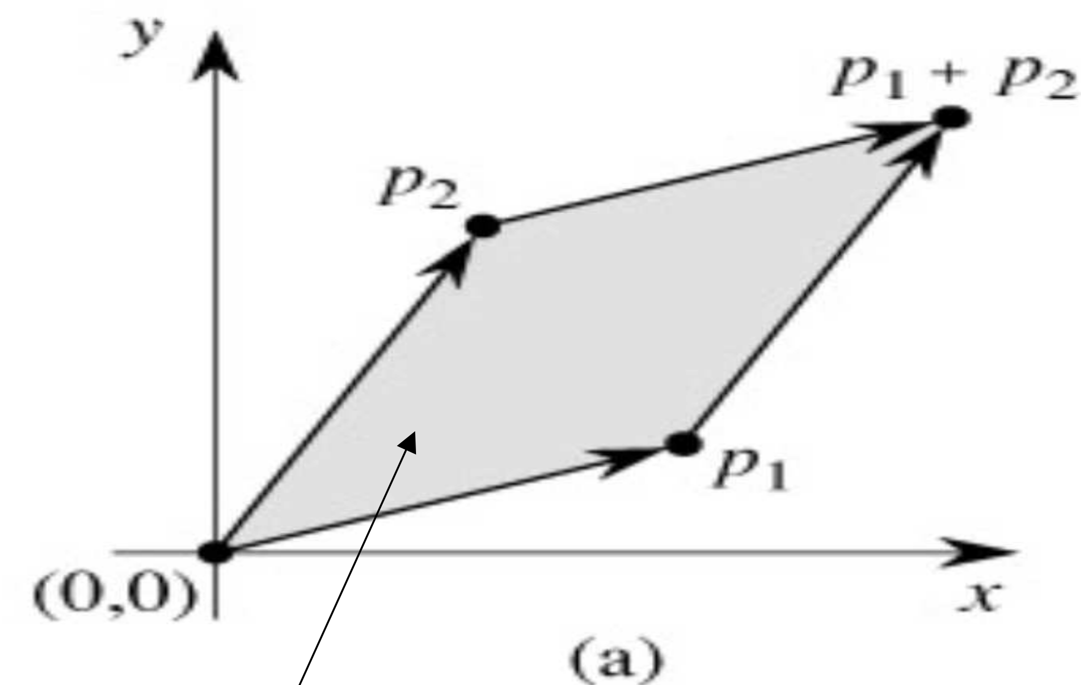
Algoritmos geométricos

Producto cruzado

Los problemas vinculados con segmentos se basan en el cálculo de productos cruzados.

Dados dos vectores p_1 y p_2, el producto cruzado $p_1 x p_2$ puede interpretarse como el área del paralelogramo formado por (0,0), p_1, p_2 y p_1+p_2.

Algoritmos geométricos



Producto cruzado

Algoritmos geométricos

Otra definición de producto cruzado

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 . \end{aligned}$$

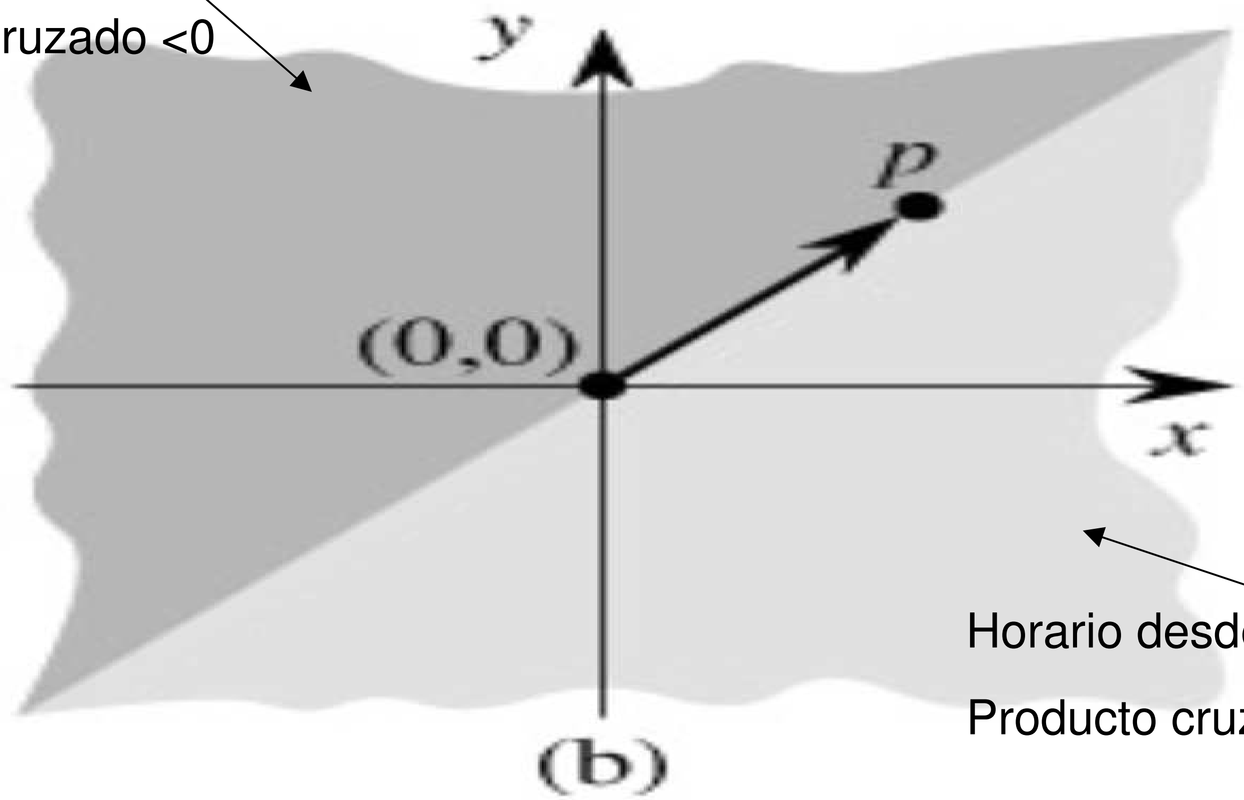
Si p_1 x p_2 >0, p_1 está en sentido horario desde p_2 con respecto al origen (0,0)

Si p_1x p_2 < 0, p_1 está en sentido antihorario desde p_2 con respecto al origen (0,0)

Algoritmos geométricos

Antihorario desde p

Producto cruzado <0



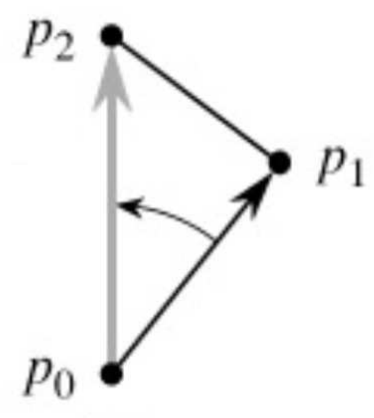
Horario desde p

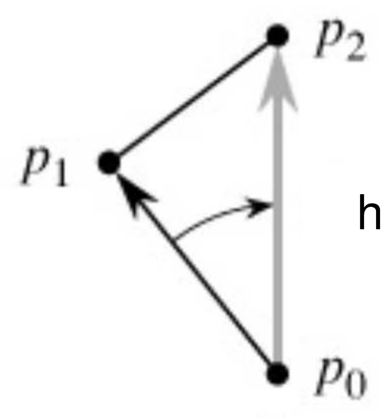
Producto cruzado>0

Algoritmos geométricos

Dados $\overline{p_0p_1}$ y $\overline{p_1p_2}$, si recorremos $\overline{p_0p_1}$ y luego $\overline{p_1p_2}$, giramos a la izquierda o a la derecha en p_2?

antihorario





horario

$(p_2-p_0) \times (p_1-p_0) < 0$

$(p_2-p_0) \times (p_1-p_0) > 0$

(p2-p0) x (p1-p0) = 0   Colineales

Algoritmos geométricos

Ejemplos de problemas que podemos resolver simplemente aplicando producto cruzado

Dado un punto p el segmento $\overline{p1p2}$ *está a la derecha o a la izquierda de p.*

Dado un polígono convexo, determinar si un punto es interior

Algoritmos geométricos

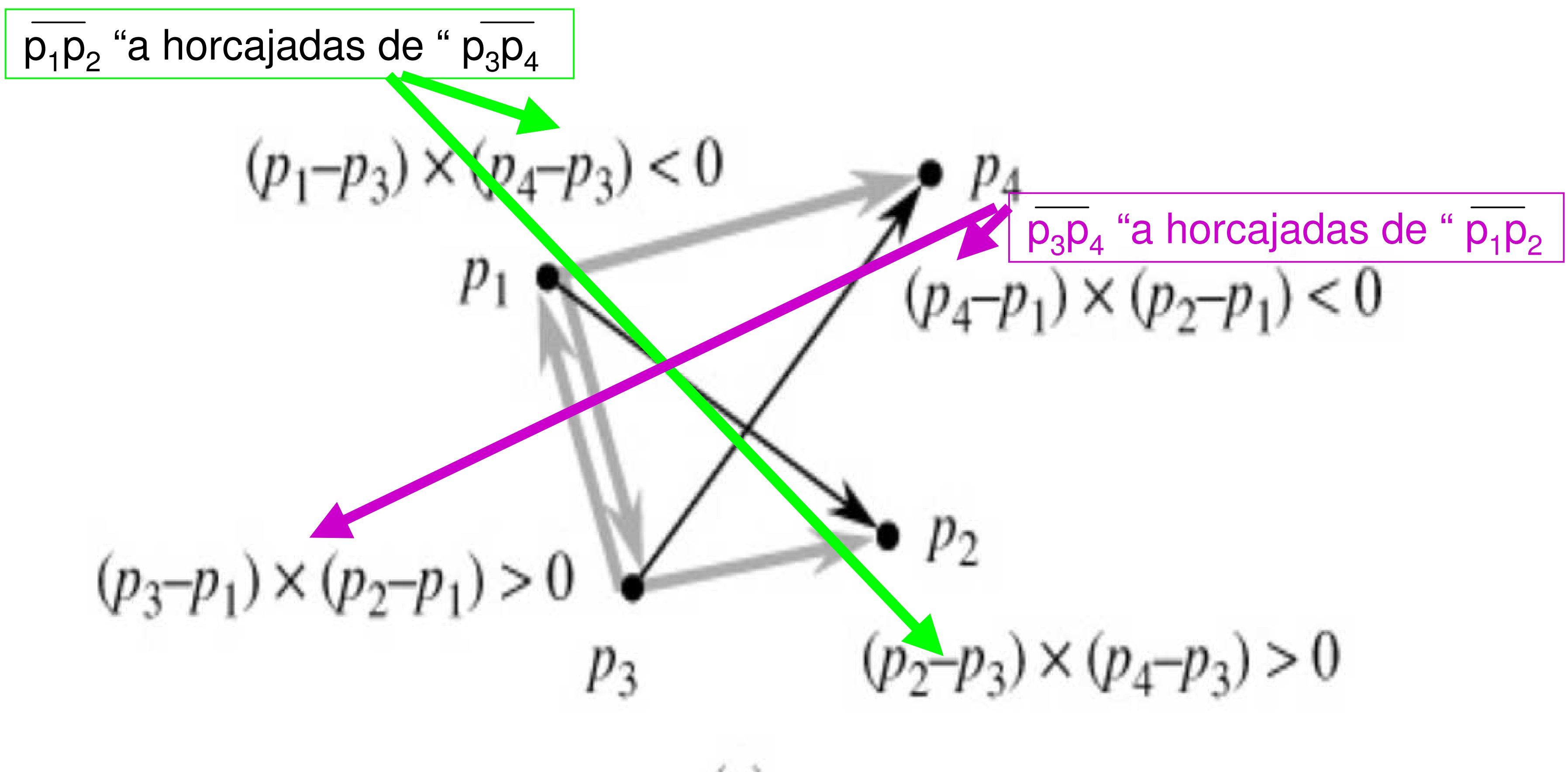
Determinar la intersección entre dos segmentos

Hay intersección entre dos segmentos $\overline{p_1p_2}$ y $\overline{p_3p_4}$ si y sólo sí se cumplen una (o ambas) de las siguientes condiciones:

p_1 queda a uno de los lados de la línea que contiene a $\overline{p_3p_4}$ y p_2 en el otro ($\overline{p1p}2$ " a horcajadas de " $\overline{p_3p_4}$); p_3 queda a uno de los lado de la línea que contiene a $\overline{p_1p_2}$ y p_4 en el otro.

El extremo de un segmento cae sobre el otro segmento.

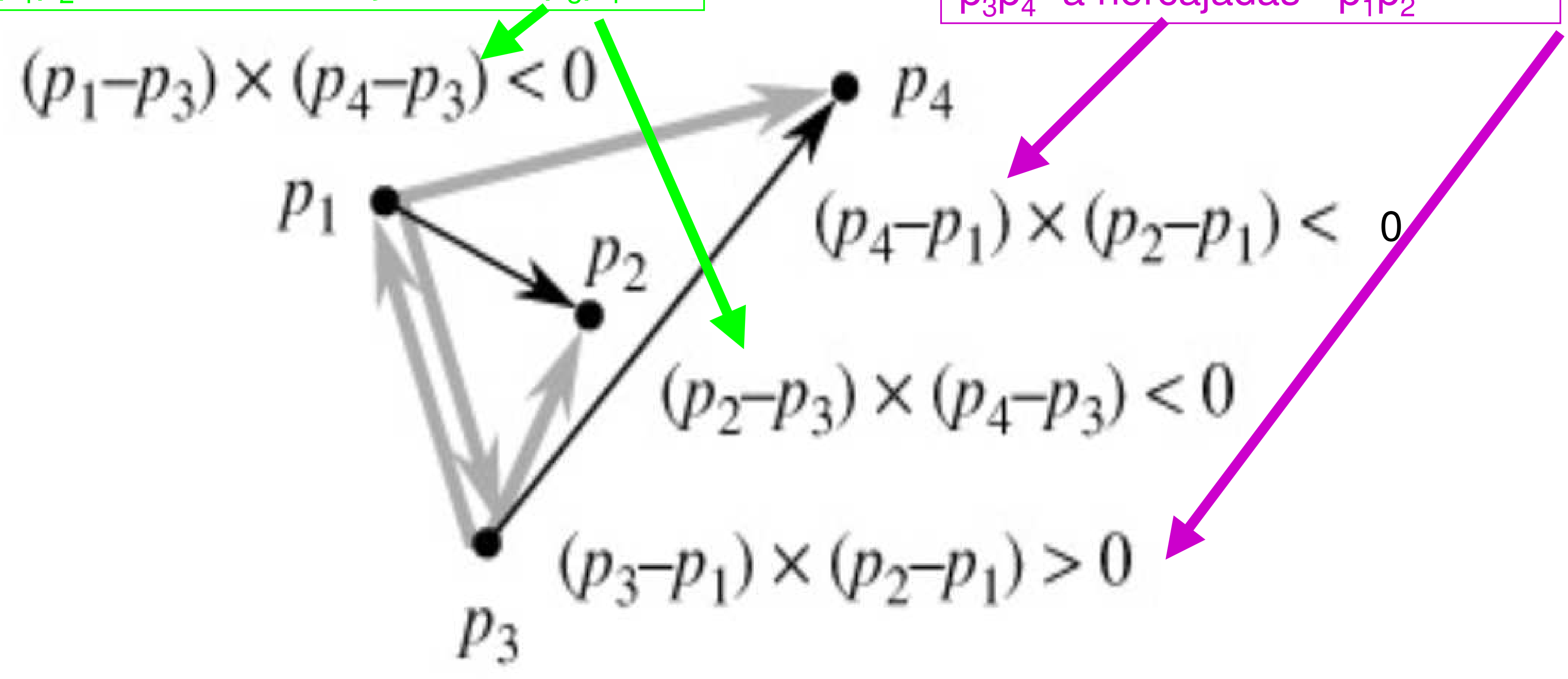
Algoritmos geométricos
Intersección de segmentos



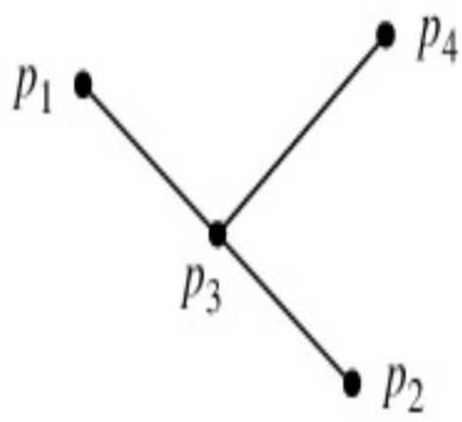
Algortimos geométricos
Intersección de segmentos

$\overline{p_1p_2}$ no está a horcajadas de $\overline{p_3p_4}$

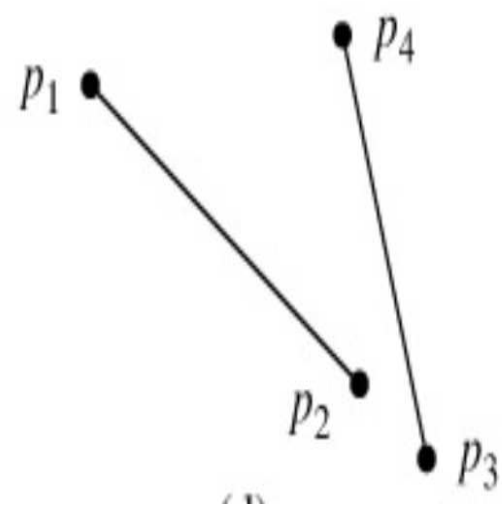
$\overline{p_3p_4}$ “a horcajadas “ $\overline{p_1p_2}$



Algoritmos geométricos
Intersección de segmentos



p3 colineal con $\overline{p1p2}$ y entre p1 y p2



p3 colineal con $\overline{p1p2}$, pero no está entre p1 y p2

Algoritmos geométricos
Intersección de segmentos

DIRECTION(pi, pj, pk)
{**return** $(pk - pi) \times (pj - pi)$;}

ON-SEGMENT(pi, pj, pk)
{**if** $\min(xi, xj) \leq xk \leq \max(xi, xj)$ and
$\min(yi, yj) \leq yk \leq \max(yi, yj)$
return TRUE;
else return FALSE ;
}

Algoritmos geométricos
Intersección de segmentos

SEGMENTS-INTERSECT($p1$, $p2$, $p3$, $p4$)

{

d1= DIRECTION($p3$, $p4$, $p1$);

d2= DIRECTION($p3$, $p4$, $p2$);

$d3$ = DIRECTION($p1$, $p2$, $p3$);

$d4$ = DIRECTION($p1$, $p2$, $p4$);

Algoritmos geómetricos
Intersección de segmentos

```
 if ((d1 > 0 and d2 < 0) or (d1 < 0 and d2 > 0)) and
      ((d3 > 0 and d4 < 0) or (d3 < 0 and d4 > 0))
return TRUE;
else if  d1 = 0 and ON-SEGMENT(p3, p4, p1)
      return TRUE;
     else if d2 = 0 and ON-SEGMENT(p3, p4, p2)
           return TRUE;
            else if d3 = 0 and ON-SEGMENT(p1, p2, p3)
                   return TRUE;
                    else if d4 = 0 and ON-SEGMENT(p1, p2, p4)
                          return TRUE;
                         else return FALSE;
 }
```

Algoritmos geométricos
Intersección entre varios segmentos

"Dados n segmentos, determinar si existe o no intersección entre pares de segmentos"

Se resolverá mediante una técnica de barrido, común en aplicaciones de geometría computacional.

Se supone una línea imaginaria que pasa a través de objetos geométricos, usualmente para ubicarlos en una estructura dinámica, y detectar relaciones entre ellos.

Simplificaciones:

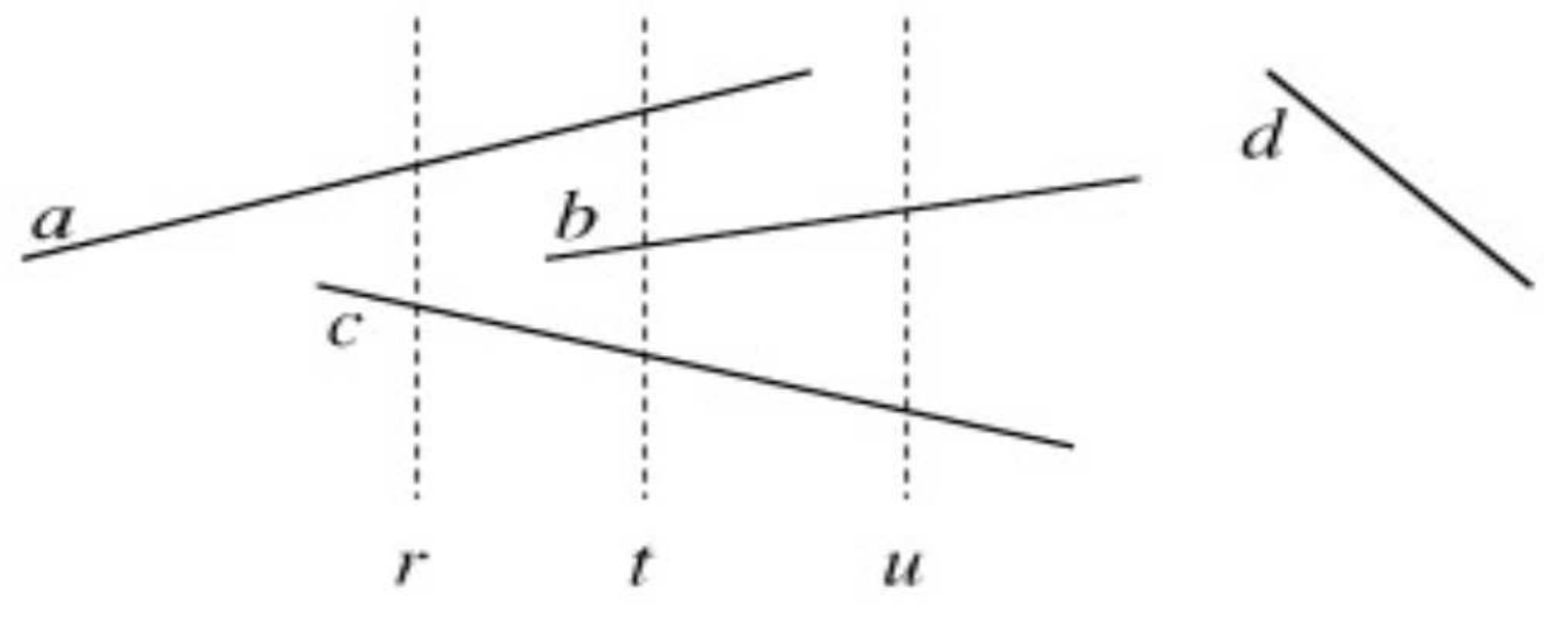
- No existen segmentos verticales
- No se cruzan 3 segmentos en el mismo punto

Algoritmos geométricos
Intersección entre segmentos

Dado que no hay segmentos verticales, la intersección de un segmento con una línea de barrido es un punto.

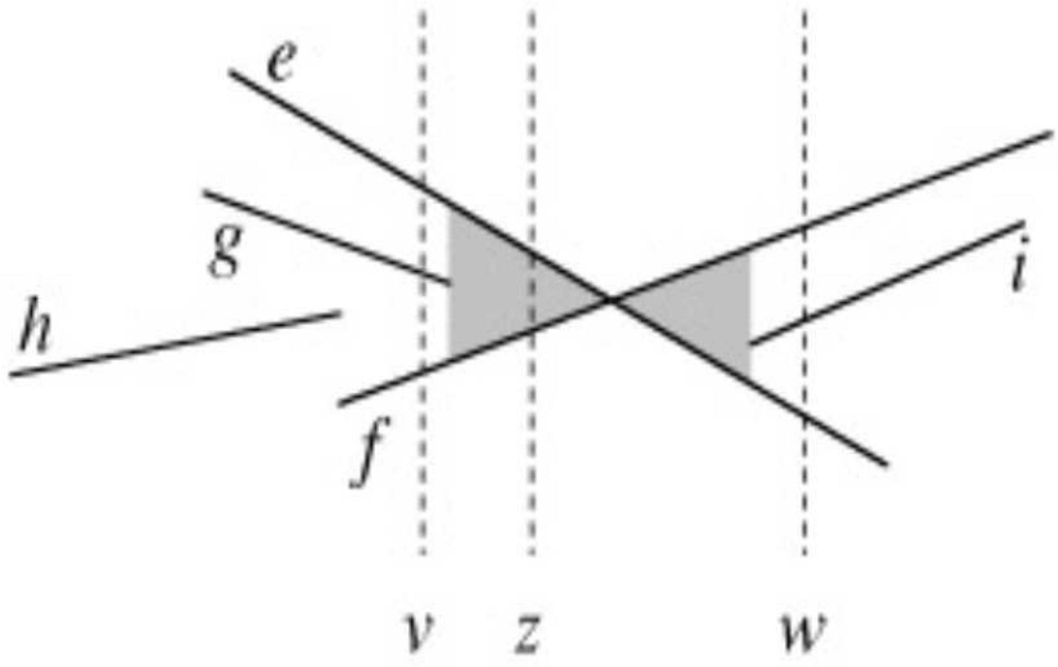
Las intersecciones sobre una misma línea de barrido pueden ordenarse por su coordenada y.

Algoritmos geométricos
Intersección entre segmentos



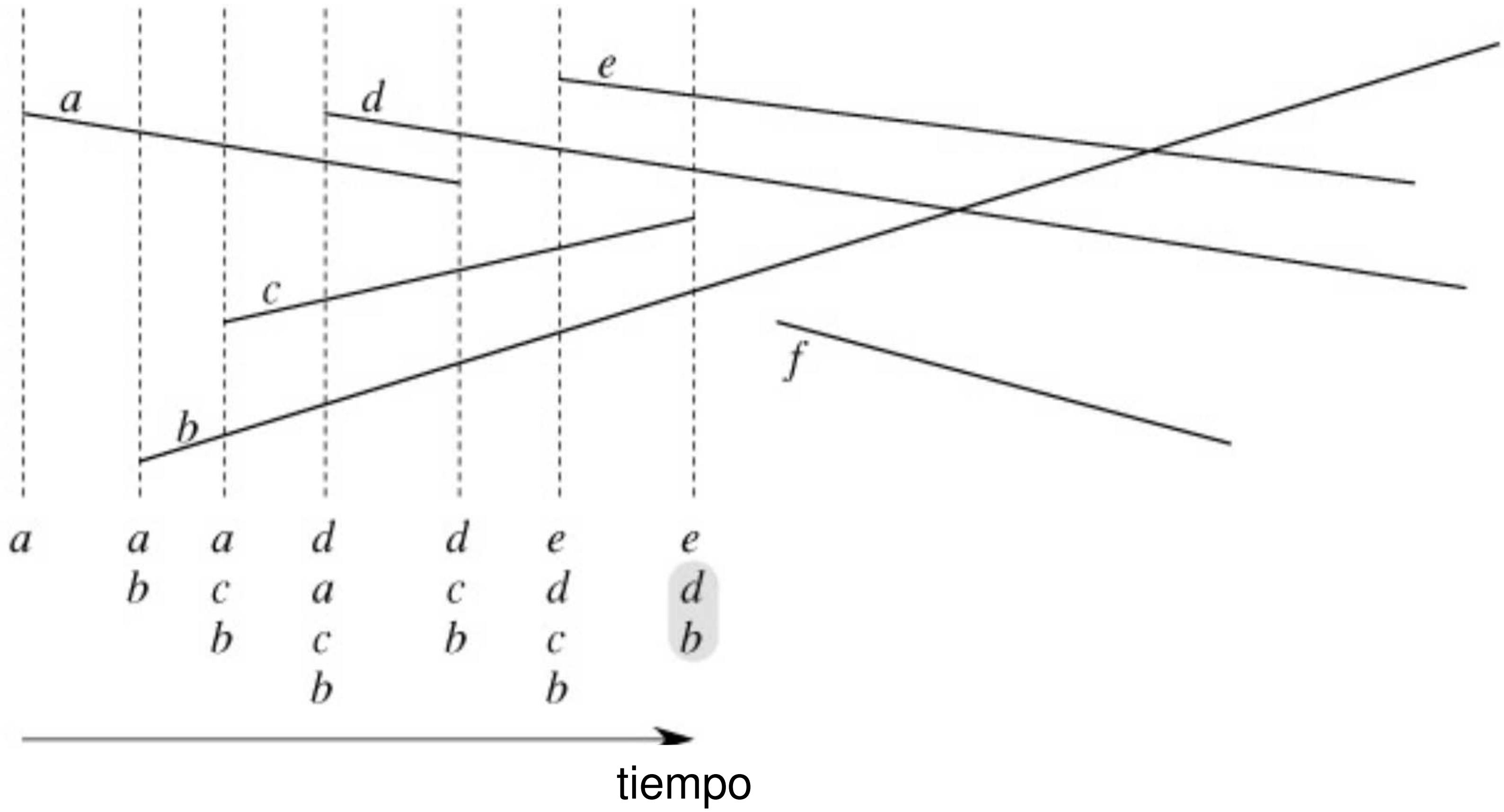
$a >_r c$, $a >_t b$, $b >_t c$, $b >_u c$ d no es comparable con a, b y c

Algoritmos geométricos Intersección entre segmentos



Cuando e y f se cruzan, sus órdenes se invierten ($e >_v f$ y $f >_w v$)

Algoritmos geométricos
Intersección entre segmentos



Algoritmos geométricos Intersección entre segmentos

Los algoritmos de barrido administran dos tipos de datos:

- El estado de la línea de barrido
- La secuencia de puntos de eventos

El estado de la línea de barrido es un orden total T, con las siguientes operaciones asociadas

Insert(T,s): inserta el segmento s en T

Delete(T,s) elimina el segmento s de T

Above(T,s): retorna el segmento predecesor de s en T

Below(T,s): retorna el segmento sucesor de s en T

Algoritmos geométricos
Intersección de segmentos

ANY-SEGMENTS-INTERSECT(*S*)
T = Ø;
-- ordenar los extremos de segmentos en S
for cada punto *p* en la lista ordenada de extremos
{**if** *p* es extremo izquierdo de un segmento s
 Insert(*T, s*);
  if (Above(*T, s*) existe e interseca a *s*) o (Below(*T, s*) existe e interseca *s*)
   return TRUE;
   if *p* es extremo derecho del segmento s
     if ambos Above (*T, s*) y Below (*T, s*) existen y
       Above(*T, s*) interseca BELOW(*T, s*)
     return TRUE;
     DELETE(*T, s*);
return FALSE;
 }

Algoritmos geométricos
Intersección de segmentos

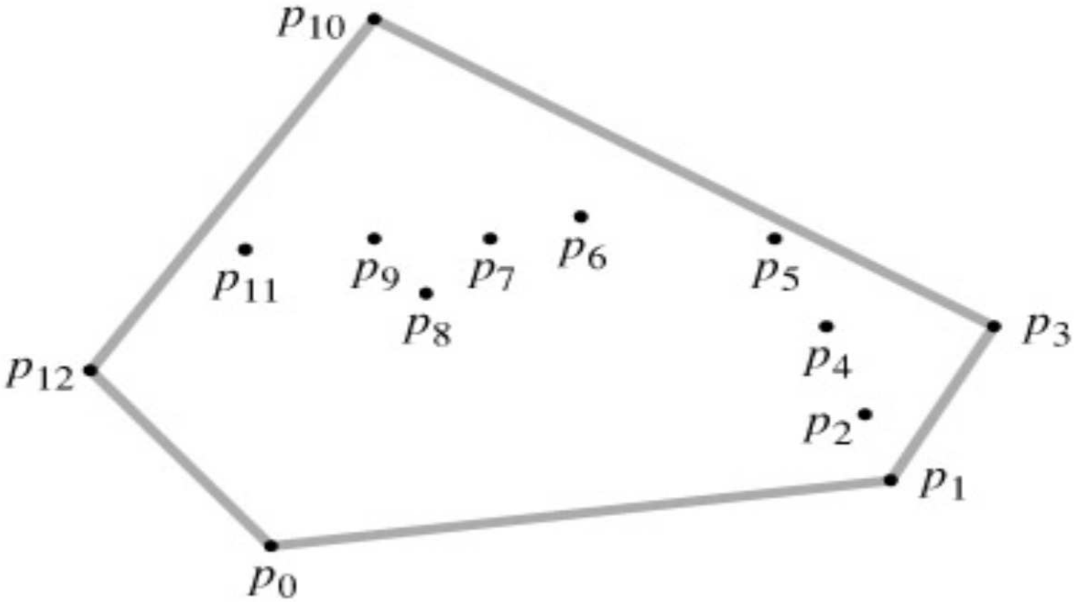
Sugerencias

Identifique los TDA que intervienen en este problema.

Analice diferentes representaciones para los TDA involucrados.

¿Es posible lograr una implementación cuyo tiempo de ejecución esté acotado, para n segmentos, por O(n log n)?

Algoritmos geométricos
Problema del “polígono convexo”



Encontrar el “menor” polígono convexo que contenga a los puntos de un conjunto Q. Cada punto es interior al polígono o pertenece a uno de sus lados.

Un polígono es convexo si todo segmento cuyos extremos son puntos interiores del polígono no corta a un segmento de borde

Algoritmos geométricos
Problema del "polígono convexo"

Algoritmos basados en "barrido rotacional":

procesan vértices en el orden de sus coordenadas polares

los ángulos que forman con referencia a un vértice

Analizaremos dos algoritmos:

Graham O(n log n)

Jarvis O(n h)

(h es el número de vértices en el menor polígono)

ALGORITMOS GEOMÉTRICOS ALGORITMO DE GRAHAM

GRAHAM-SCAN(*Q*)

{Sea *p*0 el punto en Q con mínima coordenada y, o el más a la izquierda (si es una cuerda) ;

Sea ⟨*p*1, *p*2, ..., *pm*⟩ los restantes puntos de Q ordenados por sus ángulos en sentido antihorario con respecto a *p*0 (si hay más de uno, remover a todos y dejar sólo al más alejado) ;

PUSH(*p*0, *S*);

PUSH(*p*1, *S*);

PUSH(*p*2, *S*);

for(*i* =3; i <=m; i++)

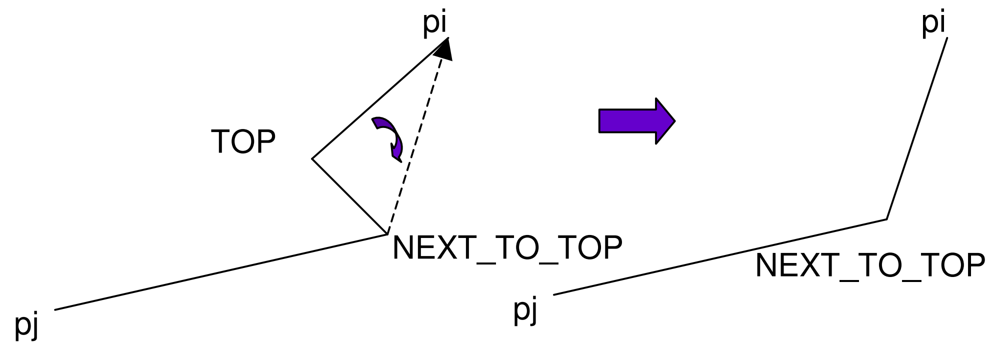
{while (el ángulo formado por los puntos NEXT-TO-TOP(*S*), TOP(*S*), y *pi* gira a la derecha)

POP(*S*);

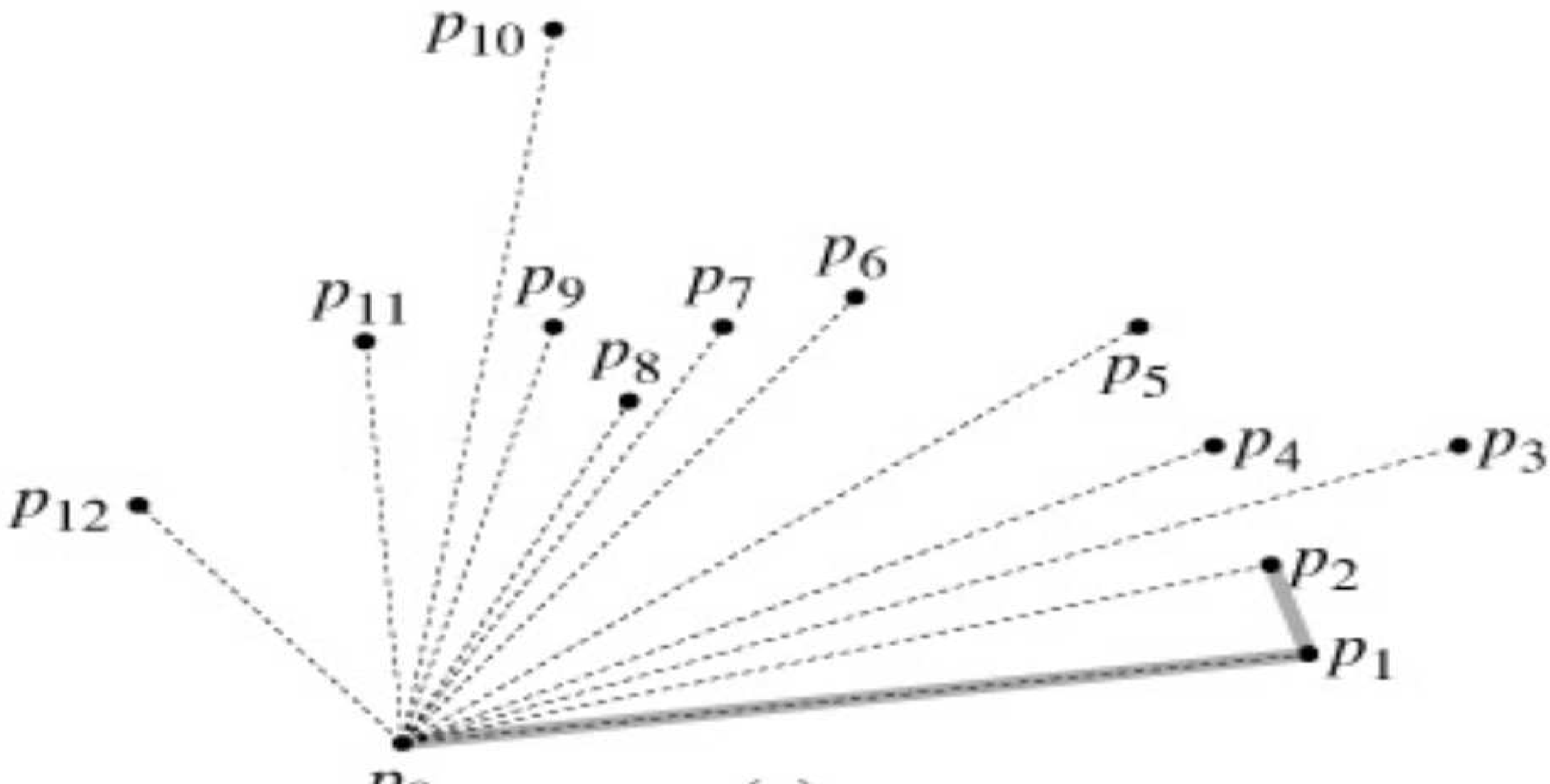
PUSH(*pi, S*);}

return *S;*

}

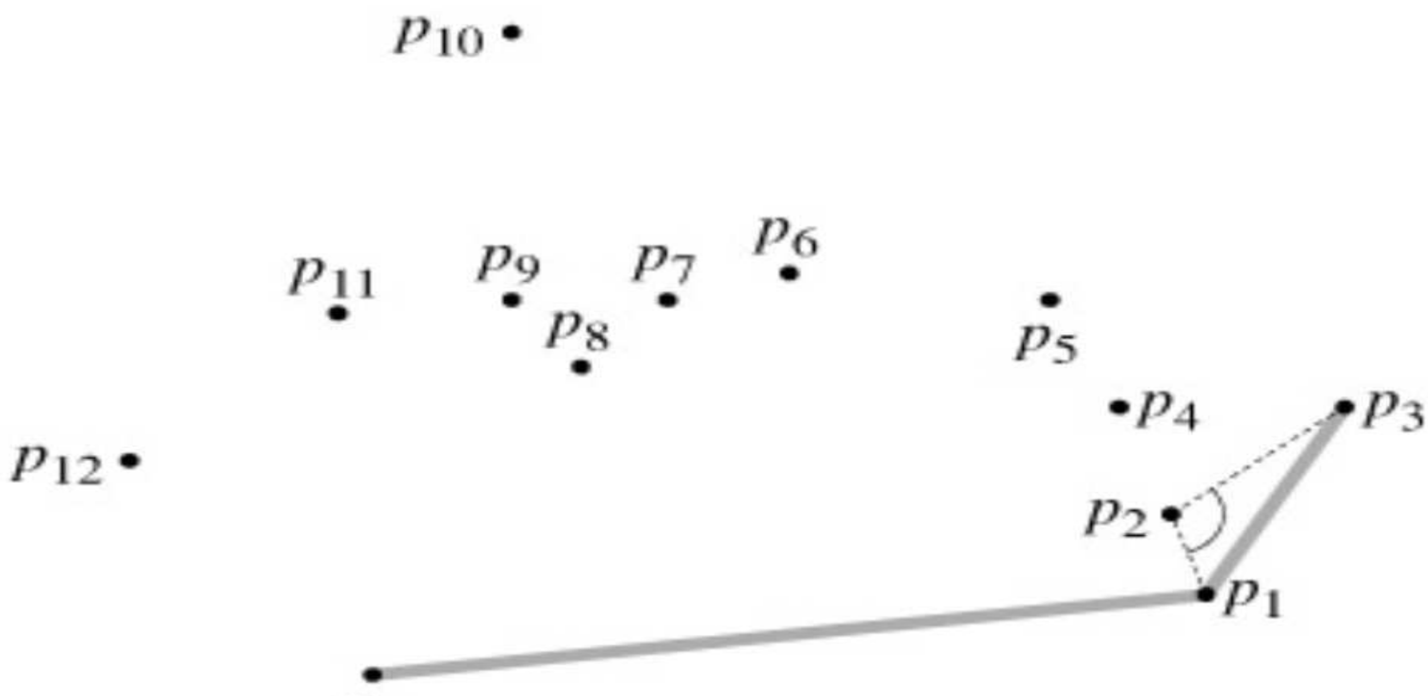


Algoritmos geométricos
Algoritmo de Graham



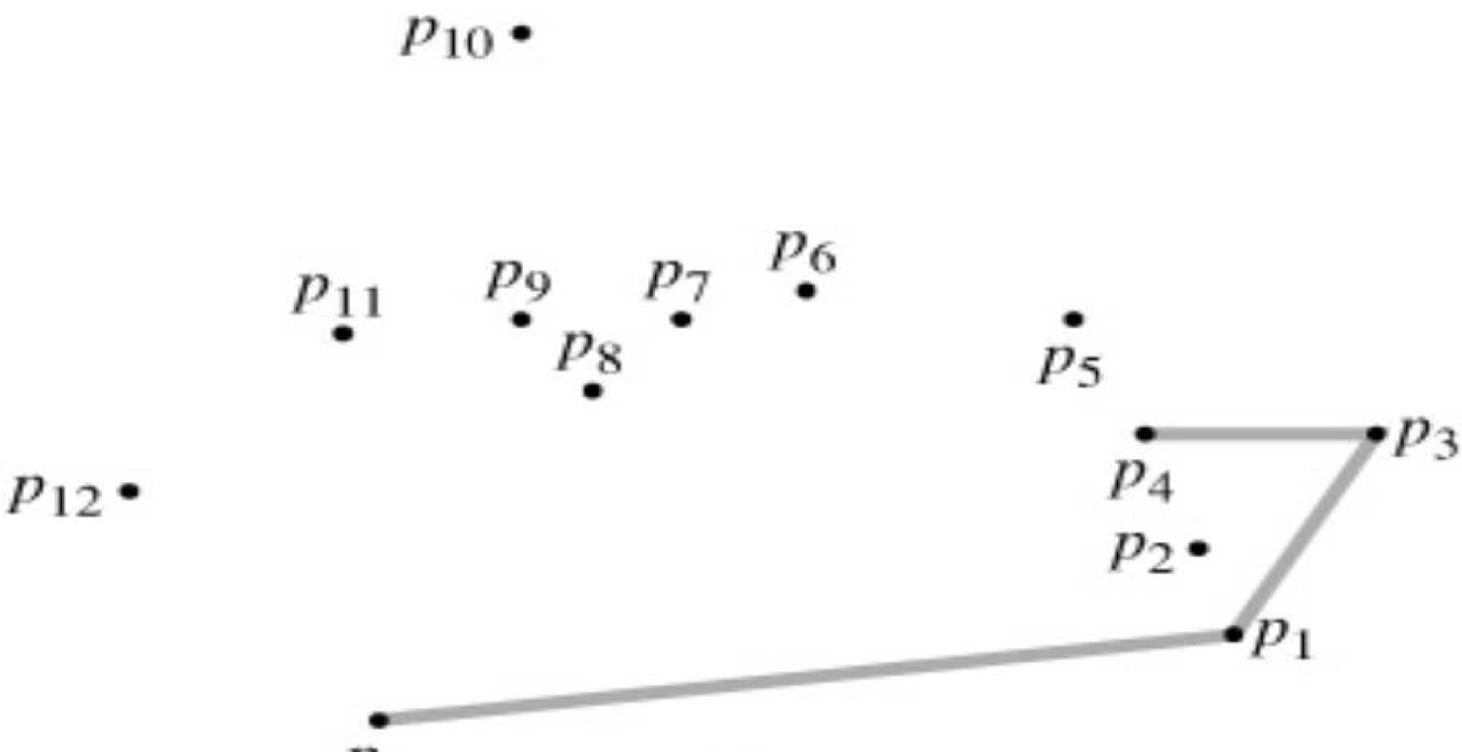
(a)

Algoritmos geométricos
Algoritmo de Graham



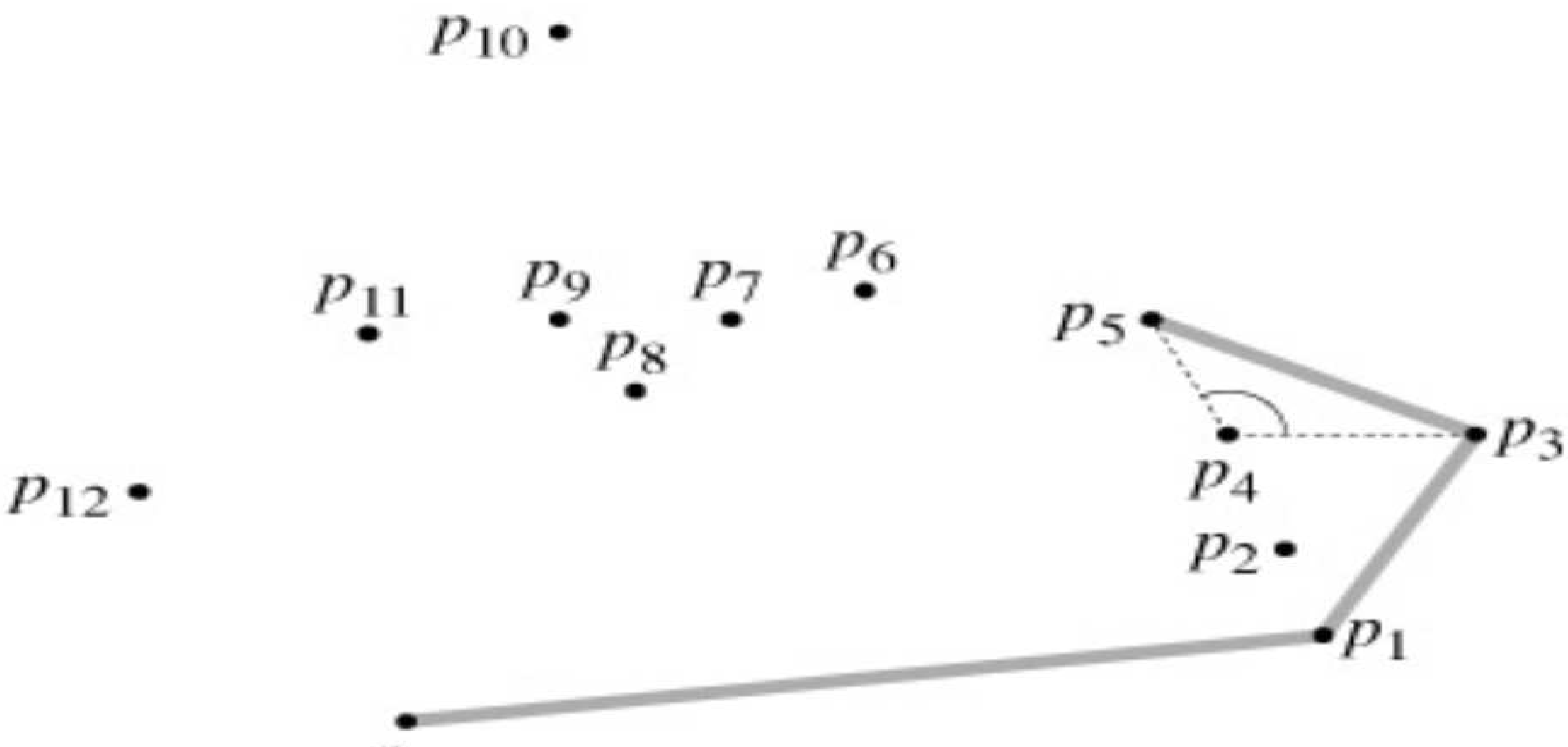
(b)

Algoritmos geométricos
Algoritmo de Graham



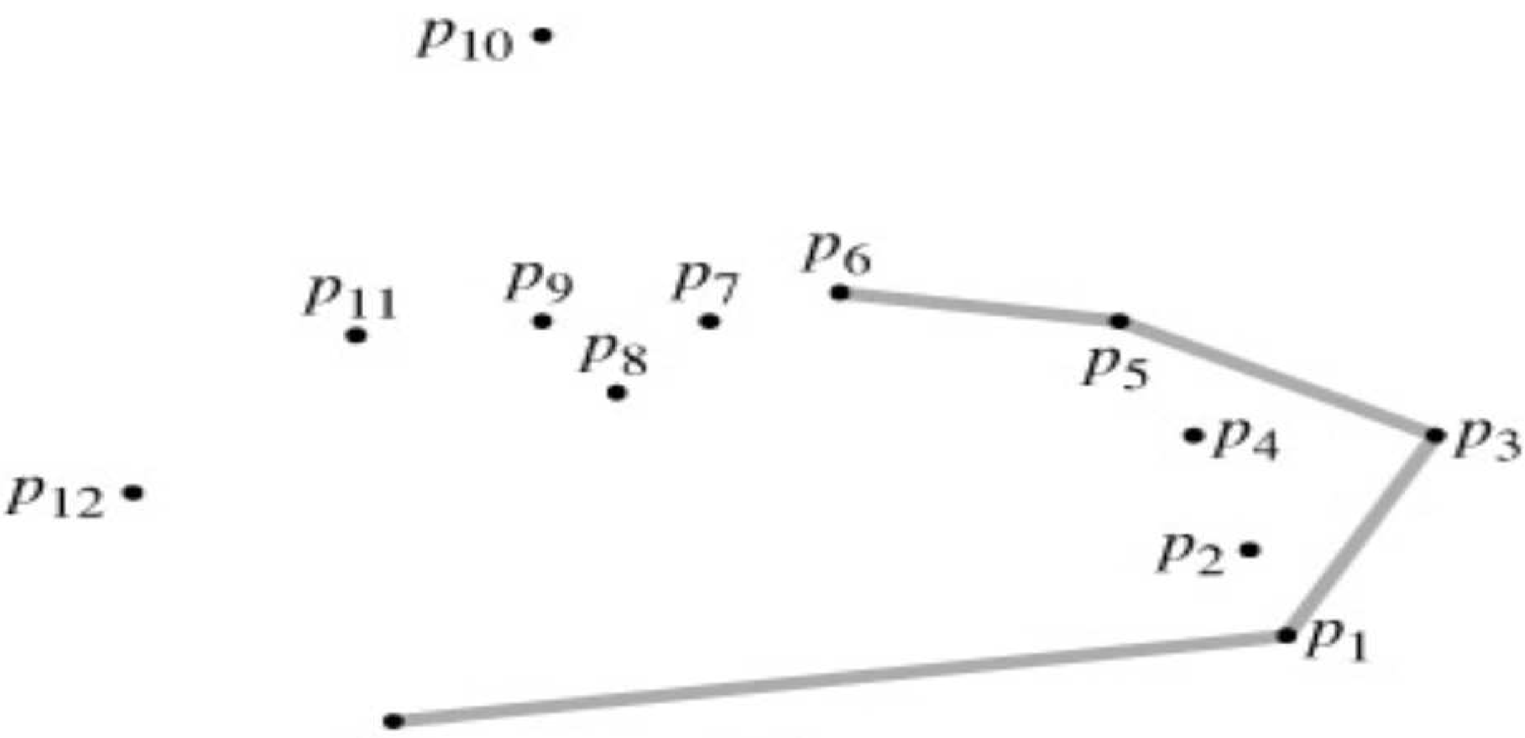
(c)

Algoritmos geométricos
Algoritmo de Graham



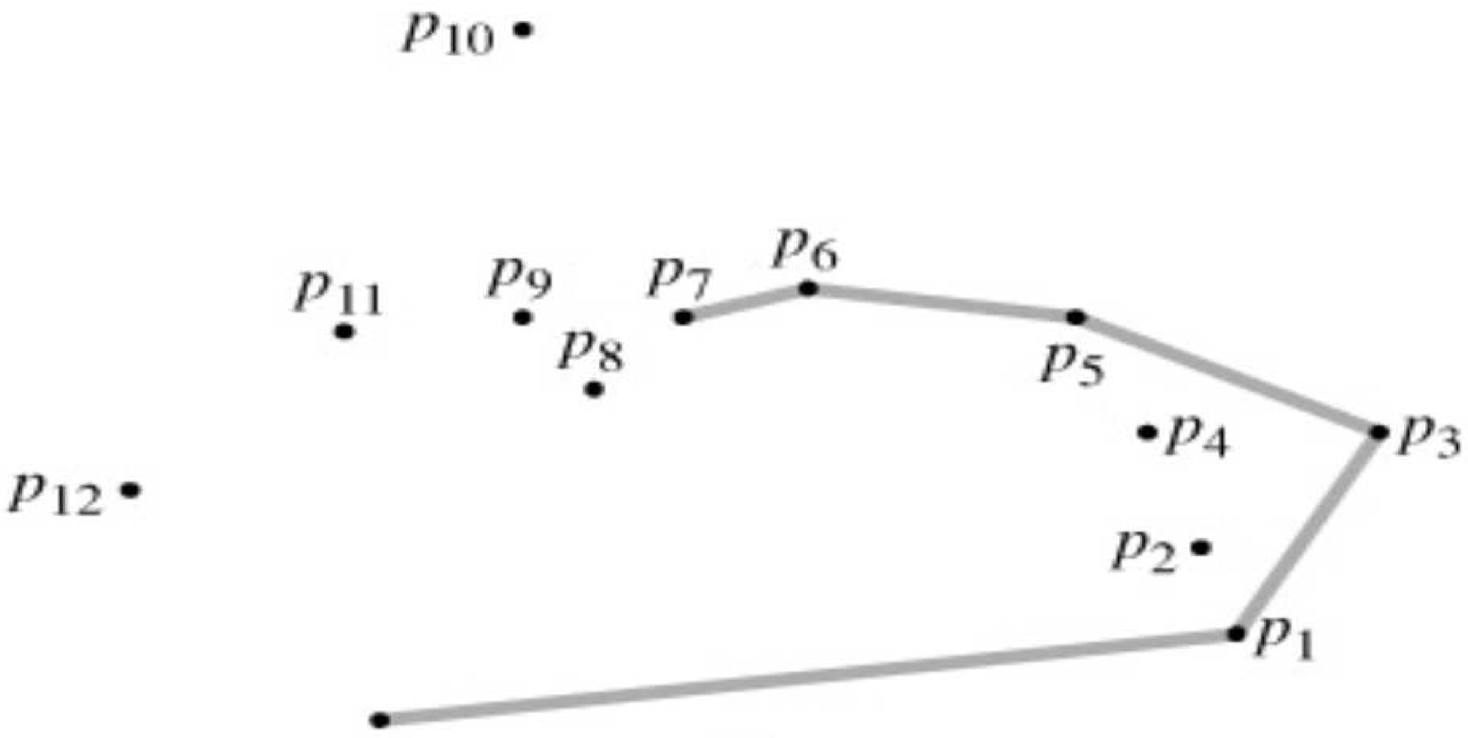
(d)

Algoritmos geométricos
Algoritmo de Graham



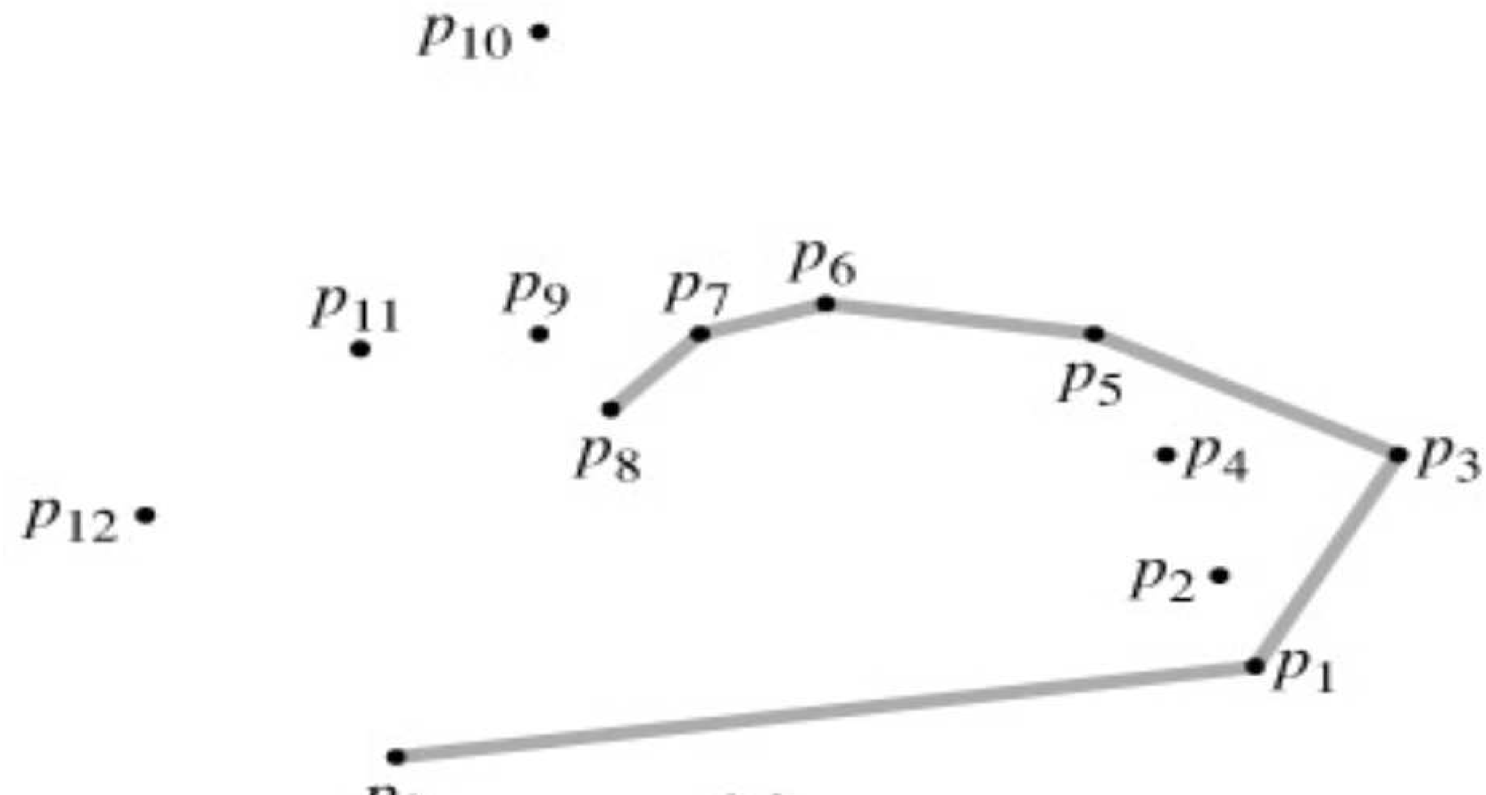
(e)

Algoritmos geométricos
Algoritmo de Graham



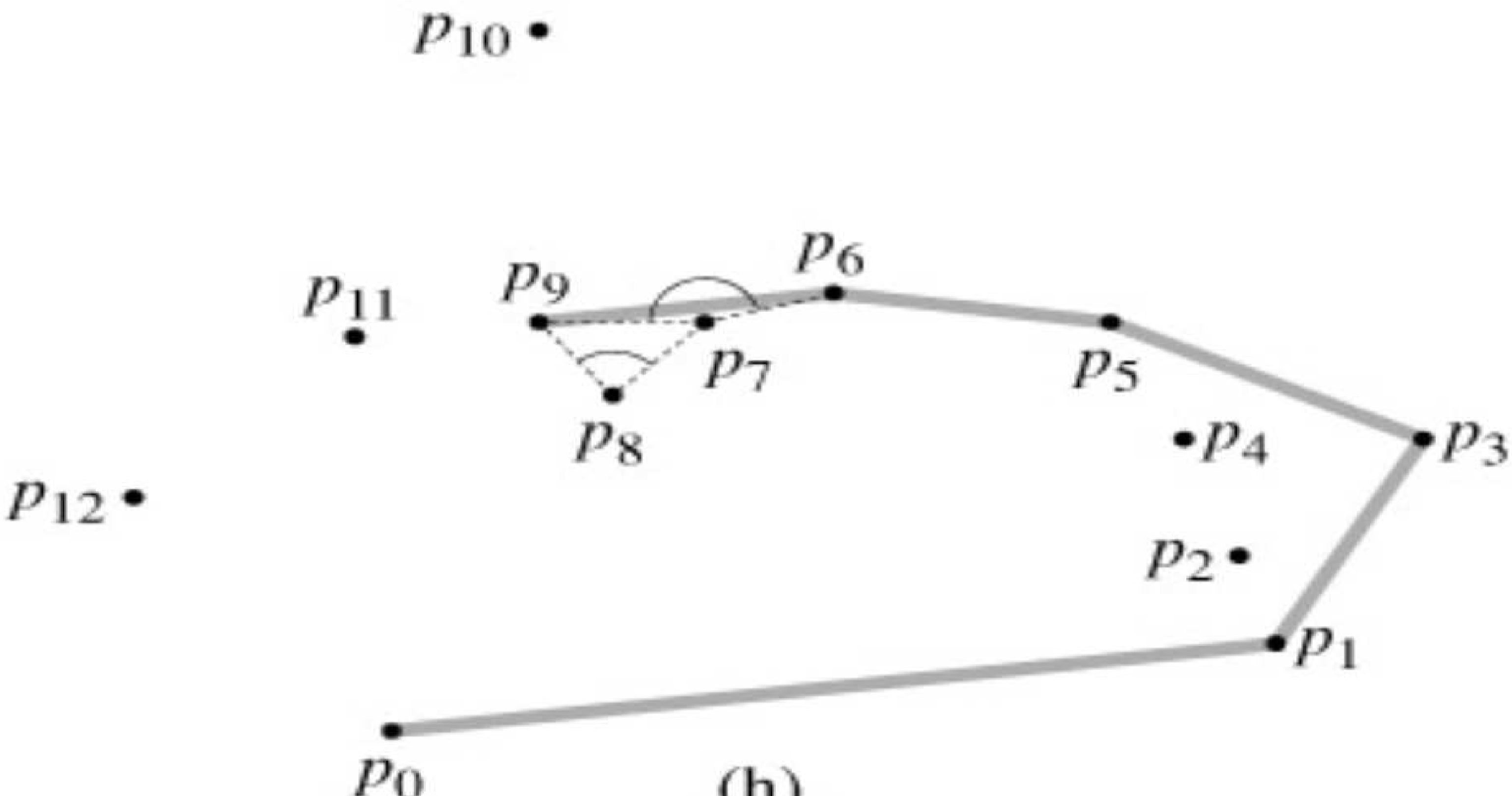
(f)

Algoritmos geométricos
Algoritmo de Graham



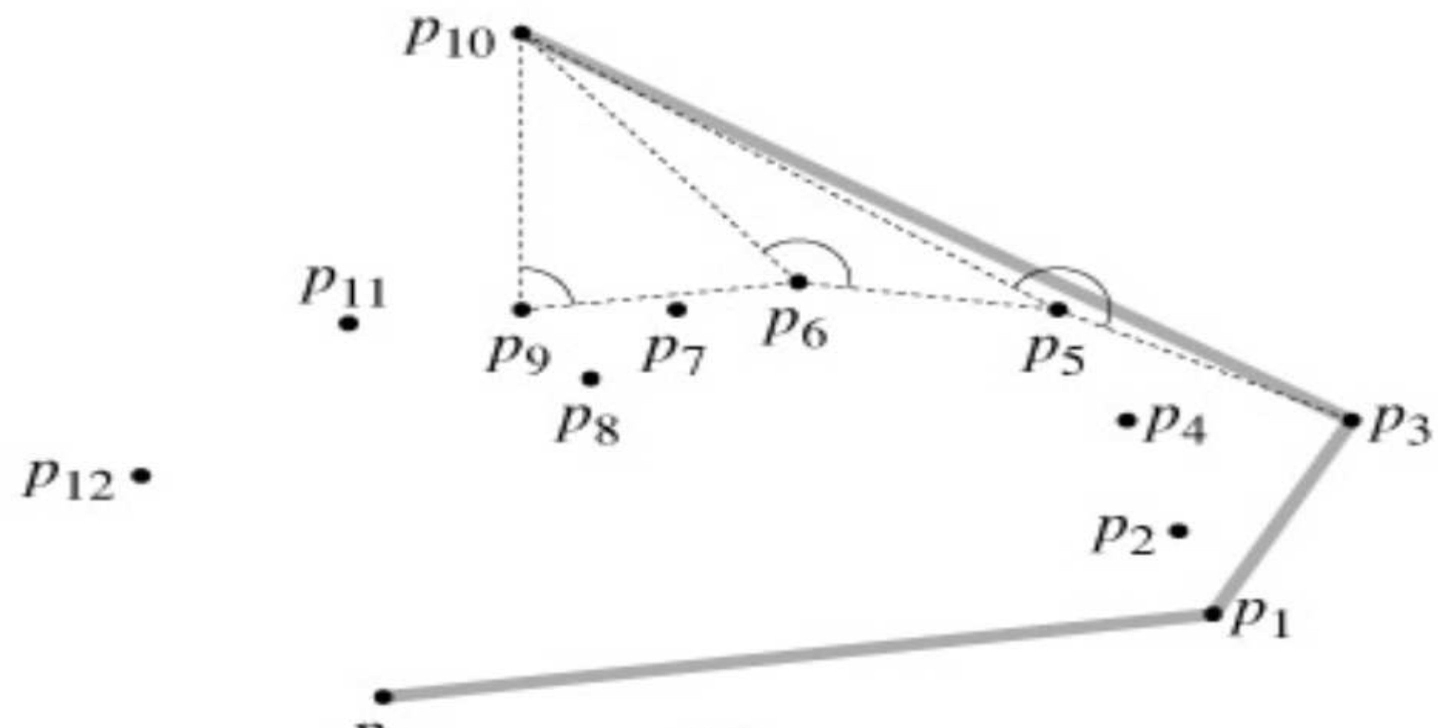
(g)

Algoritmos geométricos
Algoritmo de Graham



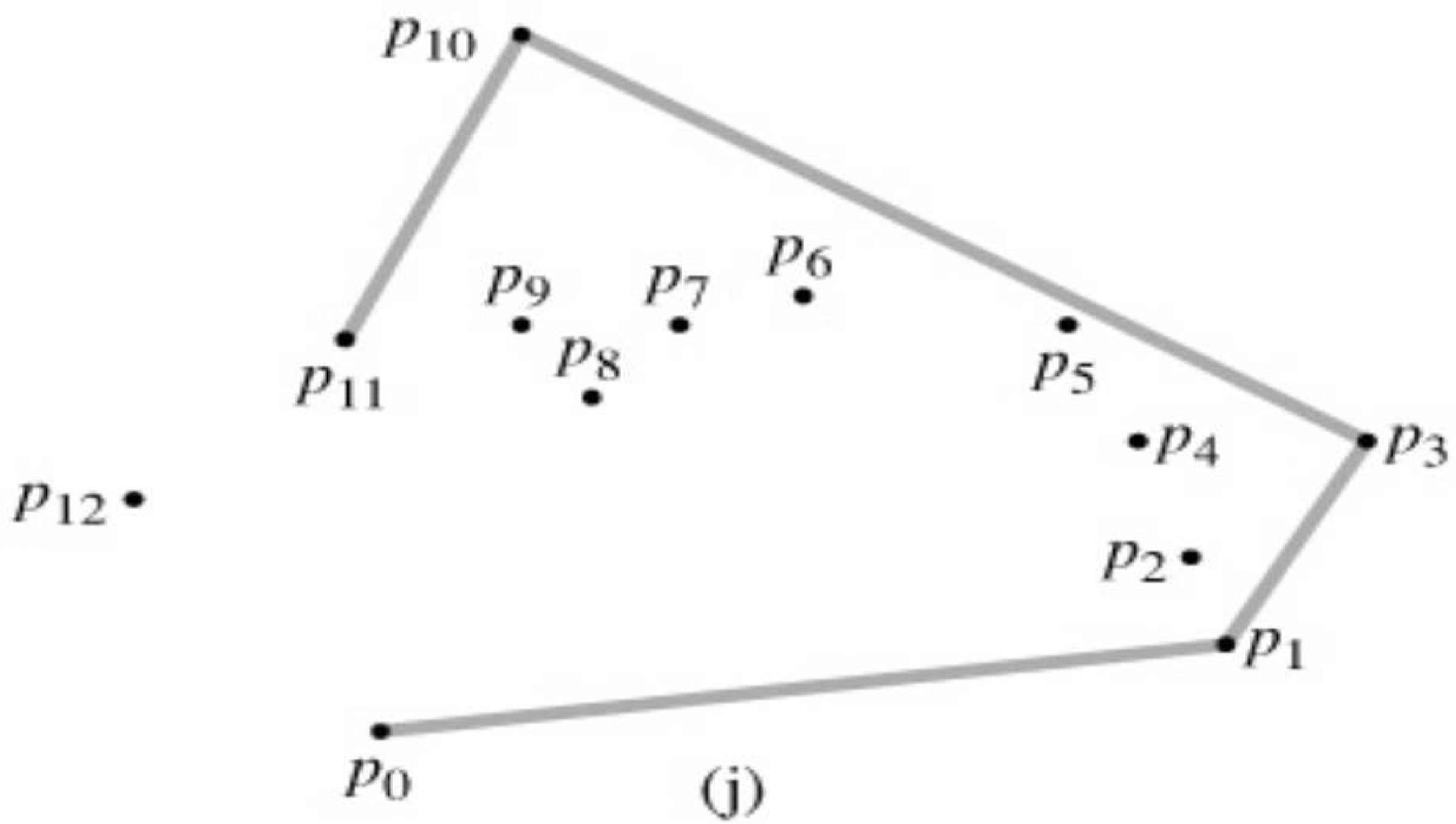
(h)

Algoritmos geométricos
Algoritmo de Graham

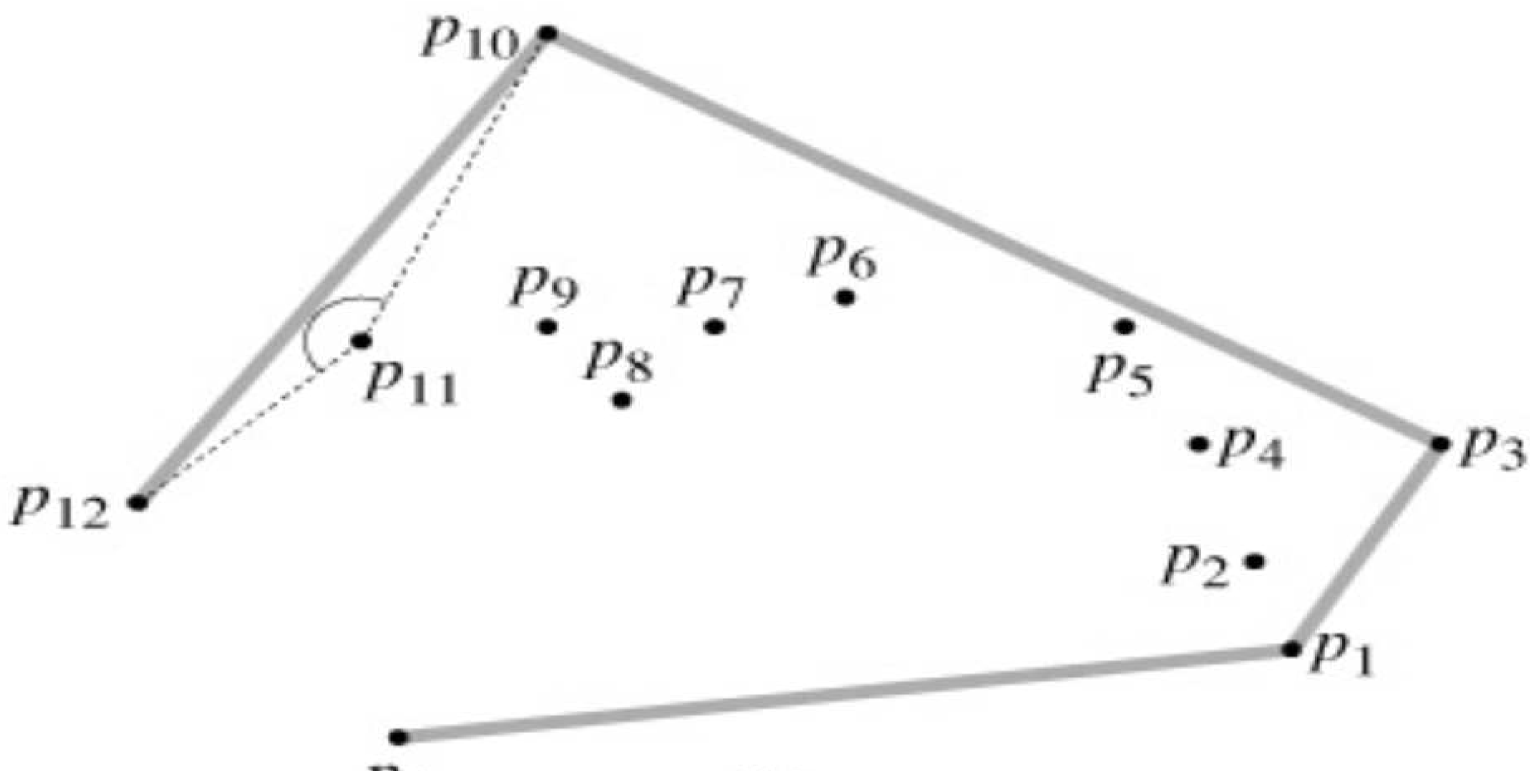


(i)

Algoritmos geométricos
Algoritmo de Graham

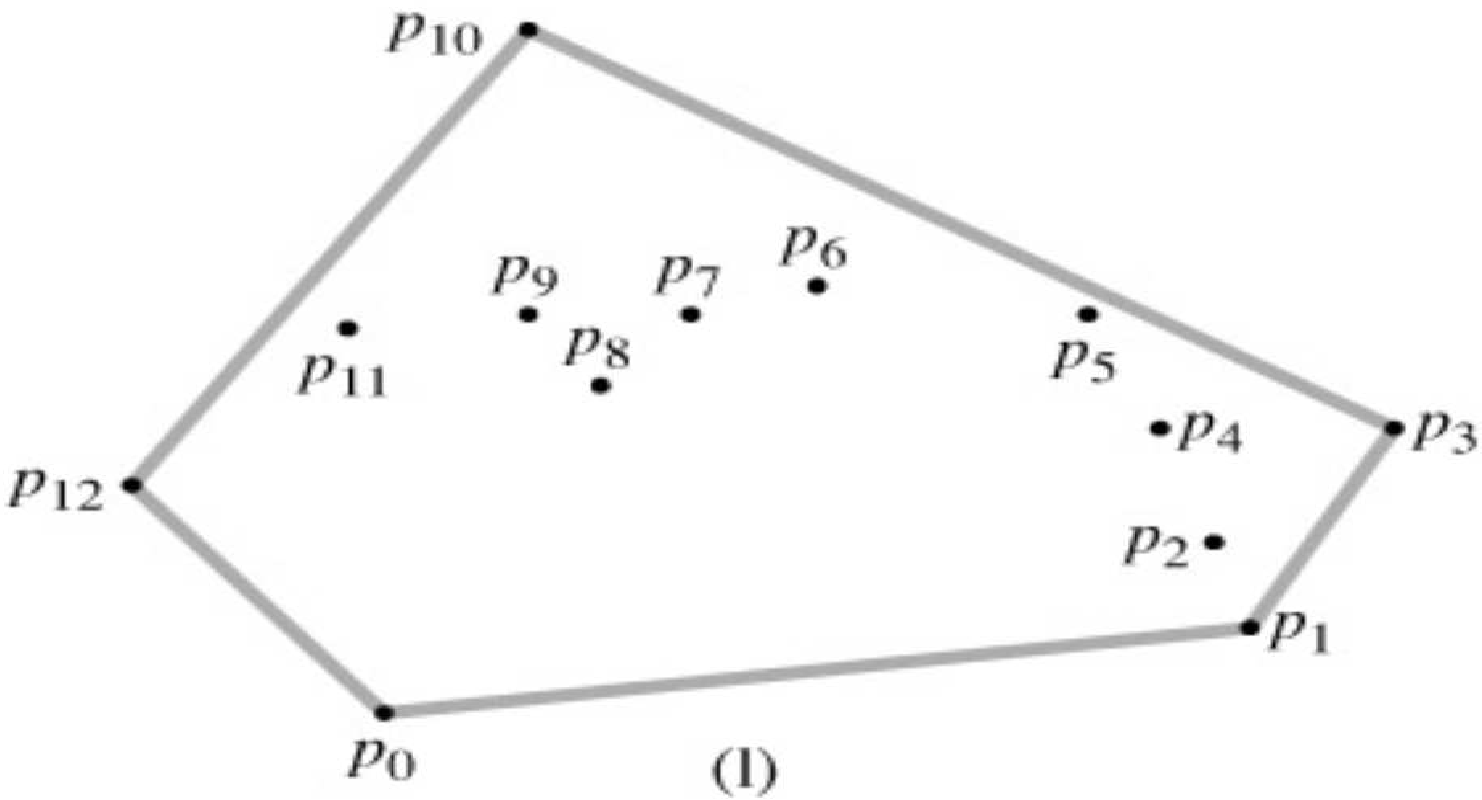


Algoritmos geométricos
Algoritmo de Graham



(k)

Algoritmos geométricos
Algoritmo de Graham



Algoritmos geométricos
Algoritmo de Graham

GRAHAM-SCAN(*Q*)

{Sea *p*0 el punto en Q con mínima coordenada y, o el más a la izquierda (si es una cuerda) ;

Sea ⟨*p*1, *p*2, ..., *pm*⟩ los restantes puntos de Q ordenados por sus ángulos en sentido antihorario con respecto a *p*0 (si hay más de uno, remover a todos y dejar sólo al más alejado) ;

PUSH(*p*0, *S*);

PUSH(*p*1, *S*);

PUSH(*p*2, *S*);

for(*i* =3; i <=m; i++)

{**while** (el ángulo formado por los puntos NEXT-TO-TOP(*S*), TOP(*S*), y *pi* gira a la derecha)

POP(*S*);

PUSH(*pi, S*);}

return *S;*

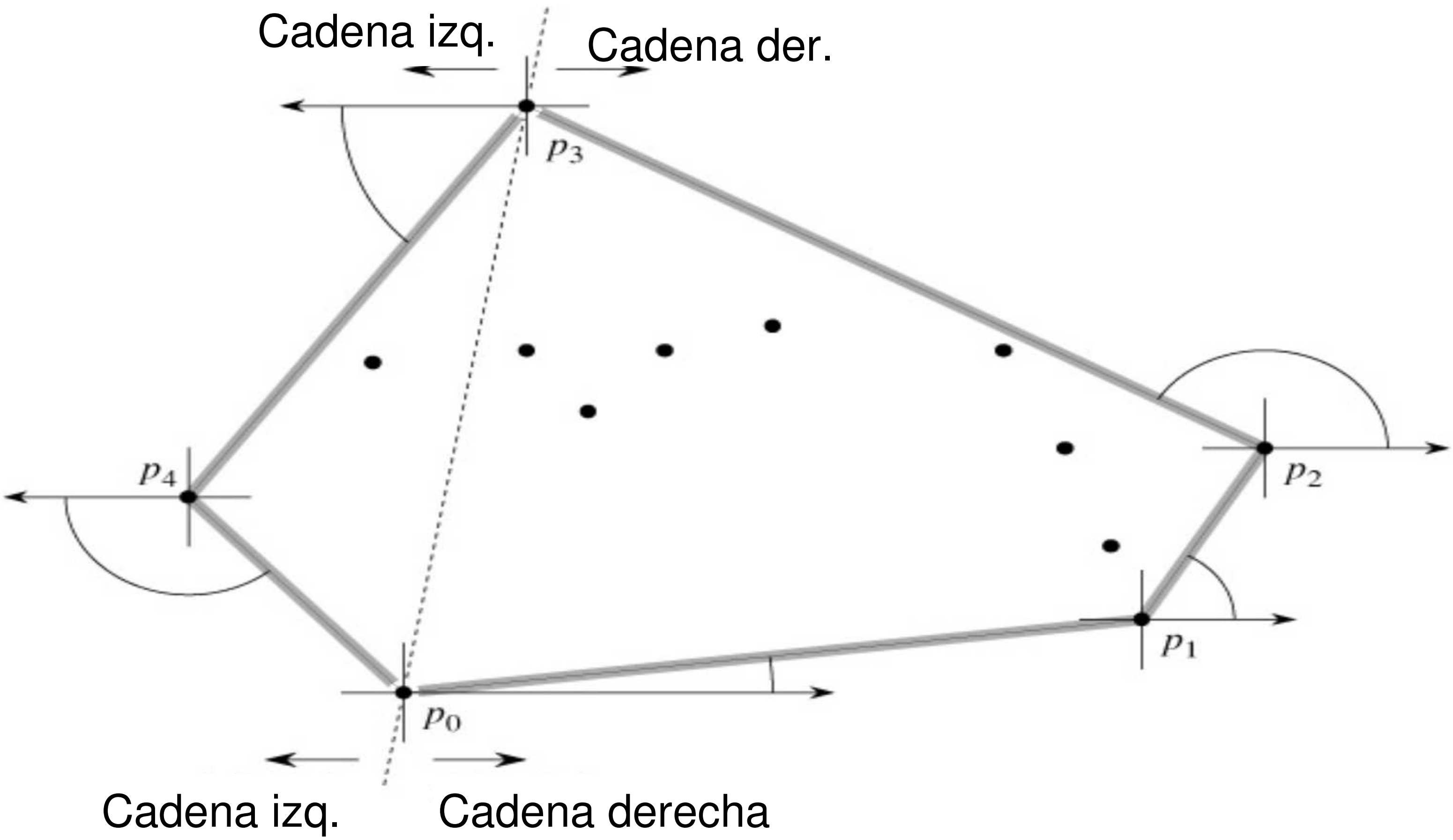
}

Algoritmos geométricos
Algoritmo de Jarvis

El algoritmo de Jarvis construye el menor polígono convexo mediante una técnica denominada “package wrapping”

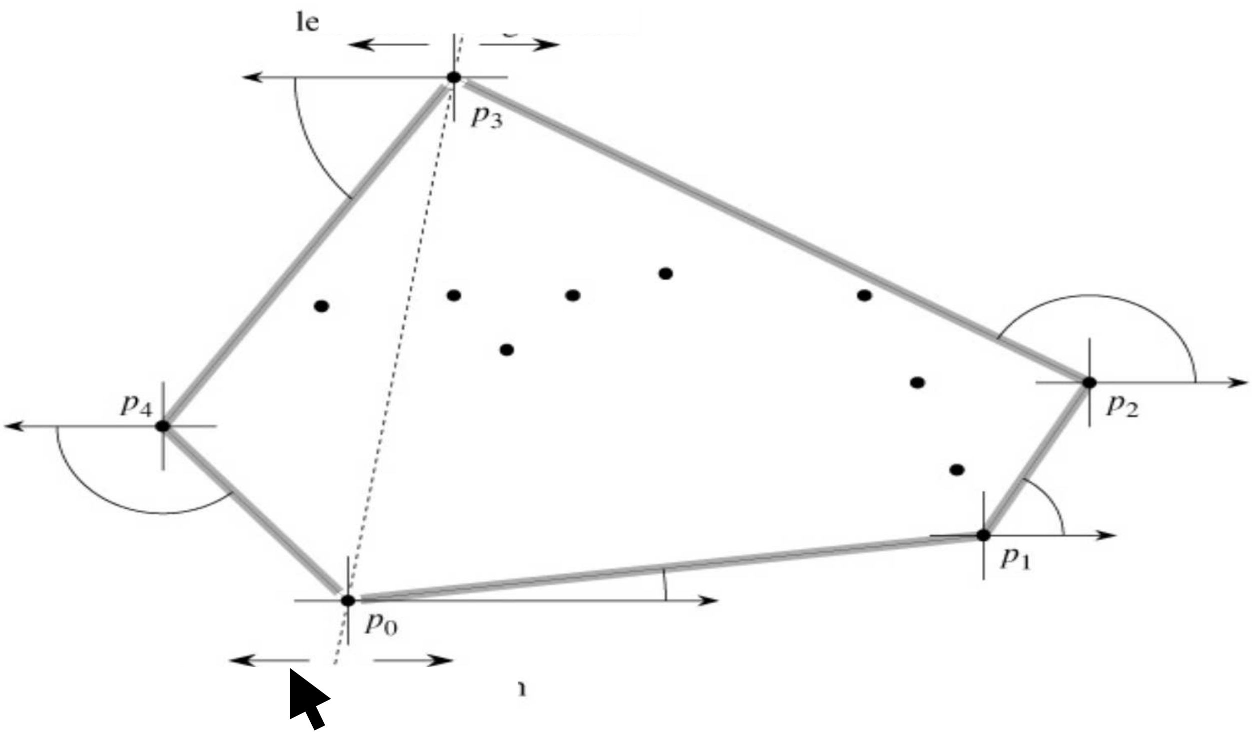
Construye una cadena derecha de vértices y una cadena izquierda seleccionado vértices a partir de un orden establecido con respecto a los vértices recientemente incorporados en la frontera.

Algoritmos geométricos
Algoritmo de Jarvis



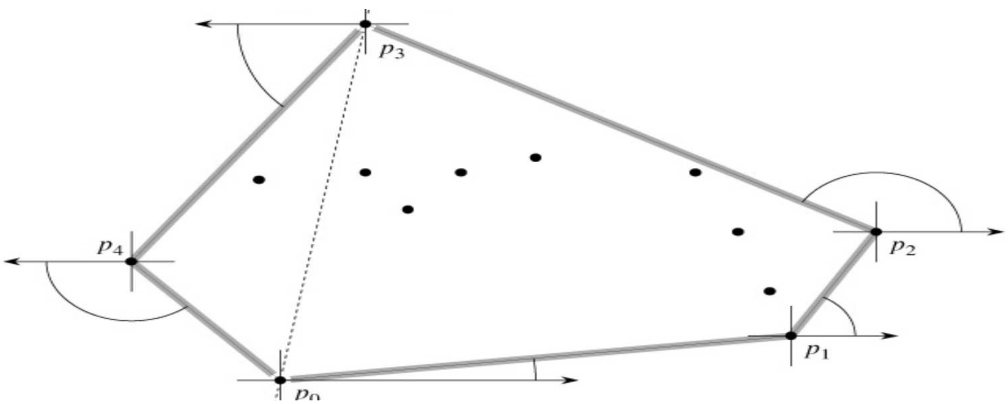
Algoritmos geométricos
Algoritmo de Jarvis

Comienza en p_o a construir la cadena derecha



Algoritmos geométricos
Algoritmo de Jarvis

El siguiente vértice p_1, forma el menor ángulo con respecto a p_0. Similarmente, p2 tiene el menor ángulo con respecto a p1…



Cuando logra el vértice más alto, en el ejemplo p_3 construyó la cadena derecha completa.Luego comienza con la izquierda eligiendo puntos que formen el menor ángulo con respecto a p_3

Algoritmos geométricos
Encontrar "los puntos más cercanos"

Dado un conjunto de puntos Q, encontrar los puntos más cercanos.

Un algoritmo de "fuerza bruta" requiere analizar $O(n^2)$ pares de puntos.

Analizaremos un algoritmo por "divide y conquista" de complejidad temporal $O(n \log n)$

Algoritmos geométricos
Encontrar “los puntos más cercanos”

Algoritmo por divide y conquista

Entrada:

un subconjunto $P \subseteq Q$

dos arreglos X e Y .

X e Y contienen a los puntos del subconjunto P ordenados crecientemente por su cordenada X y por su coordenada Y respectivamente. Usa una estrategia de pre-ordenamiento para mantener el orden sin ordenar en cada llamada recursiva.

Algoritmos geométricos
Encontrar los puntos más cercanos

Algoritmo por divide y conquista

Definición explícita

Si $|P| \leq 3$ lo resuelve por "fuerza bruta": analiza los $\binom{|P|}{2}$ pares y devuelve el mínimo.

Algoritmos geométricos
Encontrar los puntos más cercanos

Algoritmo por divide y conquista

Dividir

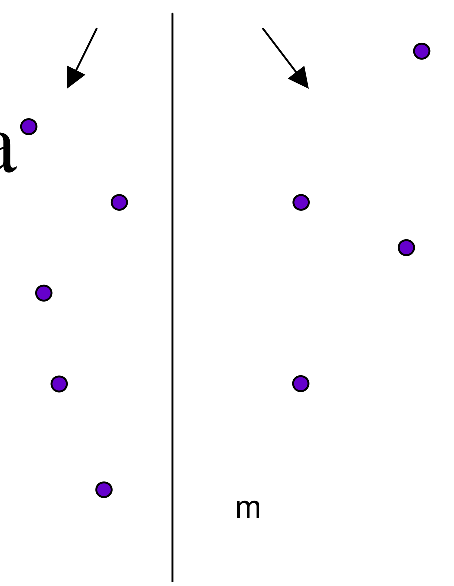
Dividir al conjunto P en dos conjuntos P_L y P_R

tal que $|P_L| = \lceil |P|/2 \rceil$ $P_R = \lfloor |P/2| \rfloor$.

Sea m la línea vertical que los separa

Dividir X en X_R y X_L

Dividir Y en Y_R y Y_L



Algoritmos geométricos
Encontrar los puntos más cercanos

Algoritmo por divide y conquista

Conquistar

Resuelve dos subproblemas

Subproblema 1

Entrada: P_L, X_L y Y_L.

Salida: el par de puntos más cercanos en P_L y la distancia δ_L.

Subproblema 2

Entrada: P_R, X_R y Y_R.

Salida: el par de puntos más cercanos en P_R y la distancia δ_R.

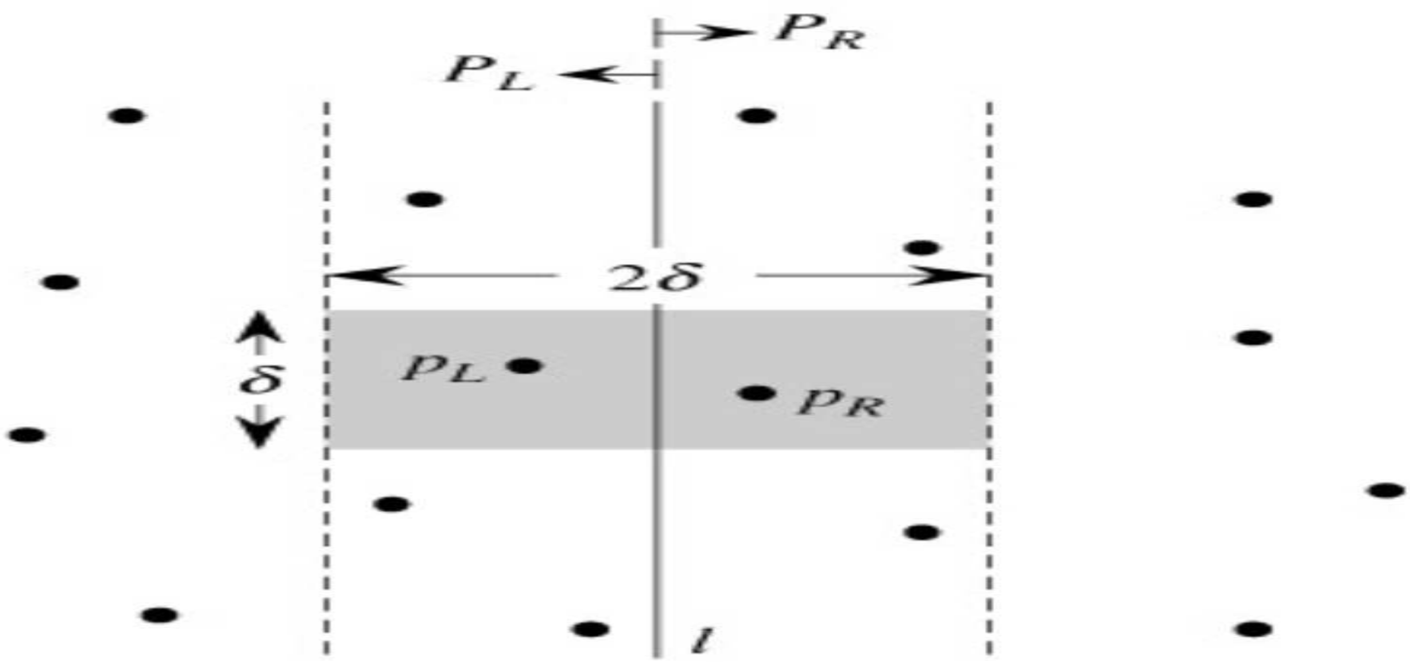
Algoritmos geométricos
Encontrar los puntos más cercanos

Algoritmo por divide y conquista

Combinar

Sea $\delta = \min(\delta_L, \delta_R)$. Determina si hay puntos a menor distancia que δ. Si existiese par de puntos, ambos puntos del par deberían estar a una distancia entre δ unidades de m

Algoritmos geométricos
Encontrar los puntos más cercanos



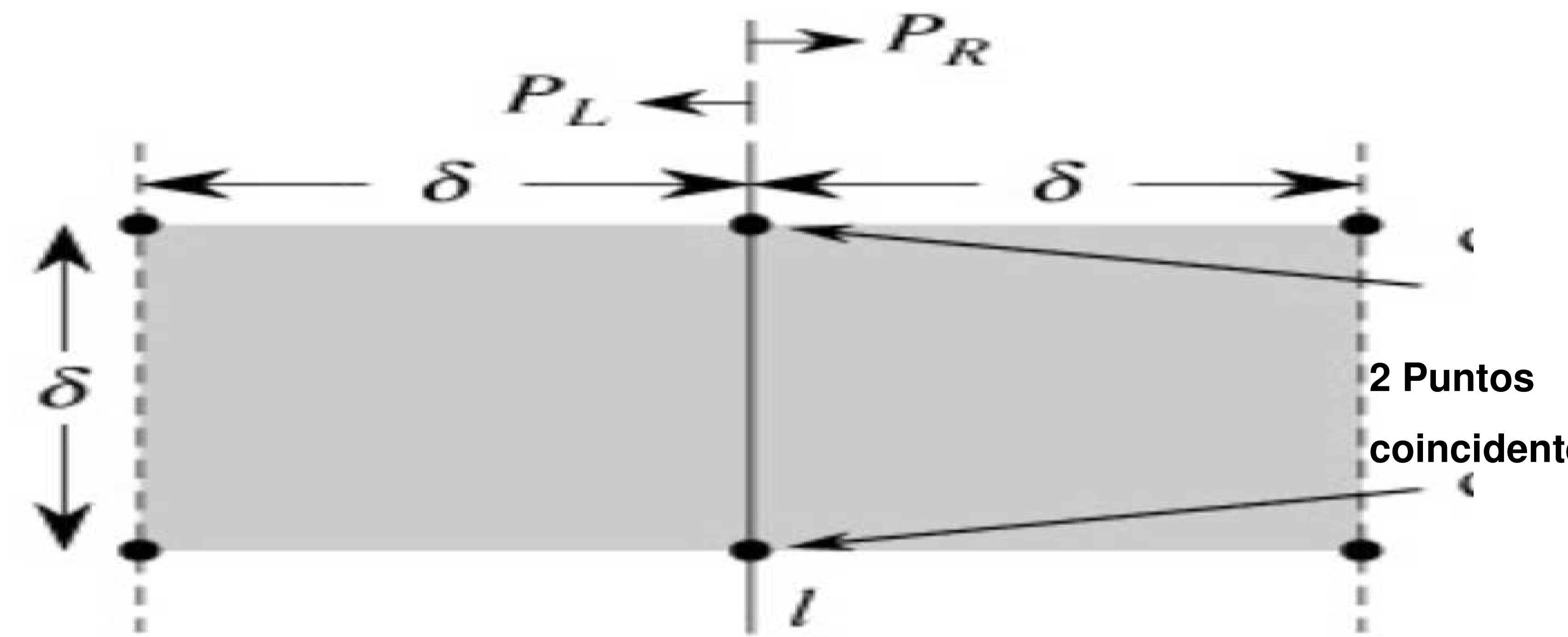
(a)

Algortimos geométricos
Encontrar los puntos más cercanos

Crea un arreglo Y' con todos los puntos que están a 2 δ. Y' es ordenado por su coordenada y. Para cada punto p en Y', el algoritmo trata de encontrar puntos en Y' que están entre las δ unidades de p.

Para cada punto p sólamente considera 7 puntos. Sea δ' la mínima distancia entre p y esos 7 puntos. Si δ'< δ, se detectó un par más cercano.

Algoritmos geométricos
Encontrar los puntos más cercanos



Cada punto sólo debe compararse con 7 puntos!!!

Algoritmos geométricos
Encontrar los puntos más cercanos

Sugerencias

Implemente en C++

¿Es posible lograr implementaciones de complejidad temporal O(n log n)?