Algoritmos Geométricos

Héctor Navarro. UCV

Estructuras de Datos

- Puntos

```
struct Punto2D{
 float x, y;
};

struct Punto3D{
 float x,y,z;
};
```

Estructuras de Datos

- Polígonos

```
Punto2D Poly[N];
```

- Rectas

```
struct Recta{
 float m;
 float b;
};
```

$y = mx + b$

Estructuras de Datos

- Rectas

```
struct Recta{
 float p0[2];
 float d[2];
};
```

p = p0 + t * d

Estructuras de Datos

Ventajas de la ecuación paramétrica

- No existen casos de borde (pendiente infinita)
- Al existir un solo parámetro es más fácil intersectar rectas con otros objetos

Estructuras de Datos

- Segmentos

```
struct Segmento{
 Punto2D p1, p2;
};
```

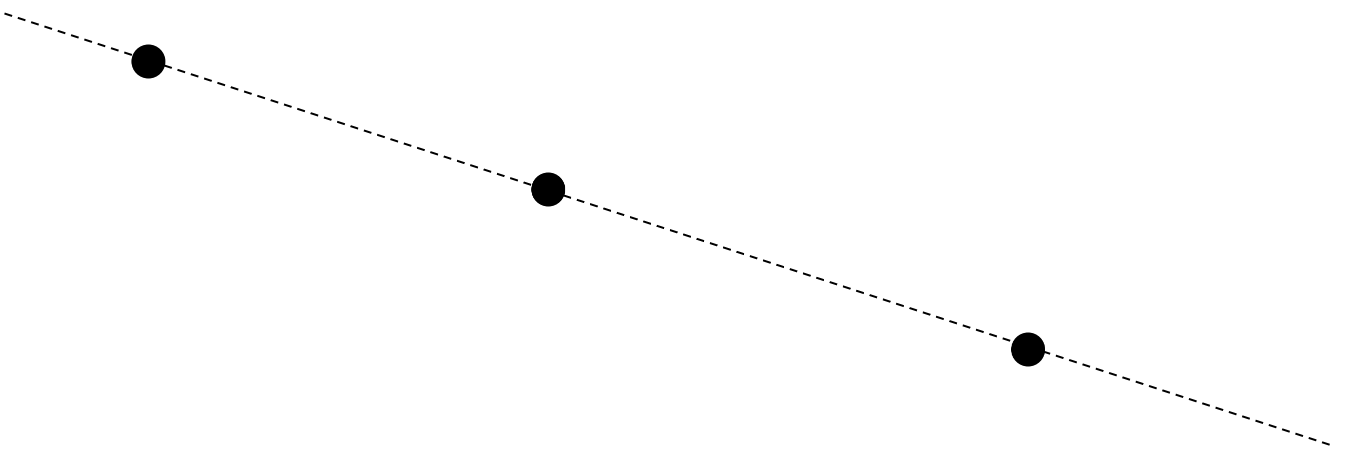
p = p0 + t * d

Intersección de segmentos

- Una forma de intersectar segmentos es usando la ecuación paramétrica de las rectas que contienen a los segmentos
- Si no se requiere conocer el punto exacto de intersección, una forma común está basada en el concepto de orientación de puntos

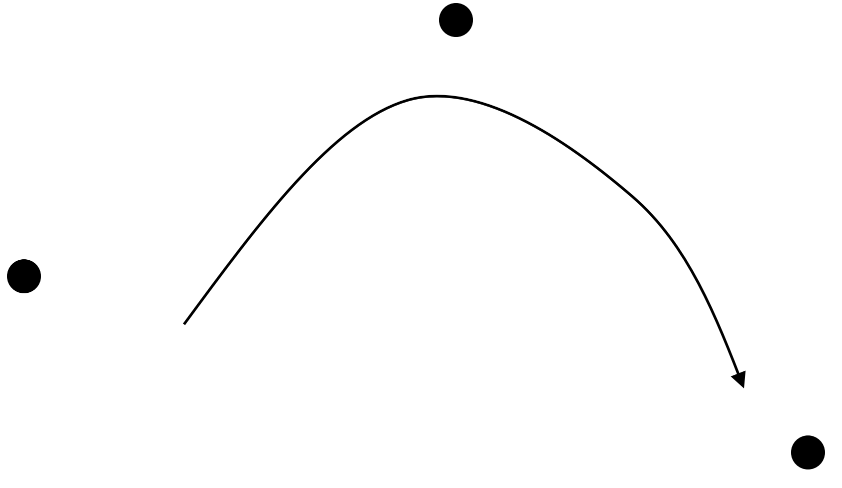
Intersección de segmentos

- Tres puntos sobre un plano pueden estar siempre:
  - Sobre una misma recta



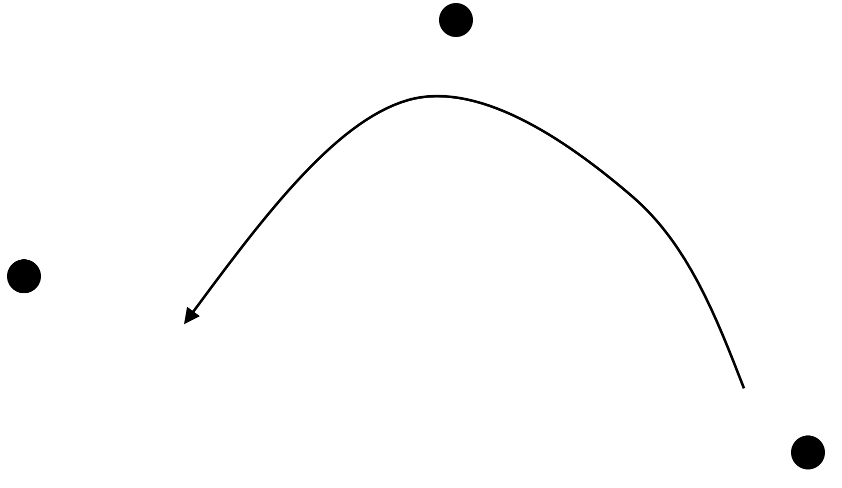
Intersección de segmentos

- Tres puntos sobre un plano pueden estar siempre:
  - Orientados en el sentido de las agujas del reloj (CW – Clockwise)



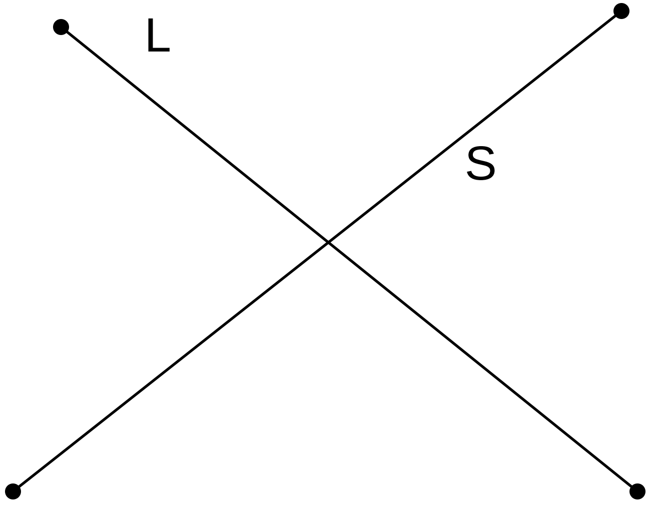
Intersección de segmentos

- Tres puntos sobre un plano pueden estar siempre:
  - Orientados en sentido contrario a las agujas del reloj (CCW – Counter Clockwise)



Intersección de segmentos

- Veamos que sucede cuando dos segmentos se intersectan:

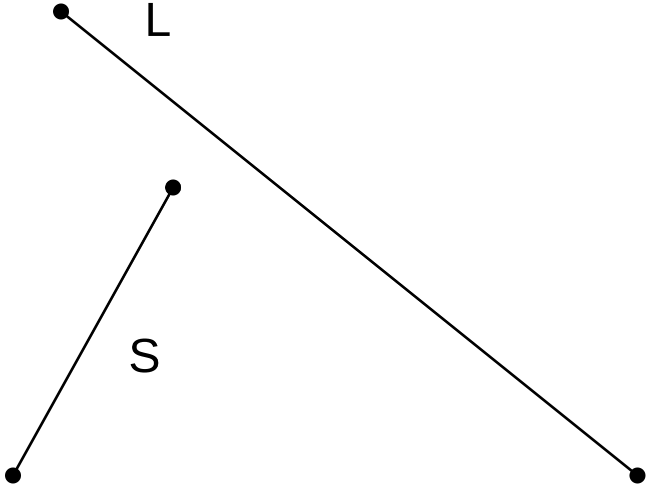


Los dos extremos de L están en lados distintos de S

Los dos extremos de S están en lados distintos de L

Intersección de segmentos

- Veamos que sucede cuando dos segmentos se no intersectan:

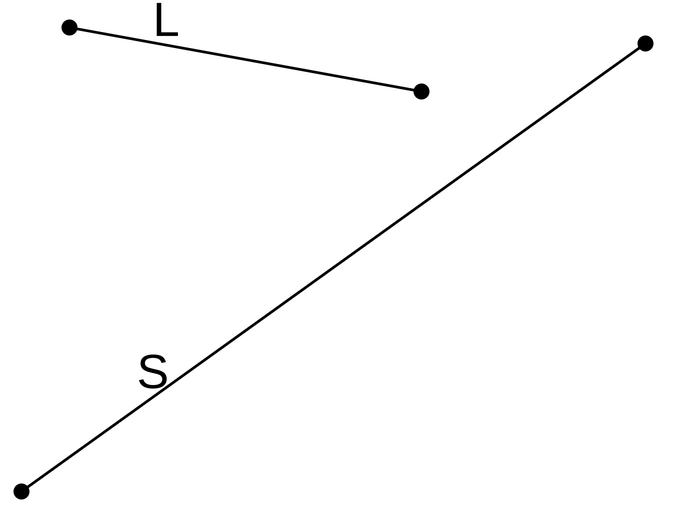


Los dos extremos de L están en lados distintos de S

Los dos extremos de S están en el mismo lado de L

Intersección de segmentos

- Veamos que sucede cuando dos segmentos se no intersectan:



Los dos extremos de L están en el mismo lado de S

Los dos extremos de S están en lados distintos de L

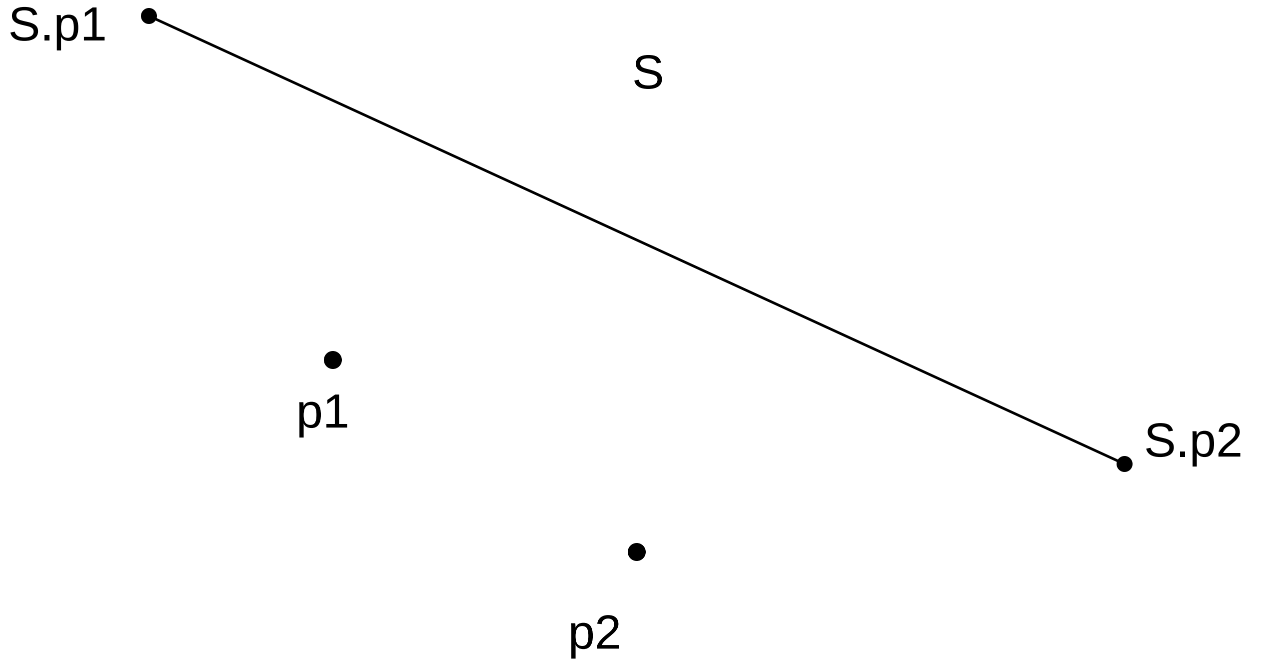
Intersección de segmentos

```
bool intersectar(Segmento &L, Segmento &S){
 return !mismoLado(L.p1, L.p2, S) &&
        !mismoLado(S.p1, S.p2, L);
}
```

¿Cómo saber si dos puntos están del mismo lado de un segmento?

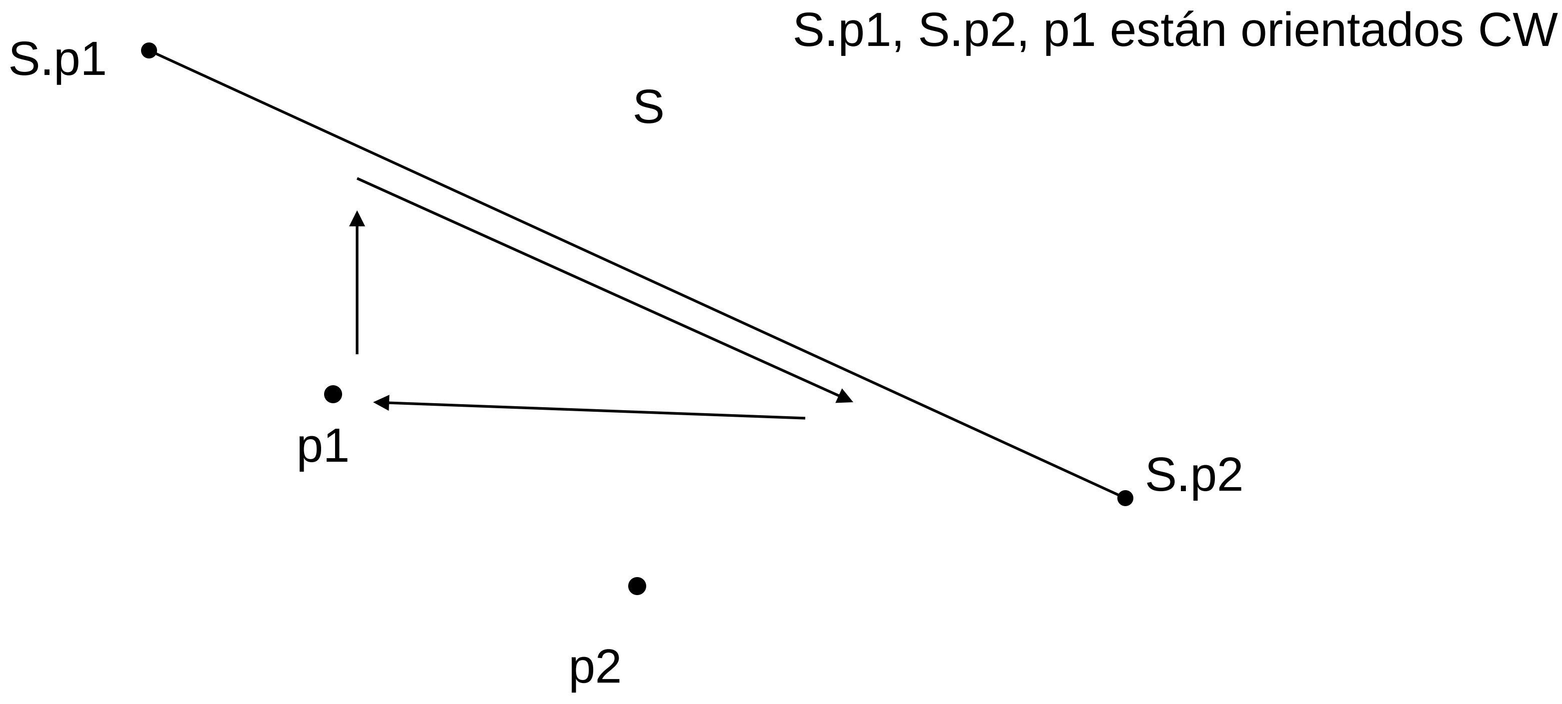
Intersección de segmentos

¿Cómo saber si dos puntos están del mismo lado de un segmento?



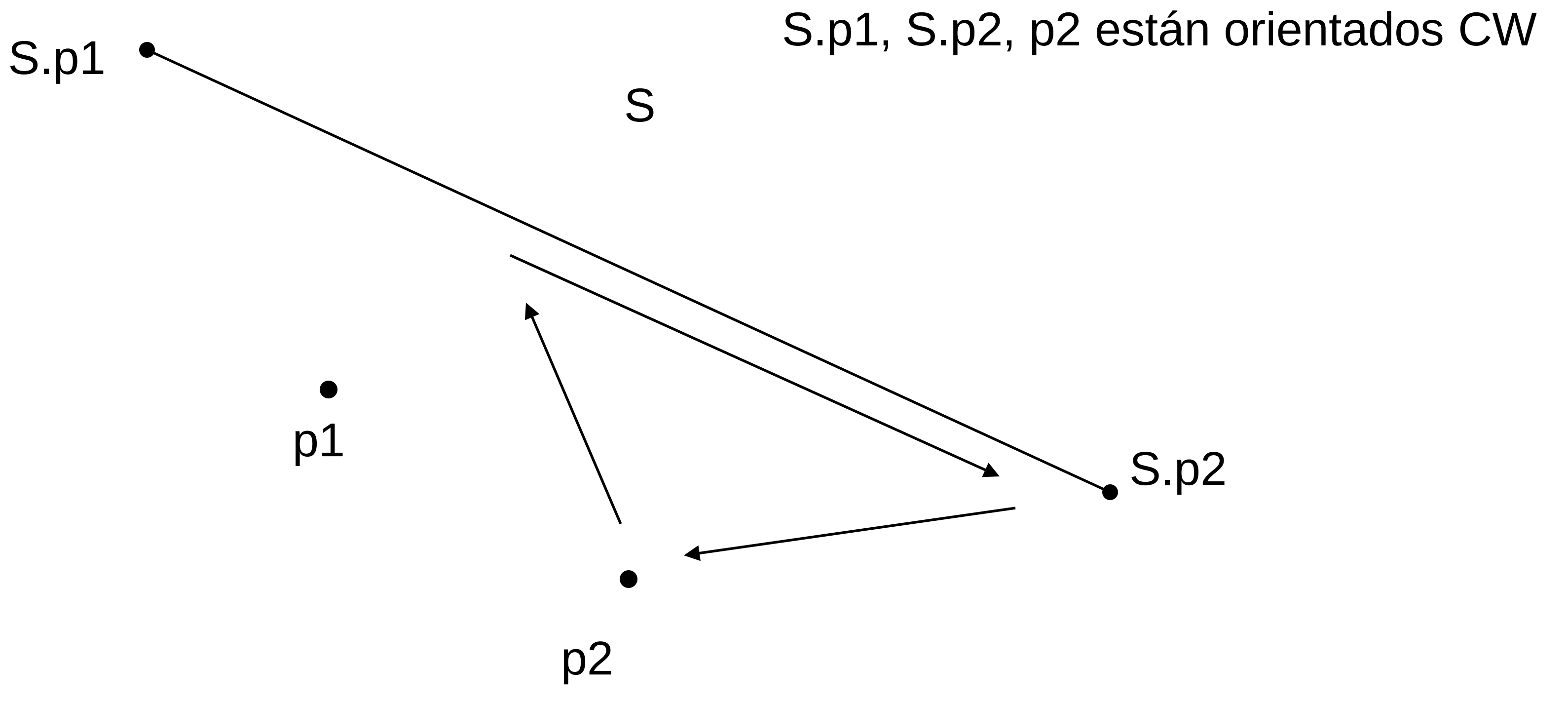
Intersección de segmentos

¿Cómo saber si dos puntos están del mismo lado de un segmento?



Intersección de segmentos

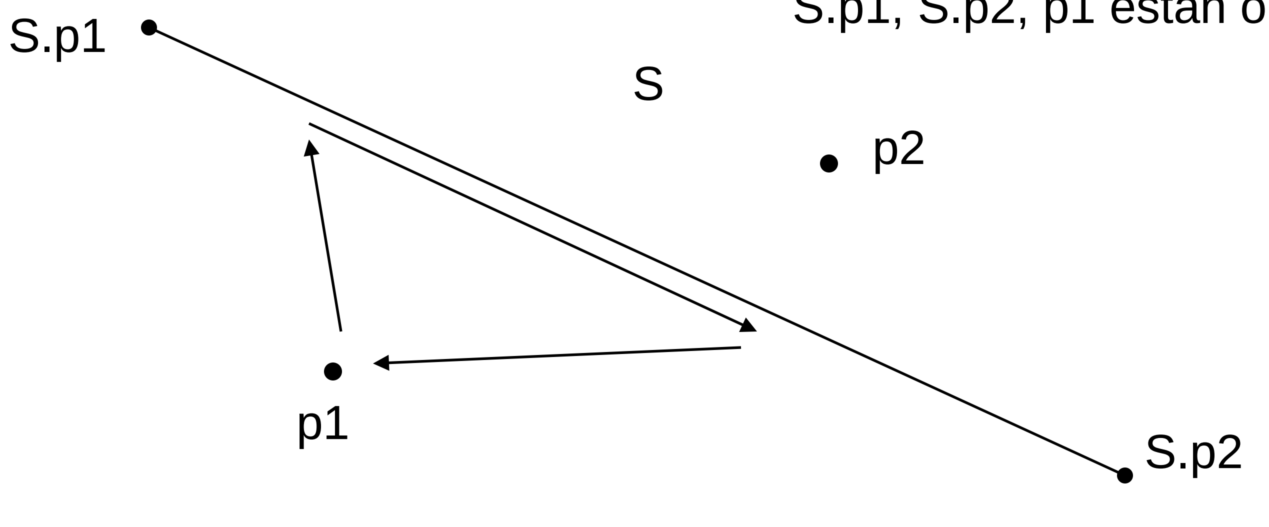
¿Cómo saber si dos puntos están del mismo lado de un segmento?



Intersección de segmentos

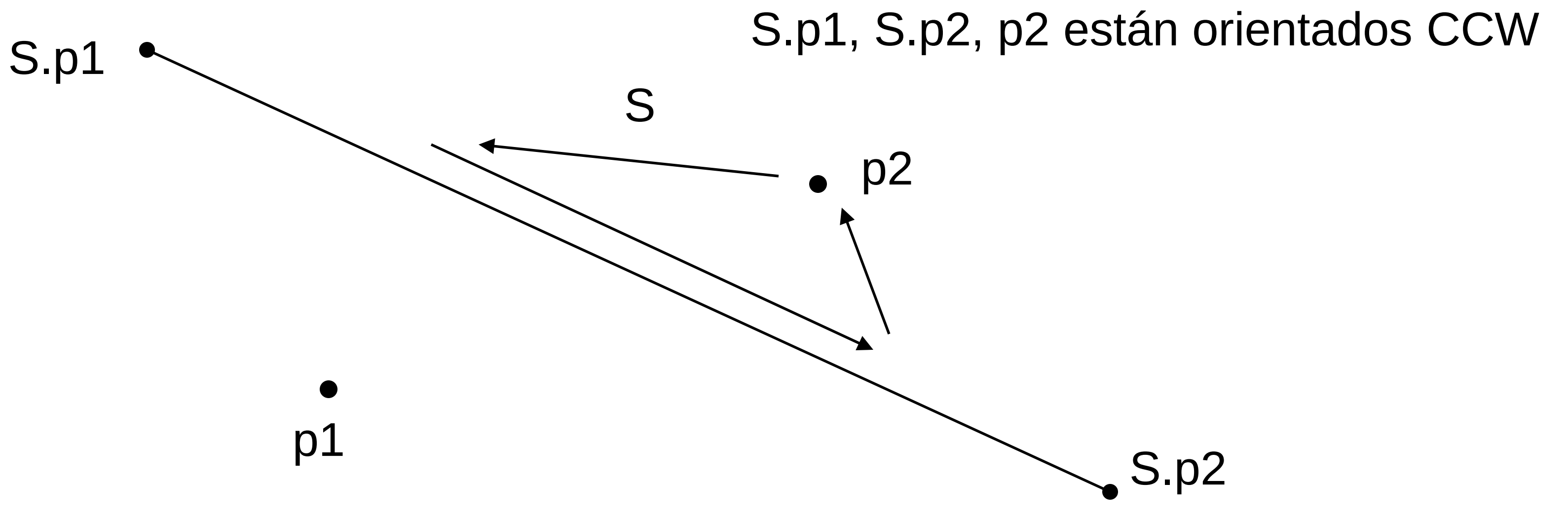
¿Cómo saber si dos puntos están del mismo lado de un segmento?

S.p1, S.p2, p1 están orientados CW



Intersección de segmentos

¿Cómo saber si dos puntos están del mismo lado de un segmento?



Intersección de segmentos

¿Cómo saber si dos puntos están del mismo lado de un segmento?

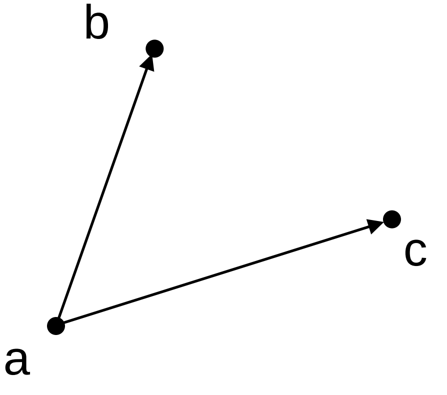
```
bool mismoLado(Punto2D p1, Punto2D p2, Segmento S){
 return ccw(S.p1, S.p2, p1)==ccw(S.p1, S.p2, p2);
}

bool intersectar(Segmento &L, Segmento &S){
 return ccw(L.p1, L.p2, S.p1)!=ccw(L.p1, L.p2, S.p2) &&
        ccw(S.p1, S.p2, L.p1)!=ccw(S.p1, S.p2, L.p2);
}
```

Intersección de segmentos

CCW

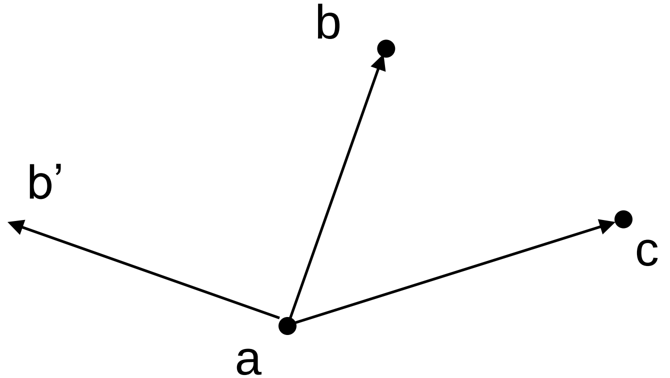
Veamos que sucede con tres puntos a,b,c, suponiendo que a está en el origen:



Intersección de segmentos

CCW

Construyamos el vector ortogonal a b, b’=[-by, bx]



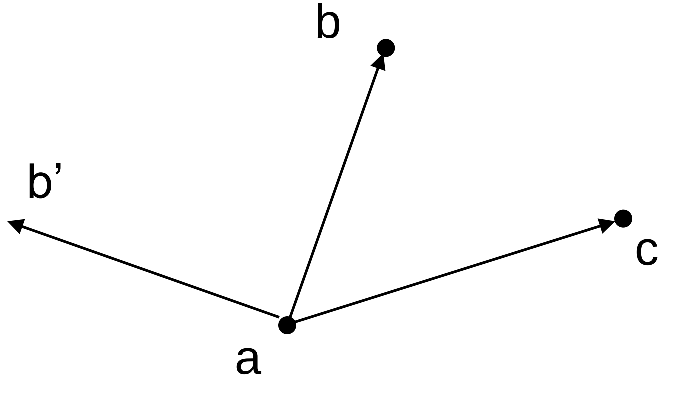
b’ · c = 0 si son colineales
b’ · c > 0 si el orden es ccw
b’ · c < 0 si el orden es cw

Intersección de segmentos

CCW

¿Qué pasa si *a* no está en el origen?

Es necesario trasladar a,b,c para hacer que siempre *a* esté en el origen



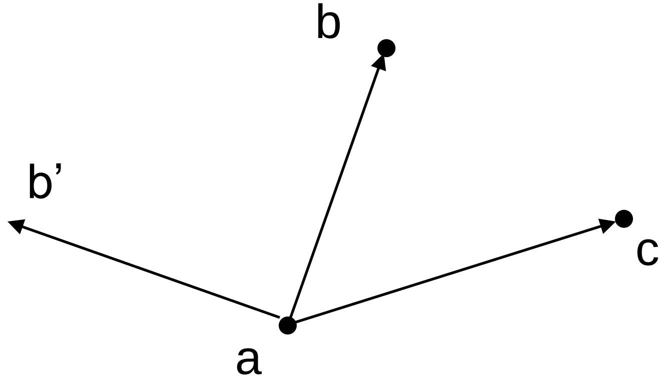
Intersección de segmentos

CCW

(b-a)’ · (c-a) = [bx-ax,by-ay]’ · [cx-ax,cy-ay]

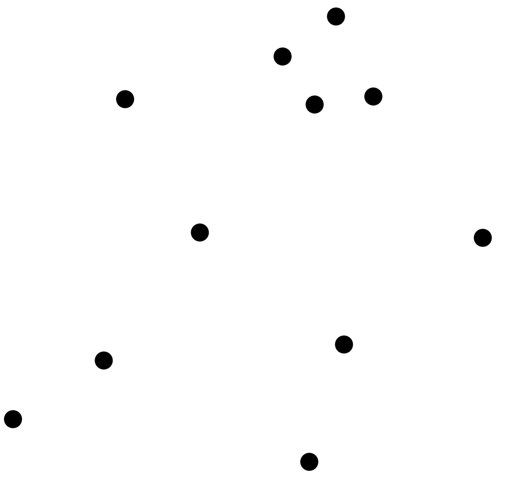
= [ay-by,bx-ax] · [cx-ax,cy-ay]

= (ay-by)(cx-ax)+(bx-ax)(cy-ay)



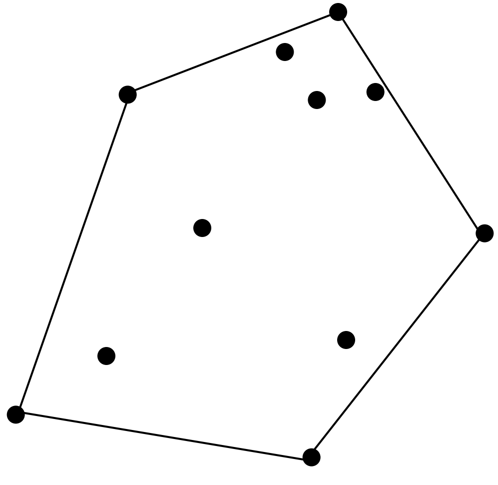
Cápsula Convexa

- La cápsula convexa (convex hull) de un conjunto de puntos es el polígono convexo de área menor que encierra a los puntos:



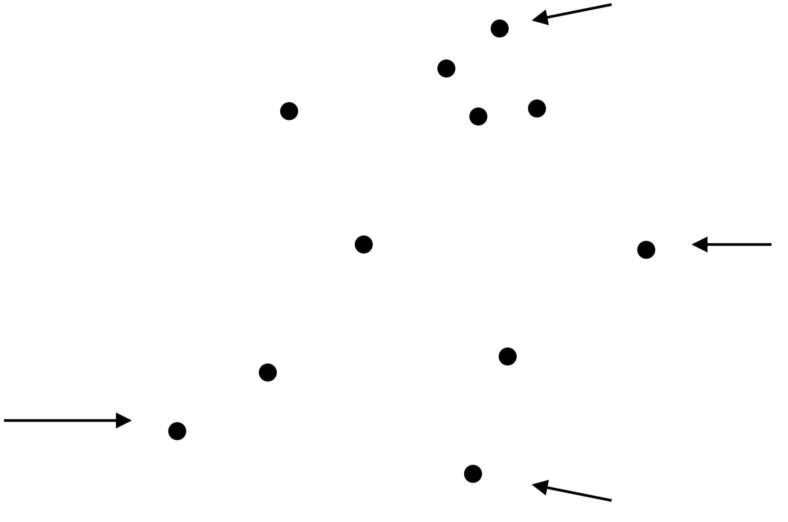
Cápsula Convexa

- La cápsula convexa (convex hull) de un conjunto de puntos es el polígono convexo de área menor que encierra a los puntos:



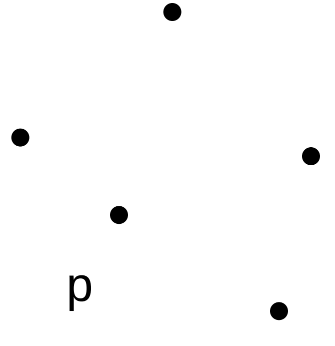
Gift Wrap

- Es el método más natural para los humanos
- Comienza con un punto que pertenece a la CC con seguridad. Cualquier punto extremo servirá (mínimo x, mínimo y, máximo x ó máximo y)



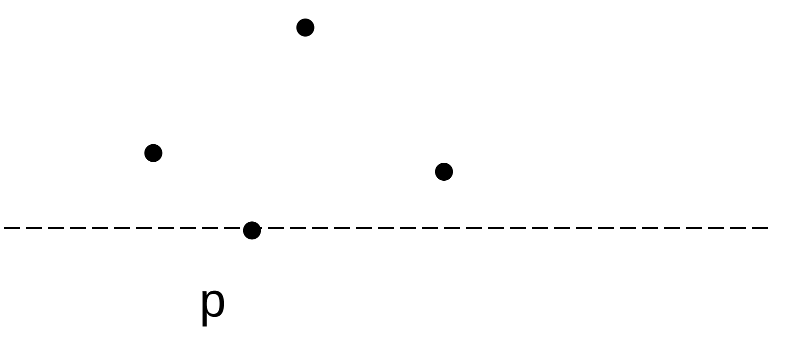
Gift Wrap

- Encontrar este punto **p** es una operación de O(N)
- A partir de ahí se agrega como próximo punto a la CC, aquel que forme un ángulo menor con la recta horizontal que pasa por **p**



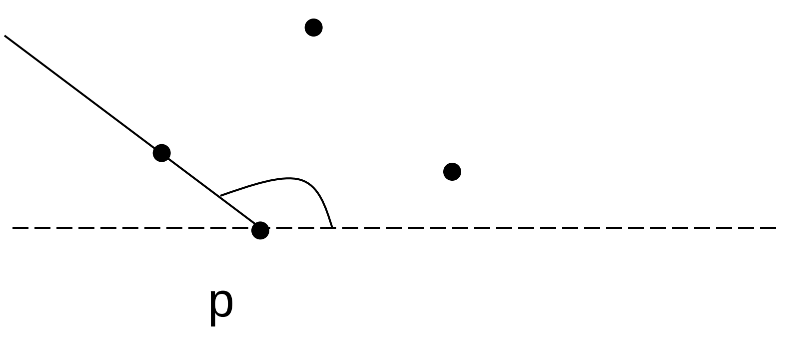
Gift Wrap

- Encontrar este punto **p** es una operación de O(N)
- A partir de ahí se agrega como próximo punto a la CC, aquel que forme un ángulo menor con la recta horizontal que pasa por **p**



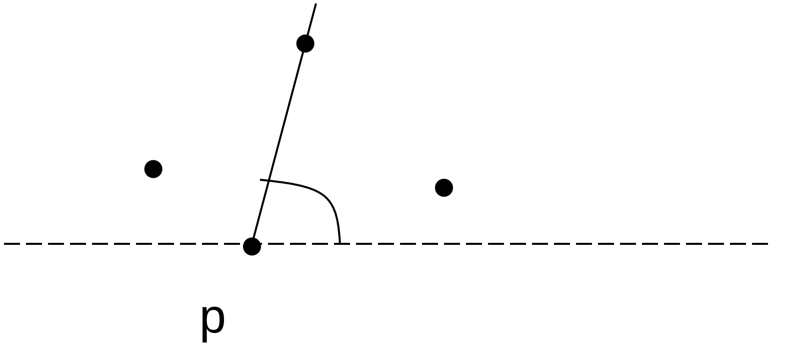
Gift Wrap

- Encontrar este punto **p** es una operación de O(N)
- A partir de ahí se agrega como próximo punto a la CC, aquel que forme un ángulo menor con la recta horizontal que pasa por **p**



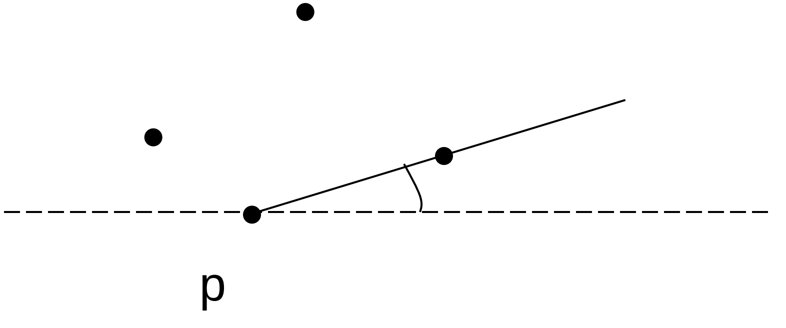
Gift Wrap

- Encontrar este punto **p** es una operación de O(N)
- A partir de ahí se agrega como próximo punto a la CC, aquel que forme un ángulo menor con la recta horizontal que pasa por **p**



Gift Wrap

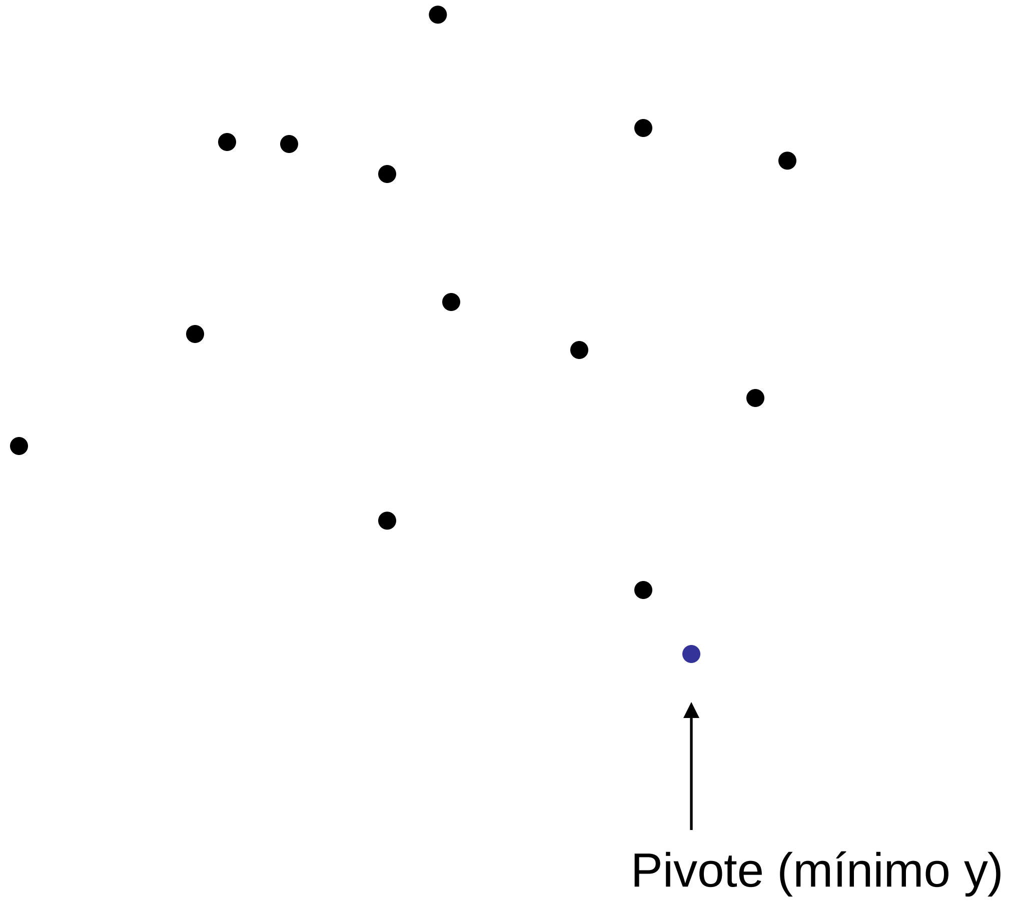
- Encontrar este punto **p** es una operación de O(N)
- A partir de ahí se agrega como próximo punto a la CC, aquel que forme un ángulo menor con la recta horizontal que pasa por **p**



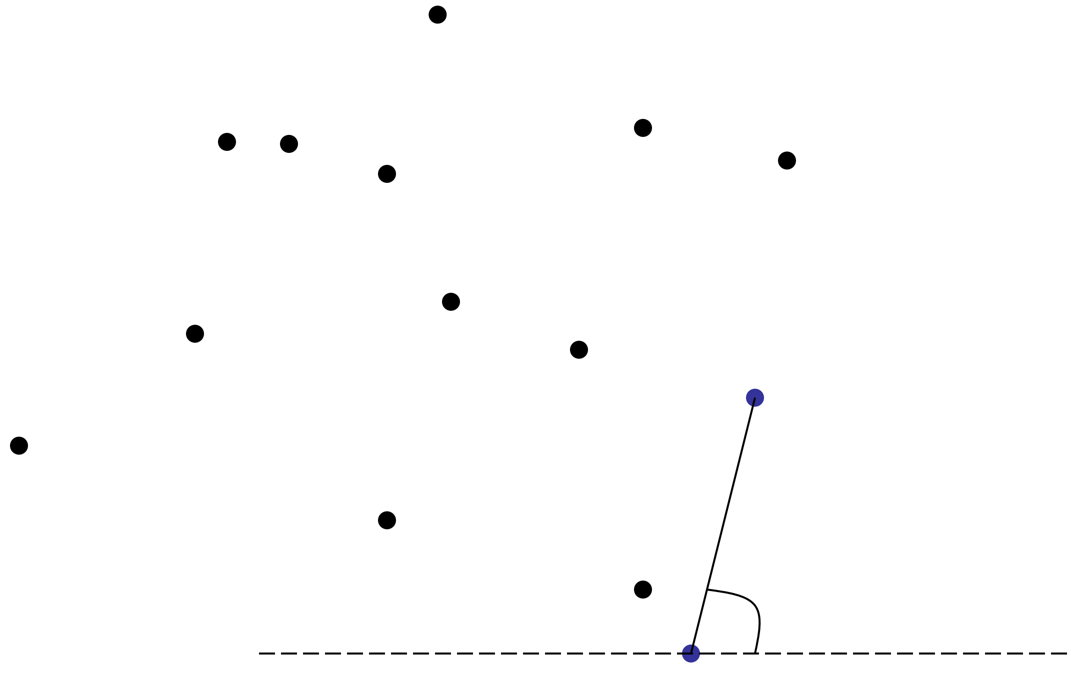
Gift Wrap

- Luego, en cada iteración del algoritmo es necesario ver cual de los restantes N puntos tiene el ángulo menor
- Se hacen a lo sumo N iteraciones
- Luego, el algoritmo es de $O(N^2)$

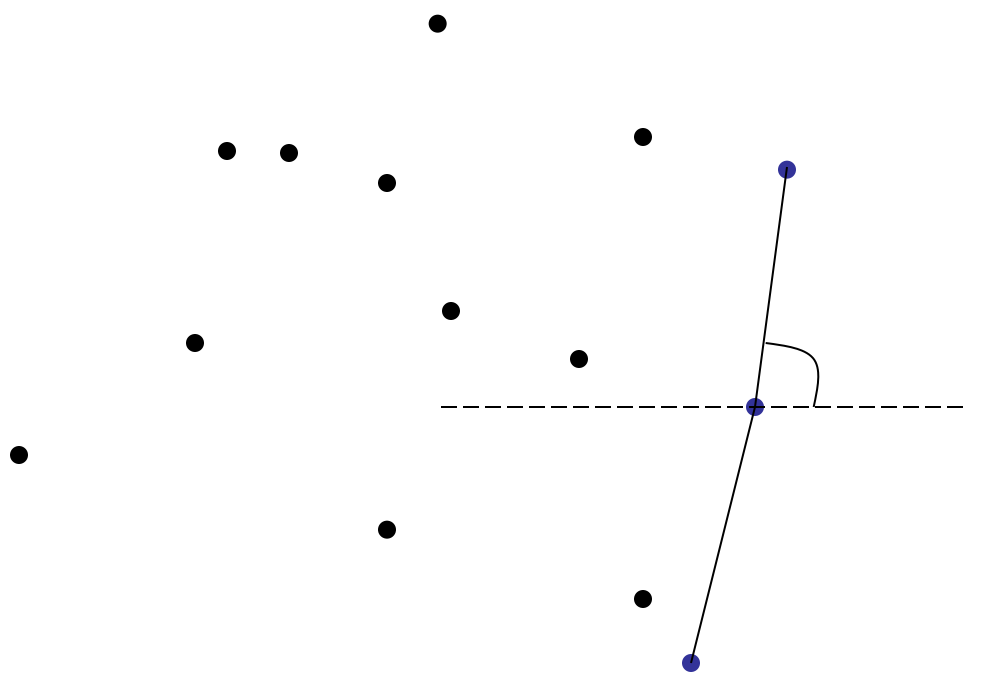
Gift Wrap



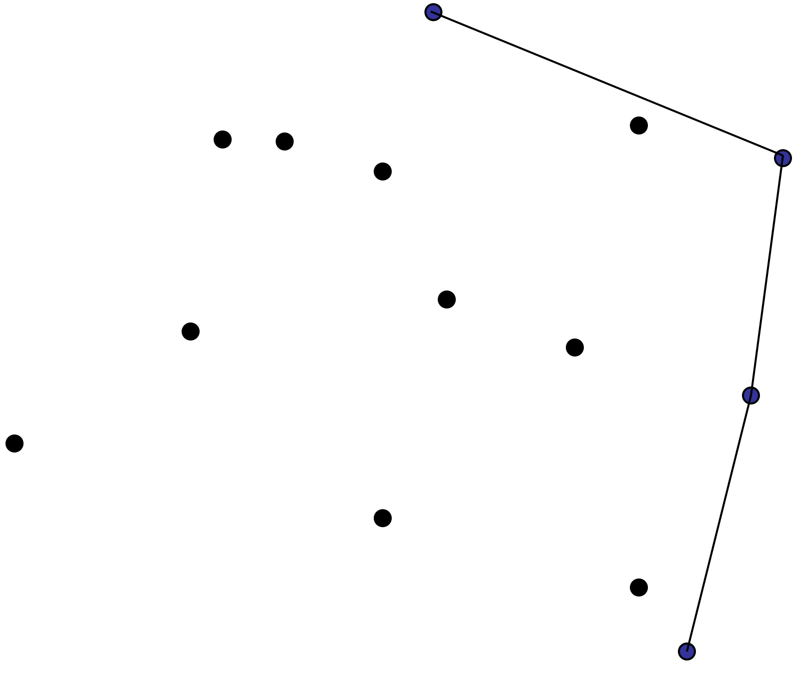
Gift Wrap



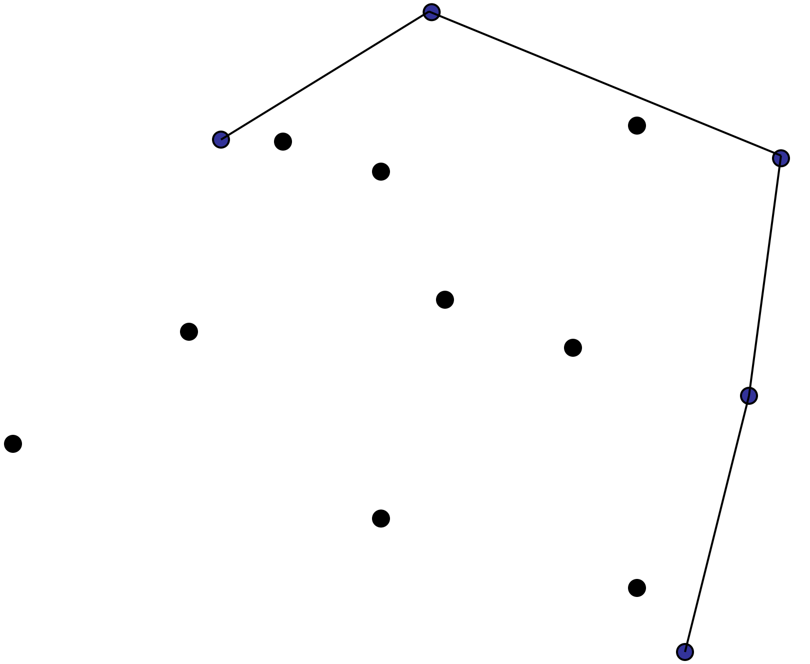
Gift Wrap



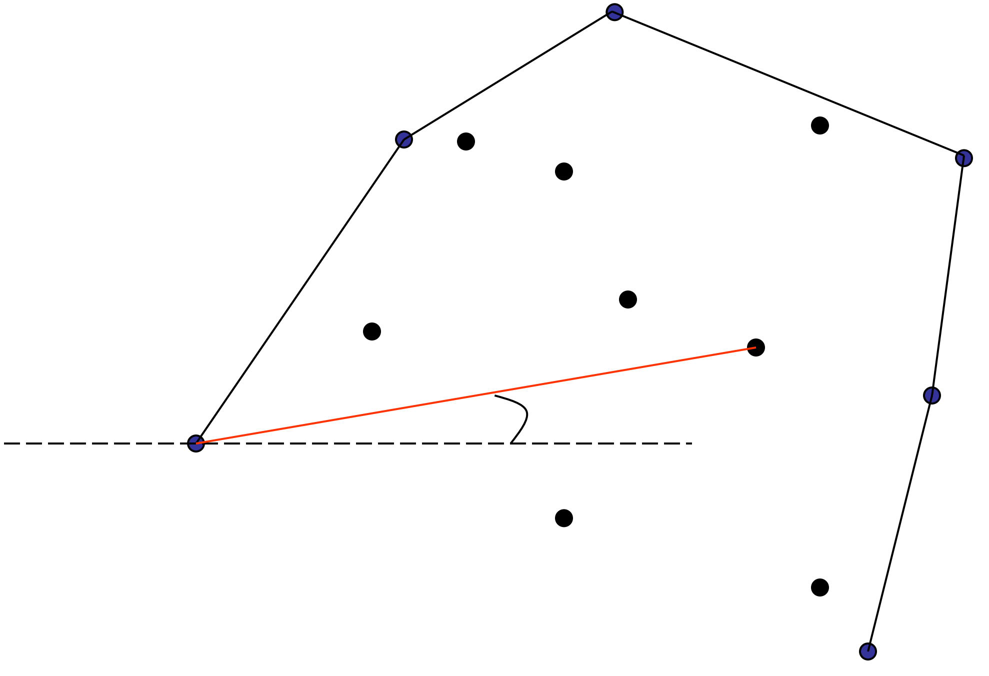
Gift Wrap



Gift Wrap



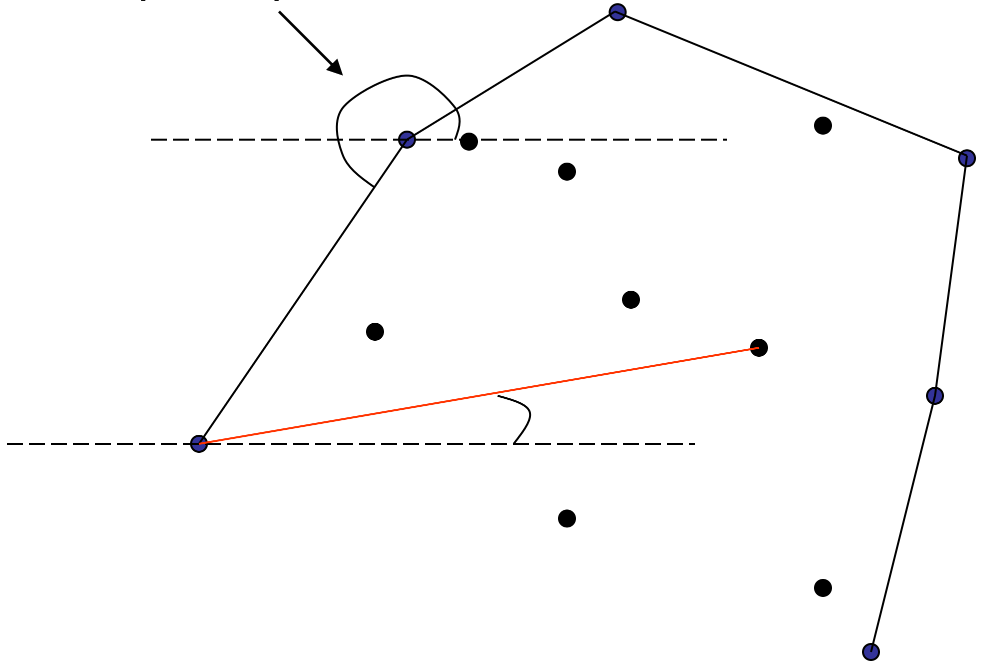
Gift Wrap



¿cómo se descarta este punto?

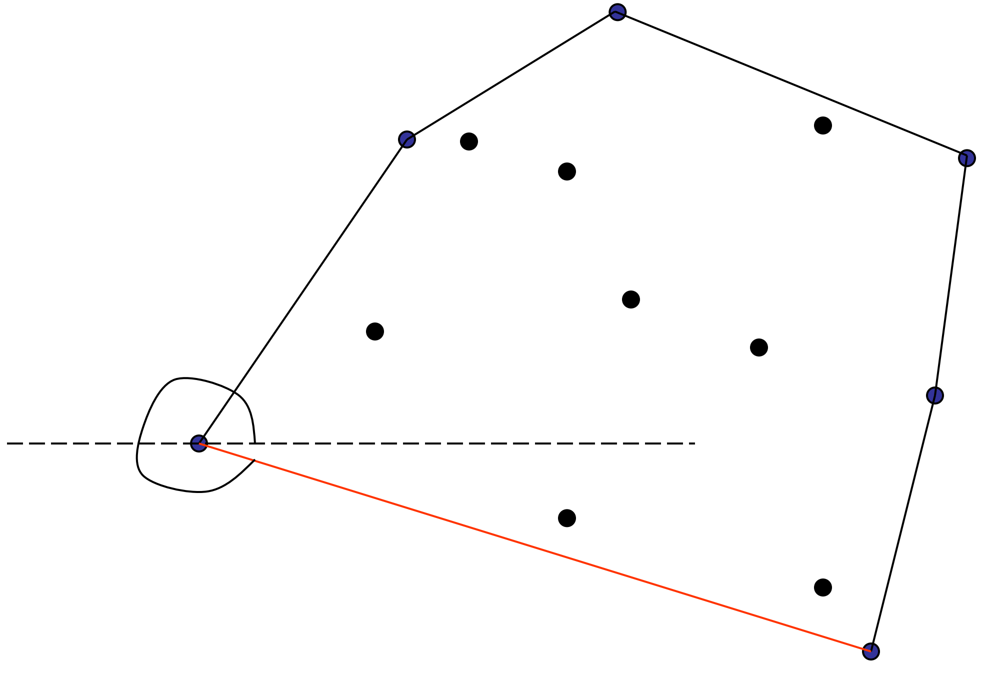
Gift Wrap

Ángulo formado por el punto anterior

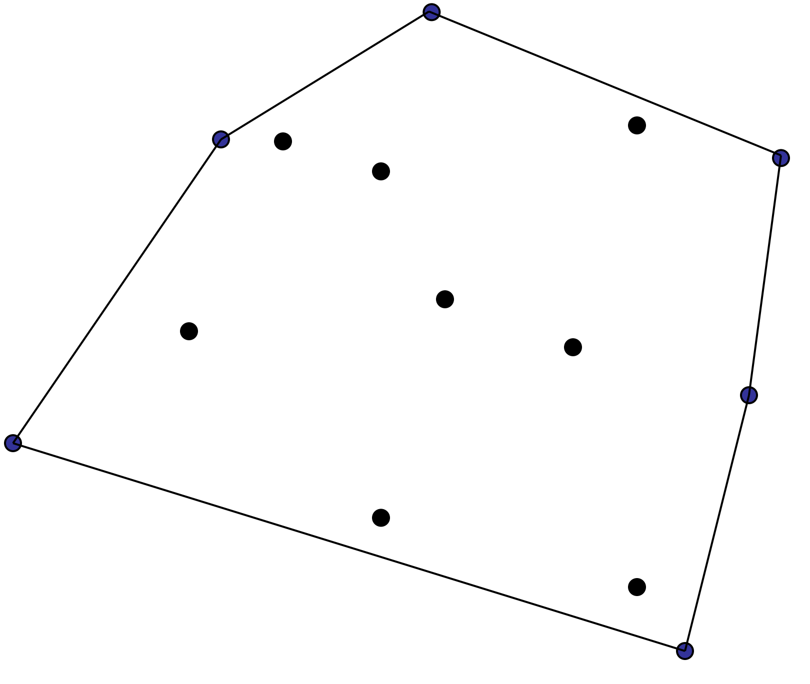


Hay que considerar el menor ángulo que sea mayor al ángulo formado por el punto anterior

Gift Wrap



Gift Wrap



Función Theta

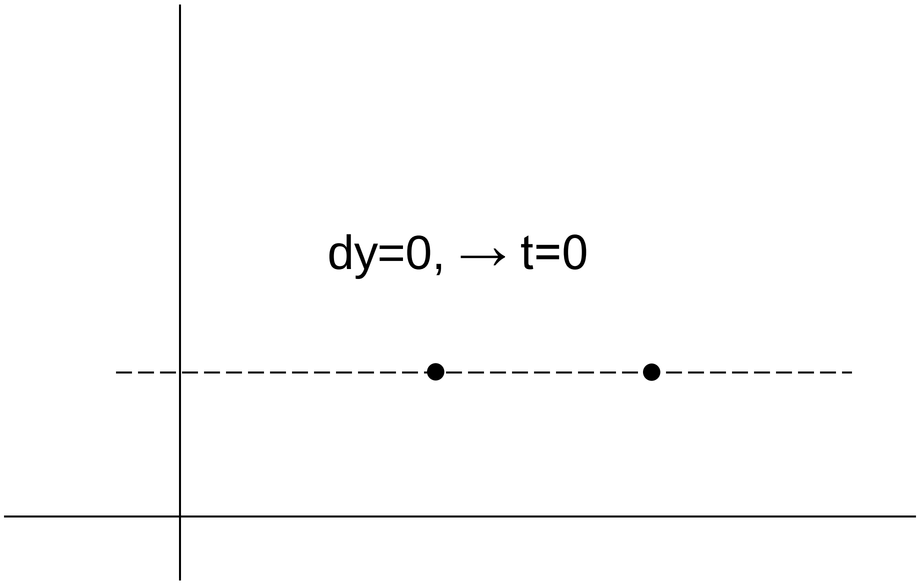
- Es posible calcular el ángulo entre dos puntos y la recta horizontal con funciones trigonométricas
- Si no se necesita el ángulo exacto, sino una función que se comporte igual podemos usar la función theta, que es más económica computacionalmente:

Función Theta

```
float theta(Punto2D &p1, Punto2D &p2){
 int dx,dy,ax,ay;
 float t;
 dx = p2.x - p1.x; ax = abs(dx);
 dy = p2.y - p1.y; ay = abs(dy);
 t = (ax+ay==0) ? 0 : (float) dy/(ax+ay);
 if(dx<0) t = 2-t;
 else if(dy < 0) t = 4+t;
 return t*90.0;
}
```

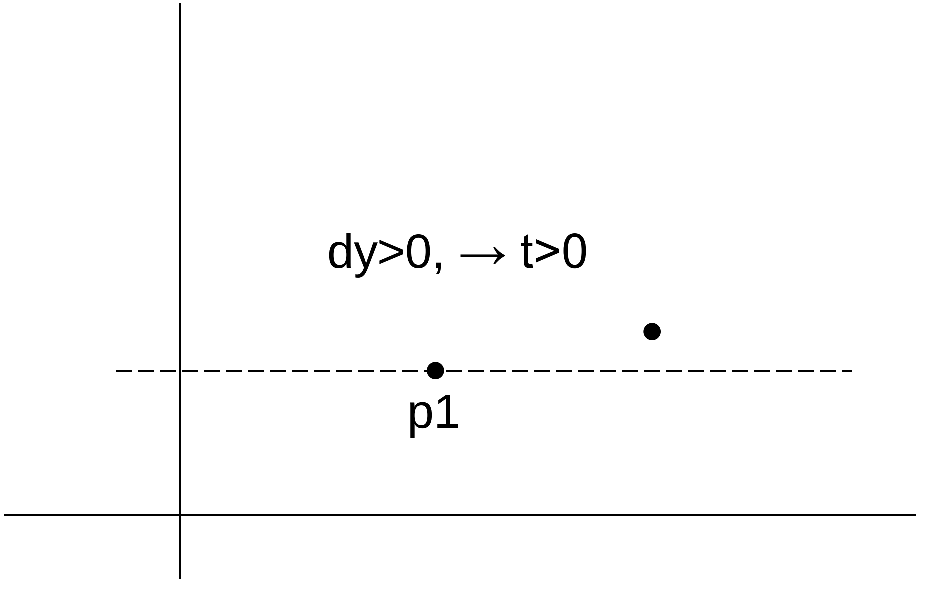
Función Theta

```
float theta(Punto2D &p1, Punto2D &p2){
 int dx,dy,ax,ay;
 float t;
 dx = p2.x - p1.x; ax = abs(dx);
 dy = p2.y - p1.y; ay = abs(dy);
 t = (ax+ay==0) ? 0 : (float) dy/(ax+ay);
 if(dx<0) t = 2-t;
 else if(dy < 0) t = 4+t;
 return t*90.0;
}
```



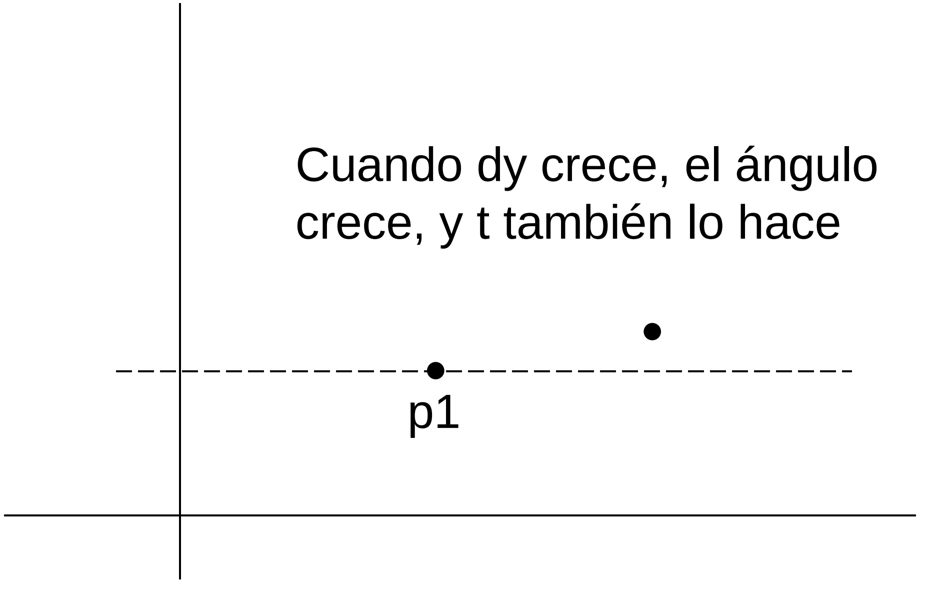
Función Theta

```
float theta(Punto2D &p1, Punto2D &p2){
 int dx,dy,ax,ay;
 float t;
 dx = p2.x - p1.x; ax = abs(dx);
 dy = p2.y - p1.y; ay = abs(dy);
 t = (ax+ay==0) ? 0 : (float) dy/(ax+ay);
 if(dx<0) t = 2-t;
 else if(dy < 0) t = 4+t;
 return t*90.0;
}
```



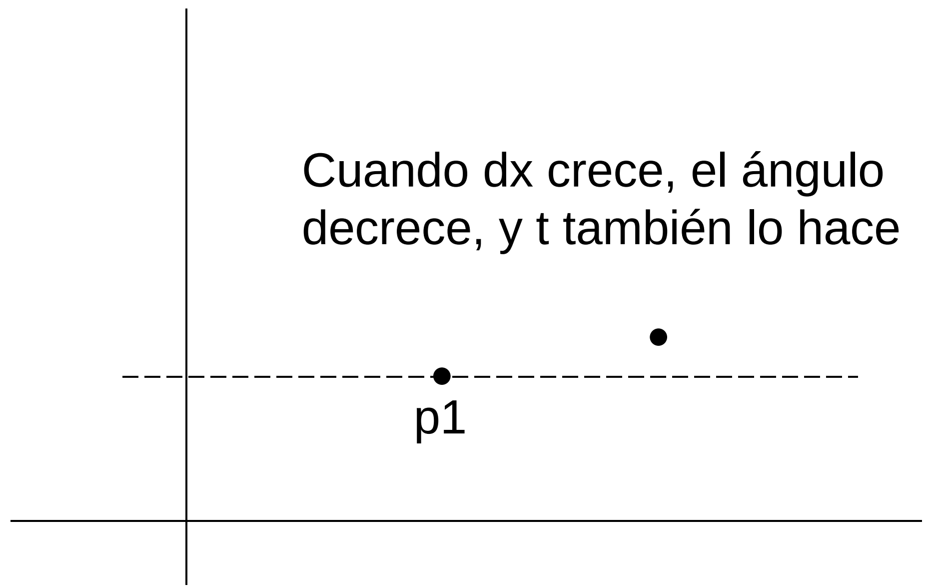
Función Theta

```
float theta(Punto2D &p1, Punto2D &p2){
 int dx,dy,ax,ay;
 float t;
 dx = p2.x - p1.x; ax = abs(dx);
 dy = p2.y - p1.y; ay = abs(dy);
 t = (ax+ay==0) ? 0 : (float) dy/(ax+ay);
 if(dx<0) t = 2-t;
 else if(dy < 0) t = 4+t;
 return t*90.0;
}
```



Función Theta

```
float theta(Punto2D &p1, Punto2D &p2){
 int dx,dy,ax,ay;
 float t;
 dx = p2.x - p1.x; ax = abs(dx);
 dy = p2.y - p1.y; ay = abs(dy);
 t = (ax+ay==0) ? 0 : (float) dy/(ax+ay);
 if(dx<0) t = 2-t;
 else if(dy < 0) t = 4+t;
 return t*90.0;
}
```



Función Theta

```
float theta(Punto2D &p1, Punto2D &p2){
 int dx,dy,ax,ay;
 float t;
 dx = p2.x - p1.x; ax = abs(dx);
 dy = p2.y - p1.y; ay = abs(dy);
 t = (ax+ay==0) ? 0 : (float) dy/(ax+ay);
 if(dx<0) t = 2-t;
 else if(dy < 0) t = 4+t;
 return t*90.0;
}
```

Manejo de los 4 cuadrantes

Función Theta

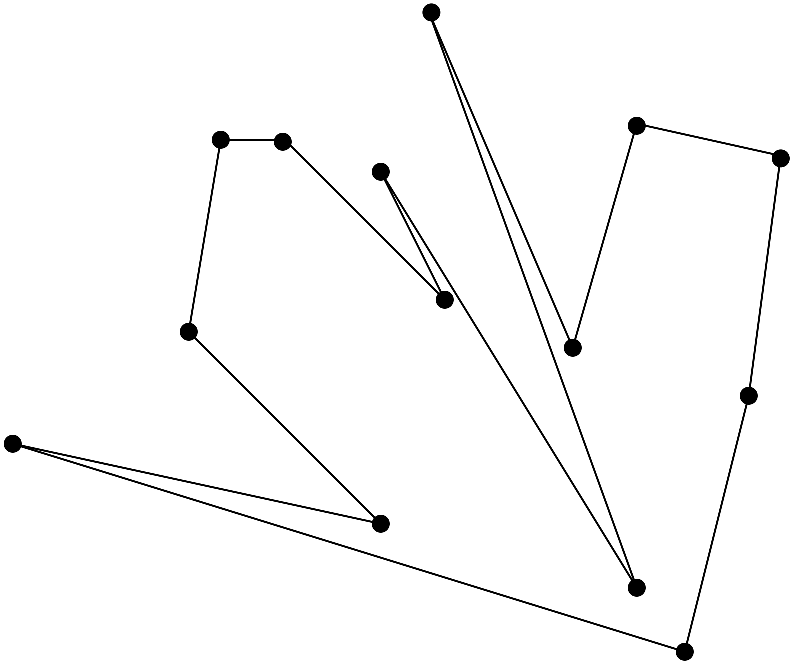
```
float theta(Punto2D &p1, Punto2D &p2){
 int dx,dy,ax,ay;
 float t;
 dx = p2.x - p1.x; ax = abs(dx);
 dy = p2.y - p1.y; ay = abs(dy);
 t = (ax+ay==0) ? 0 : (float) dy/(ax+ay);
 if(dx<0) t = 2-t;
 else if(dy < 0) t = 4+t;
 return t*90.0;
}
```

Finalmente el resultado se lleva del rango [0,4] al rango [0,360], aunque esto no es estrictamente necesario

Graham

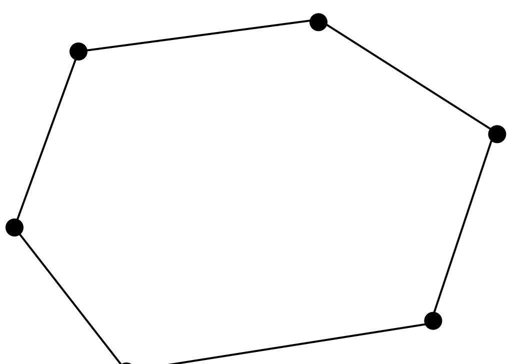
- Este método comienza ubicando un punto pivote que con certeza pertenece al CC (igual que el método de Gift wrap) – O(N)
- A partir de ahí ordena todos los puntos según el ángulo que forman con la recta horizontal que pasa por el pivote
  – O(N log N)

Graham



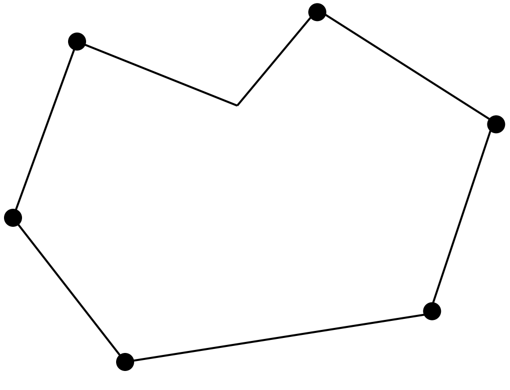
Graham

- A continuación se evaluán grupos de tres puntos consecutivos comenzando por el pivote
- En un polígono convexo todos los puntos consecutivos deberían estar orientados de la misma forma (CCW o CW)

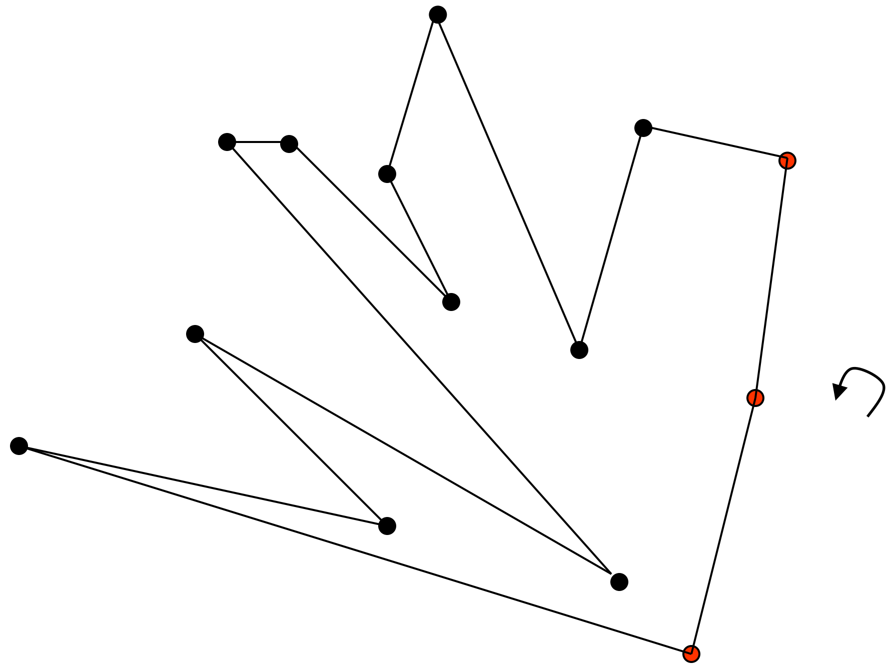


Graham

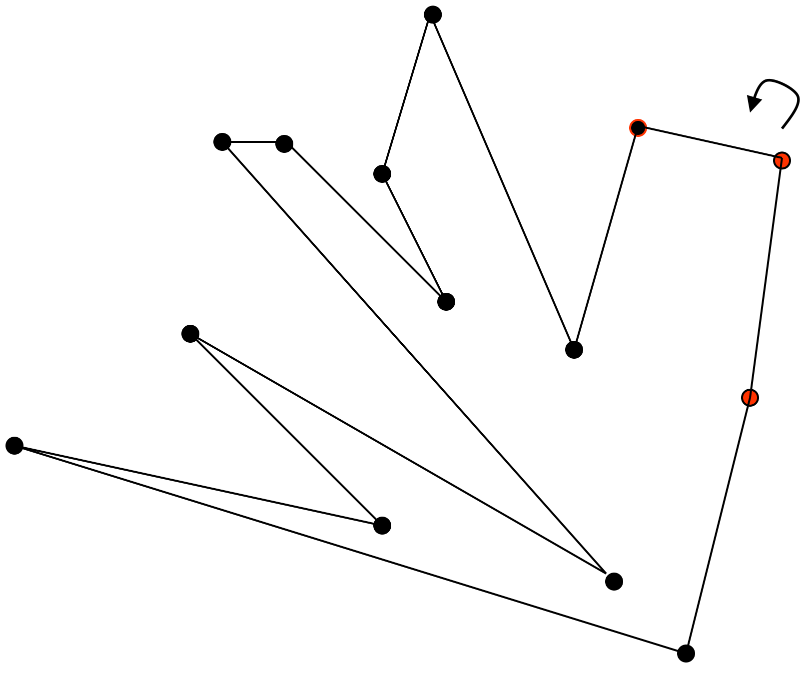
- A continuación se evalúan grupos de tres puntos consecutivos comenzando por el pivote
- En un polígono convexo todos los puntos consecutivos deberían estar orientados de la misma forma (CCW o CW)



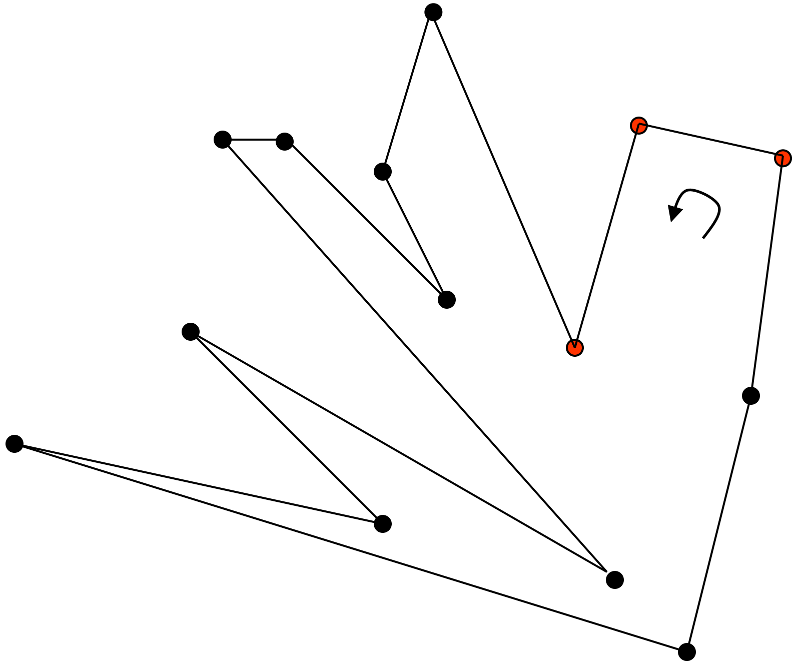
Graham



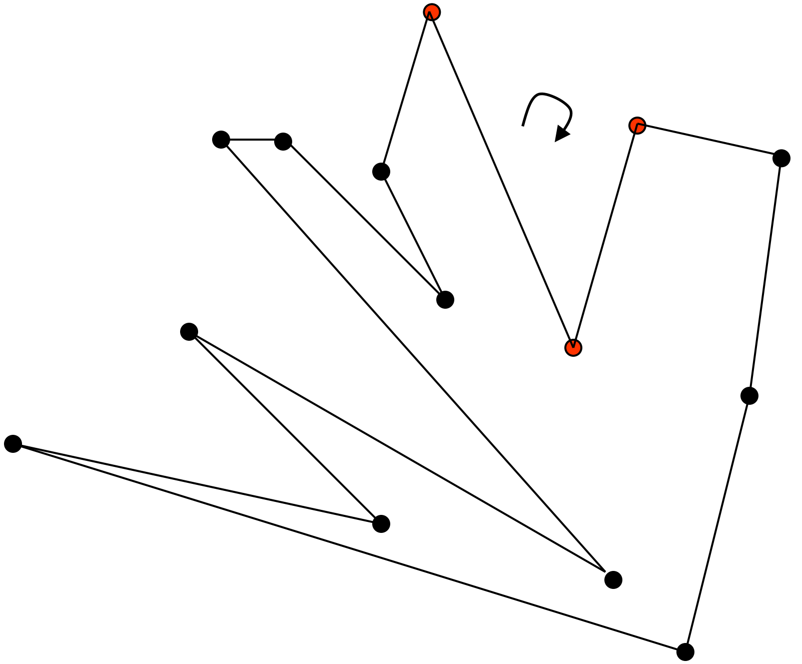
Graham



Graham

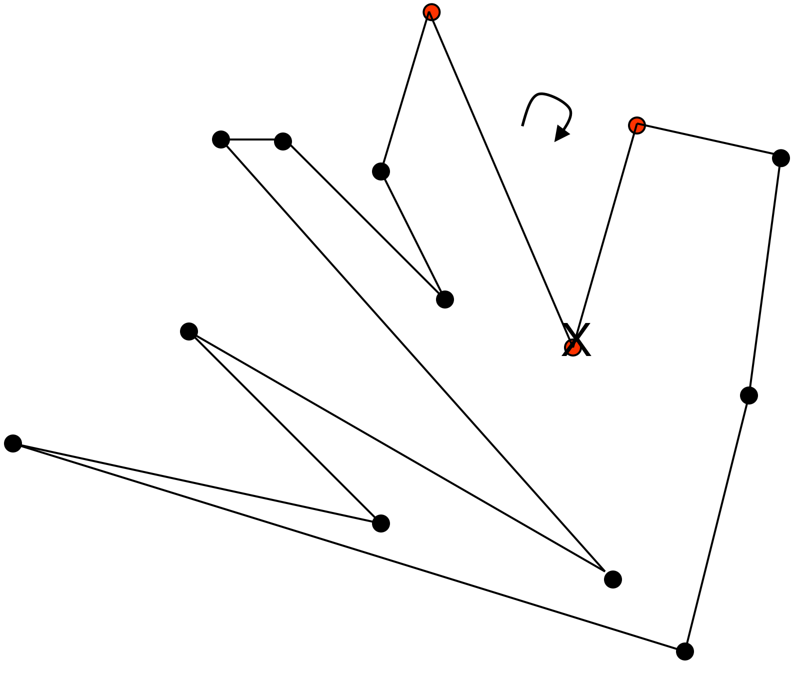


Graham



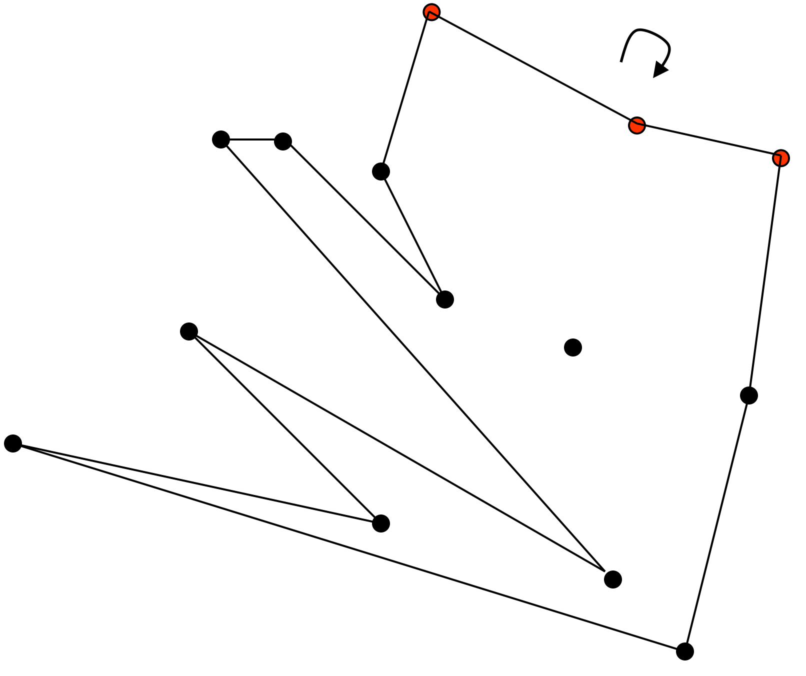
Hay un cambio de orientación. ¿Cuál de los tres puntos no está en la CC?

Graham

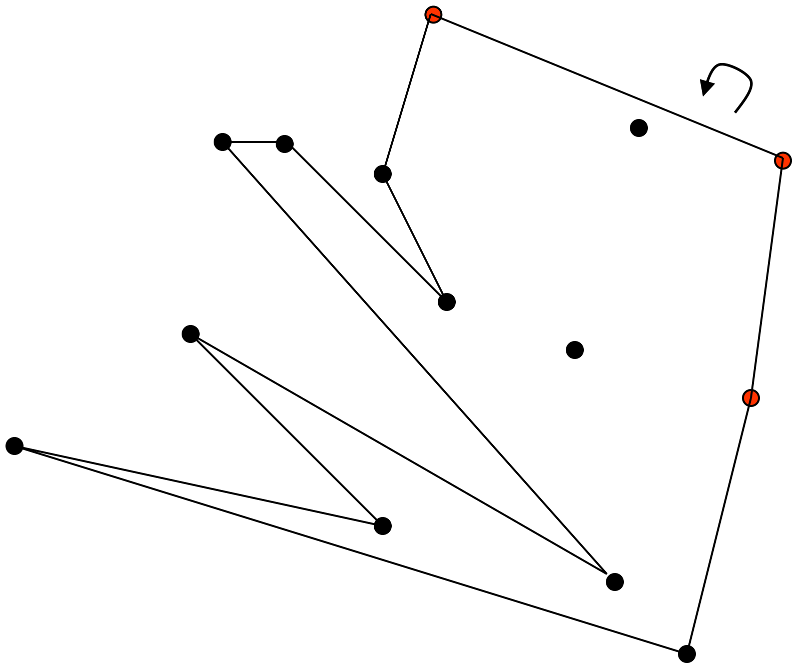


Hay un cambio de orientación. ¿Cuál de los tres puntos no está en la CC?

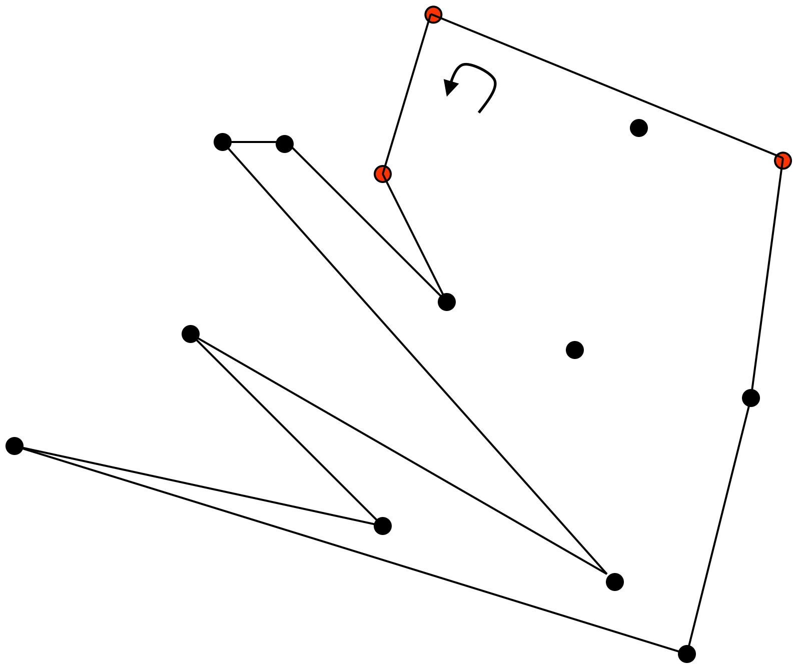
Graham



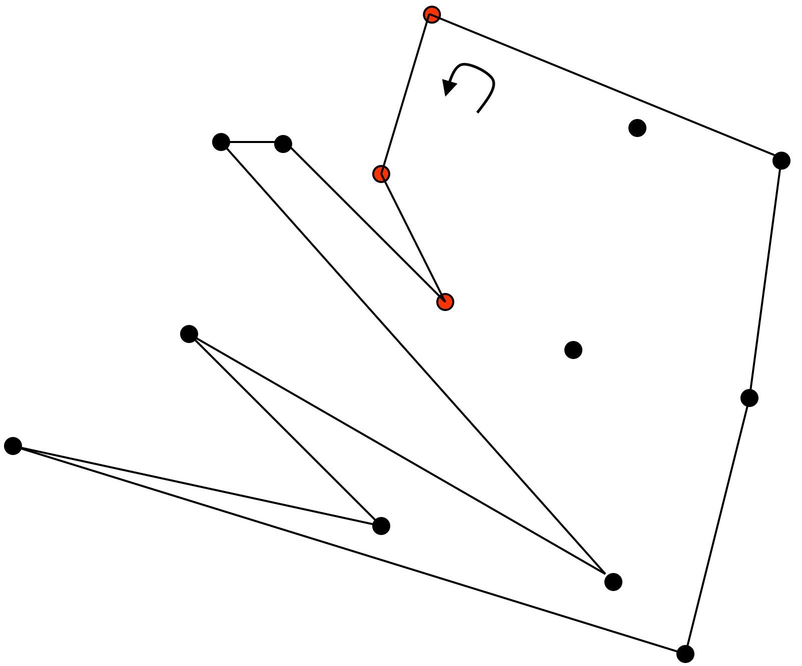
Graham



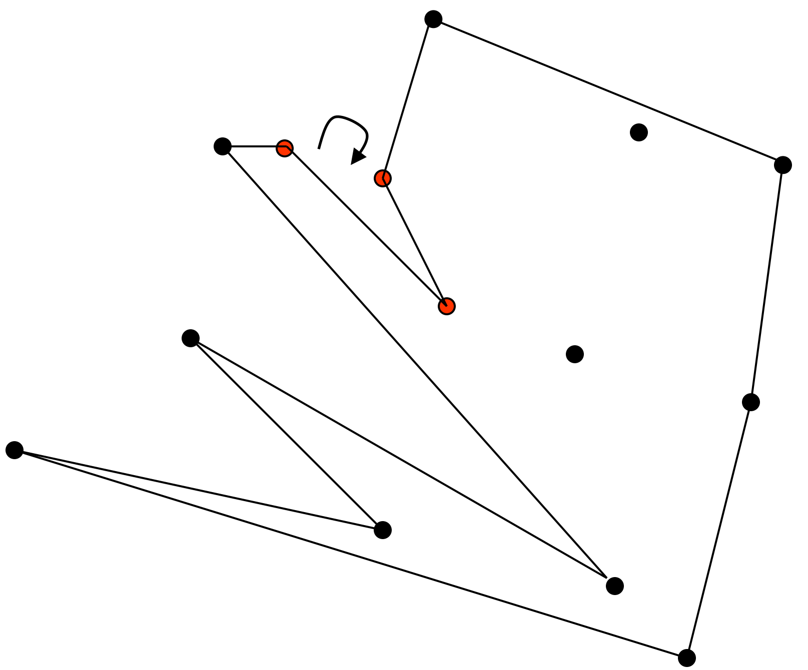
Graham



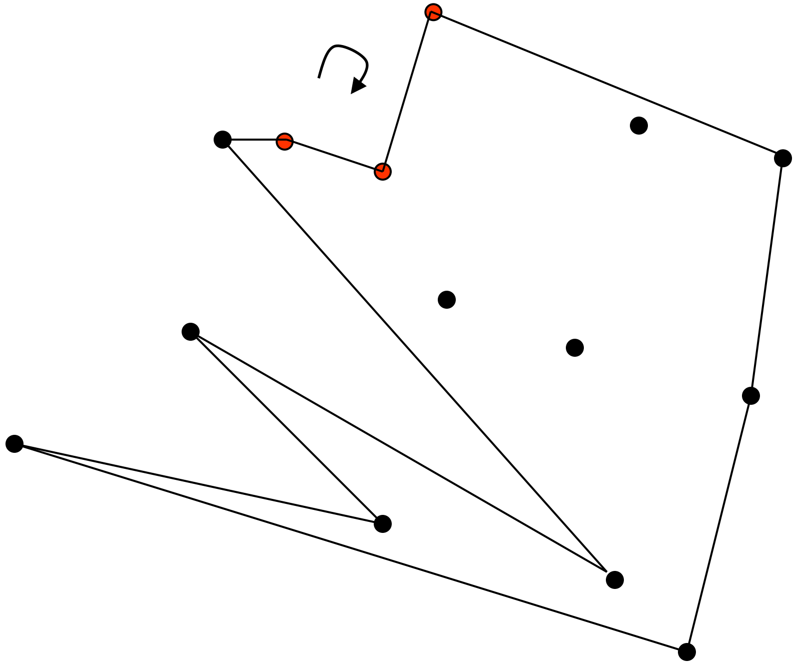
Graham



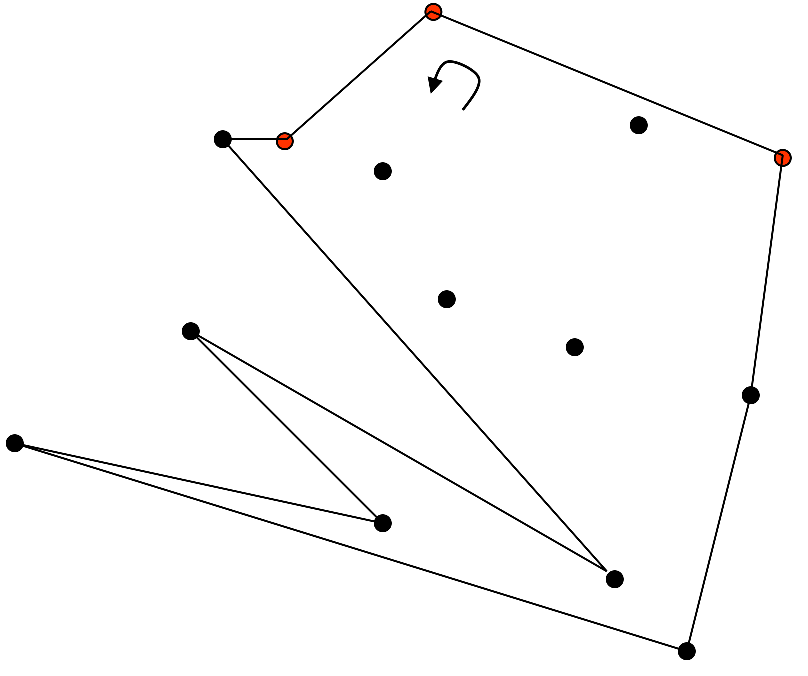
Graham



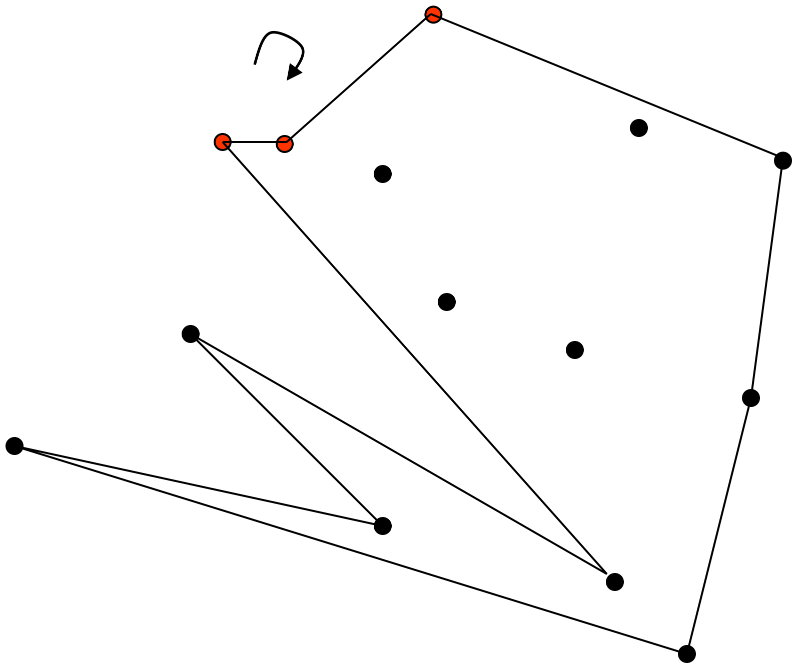
Graham



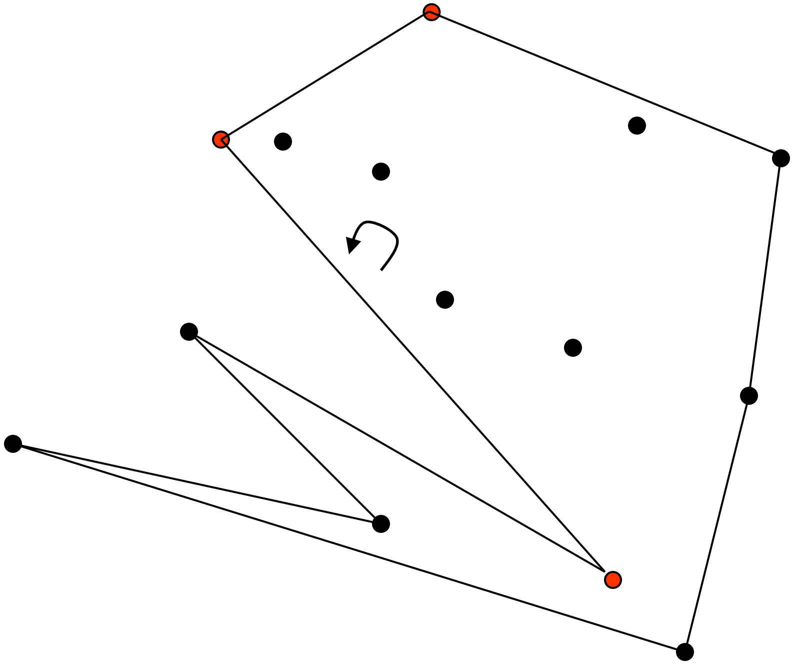
Graham



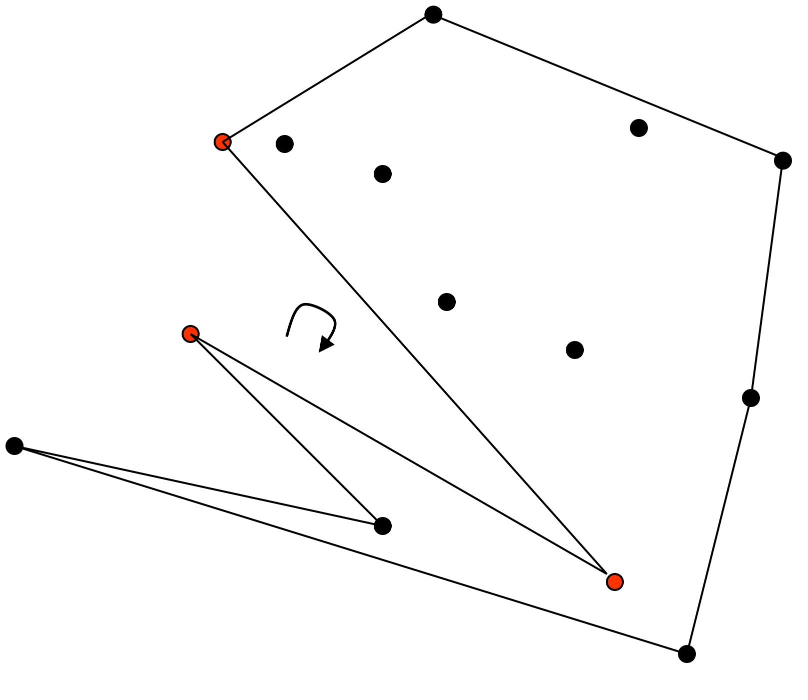
Graham



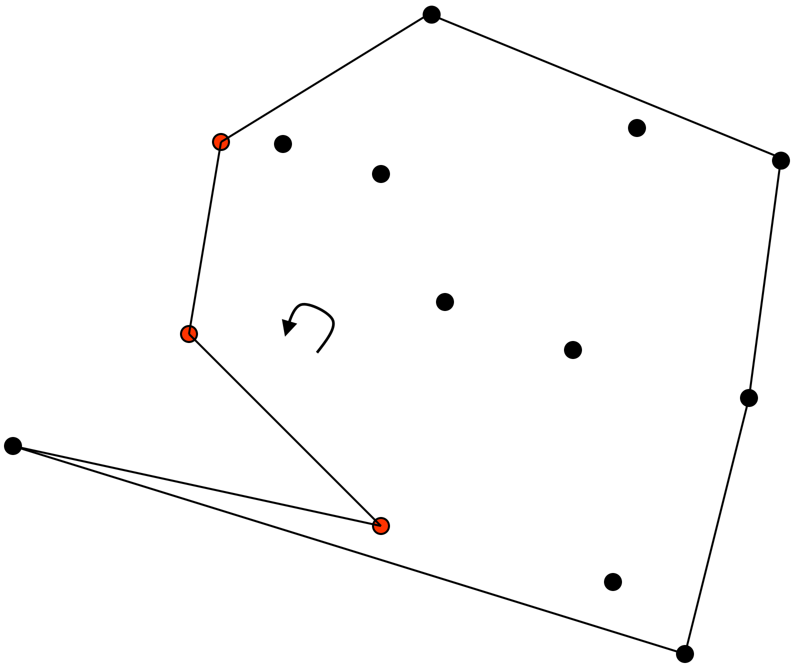
Graham



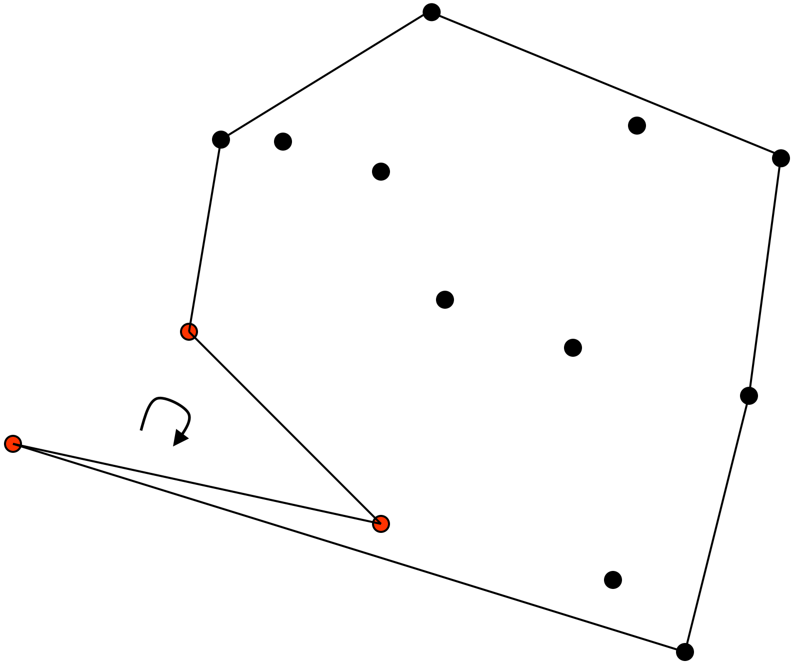
Graham



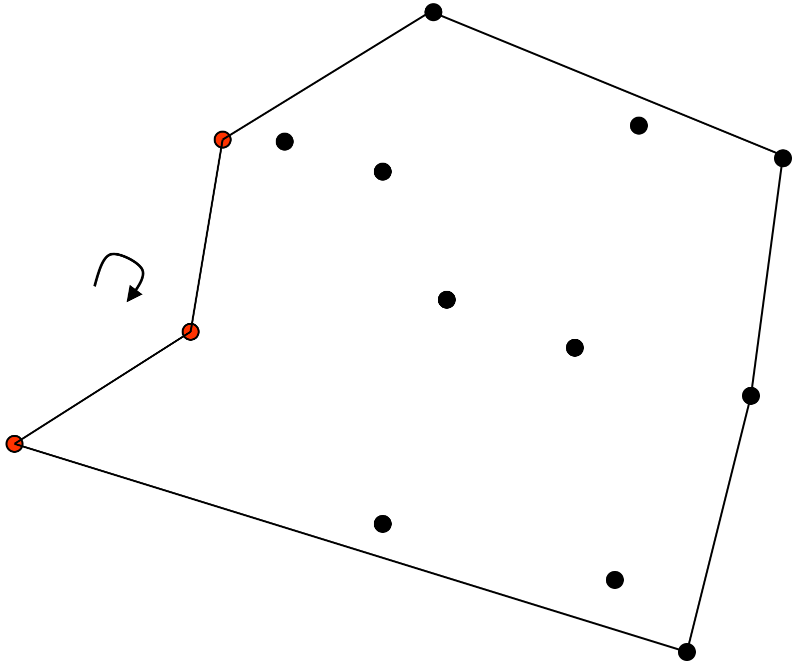
Graham



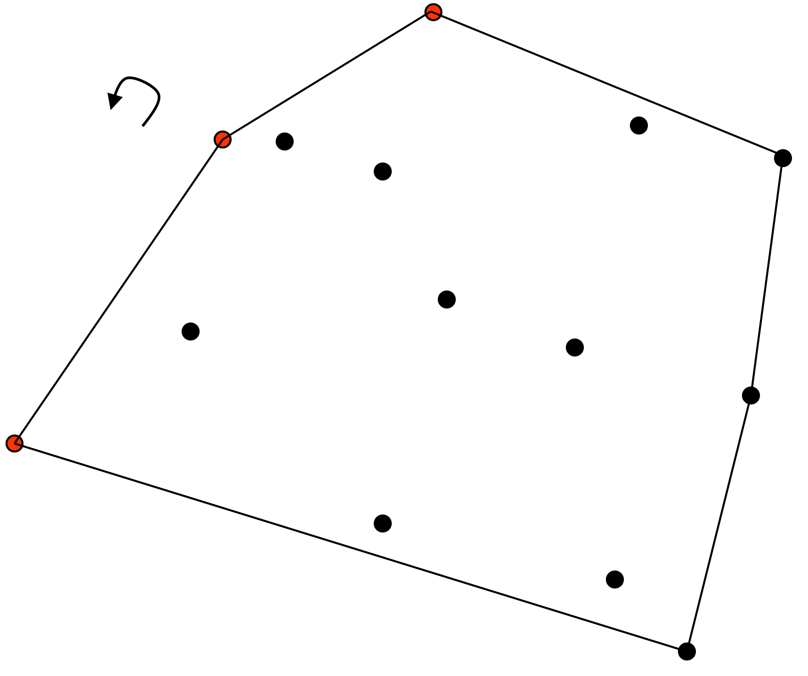
Graham



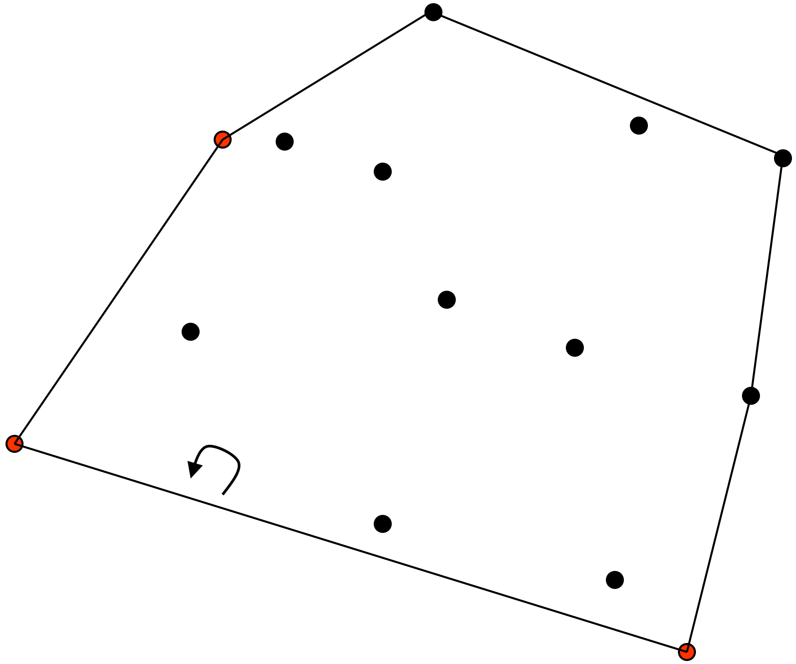
Graham



Graham



Graham



Graham

- En esta última etapa cada punto es examinado a lo sumo 2 veces – O(N)
- Finalmente el algoritmo es de:

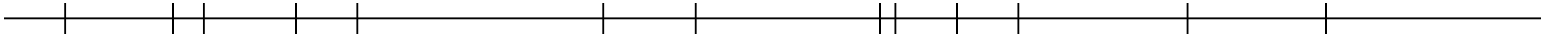
$$O(N)+O(N \log N) + O(N) = O(N \log N)$$

Par más cercano

- Dado un conjunto de puntos sobre el plano, ¿cuál es la distancia entre el par más cercano de puntos?
- Solución trivial de $O(N^2)$, verificando todos los pares de puntos del conjunto

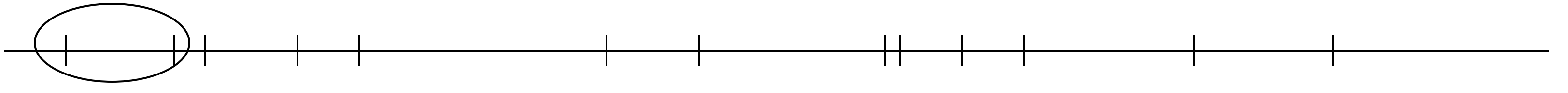
Par más cercano

- Pensemos primero en la solución del problema 1D:



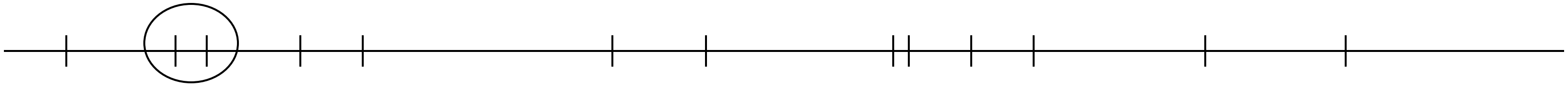
Par más cercano

- Podemos resolver el problema en O(N) explorando los puntos en orden:



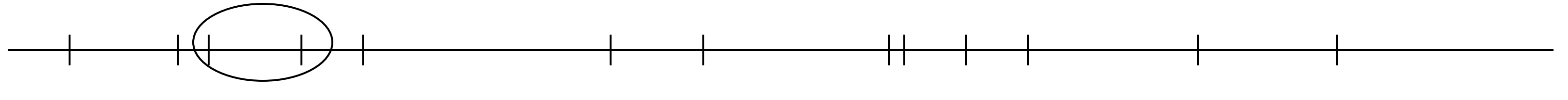
Par más cercano

- Podemos resolver el problema en O(N) explorando los puntos en orden:



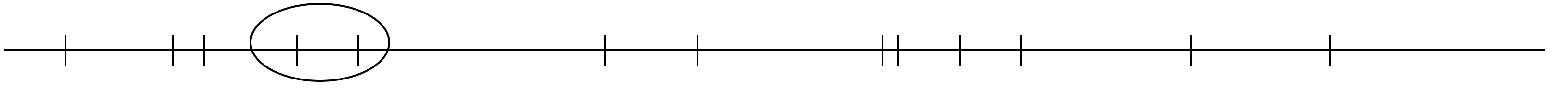
Par más cercano

- Podemos resolver el problema en O(N) explorando los puntos en orden:



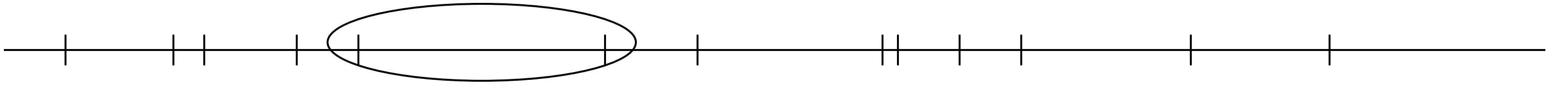
Par más cercano

- Podemos resolver el problema en O(N) explorando los puntos en orden:



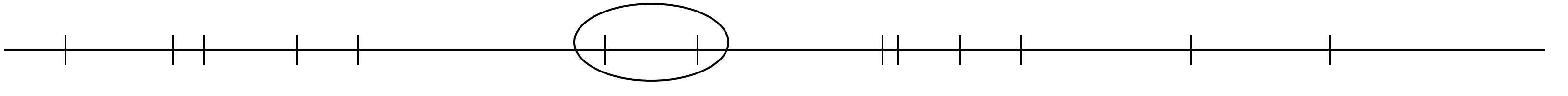
Par más cercano

- Podemos resolver el problema en O(N) explorando los puntos en orden:



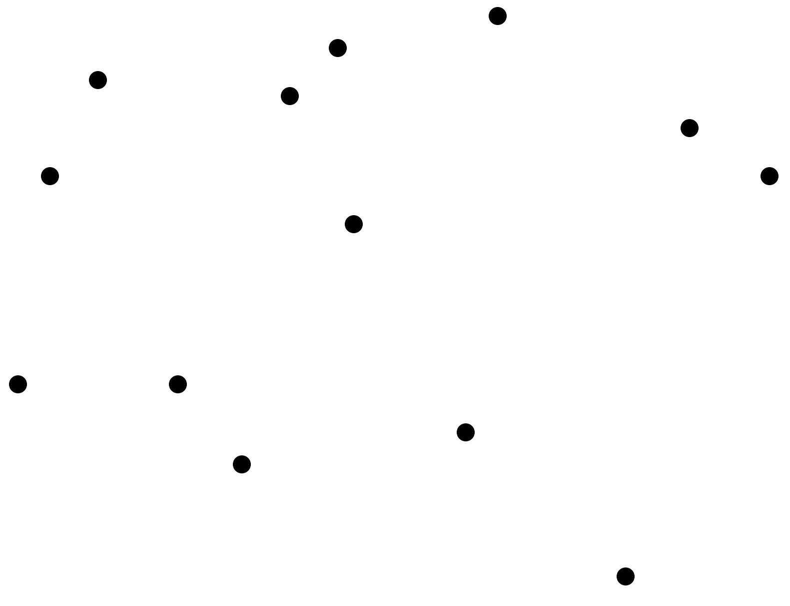
Par más cercano

- Podemos resolver el problema en O(N) explorando los puntos en orden:



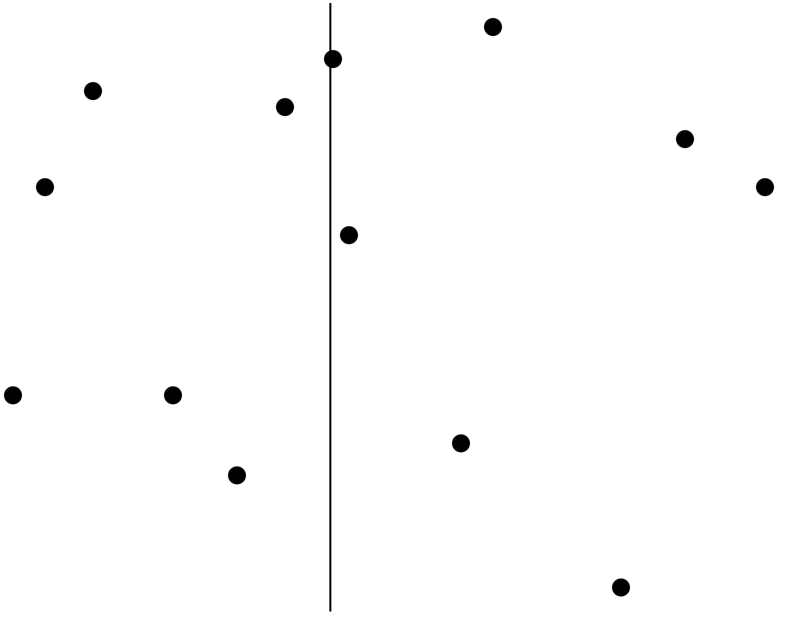
Par más cercano

- En 2D esto no es posible, ya que no existe un ordenamiento de los puntos



Par más cercano

- Podemos usar un esquema Divide and Conquer separando los puntos en la mediana según alguna coordenada

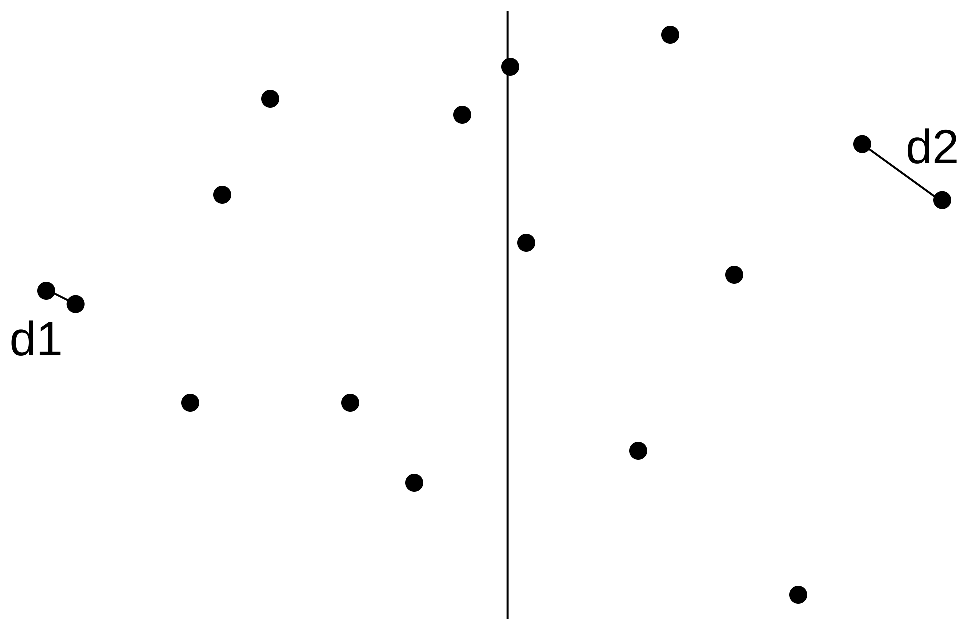


Par más cercano

- Si llegamos a dos puntos, el par más cercano es justamente ese par de puntos
- Si llegamos a un solo punto, no existe ahí un par más cercano
- ¿cómo unimos dos soluciones?

Par más cercano

- ¿cómo unimos dos soluciones?



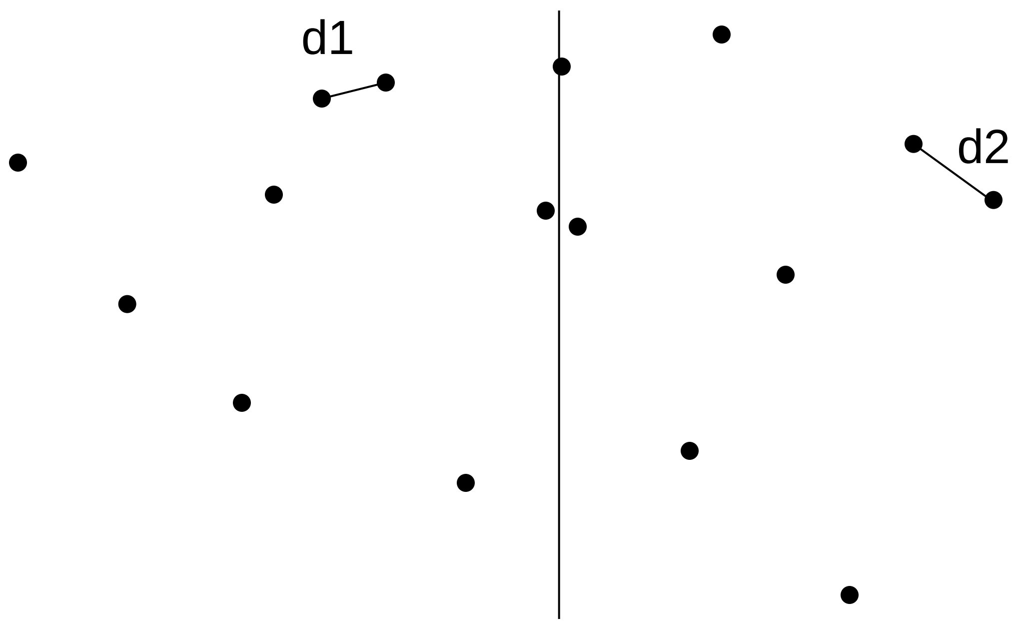
Distancia mínima d1

Distancia mínima d2

d = min(d1, d2)

Par más cercano

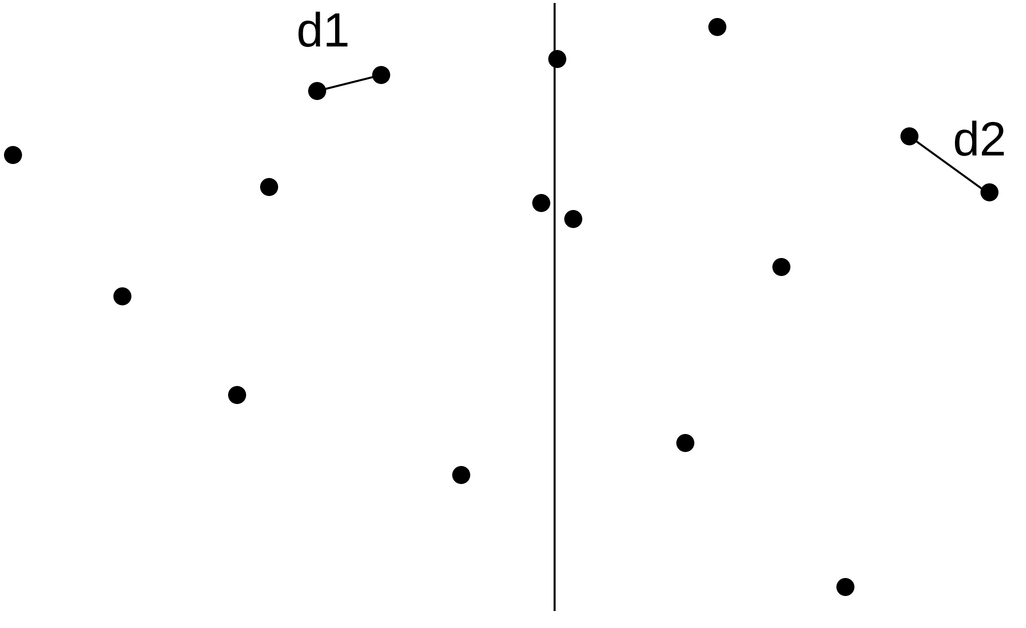
- ¿cómo unimos dos soluciones?



Pero es posible que la nueva distancia mínima no sea d1 ni d2, sino una nueva distancia entre algún punto del lado izquierdo y otro del lado derecho

Par más cercano

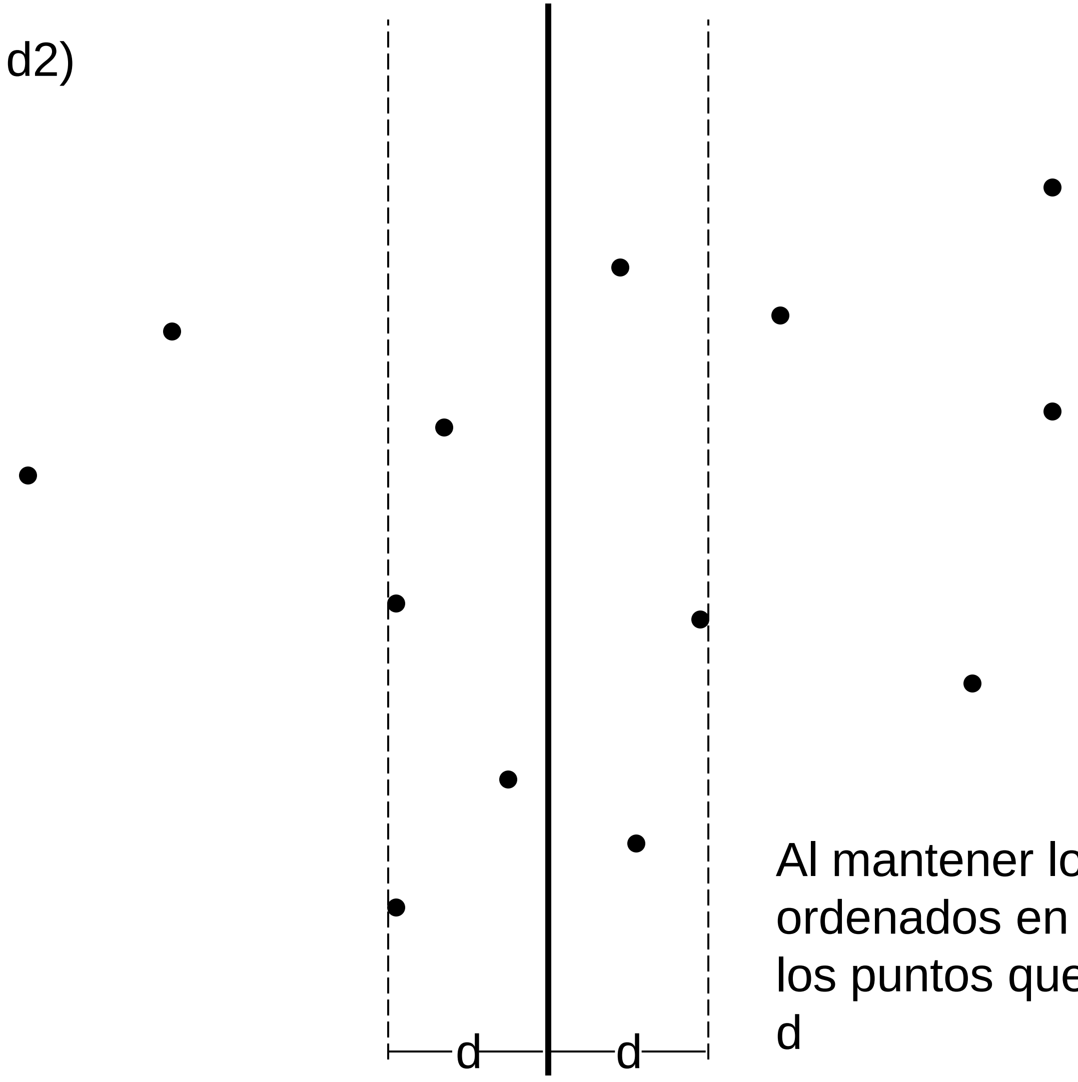
- ¿cómo unimos dos soluciones?



¿Es necesario probar todos los puntos del lado izquierdo contra todos los puntos del lado derecho?

Par más cercano

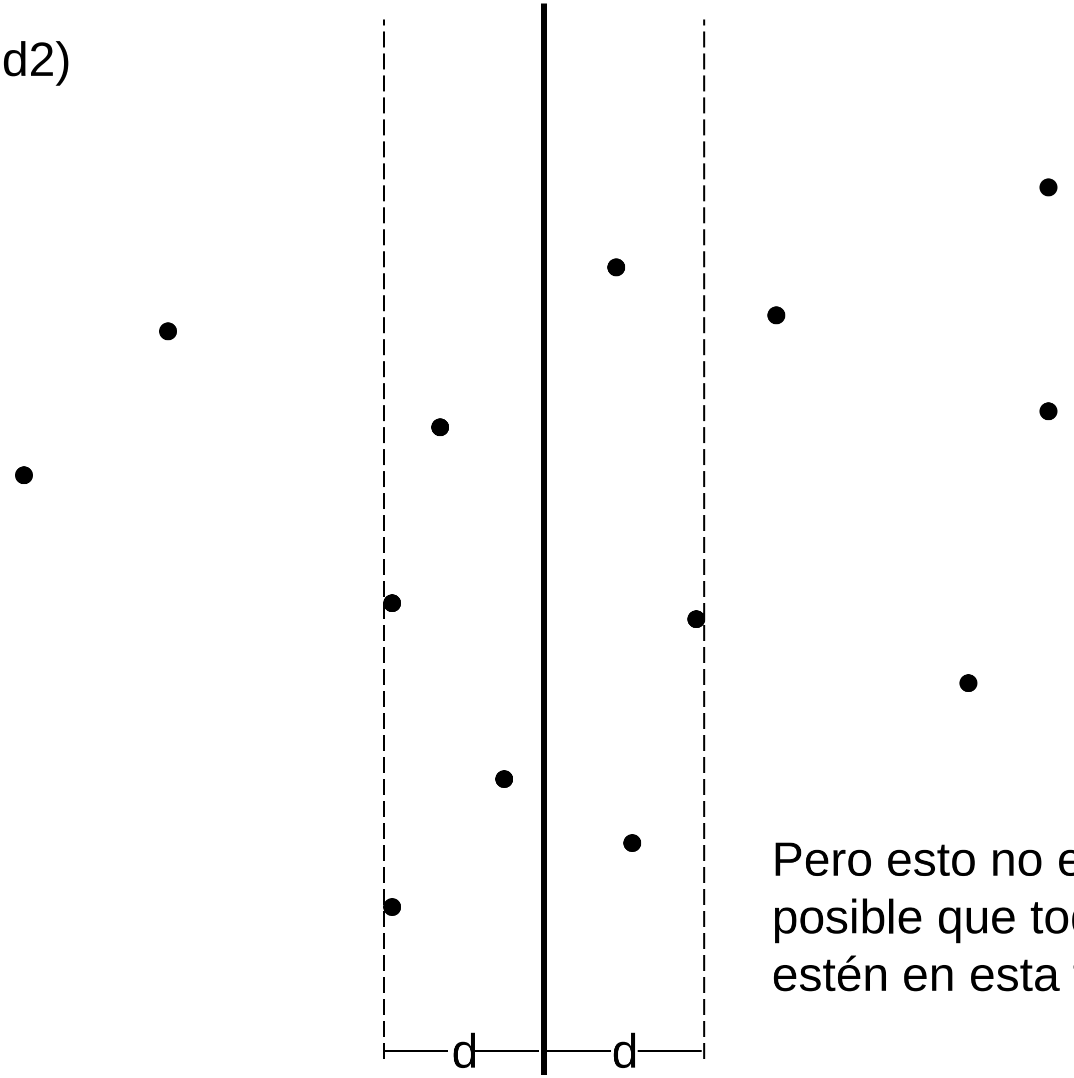
d = min(d1, d2)



Al mantener los puntos ordenados en X es fácil obtener los puntos que caen en la franja d

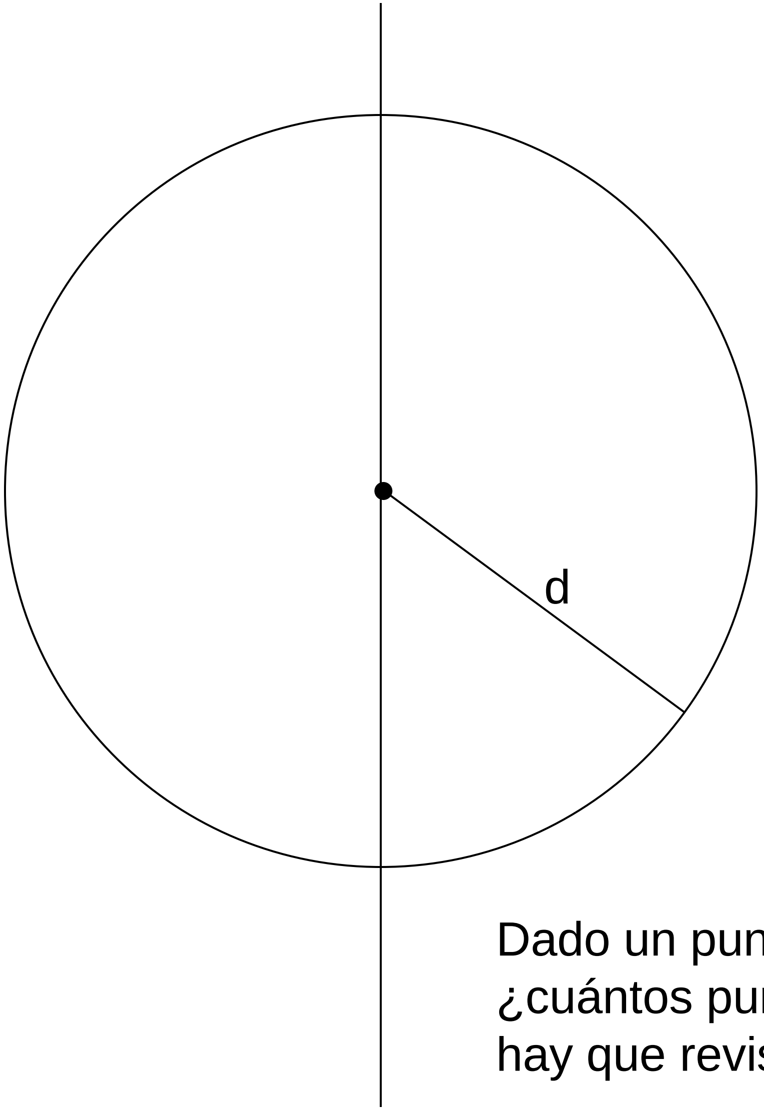
Par más cercano

d = min(d1, d2)



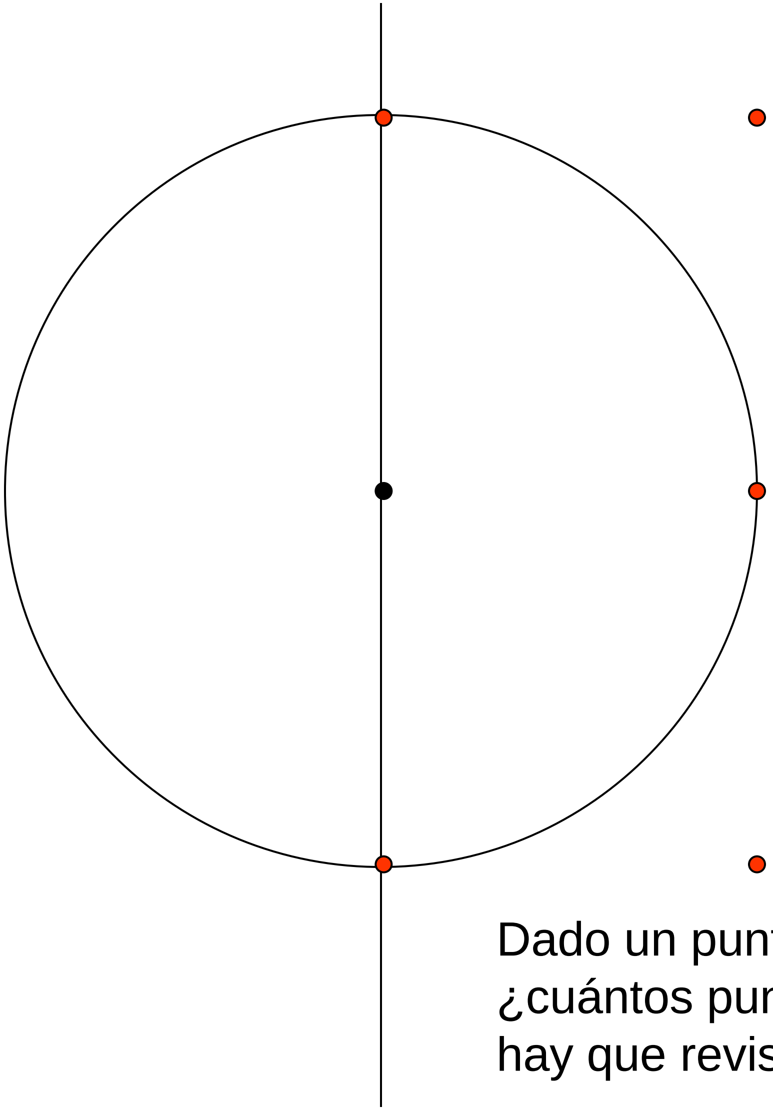
Pero esto no es suficiente, es posible que todos los puntos estén en esta franja!

Par más cercano



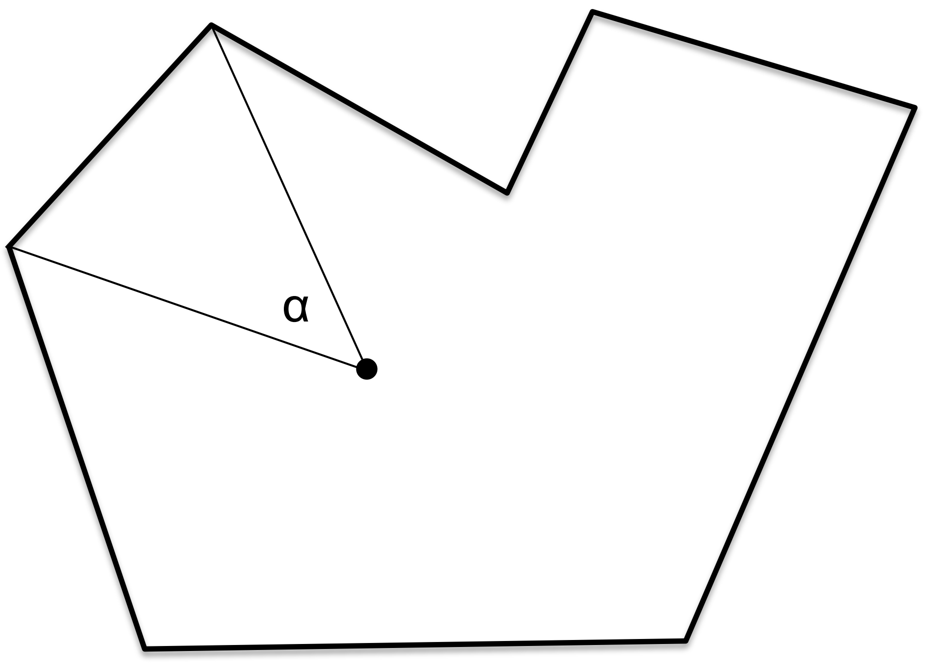
Dado un punto del lado izquierdo,
¿cuántos puntos del lado derecho
hay que revisar en el peor caso?

Par más cercano

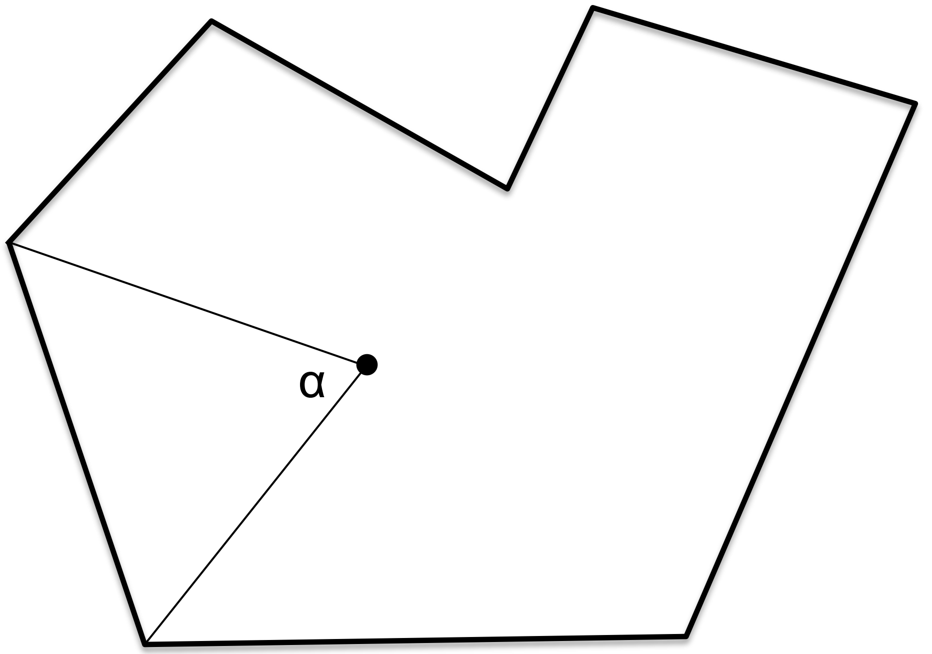


Dado un punto del lado izquierdo, ¿cuántos puntos del lado derecho hay que revisar en el peor caso?

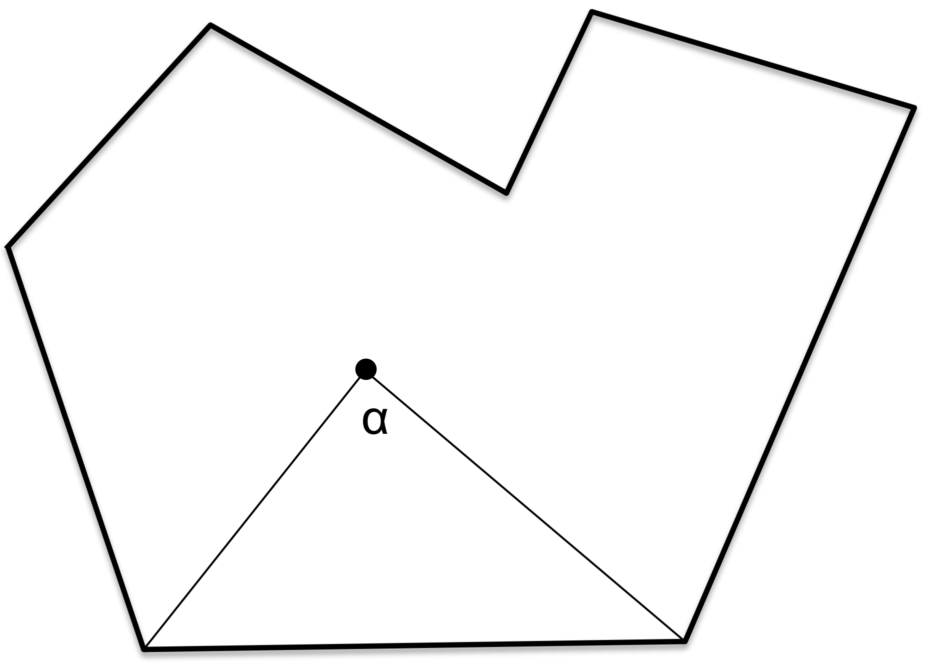
Punto en polígono



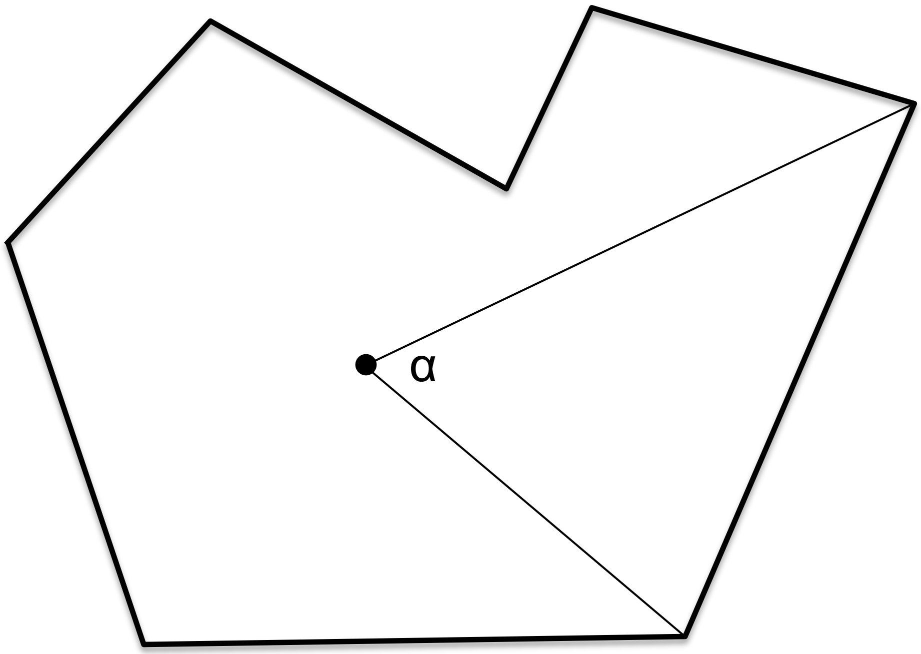
Punto en polígono



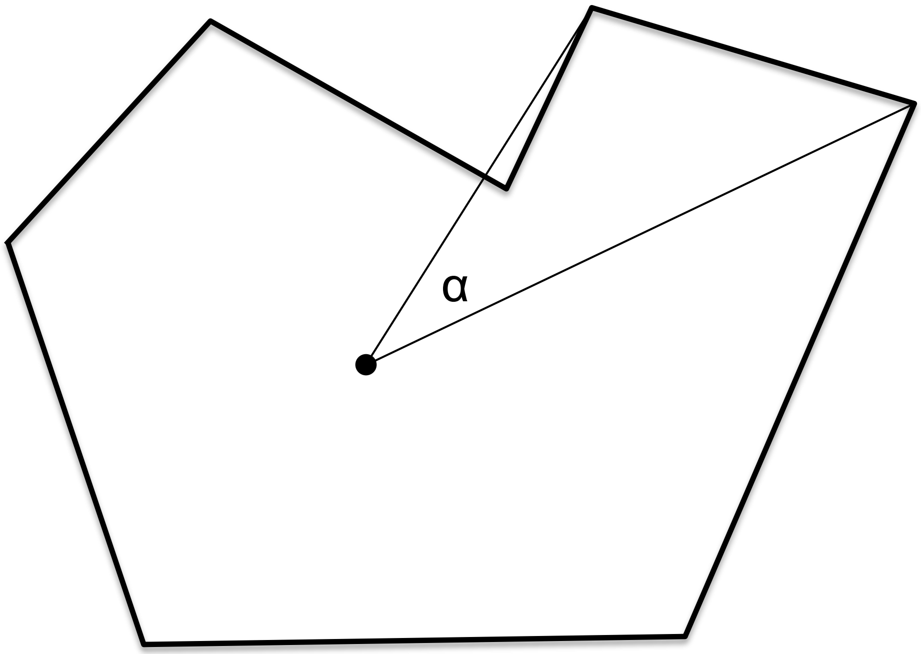
Punto en polígono



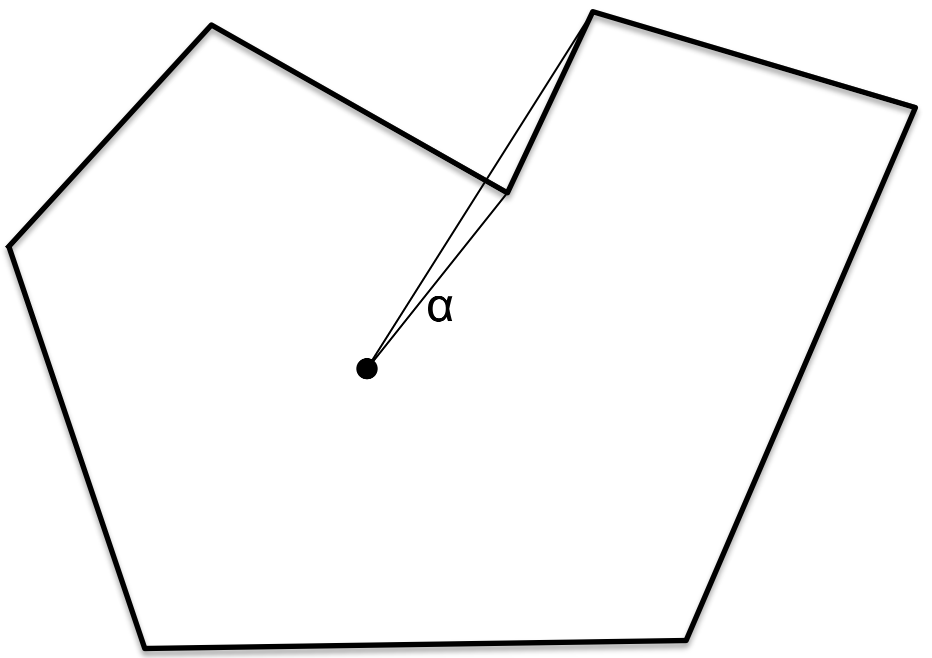
Punto en polígono



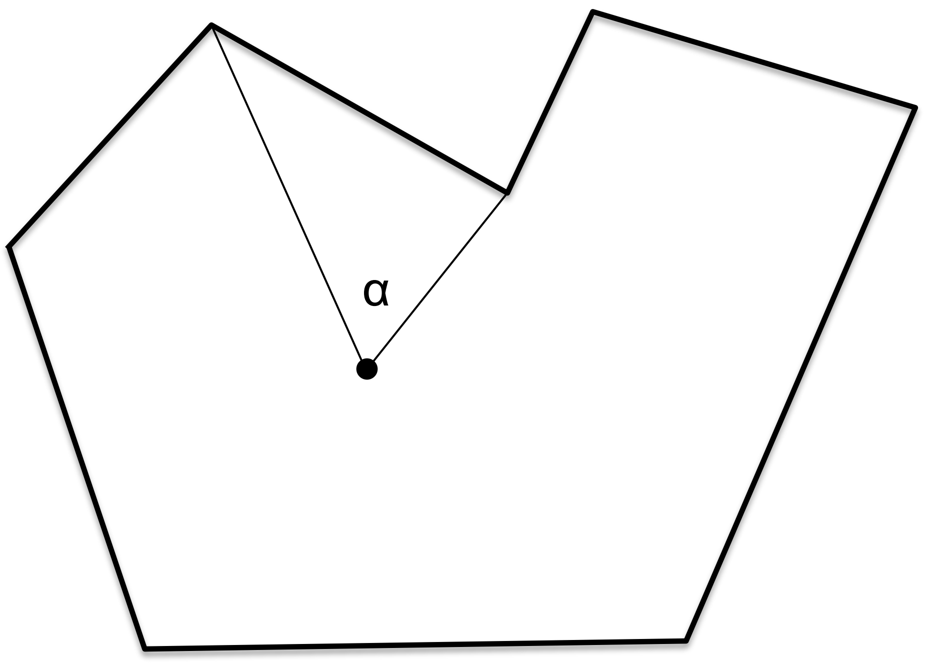
Punto en polígono



Punto en polígono

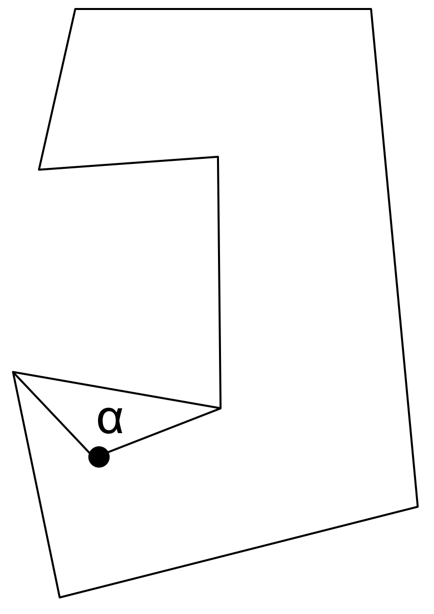


Punto en polígono

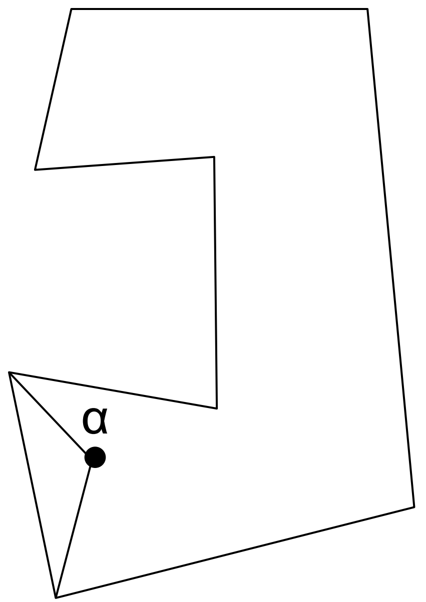


¿cuánto suman los ángulos?

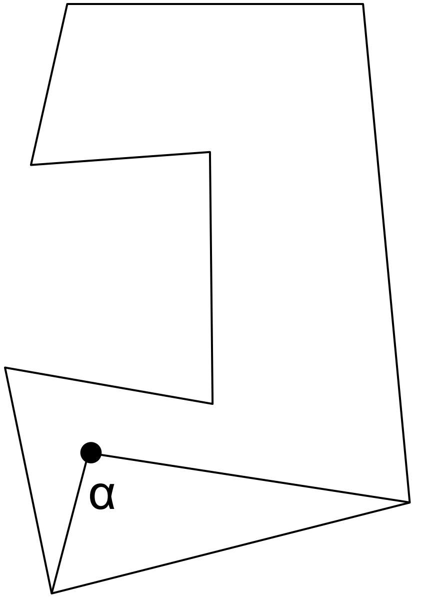
Punto en polígono



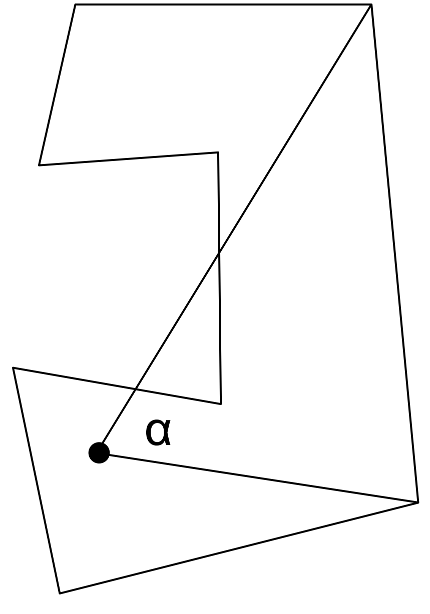
Punto en polígono



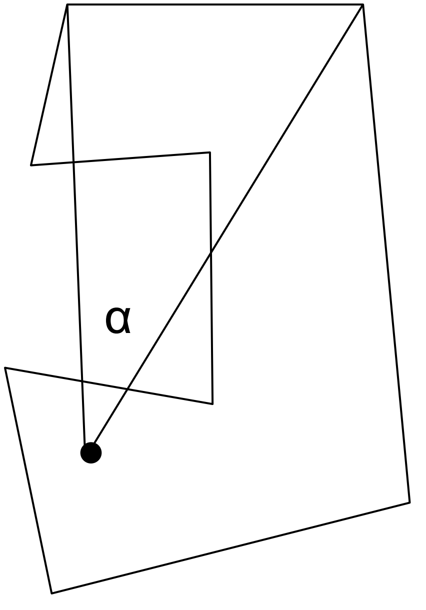
Punto en polígono



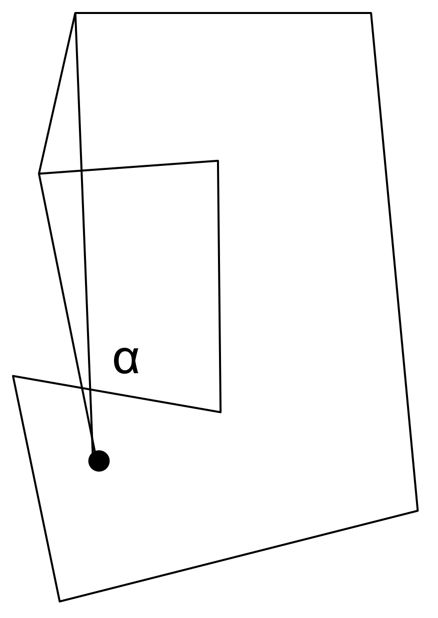
Punto en polígono



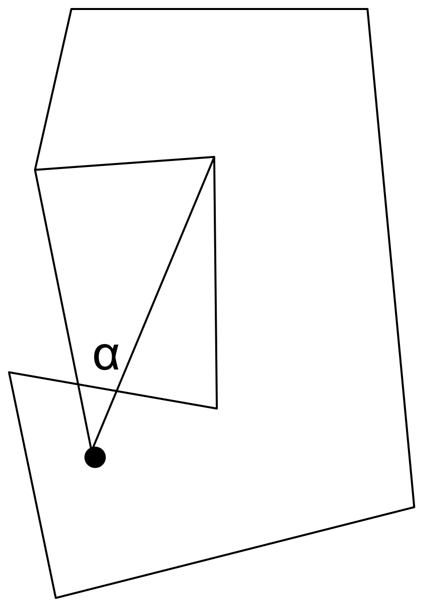
Punto en polígono



Punto en polígono

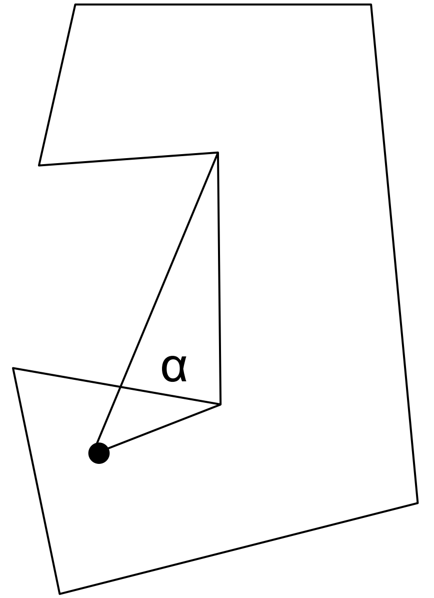


Punto en polígono



En este momento los puntos
están orientados CCW
El ángulo no se debe sumar
sino restar

Punto en polígono

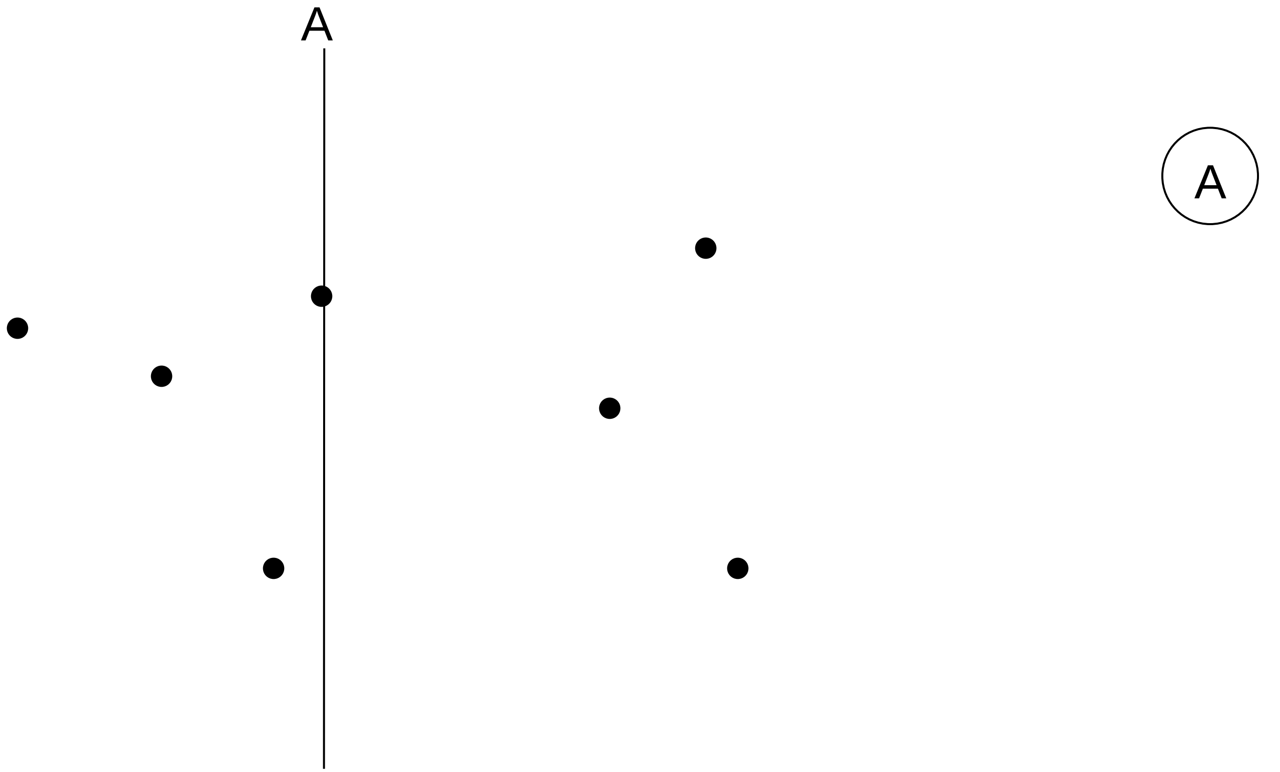


En este momento los puntos están orientados CCW
El ángulo no se debe sumar sino restar

Árboles KD

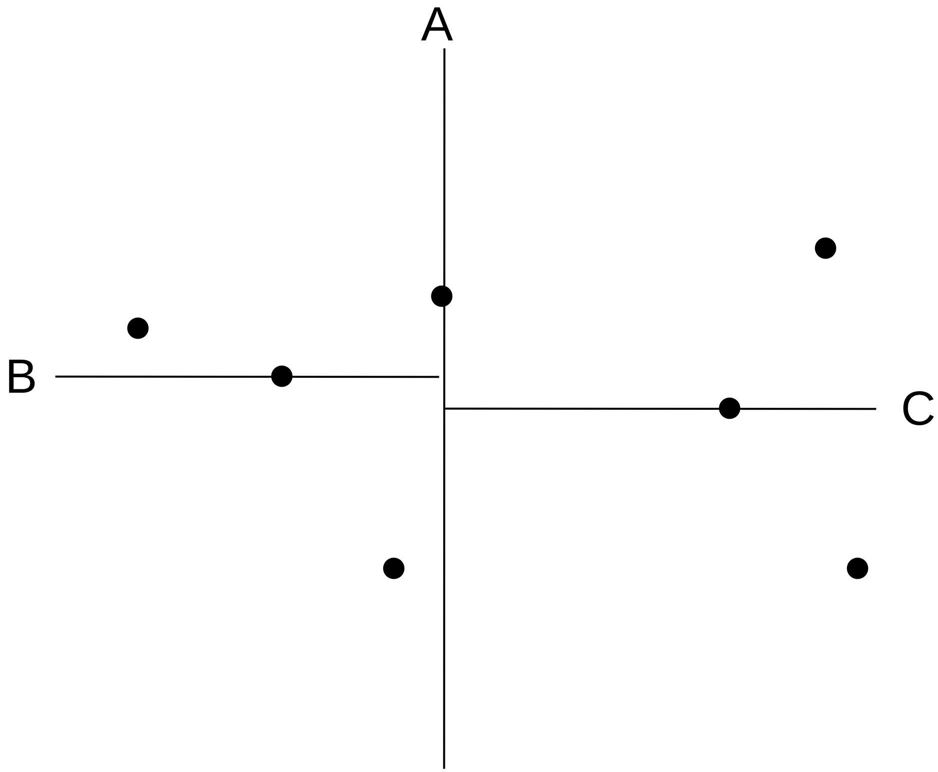
- Permiten dividir el espacio jerárquicamente
- En cada nodo del árbol se almacena una recta paralela al eje X o Y
- El eje al cual las rectas son paralelas se alterna en cada nivel

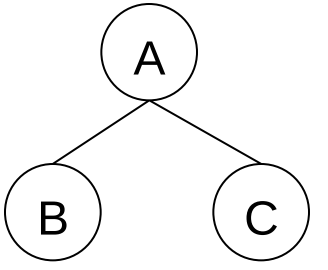
Árboles KD



Primer nivel: Dividir el espacio con una recta paralela a Y

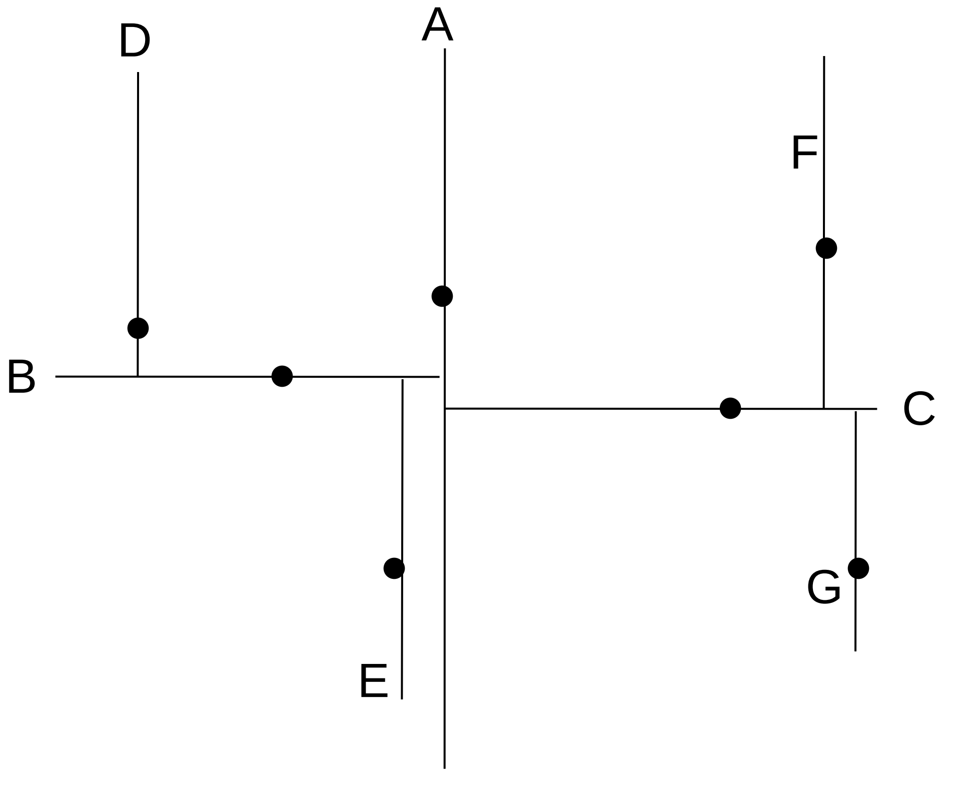
Árboles KD

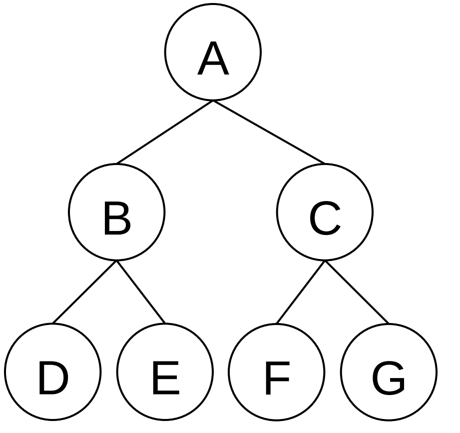




Segundo nivel: Dividir el espacio con rectas paralelas a X

Árboles KD





Tercer nivel: Dividir el espacio con rectas paralelas a Y

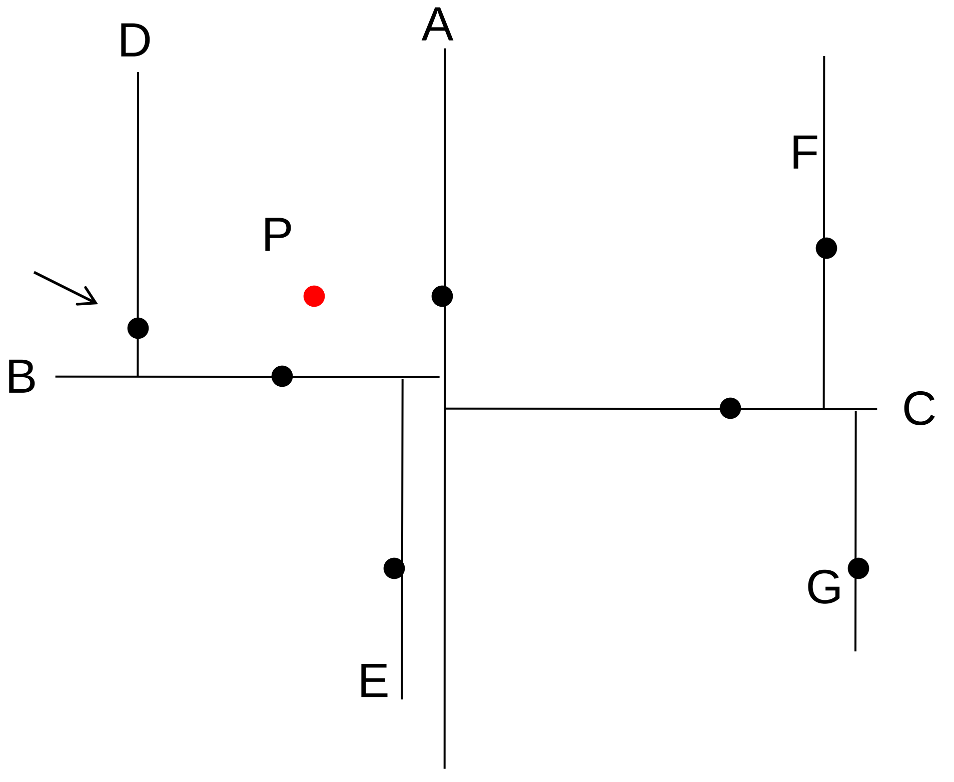
Árboles KD

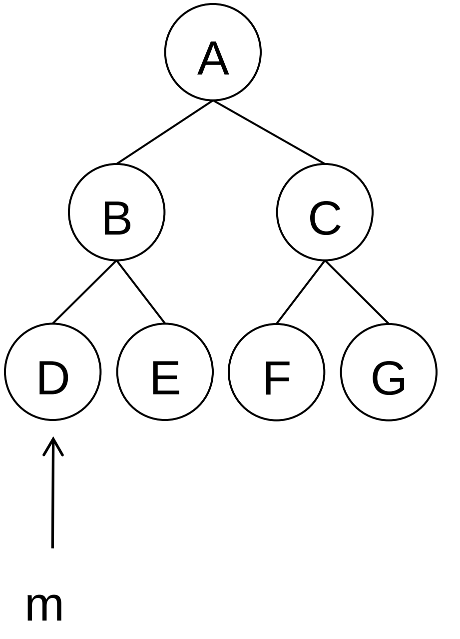
```
struct NodoKD
{
 struct *izq, *der;
 Punto2D p;
 float valor;
};
```

Punto más cercano

- Dado un conjunto de puntos C y un punto P, ¿cuál es el punto en C más cercano a P?
- Solución trivial de O(N)
- Construir el árbol KD de C (T)
- Hacer una búsqueda binaria en T
- El nodo hoja final es el más cercano (m)

Punto más cercano





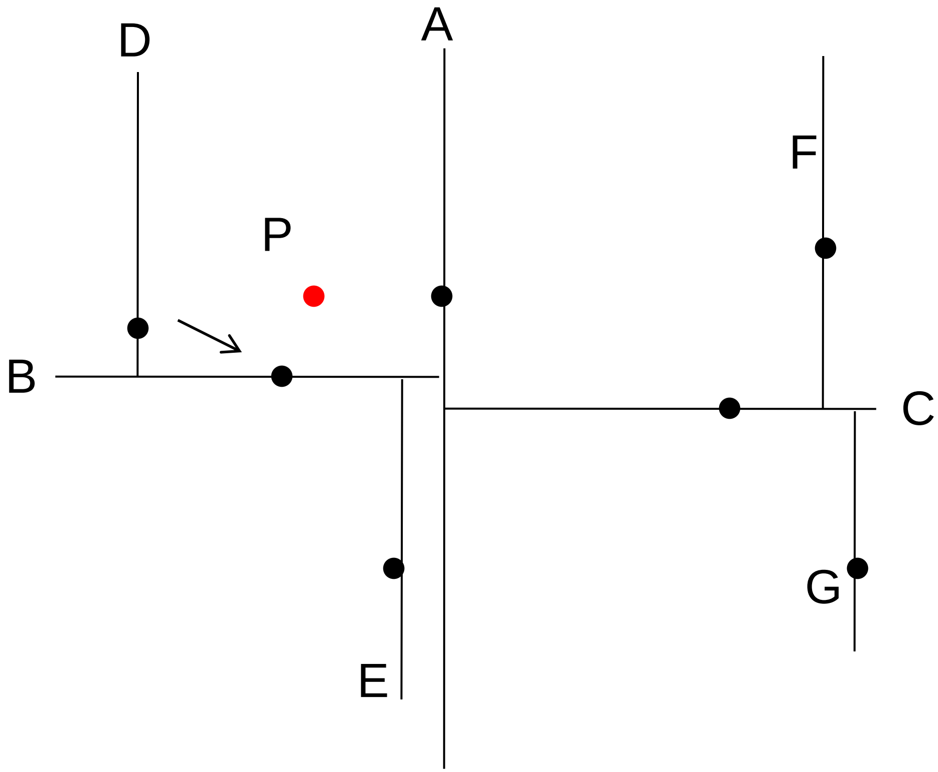
Punto más cercano

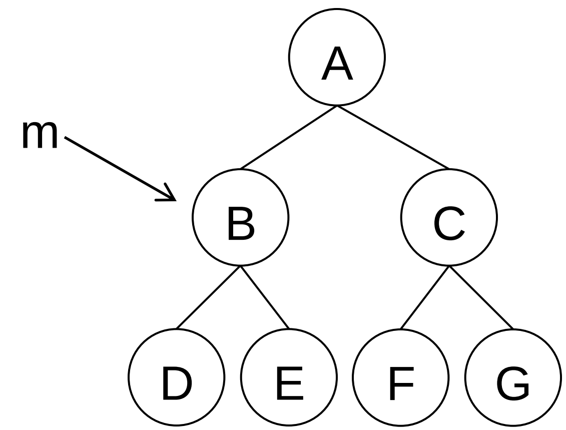
- Ahora en cada llamada recursiva ver si el nodo actual está más cerca de P, que el que es hasta ahora el más cercano
- Verificar si es posible que algún punto del otro lado del nodo esté más cerca a P
- Esto se hace viendo la distancia de P a la recta almacenada en el nodo actual **n**

Punto más cercano

- Si |n.r-P|<|m-P| entonces se aplica recursivamente el mismo procedimiento del otro lado de la recta
- En caso contrario, seguir retornando recursivamente

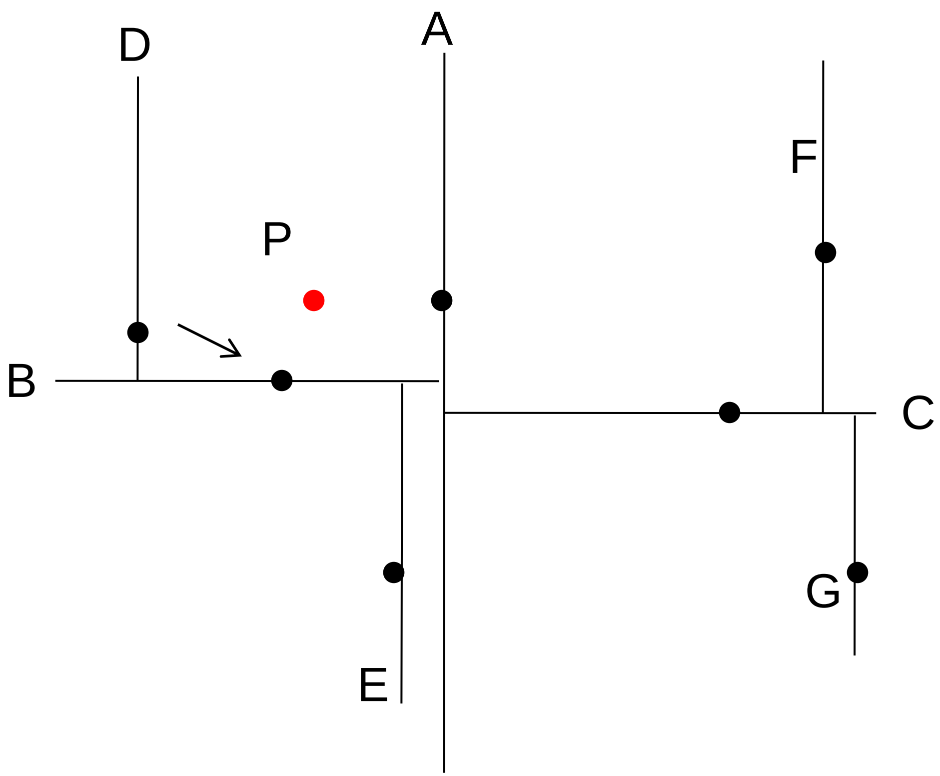
Punto más cercano

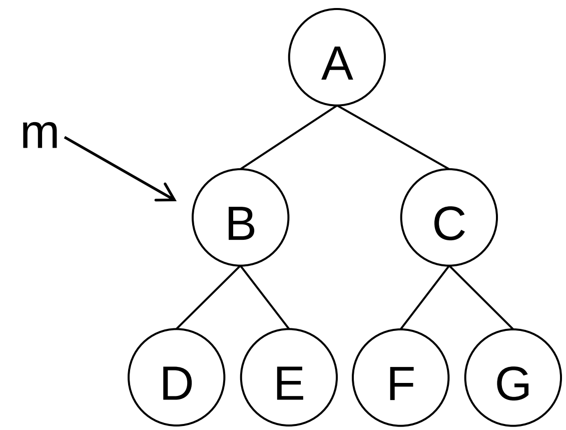




Aquí es posible que exista un punto más cercano a P del lado derecho de B

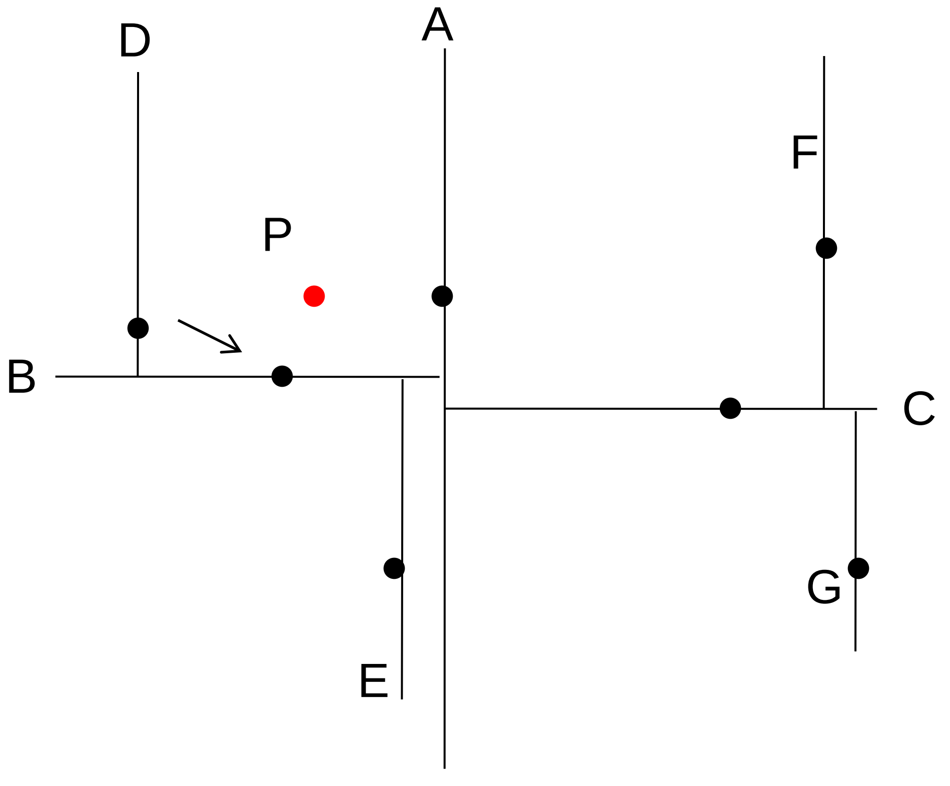
Punto más cercano

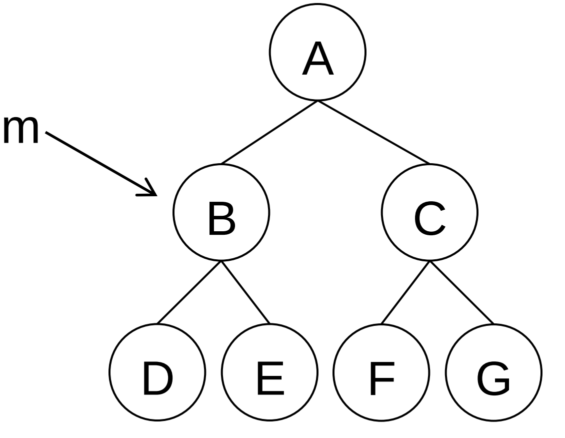




Recursivamente vemos que E está
mas lejos de P que B

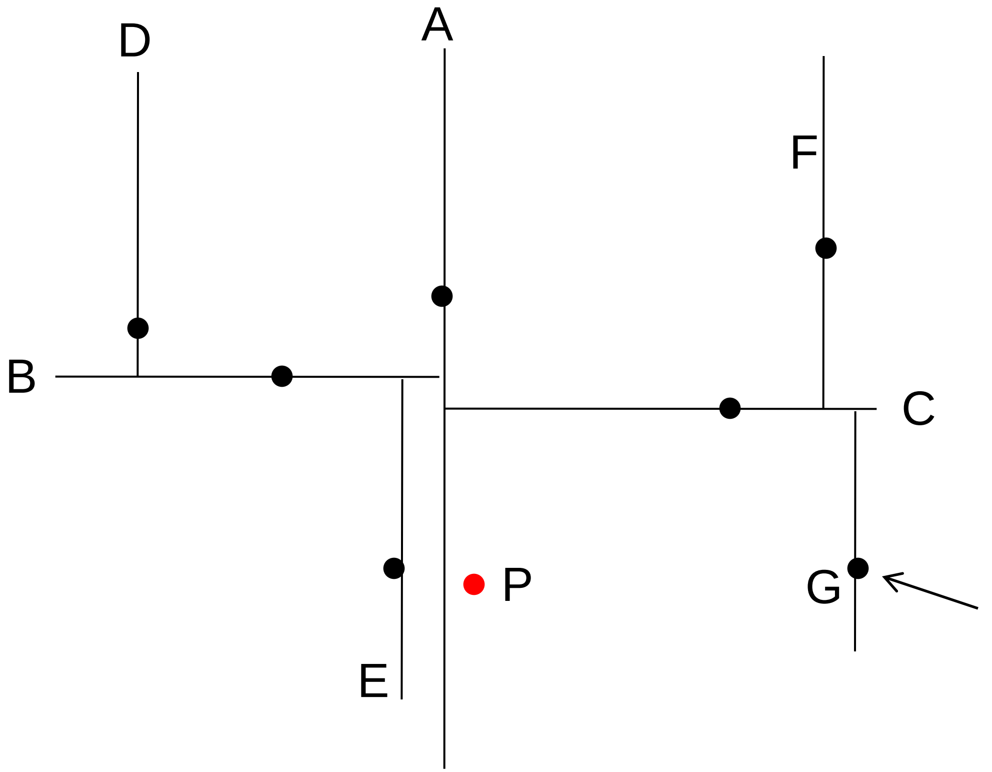
Punto más cercano

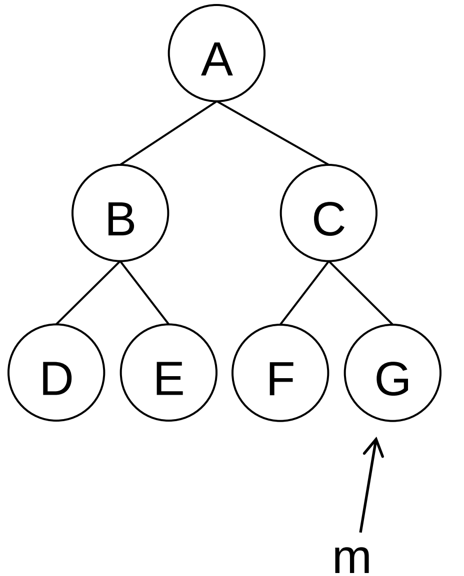




Regresamos al ambiente en donde está la raíz A. La recta A está más lejos de P que B, por lo que no hay que buscar del lado derecho de A

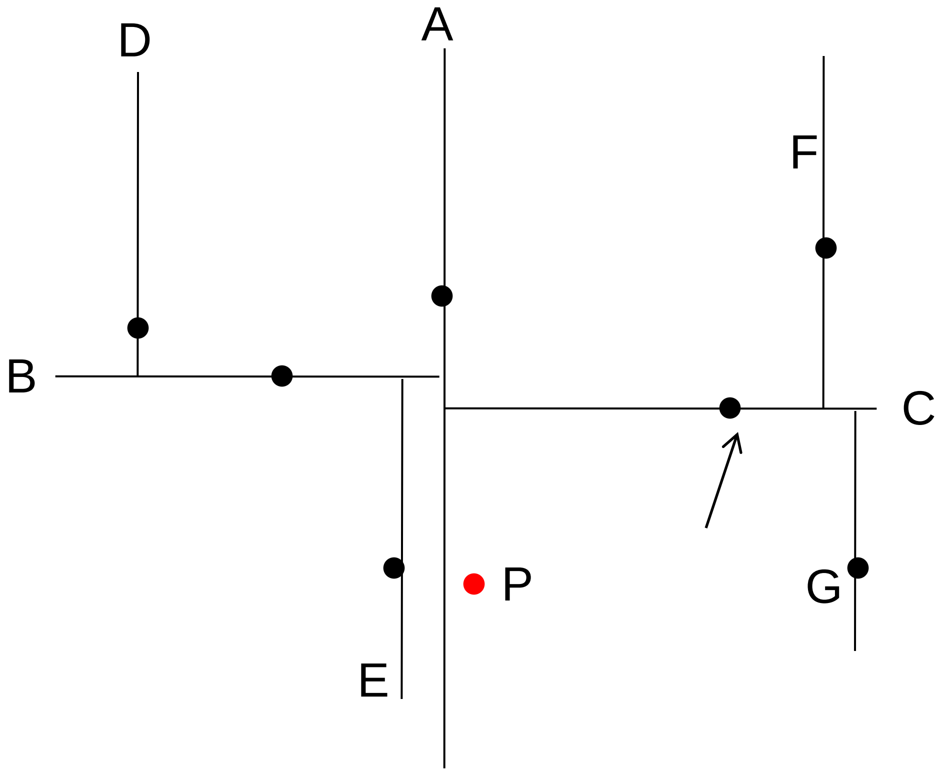
Punto más cercano

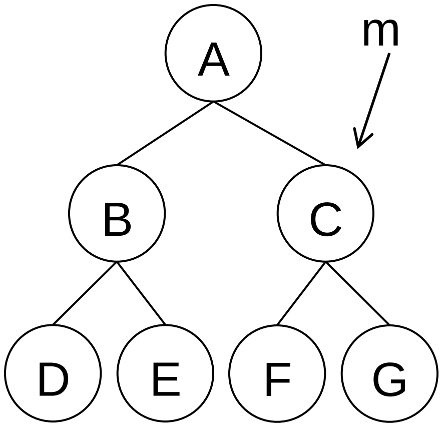




En este caso la 1ra parte del algoritmo se detiene en G

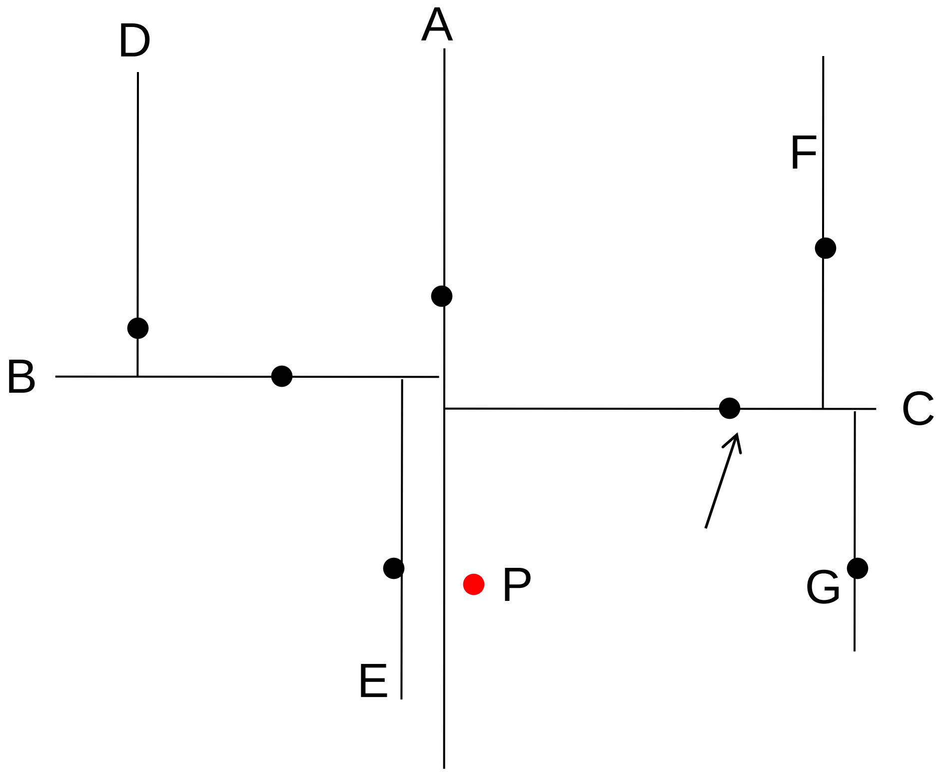
Punto más cercano

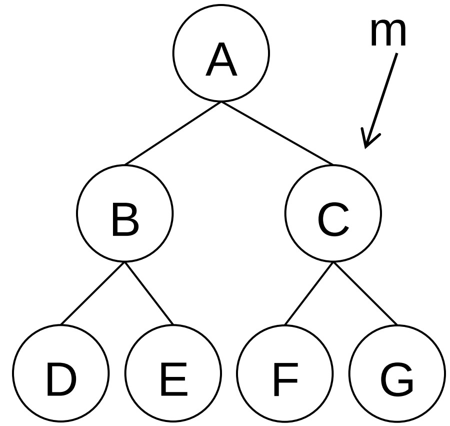




El algoritmo se devuelve a C, el cual está mas
cerca de P

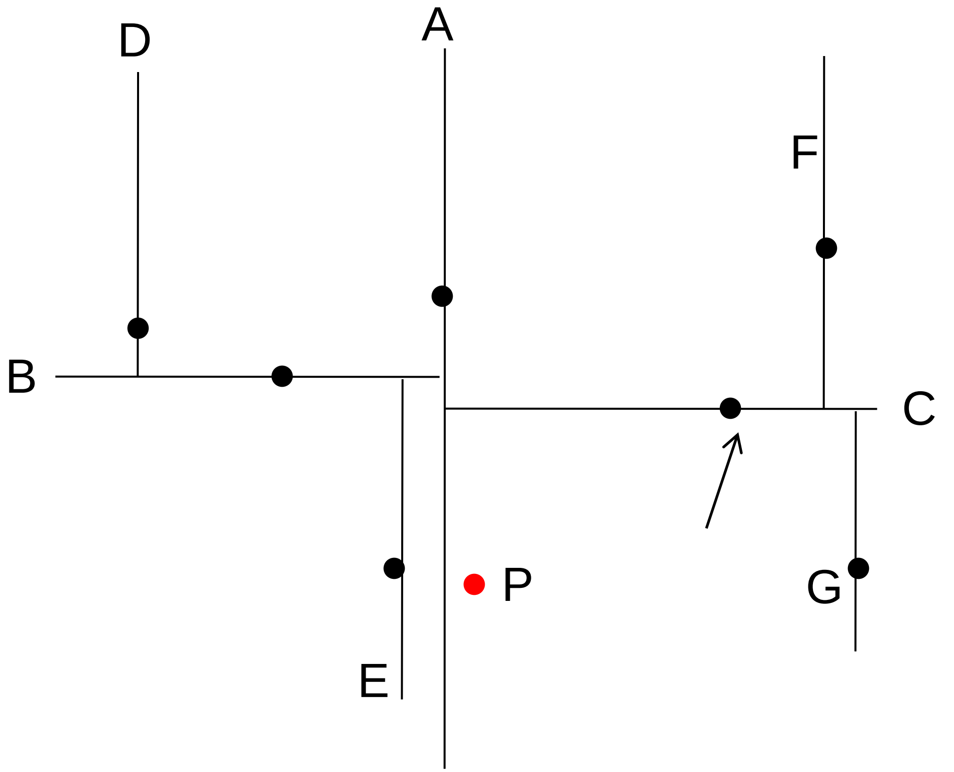
Punto más cercano

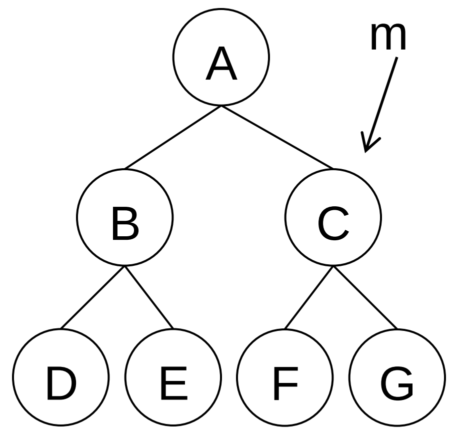




La recta que pasa por C está mas cerca de P que C, por lo que hay que buscar del otro lado de la recta

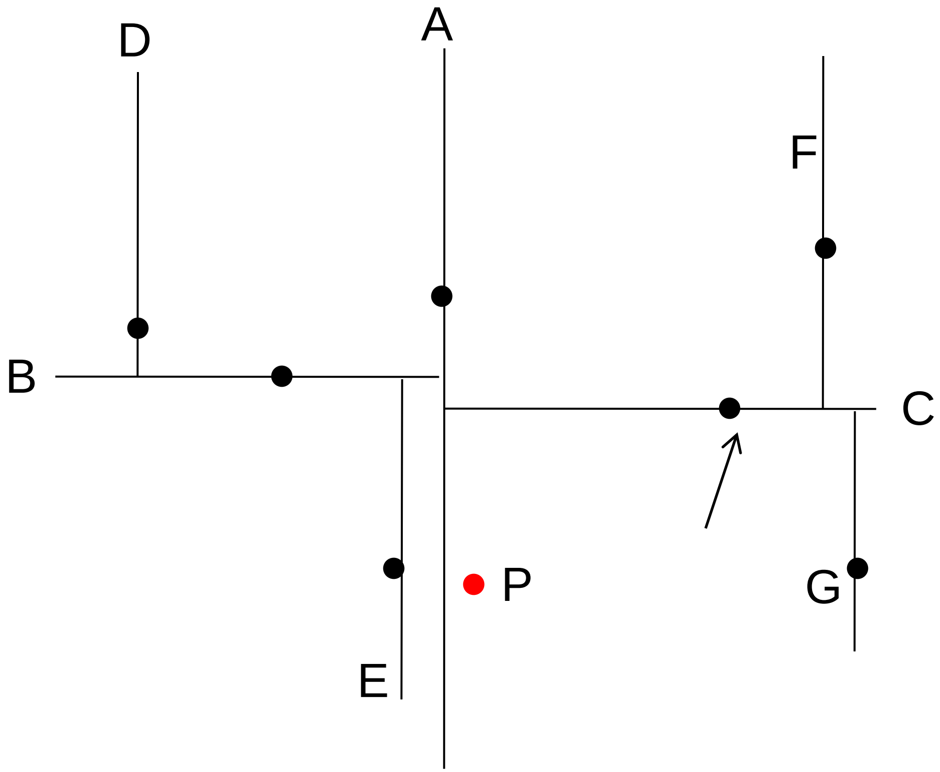
Punto más cercano

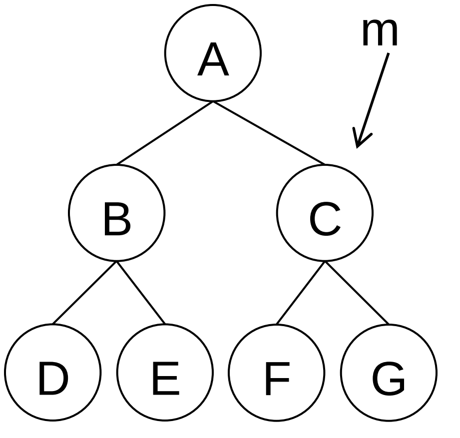




Se busca F, pero C sigue estando más cerca.
Retornar al nodo A

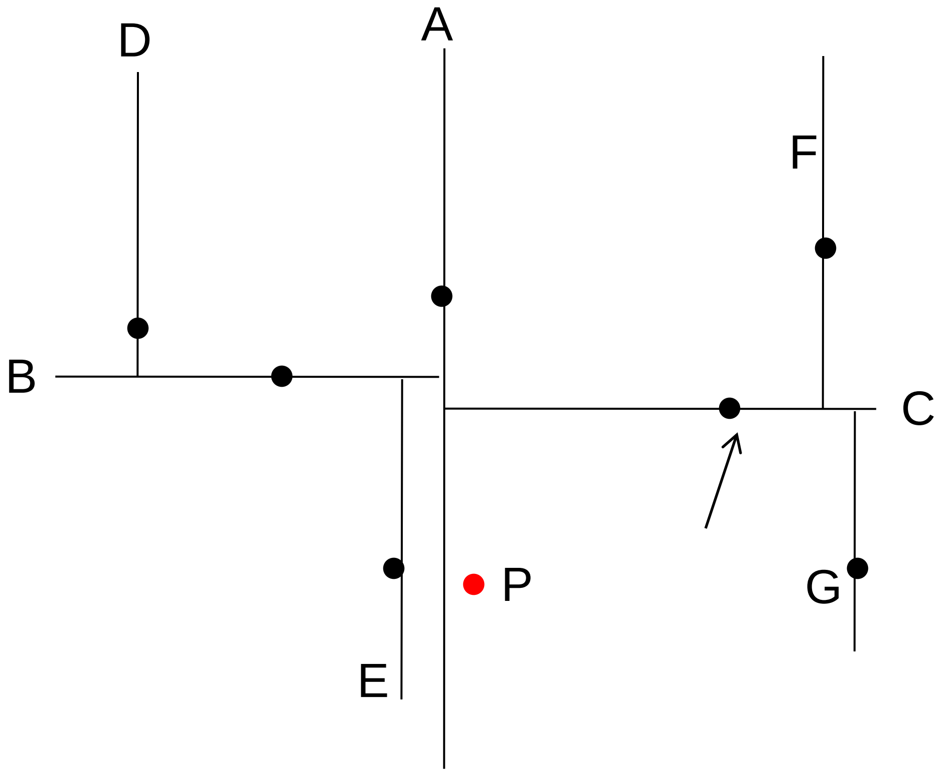
Punto más cercano

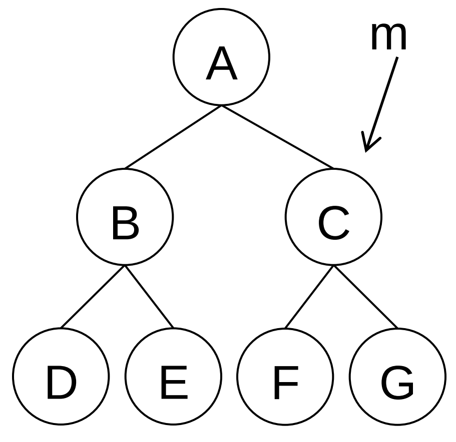




La recta que pasa por A está mas cerca de P que C, por lo que hay que buscar a la izquierda de A

Punto más cercano

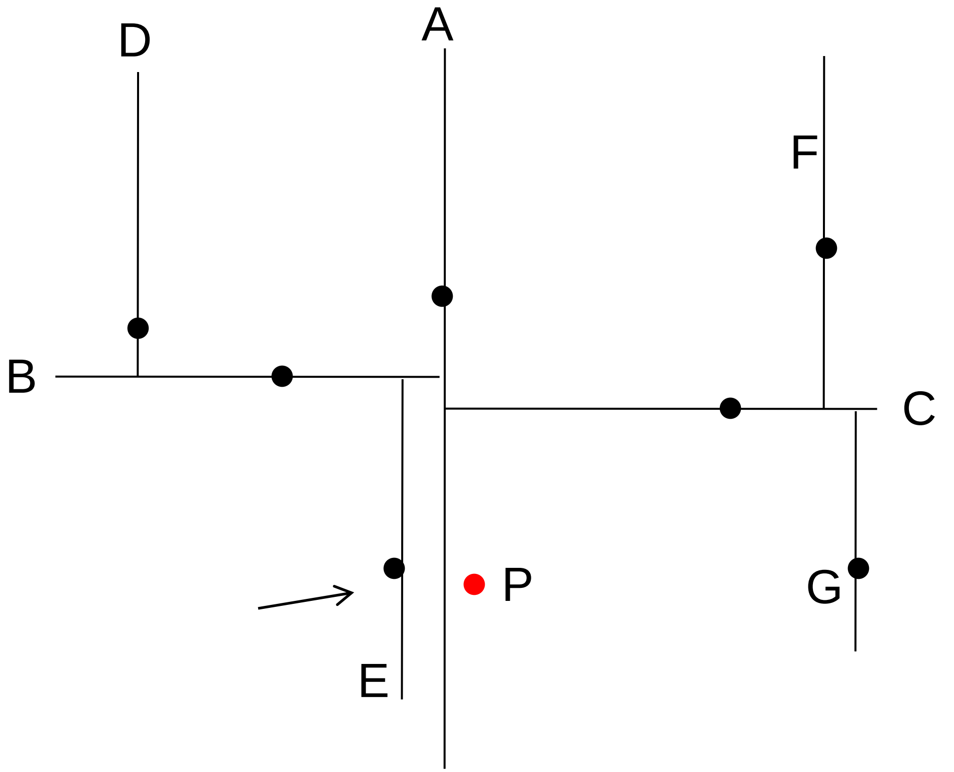


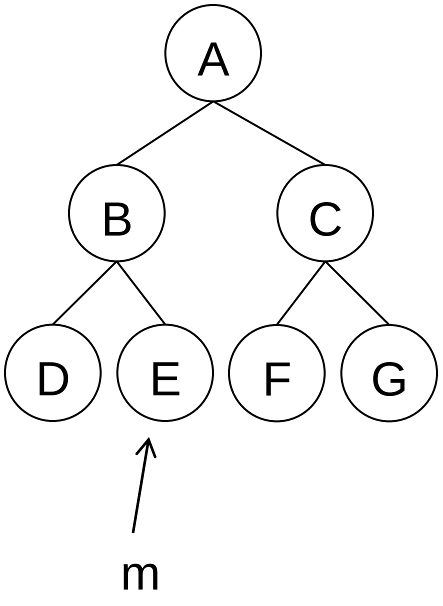


B no está más cerca de P que C

Como P está debajo de B, buscar ahora ahí

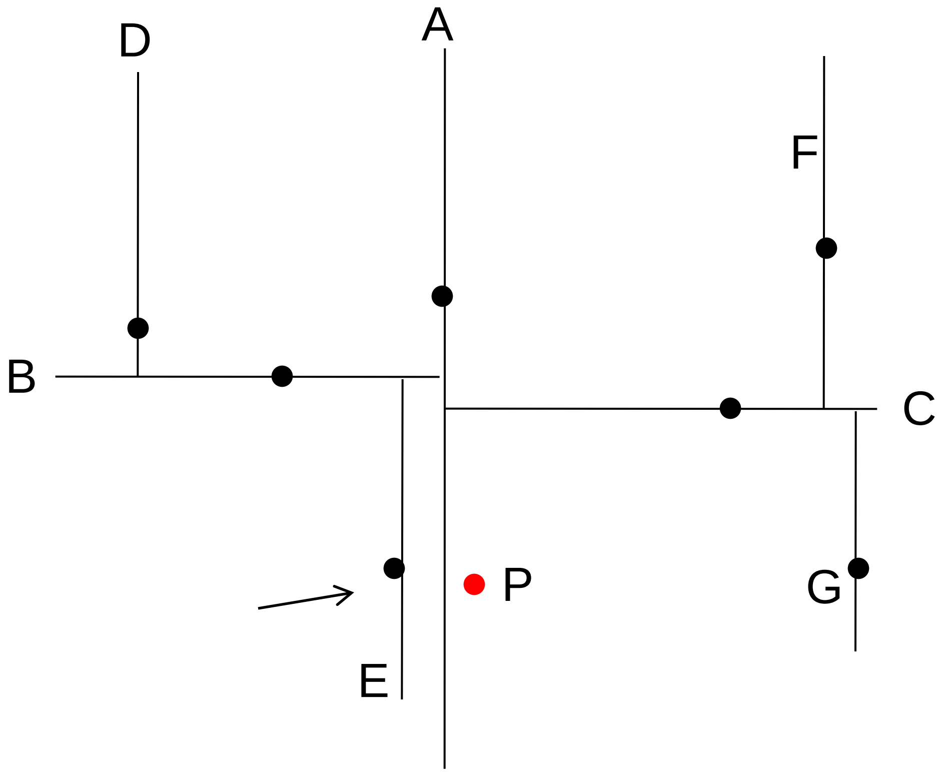
Punto más cercano

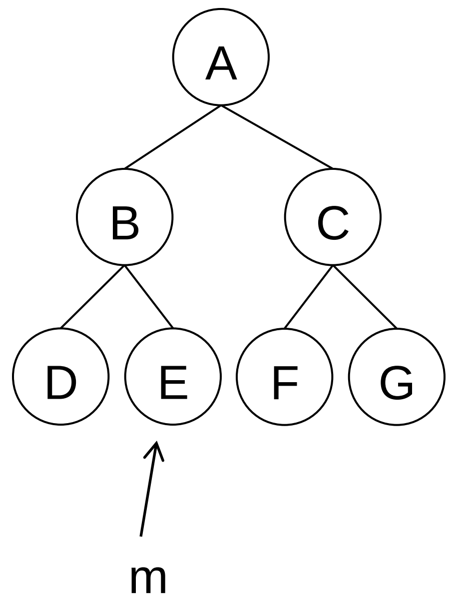




Encontramos ahora E que está más cerca de
P que C

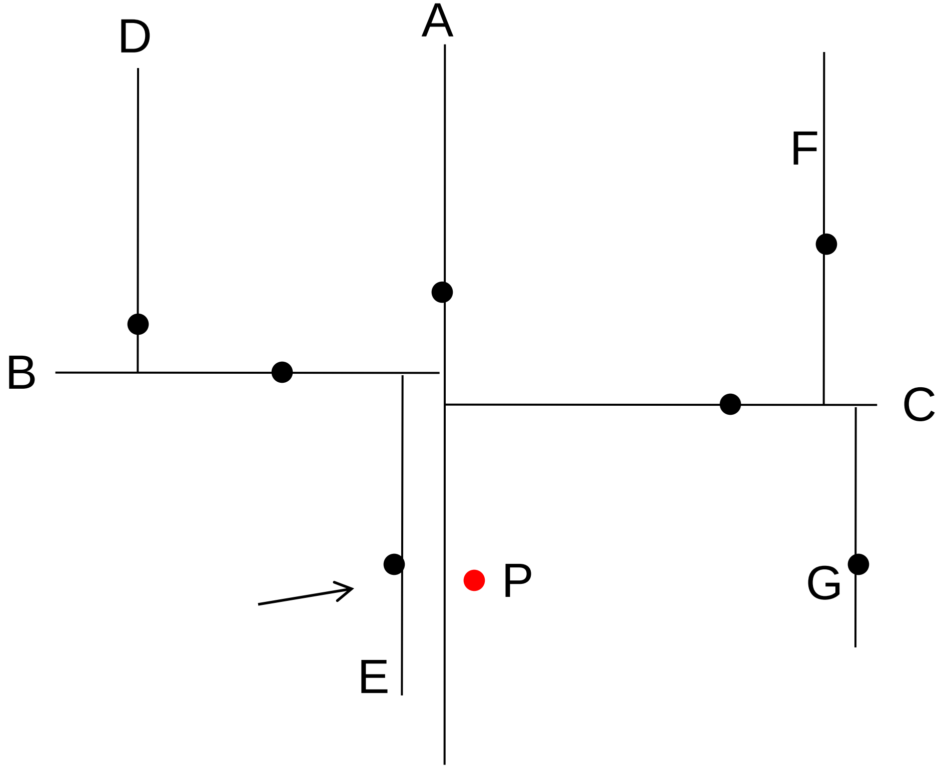
Punto más cercano

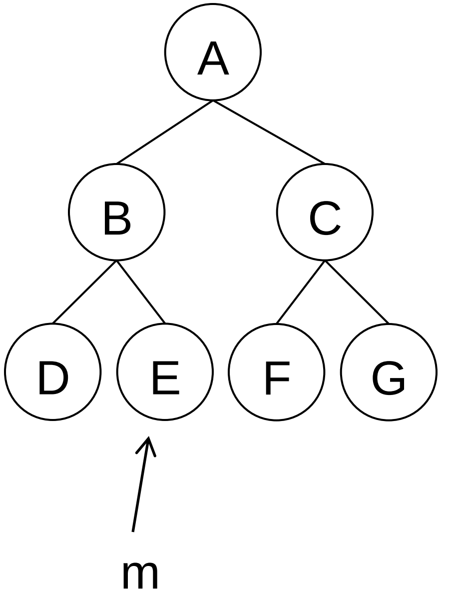




Ahora cuando regresamos a B vemos que la distancia de P a E es menor que la distancia de E a la recta que pasa por B

Punto más cercano





Eso significa que no hay que buscar en el otro lado del nodo B
Ahora regresamos a la llamada en el nodo A, y ya no hay más llamadas

Punto más cercano

- La complejidad de la búsqueda para puntos aleatorios es de O(log N)
- Pueden construirse casos en donde el algoritmo es de O(N)
- Es posible realizar cambios al algoritmo para garantizar búsquedas de O(log N)
- Es posible hacer un algoritmo aproximado que se detenga después de cierta cantidad de iteraciones

Árboles KD

- Otro problema que se resuelve con árboles KD son las búsquedas en rango:
- ¿cuáles puntos están a distancia máxima K de cierto punto P?