Geometría Computacional

Conceptos básicos

Problemas geométricos

Segmentos

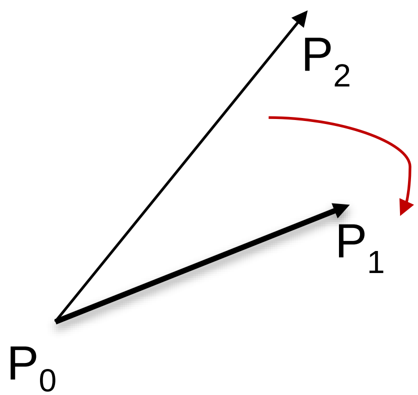
Problemas geométricos

Segmentos de línea

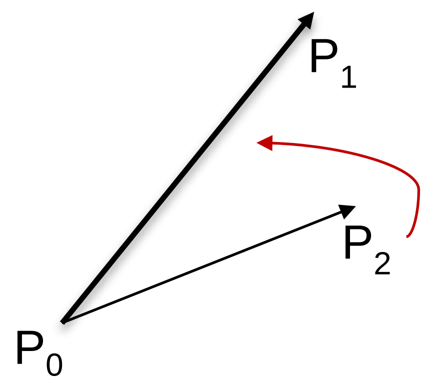
1. Segmento (o punto) está ubicado en el sentido horario o anti-horario de otro.
2. Hacia donde doblamos cuando pasamos de un segmento a otro.
3. Intersección de 2 segmentos.
   a) Visión analítica
   b) Aplicando Producto en cruz

Segmentos de línea

1. a) - Dados 2 segmentos P_0P_1 y P_0P_2; está P_0P_1 ubicado en el **sentido horario** de P_0P_2 respecto a P_0 ?



SI



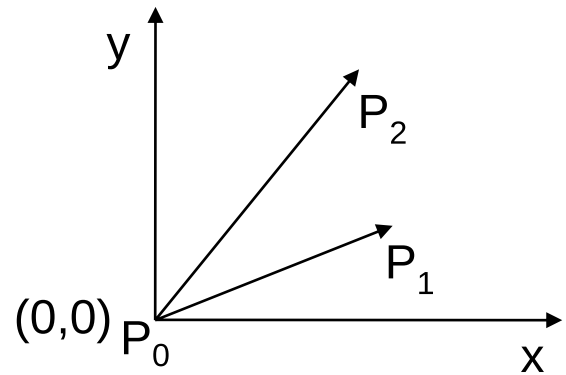
NO, está en el sentido **anti-horario**

Producto cruzado

$$P_1 \times P_2 = \det \begin{pmatrix} x1 & x2 \\ y1 & y2 \end{pmatrix}$$

$$= x1\ y2 - x2\ y1$$

$$P_2 \times P_1 = - P_1 \times P_2$$



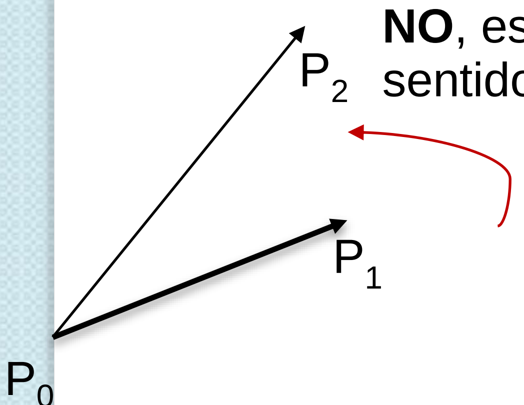
$P_1 \times P_2$ > 0 entonces P_1 está ubicado en el sentido **horario** de P_2 respecto a P_0 , en este caso el origen (0,0)

< 0 entonces P1 está ubicado en el sentido **anti-horario** de P_2

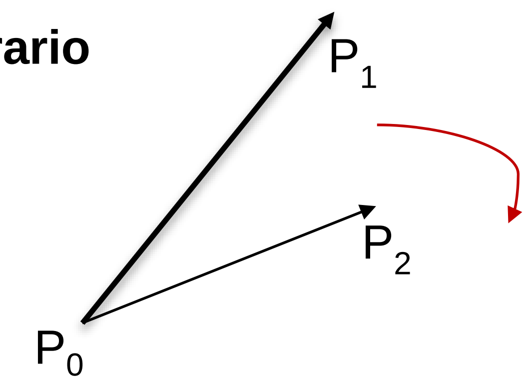
Segmentos de línea

1.b) Dados 3 puntos P_0 P_1 y P_2; está P_2 ubicado en el **sentido horario** del segmento dirigido que pasa por P_0 y P_1 ?

NO, está en el sentido **anti-horario**

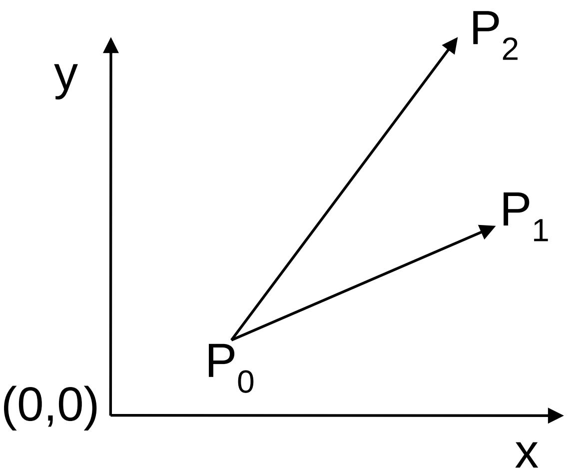


SI



Producto cruzado

Para un P_0 cualquiera, simplemente se lo traslada al origen



$(P_1 - P_0) \rightarrow P'_1 = (x'1, y'1)$

$x'1 = (x1-x0)$ y $y'1 = (y1-y0)$

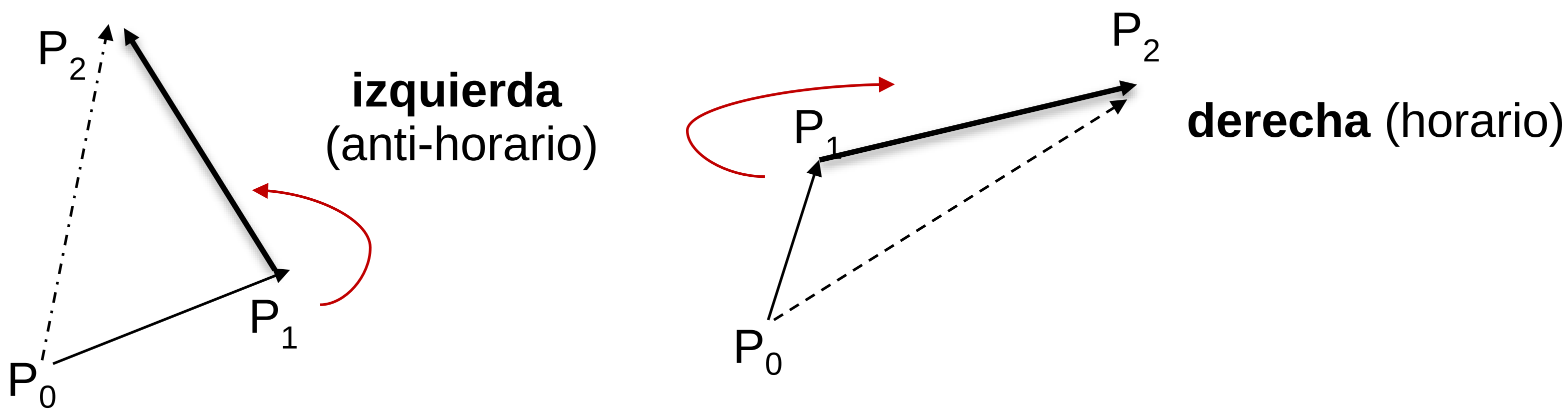
$(P_2 - P_0) \rightarrow P'_2 = (x'2, y'2)$

$$P'_1 \times P'_2 = \det \begin{pmatrix} x'1 & x'2 \\ y'1 & y'2 \end{pmatrix}$$

$$P'_1 \times P'_2 = (P_1 - P_0) \times (P_2 - P_0) = (x1-x0)(y2-y0) - (x2-x0)(y1-y0)$$

Segmentos de línea

2.- Dados 2 segmentos consecutivos P_0P_1 y P_1P_2; P_1P_2 gira a izquierda o a derecha del punto P_1 ?



Solución: Chequear si el segmento P_0P_2 está en el sentido horario o anti-horario en relación al segmento P_0P_1 usando el producto en cruz.

Producto cruzado

$P'_1 \times P'_2 = (P_1 - P_0) \times (P_2 - P_0) =$ (x1-x0)(y2-y0) – (x2-x0)(y1-y0)

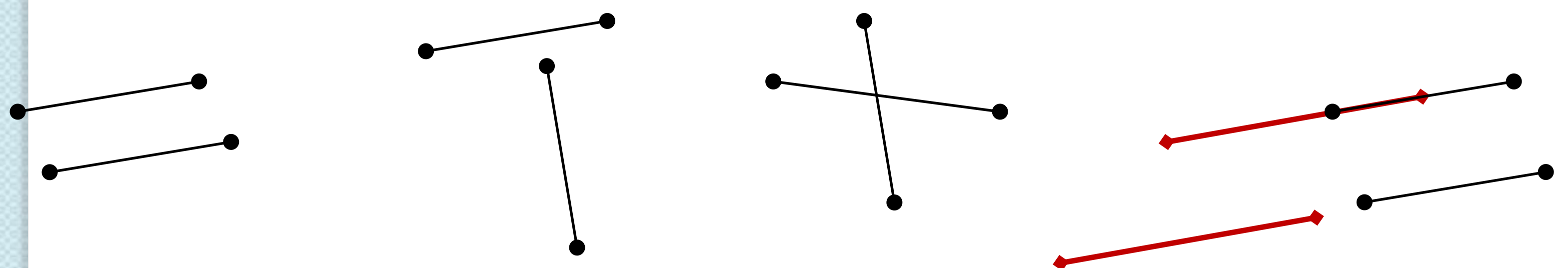
$a = P_0$, $b = P_1$ y $c = P_2$

```
double producto_en_cruz(point a, point b, point c)
{
	return( (a[X]*b[Y] - a[Y]*b[X] + a[Y]*c[X]
	- a[X]*c[Y] + b[X]*c[Y] - c[X]*b[Y]));
}
```

Segmentos de línea

Intersección de 2 segmentos (visión analítica)

```
typedef struct {
        point p1,p2;                    /* endpoints of line segment */
       } segment;
```

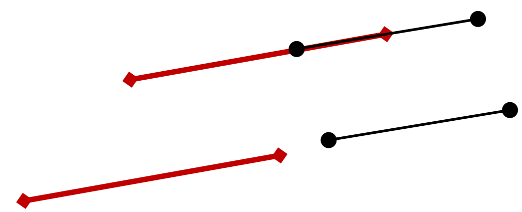


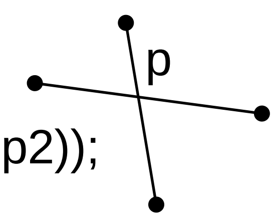
```
bool point_in_box(point p, point b1, point b2)
{
     return( (p[X] >= min(b1[X],b2[X])) && (p[X] <= max(b1[X],b2[X]))
     && (p[Y] >= min(b1[Y],b2[Y])) && (p[Y] <= max(b1[Y],b2[Y])) );
}
```

Segmentos de línea

Intersección de 2 segmentos (visión analítica)

```
bool segments_intersect(segment s1, segment s2)
{
    line l1,l2;                          /* lines containing the input segments */
    point p;                             /* intersection point */
  points_to_line(s1.p1,s1.p2,&l1);
  points_to_line(s2.p1,s2.p2,&l2);
  if (same_lineQ(l1,l2))                 /* overlapping or disjoint segments */
          return( point_in_box(s1.p1,s2.p1,s2.p2) ||
                  point_in_box(s1.p2,s2.p1,s2.p2) ||
                  point_in_box(s2.p1,s1.p1,s1.p2) ||
                  point_in_box(s2.p2,s1.p1,s1.p2) );
  if (parallelQ(l1,l2)) return(FALSE);
   intersection_point(l1,l2,p);
   return(point_in_box(p,s1.p1,s1.p2) && point_in_box(p,s2.p1,s2.p2));
}
```

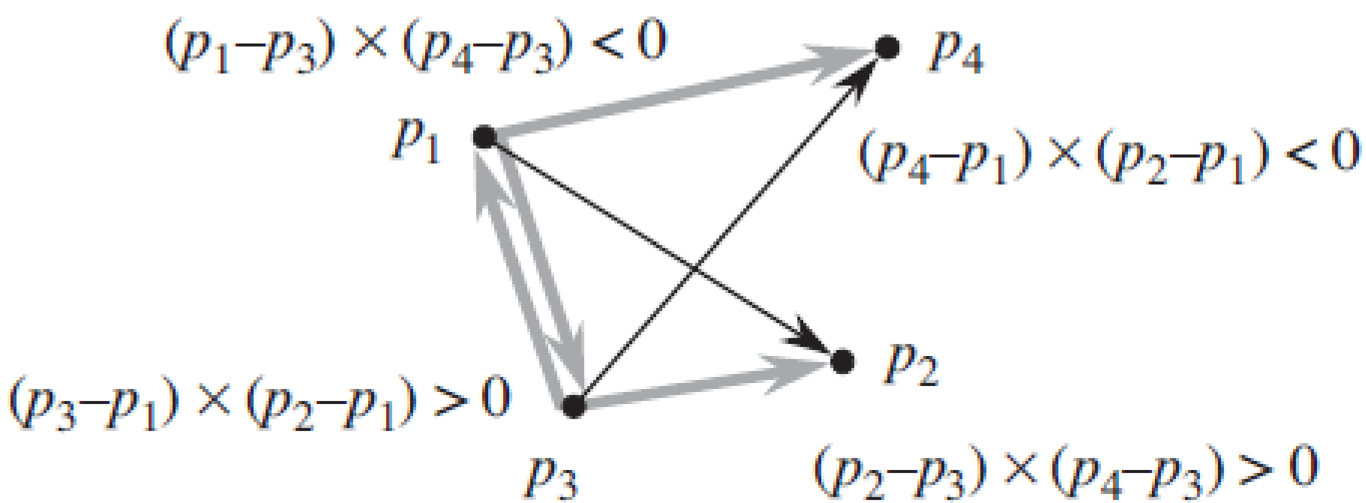




Segmentos de línea

Intersección de 2 segmentos

(visión geometría computacional)



Segmentos de línea

Intersección de 2 segmentos

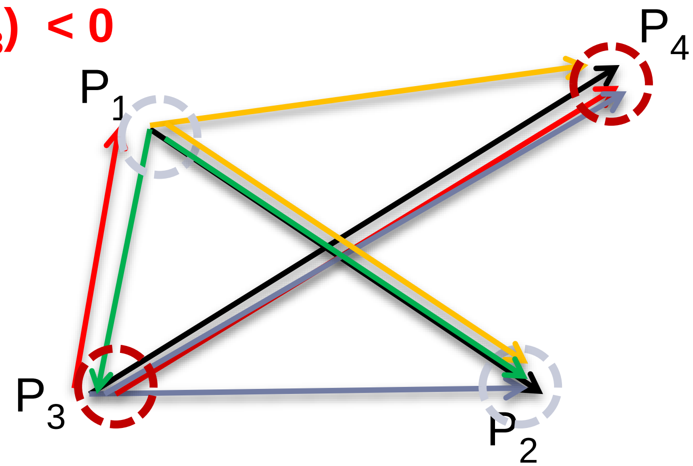
(usando Producto en cruz)

Ver si los extremos del seg P_1P_2 están a cada lado del seg P_3P_4

$(P_4 - P_1) \times (P_2 - P_1) < 0$

$(P_1 - P_3) \times (P_4 - P_3) < 0$

Ver si los extremos del seg P_3P_4 están a cada lado del segmento P_1P_2



$(P_3 - P_1) \times (P_2 - P_1) > 0$

$(P_2 - P_3) \times (P_4 - P_3) > 0$

(a)

Segmentos de línea

Intersección de 2 segmentos

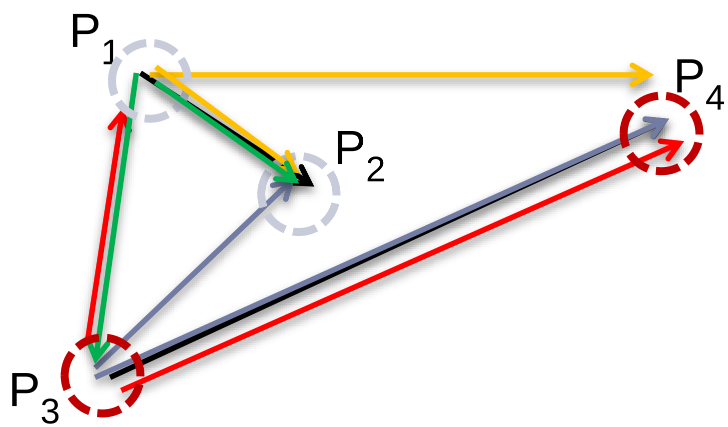
(usando Producto en cruz)

Ver si los extremos del seg P_1P_2 están a cada lado del seg P_3P_4

$(P_4 - P_1) \times (P_2 - P_1) < 0$

$(P_1 - P_3) \times (P_4 - P_3) < 0$

Ver si los extremos del seg P_3P_4 están a cada lado del segmento P_1P_2



$(P_3 - P_1) \times (P_2 - P_1) > 0$

$(P_2 - P_3) \times (P_4 - P_3) < 0$

(b)

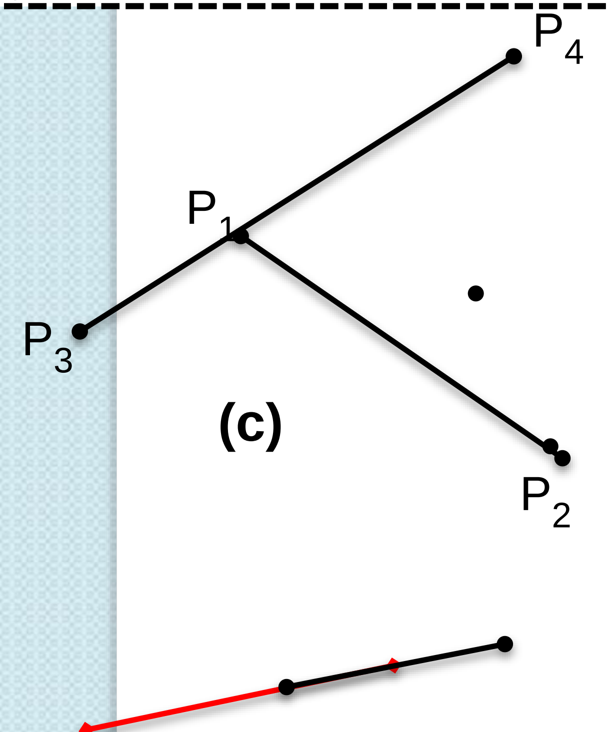
Segmentos de línea

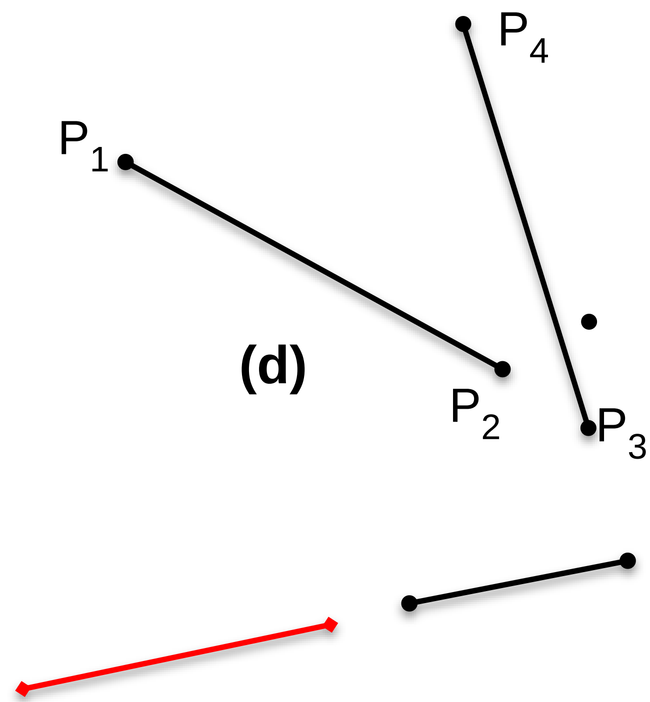
Intersección de 2 segmentos

(usando Producto en cruz)

El punto P_1 es colineal con el segmento P_3P_4 y está entre P_3 y P_4

El punto P_3 es colineal con el segmento P_1P_2 pero no está entre P_1 y P_2





Segmentos de línea

Intersección de 2 segmentos

(usando Producto en cruz)

Seudocódigo

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

Segmentos de línea

Intersección de 2 segmentos

(usando Producto en cruz)

```
DIRECTION(pi, pj, pk)
  return (pk - pi) × (pj - pi)


ON-SEGMENT(pi, pj, pk)
   if min(xi, xj) ≤ xk ≤ max(xi, xj) and min(yi, yj) ≤ yk ≤ max(yi, yj)
        then return TRUE
        else return FALSE
```

Bibliografía

Programming Challenges. The Programming Contest Training Manual. Steven S. Skiena, Miguel A. Revilla. Springer-Verlag New York, Inc., 2003 ISBN 0-387-00163-8.

Introduction to Algorithms, 2nd Ed –Thomas H. Cormen
ISBN 0-262-03293-7 (hc.: alk. paper, MIT Press).-ISBN 0-07-013151-1 (McGraw-Hill)

Problemas geométricos

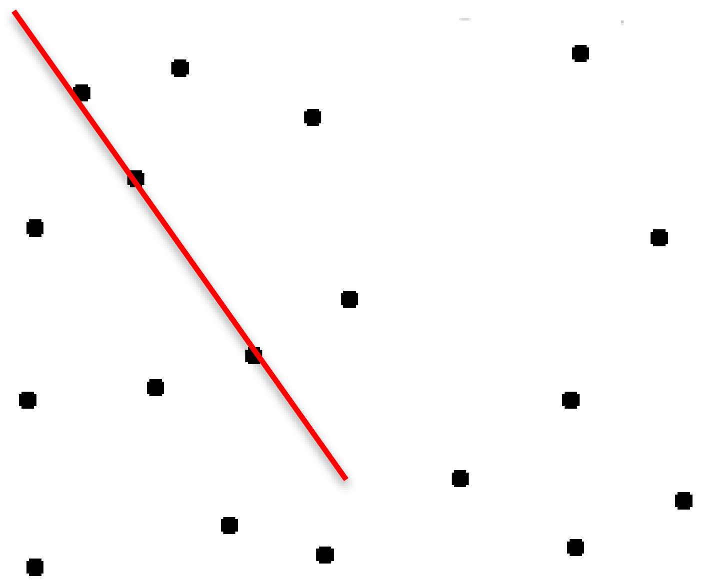
Puntos:

1. Máxima cantidad de puntos en una recta
2. Par de puntos más cercanos

Problemas geométricos: Puntos
Descripción

1. Máxima cantidad de puntos en una recta

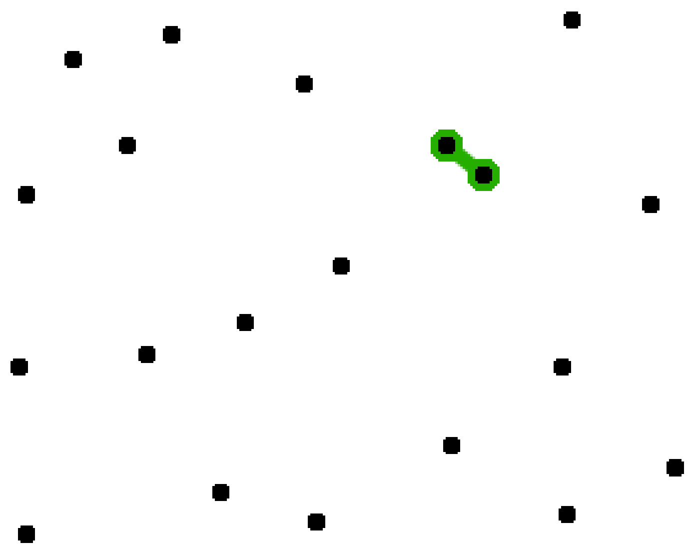
Dado un conjunto P de puntos en R^2 se debe encontrar la máxima cantidad de ellos que se encuentran alineados, es decir, que caen en una misma recta.



Problemas geométricos: Puntos
Descripción

2. Par de puntos más cercanos

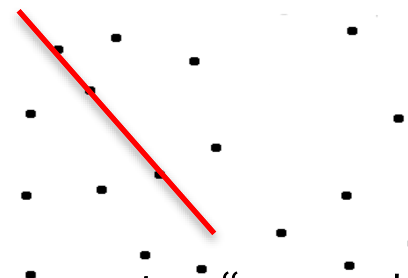
Dado un conjunto P de puntos en R^2 se debe encontrar el par de puntos más cercanos.



Resolución

Máxima cantidad de puntos en una recta

Dado un conjunto P de puntos en R^2 se debe encontrar la máxima cantidad de ellos que se encuentran alineados, es decir, que caen en una misma recta (son colineales).



Estrategia de solución ??

Qué sabemos ??

a.- Todos los puntos p_i que caen en una misma recta "$y = m*x + n$" deben satisfacer dicha ecuación.

b.- Si definimos la ec. de la recta que pasa por 2 puntos, p_i y p_j → un 3er punto p_k estará alineado con p_i y p_j si cumple que

$m_{i,j} = m_{i,k}$ siendo $m_{a,b}$ la pendiente entre p_a y p_b

Resolución

Máxima cantidad de puntos en una recta

b.- Definimos la ec. de la recta que pasa por 2 puntos, p_i y p_j →

Recta que pasa por los punto $P_1 (x_1,y_1)$ y $P_2 (x_2,y_2)$ se tiene que la

pendiente $m = (y_2 - y_1) / (x_2 - x_1)$ y ordenada al origen $n = y_1 - m\, x_1$

Como tienen que satisfacer "$y = m*x + n$" entonces

$$y = (y_2 - y_1) / (x_2 - x_1)*x + y_1 - [(y_2 - y_1) / (x_2 - x_1)]\, x_1$$

$$y - y_1 = (y_2 - y_1) / (x_2 - x_1) * (x - x_1) \qquad m_{1,2}$$

Dado otro punto p_k la ec recta que pasa por P_1 y P_k es:

$$y - y_1 = (y_k - y_1) / (x_k - x_1) * (x - x_1) \qquad m_{1,k}$$

Entonces el 3er punto p_k estará alineado con p_1 y p_2 si cumple :

$m_{1,2} = m_{1,k}$ siendo $m_{a,b}$ la pendiente entre p_a y p_b

Resolución

Máxima cantidad de puntos en una recta

Solución 1: Fuerza Bruta --- > O(n^3)

a) Dado un punto $p_i \in P$

i.- calcular la pendiente entre el p_i y los restantes n-1 puntos.

ii.- calcular la cantidad máxima de pendientes iguales.

b) Repetir a) para los restantes $p_i \in P$ e ir calculando el máximo entre las cantidades obtenidas en a) ii.-

Resolución

Máxima cantidad de puntos en una recta

Solución 1: $O(n^3)$

Código guía

```
01: Const
02:       cero= 1e-8;
03:       infinito= 1e+1000;
04:
05: Type
06:      point = record
07:             x,y: longint;
08:             end;
09:
10: Var
11:    i,n: longint;
12:    p: array [1..200] of point;
13:
14: Function PuntLinea(n: longint): longint;
15:          var i,j,k,r,t: longint;
16:              m: array [1..200] of extended;
17:          begin
18:          r:= 0;
19:          for i:= 1 to n-1 do
20:              begin
21:              if (r+2 >n-i) then break;
22:              for j:= i+1 to n do
23:                  if (p[i].x<> p[j].x)
24:                     then m[j]:= (p[i].y -p[j].y)/(p[i].x -p[j].x)
25:                     else m[j]:= infinito;
26:              for j:= i+1 to n-1 do
27:                  begin
28:                  t:= 0;
29:                  for k:= j+1 to n do
30:                      if (abs(m[j]-m[k])< cero) then Inc(t);
31:                  if (t > r) then r:= t;
32:                  end;
33:              end;
34:          PuntLinea:= r+2;
35:          end;
36:
37: Begin
38:       readln(input,n);
39:       for i:= 1 to n do
40:           readln(input,p[i].x, p[i].y);
41:       writeln(output,PuntLinea(n));
42: End.
```

ordenar el vector de pendientes antes de calcular la cantidad máxima de pendientes iguales.
Cambiar líneas 26-35

Resolución

Máxima cantidad de puntos en una recta

Solución 2: mejorada --- > O(n^2 log n)

Se modifica la solución 1 ordenando el vector de pendientes antes del paso ii.-

Código guía

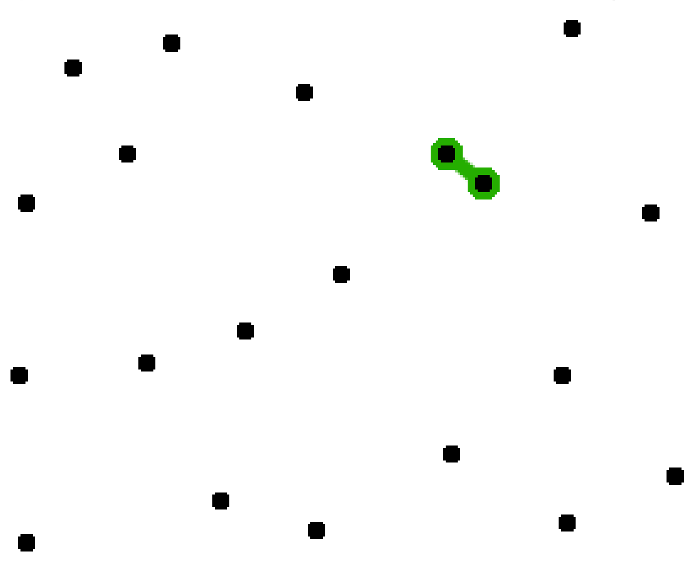
```
            quicksort(m,i,n-1);
            t:= 0;
            for j:= i+1 to n-1 do
                if (abs(m[j]-m[j+1])< cero)
                   then Inc(t)
                   else begin
                        if (t>r) then r:= t;
                        t:= 0;
                        end;
            if (t>r) then r:= t;
            end;
        PuntLinea:= r+2;
        end;
```

Resolución

Par de puntos más cercanos

2. Par de puntos más cercanos

Dado un conjunto P de puntos en R^2 se debe encontrar el par de puntos más cercanos.



Resolución

Par de puntos más cercanos

Solución 1: Fuerza Bruta --- > $O(n^2)$

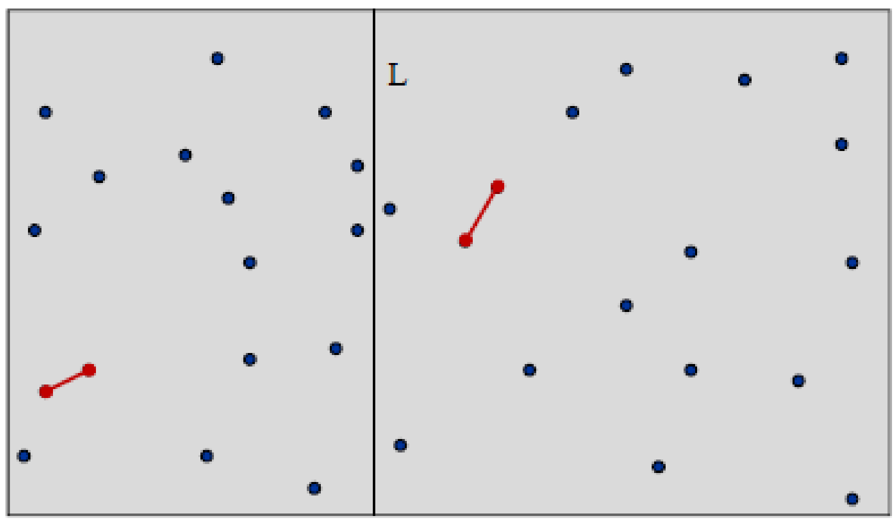
Calcula distancia entre cada punto $(p_i, p_j) \varepsilon P$ quedándose con la mínima

Solución 2: Divide & Conquer --- > $O(n \log n)$

Resolución

Par de puntos más cercanos

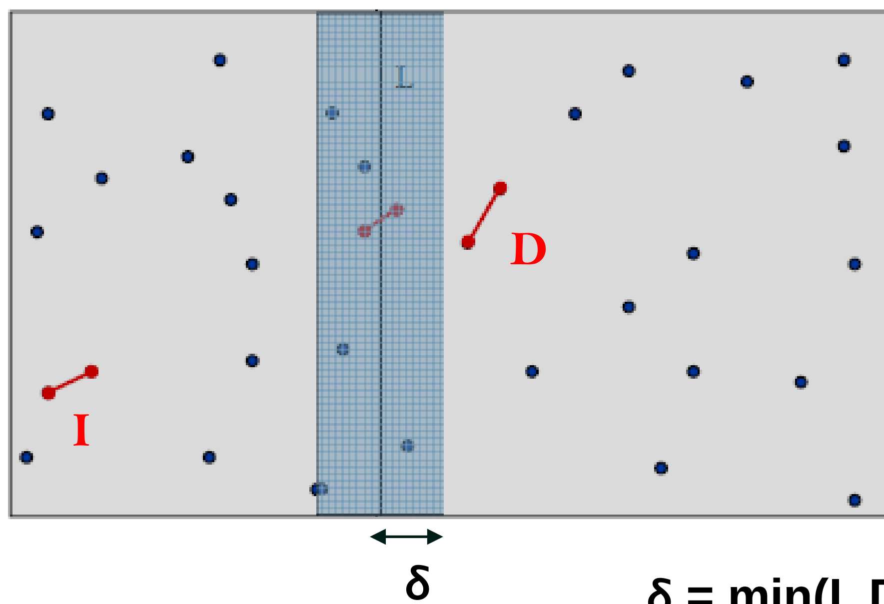
Divide : divide el conjunto de puntos P en 2 subconjuntos P_{izq} y P_{der} y se busca el ppmc en cada subconjunto, d_i y d_d



Resolución

Par de puntos más cercanos

Conquer : encontrar la pareja más cercana a cada lado de la línea



δ = min(I, D)

Ordenar los elementos de la franja 2δ en función de su coordenda y

Resolución

Par de puntos más cercanos

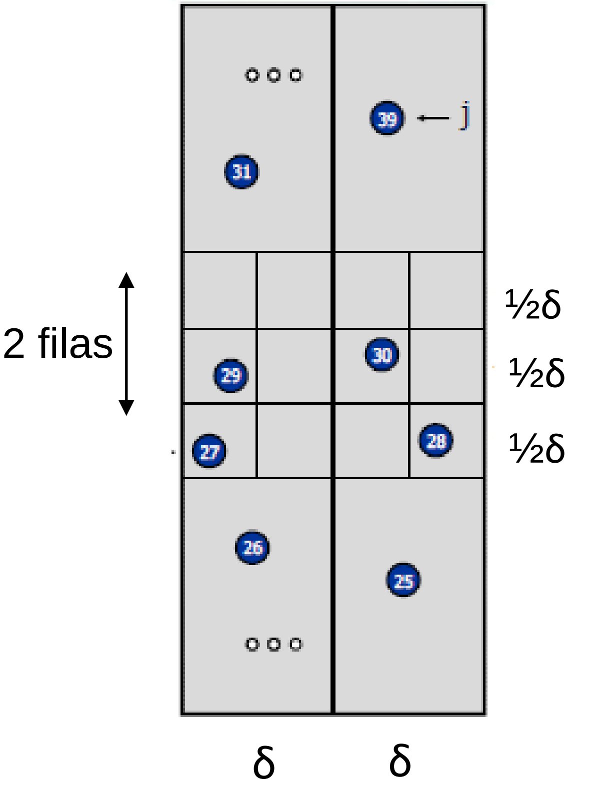
Conquer : encontrar la pareja más cercana a cada lado de la línea

Propiedad
Si s y s' son puntos de la franja 2δ tales que d(s,s')< δ, para encontrarlos no tendremos que calcular más de 11 distancias por punto si los ordenamos por su coordenada y.

Demostración:

- No hay dos puntos en la misma región de tamaño ½δ x ½δ.

- Dos puntos separados por dos filas están a una distancia ≥ 2(½δ).



Resolución

Par de puntos más cercanos

```
Closest-Pair (p1, …, pn)
{
    Compute separation line L such that half the
    points are on one side and half on the other
    side.
    δ1 = Closest-Pair(left half)
    δ2 = Closest-Pair(right half)
    δ = min(δ1, δ2)

    Delete all points further than δ from separation
    line L

    Sort remaining points by y-coordinate.

    Scan points in y-order and compare distance between
    each point and next 11 neighbors. If any of these
    distances is less than δ, update δ.

    return δ.
}
```