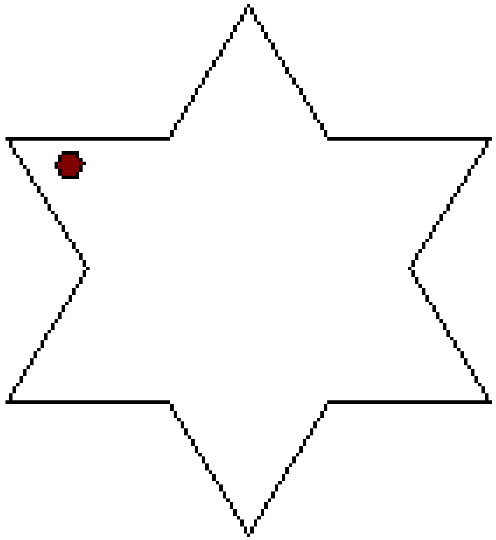
Geometría para ICPC

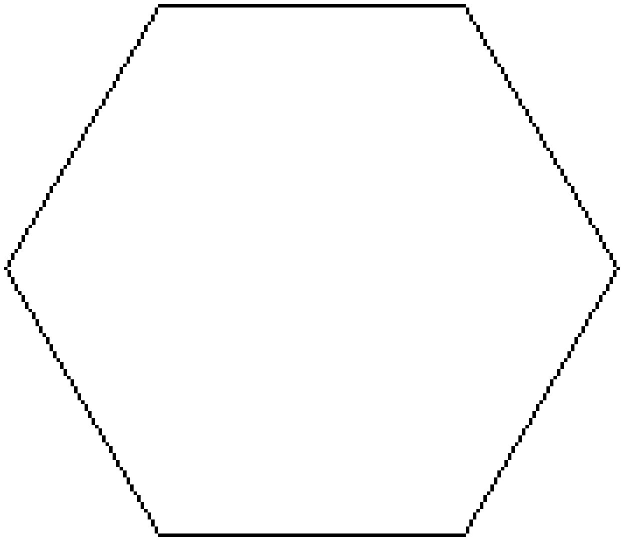
- Convex hull (cápsula convexa)
  - Polígonos convexos
  - Producto cruz
  - Algoritmo de Graham
- Superficie polígono
  - Pick
- Par de puntos más cercano
- Técnicas de sweep line (barrido)
  - Máximo rectángulo sin puntos adentro
  - Intersección de segmentos
  - Par de puntos mas cercano
  - Unión de intervalos y rectángulos

Convex hull

- “The art gallery” http://acm.uva.es/problemset/v100/10078.html
- Problema: decidir si hay un punto en un polígono desde el cual no todo el polígono sea visible.



Gallery 1

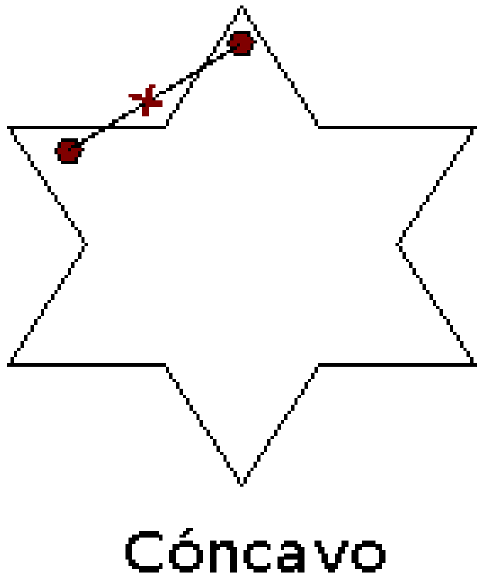


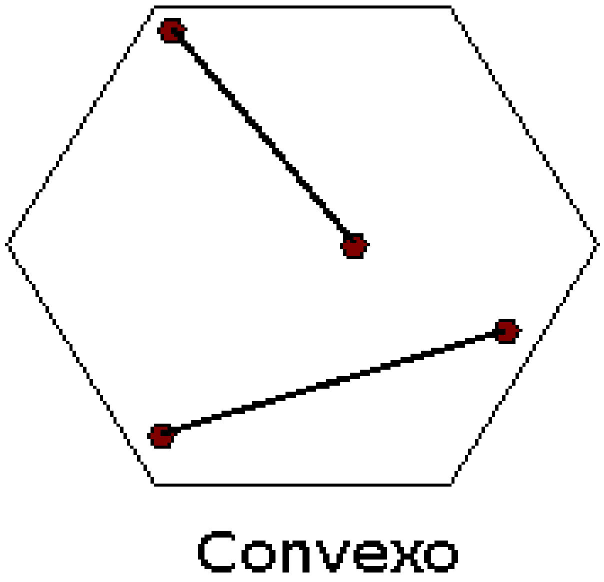
Gallery 2

Convex hull

- “The art gallery” http://acm.uva.es/problemset/v100/10078.html
- Es lo mismo que decidir si un polígono es cóncavo o convexo
- Un polígono P es convexo sii

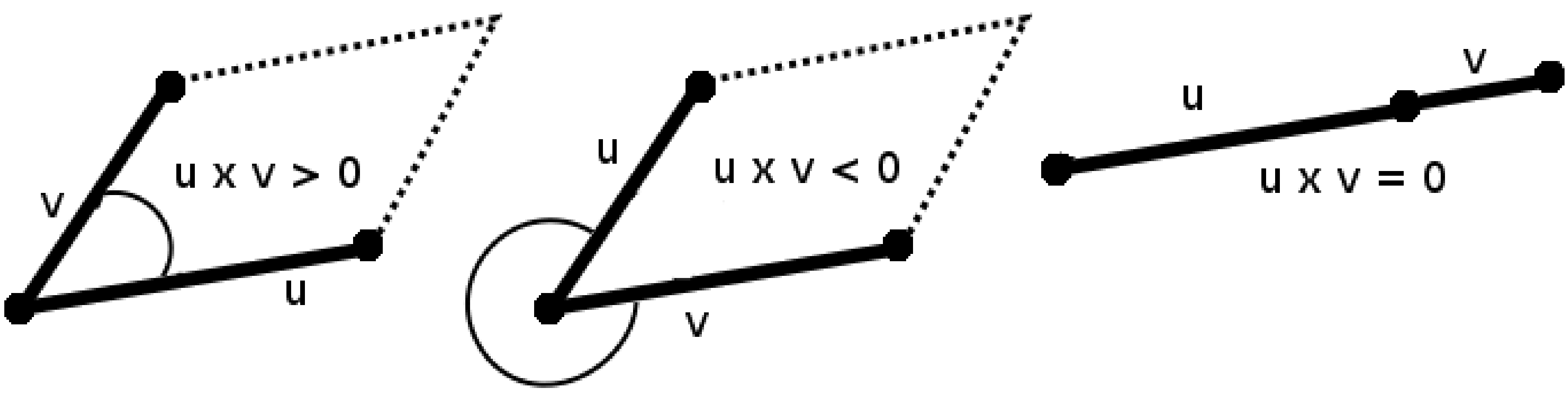
$$\text{Para todo } \forall\, A \in P,\, B \in P \Rightarrow \overline{AB} \subseteq P$$





Convex hull

- **Producto cruz** "$u_x v$"
- $u=(x1, y1)$ $v=(x2, y2)$, $u_x v = x1*y2-x2*y1 = |u||v|sen(\Phi)$
- El valor absoluto es el area del paralelogramo
- El signo es la orientación de los vectores: "Φ"



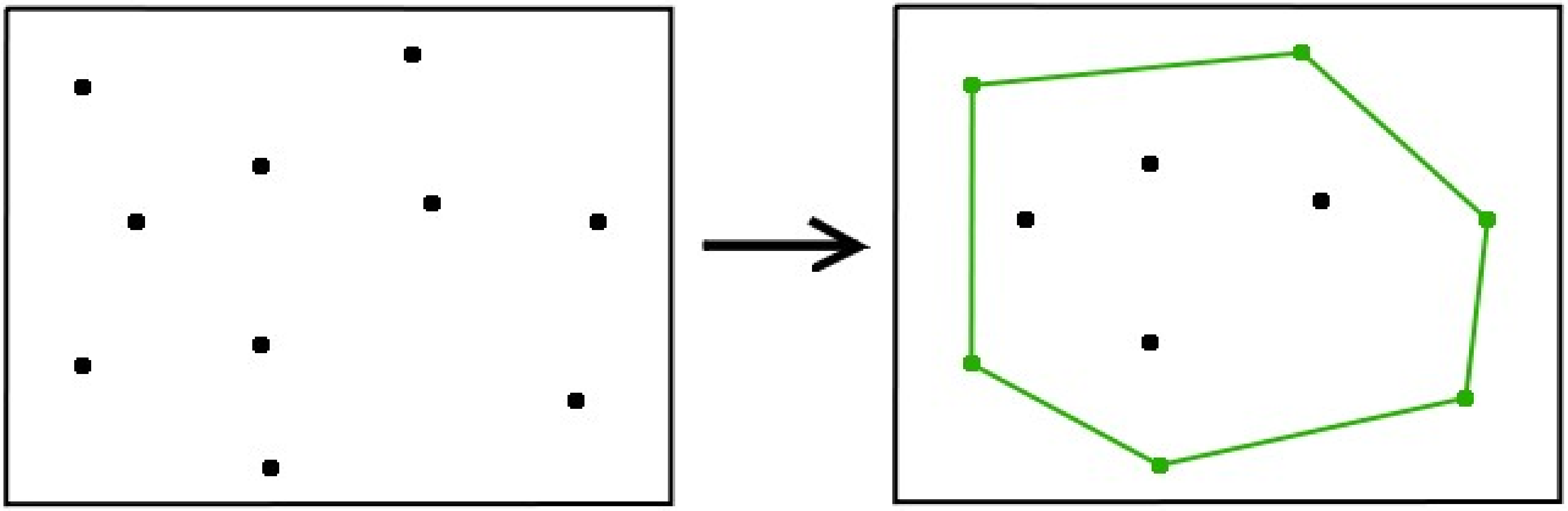
Convex hull

- “The art gallery” http://acm.uva.es/problemset/v100/10078.html
- Algoritmo **O(n)**

```
boolean isConvex(int n, int[] x, int[] y){
  int pos = 0, neg = 0;
  for(int i = 0; i < n; i++){
    int j = (i + n – 1) % n, k = (i + 1) % n;
    int pc = (x[k]-x[i])*(y[j]-y[i])-(x[j]-x[i])*(y[k]-y[i]);
    if(pc < 0){
      neg++;
    }else{
      pos++;
    }
  }
  return (neg == 0) || (pos == 0);
}
```

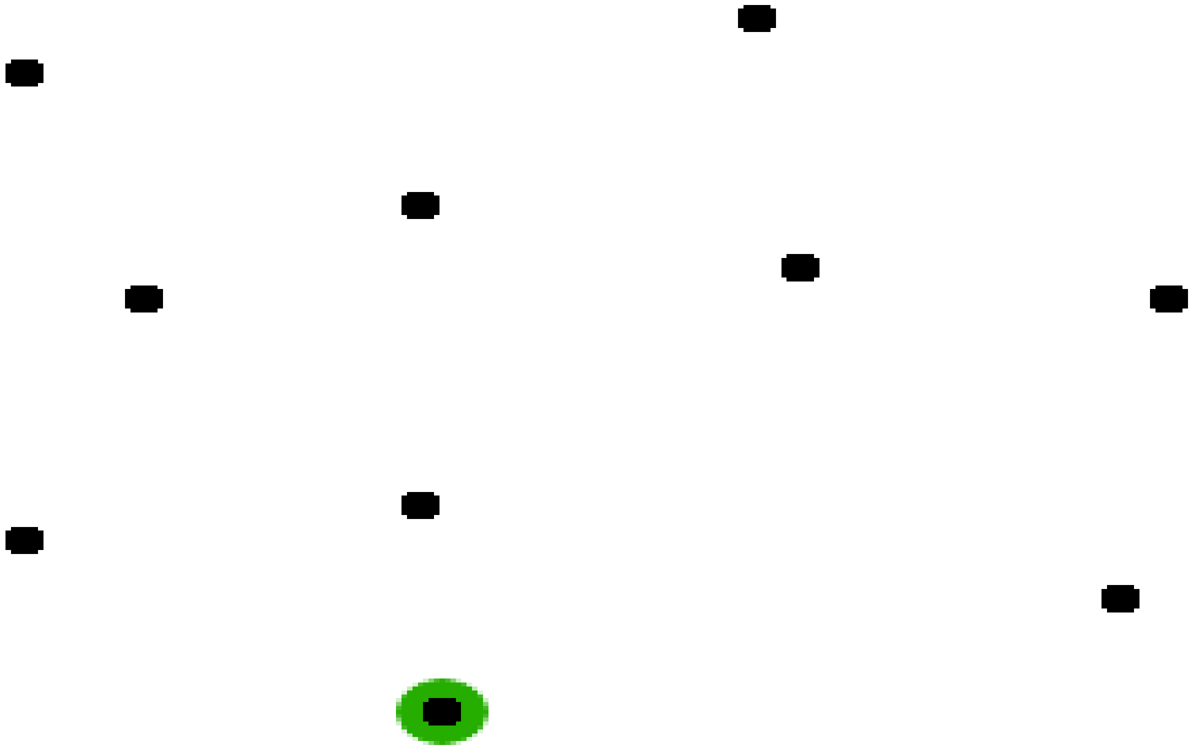
Convex hull

- El problema de la cápsula convexa
- Entrada: Un conjunto de puntos en el plano
- Salida: El póligono convexo mas pequeño que los contiene a todos



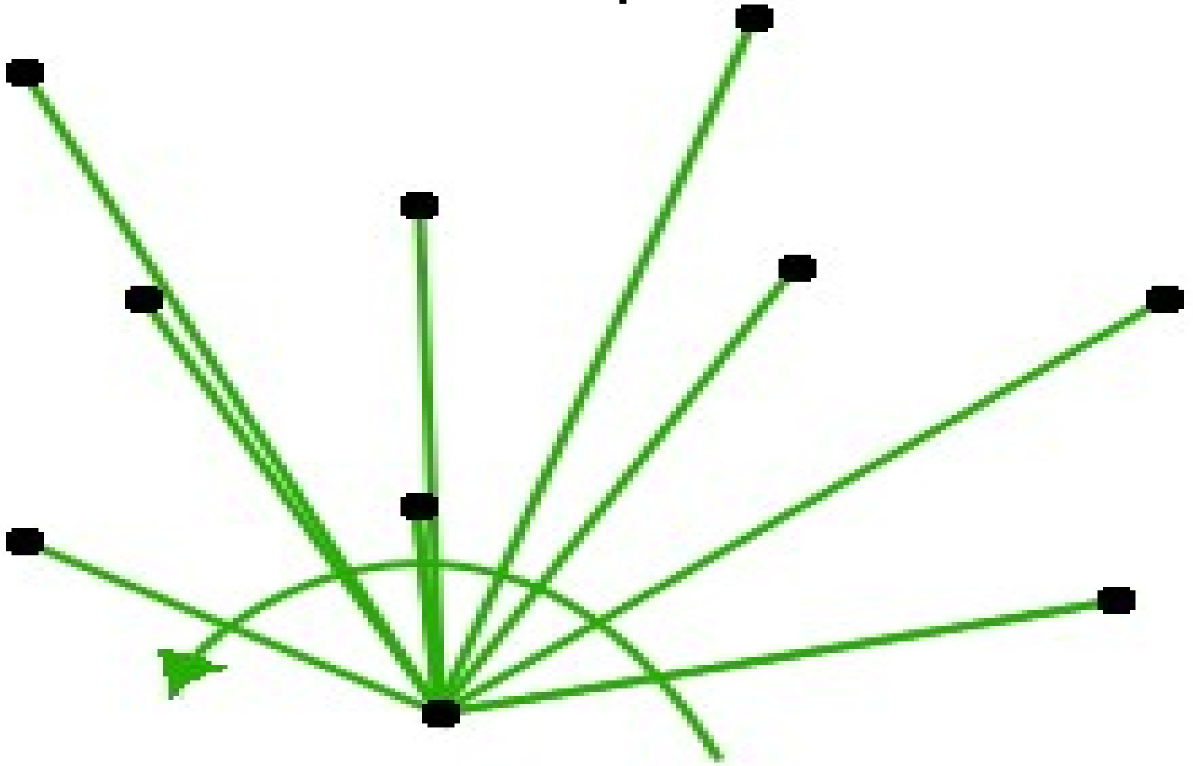
Convex hull

- Algoritmo de Graham http://en.wikipedia.org/wiki/Graham_scan
- El punto más abajo (y más a la izquierda) seguro está en la convex hull
- **O(n)**



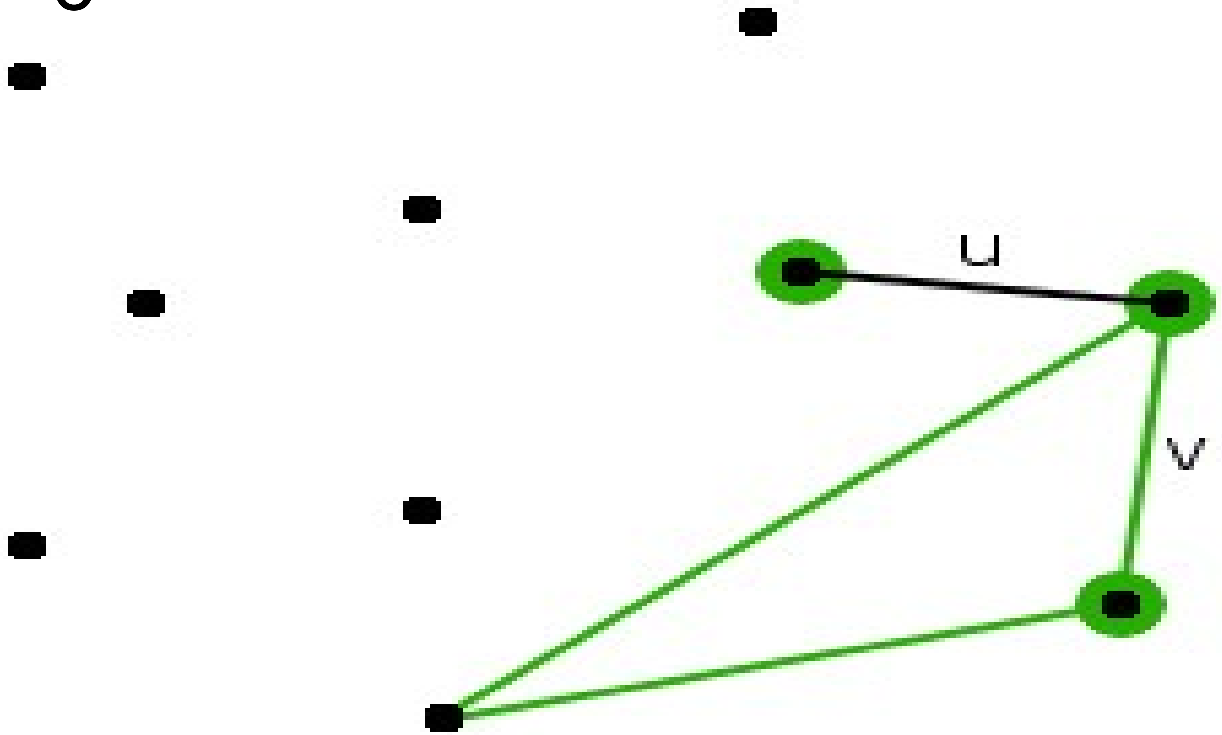
Convex hull

- Algoritmo de Graham http://en.wikipedia.org/wiki/Graham_scan
- Ordernar en sentido anti-horario (y por cercanía) respecto de este punto **O(n.log(n))**
- Invariante en el paso **i**: Convex hull “parcial” de los primeros **i** puntos.



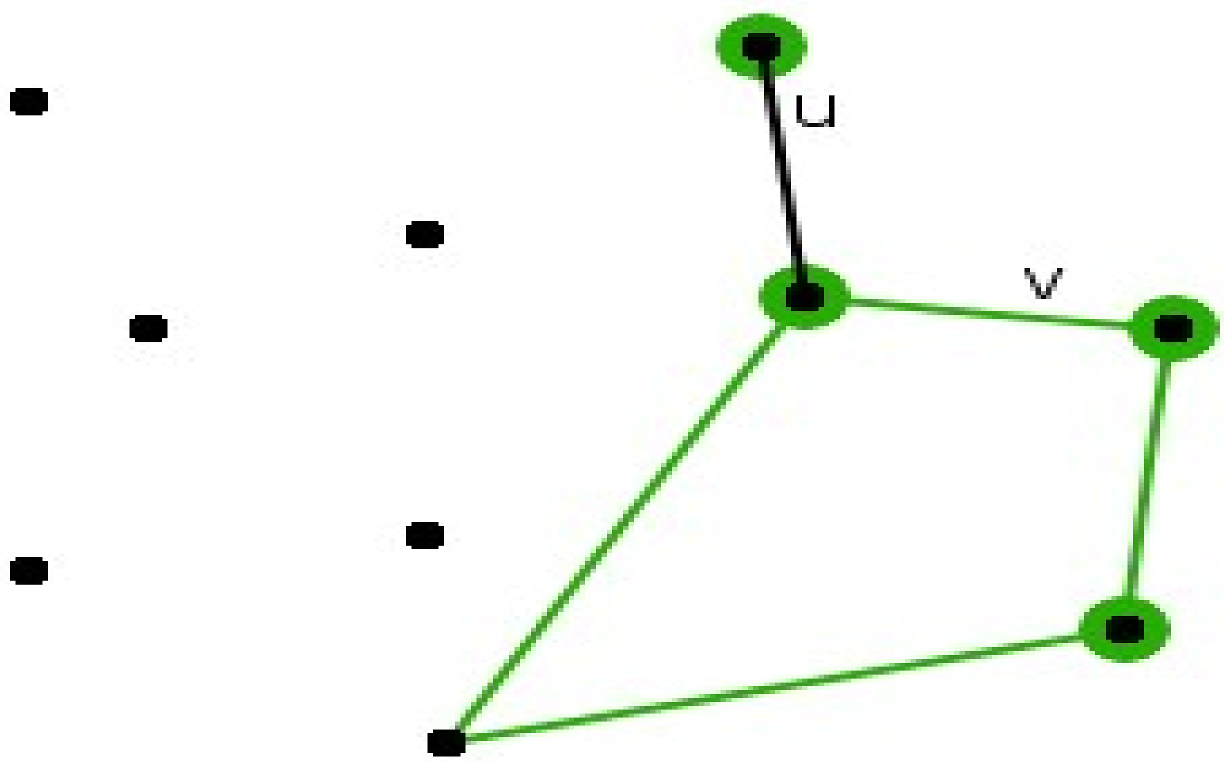
Convex hull

- Algoritmo de Graham http://en.wikipedia.org/wiki/Graham_scan
- El “próximo” siempre está en la convex hull “parcial”
- En este caso $u \times v > 0$



Convex hull

- Algoritmo de Graham http://en.wikipedia.org/wiki/Graham_scan
- Mientras $u \times v \leq 0$, se saca el anterior
- **O(n)**



Convex hull

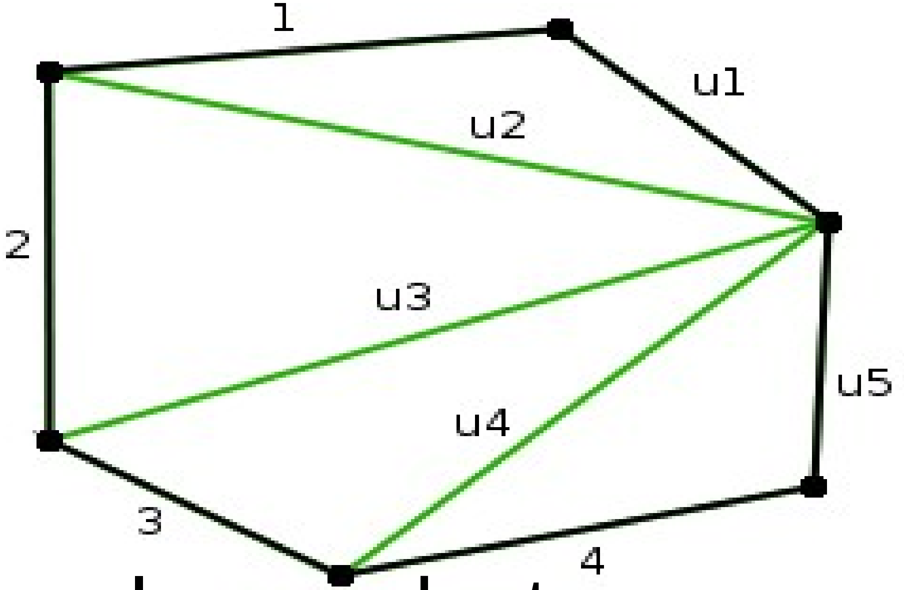
- Para practicar
  - **Onion layers** (sam06, live archive: 3655)

    http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=3655
  - **Not too Convex hull** (sam02, live archive: 2615)

    http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2615

Superficie de polígono

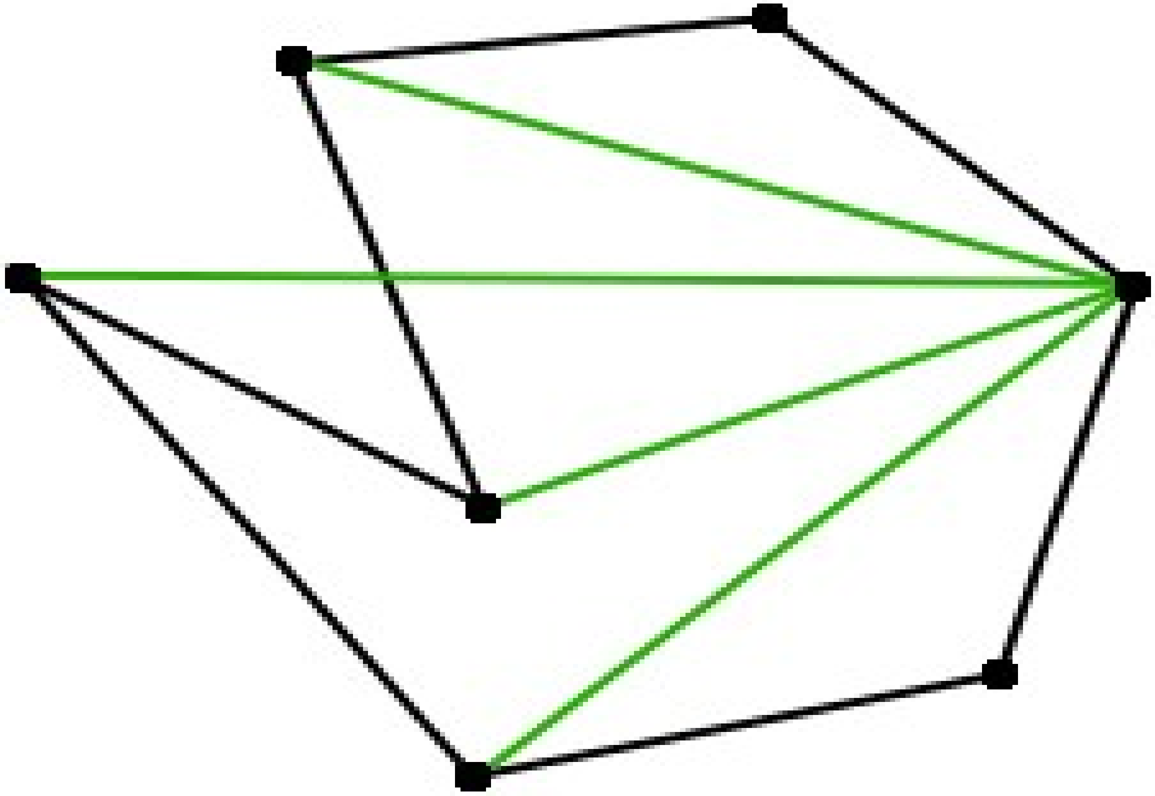
- Dado un polígono, calcular su superficie
- Si es convexo
- Se triangula desde un vértice cualquiera



- En orden, se suman los productos cruz (u1xu2) + (u2xu3) + (u3xu4) + (u4xu5)
- Se saca valor absoluto y se divide por 2

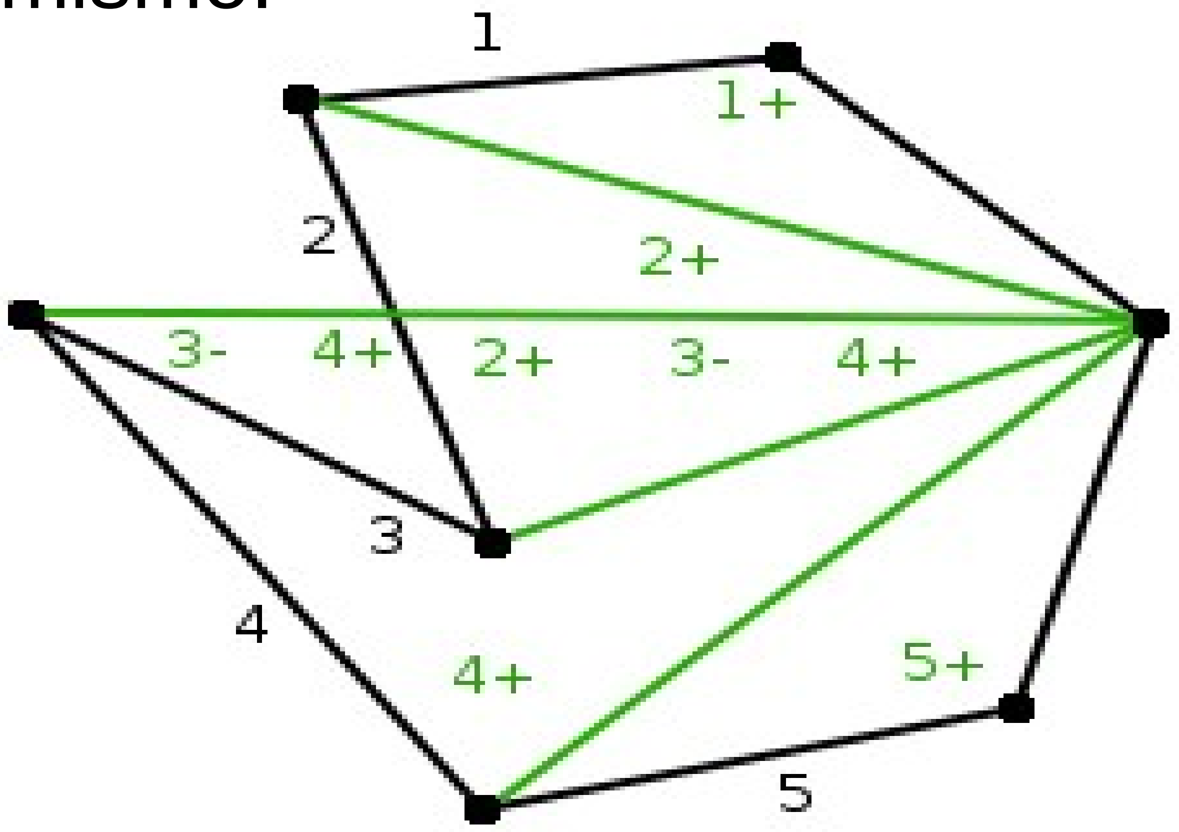
Superficie de polígono

- Dado un polígono, calcular su superficie
- Si es cóncavo?



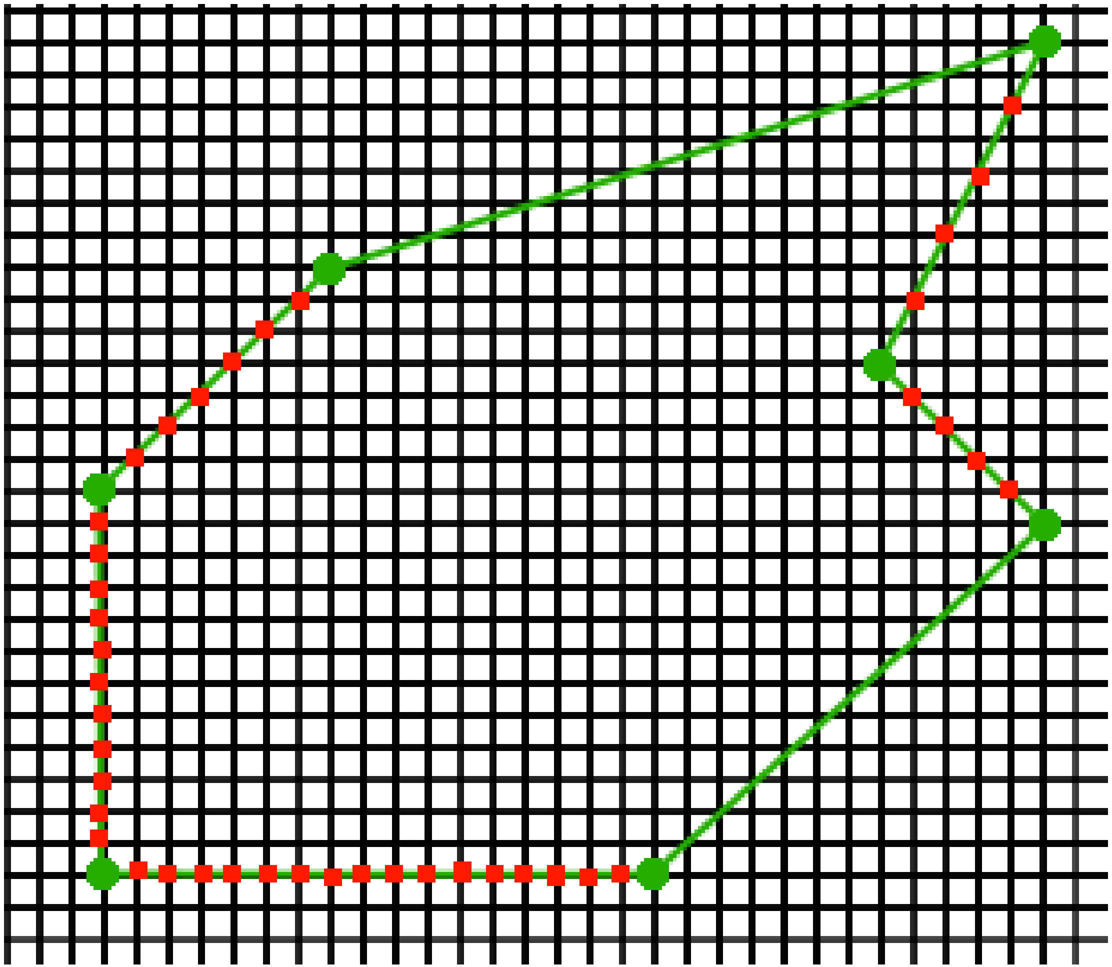
Superficie de polígono

- Dado un polígono, calcular su superficie
- Es lo mismo!



Superficie de polígono

- Teorema de Pick
- Para coordenadas enteras
- $A = I + B / 2 - 1$
- $I = A - B / 2 + 1$



Superficie de polígono

- Calcular el borde usando m.c.d.
- Algoritmo **O(n)**

```
int border(int n, int[] x, int[] y){
  int b = 0;
  for(int i = 0; i < n; i++){
    int j = (i + 1) % n;
    b += gcd(x[i]-x[j],y[i]-y[j]);
  }
  return b;
}
```

- Se puede calcular la cantidad de puntos interiores en **O(n)**

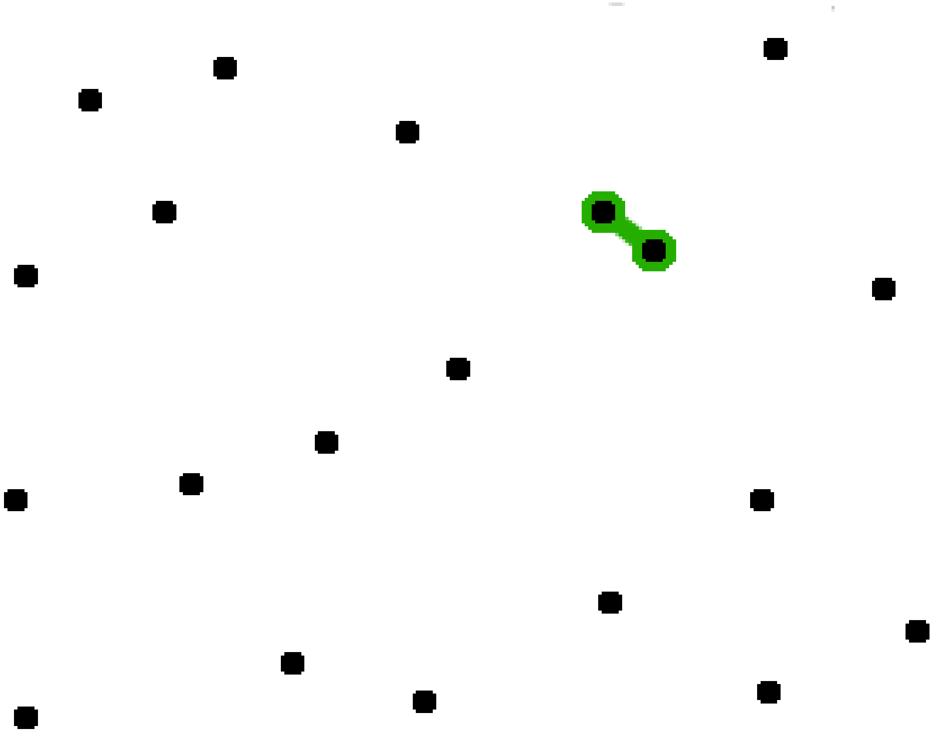
Superficie de polígono

- Para practicar
  - **Jacquard circuits** (wf07, live archive: 2395)

    http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2395

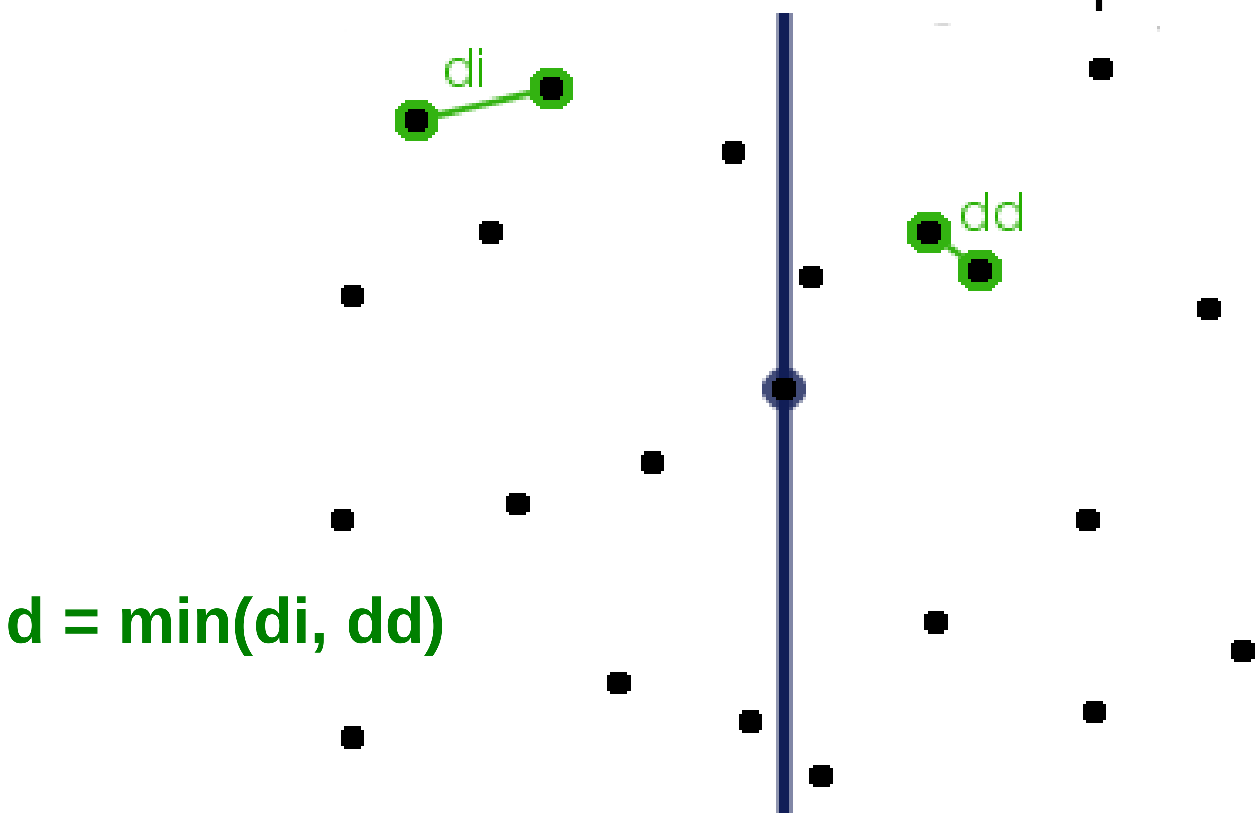
Par de puntos más cercano

- Entrada: Conjunto de puntos
- Salida: Par de puntos mas cercano



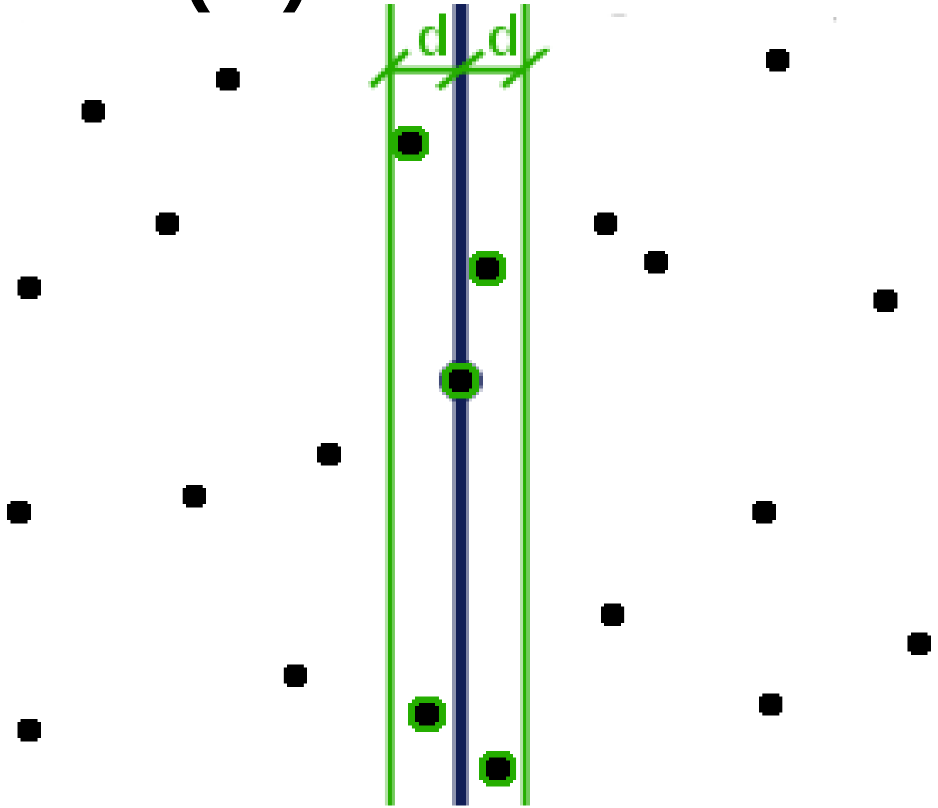
Par de puntos más cercano

- Idea: Divide & Conquer **O(n.log(n))**
- Se necesitan ordenados vertical y horizontalmente
- <u>Divide</u>: a la mitad ordenados por “x”. En **O(n')** también se calcula el orden vertical de cada parte



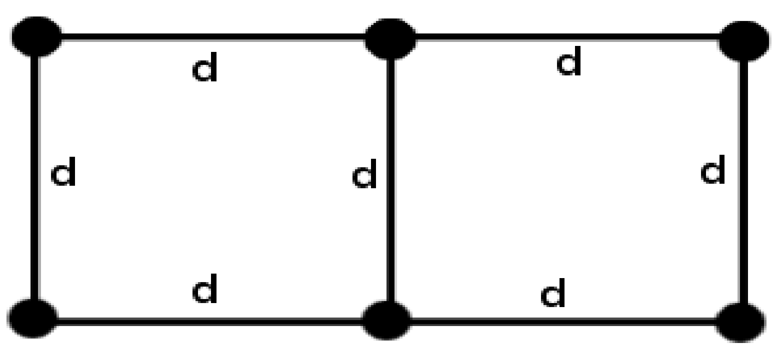
Par de puntos más cercano

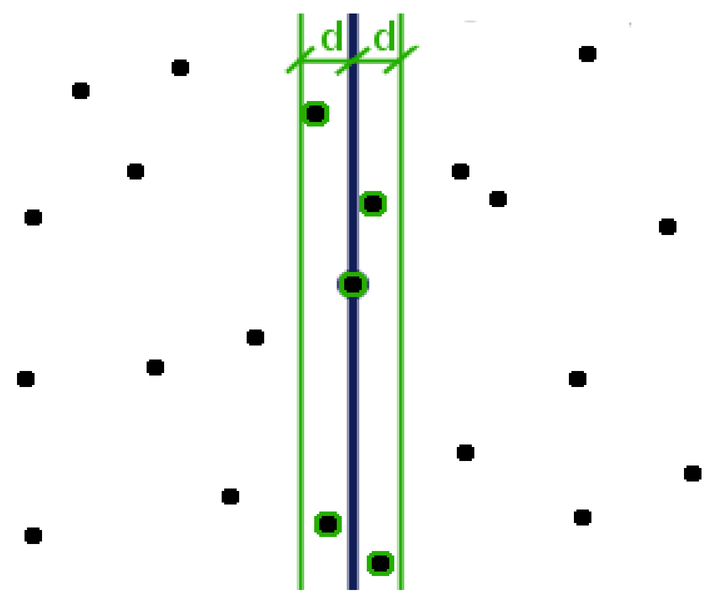
- <u>Conquer</u>: un punto de cada lado, están incluídos en la franja con distancia horizontal menor o igual a “d” del “corte”
- Se calcula en **O(n')** ordenados verticalmente



Par de puntos más cercano

- Se chequea cada punto con los que están a abajo a distancia vertical <= “d”
- Como mucho puede haber 6 puntos a distancia mayor o igual a “d” en un rectángulo de 2d x d
- Por cada punto son **O(1)** comparaciones: cada “merge” es **O(n')**





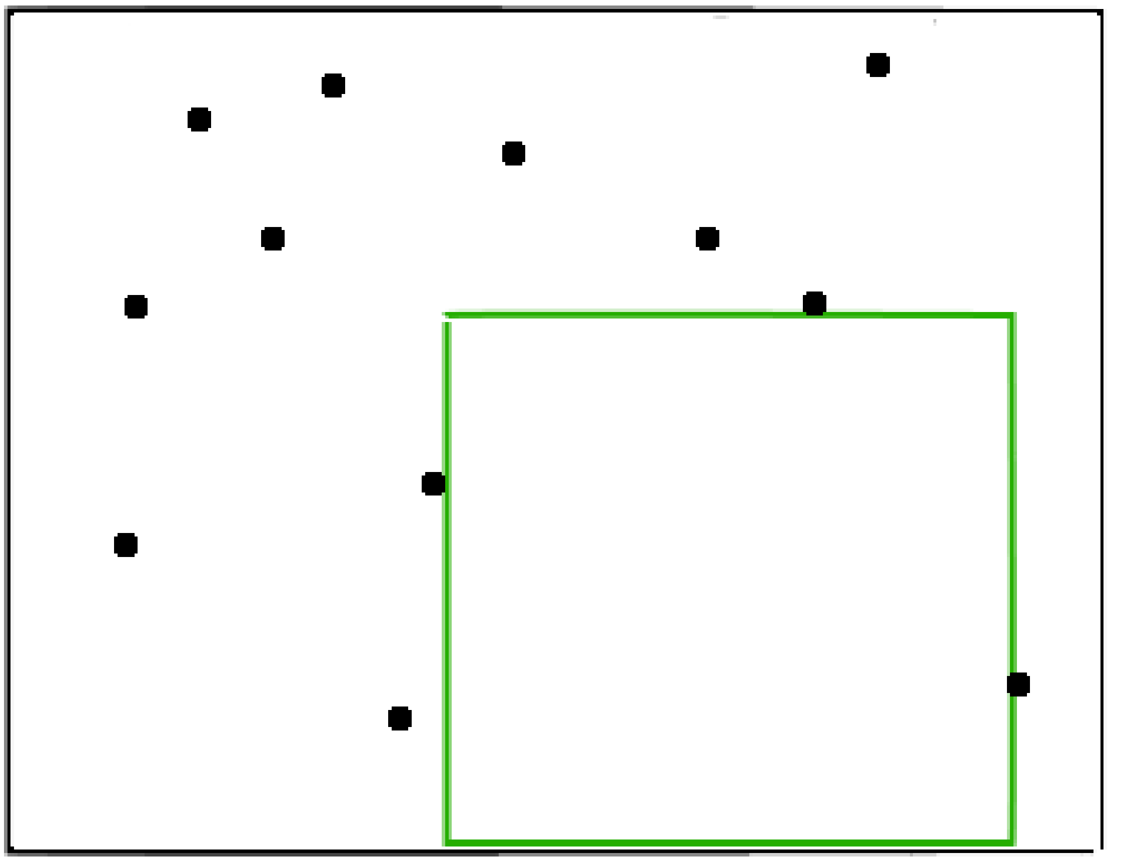
Par de puntos más cercano

```
typedef vector<point> VP;
double closest(VP points){
  VP vx = sortInXY(points), vy = sortInYX(points);
  for(int i = 1; i < vx.size(); i++) if(vx[i-1] == vx[i]) return 0.0;
  return closest_recursive(vx, vy);
}
```

```
double closest_recursive(VP vx, VP vy){
  if(vx.size()==1) return 1e20; //infinity
  if(vx.size()==2) return dist(vx[0], vx[1]);
  point cut = vx[vx.size()/2];
  VP vxL = filter(vx : x < cut.x || x == cut.x && y <= cut.y);
  VP vyL = filter(vy : x < cut.x || x == cut.x && y <= cut.y);
  double dL = closest_recursive(vxL, vyL);
  VP vxR = filter(vx : !(x < cut.x || x == cut.x && y <= cut.y));
  VP vyR = filter(vy : !(x < cut.x || x == cut.x && y <= cut.y));
  double dR = closest_recursive(vxR, vyR);
  double d = min(dL, dR);
  VP b = filter(vy : abs(x - cut.x) <= d);
  for(int i = 0; i < b.size(); i++)
    for(int j = i + 1; j < b.size() && (b[j].y - b[i].y) <= d; j++)
      d = min(d, dist(b[i], b[j])
  return d;
}
```

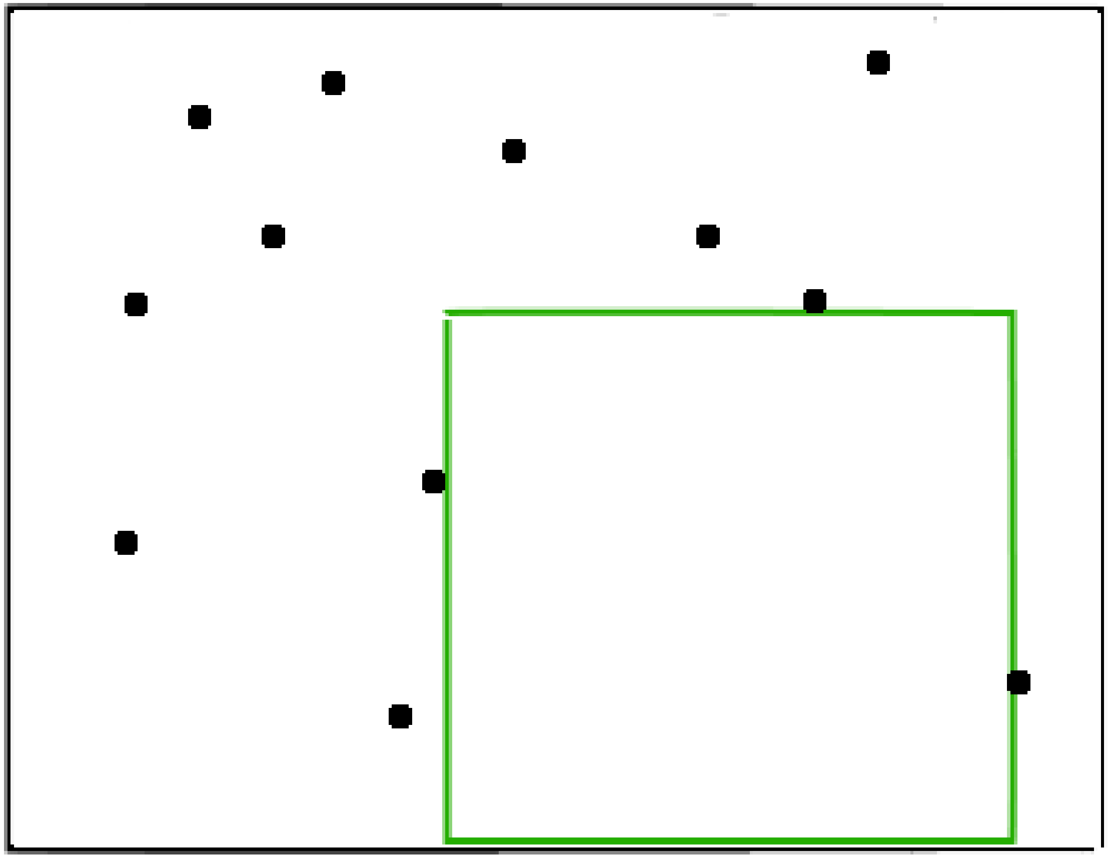
Barrido > Mayor rectángulo sin puntos adentro

- Entrada: Ancho, alto, conjunto de puntos
- Salida: Rectángulo con lados paralelos a los bordes, sin puntos adentro, con la mayor area posible



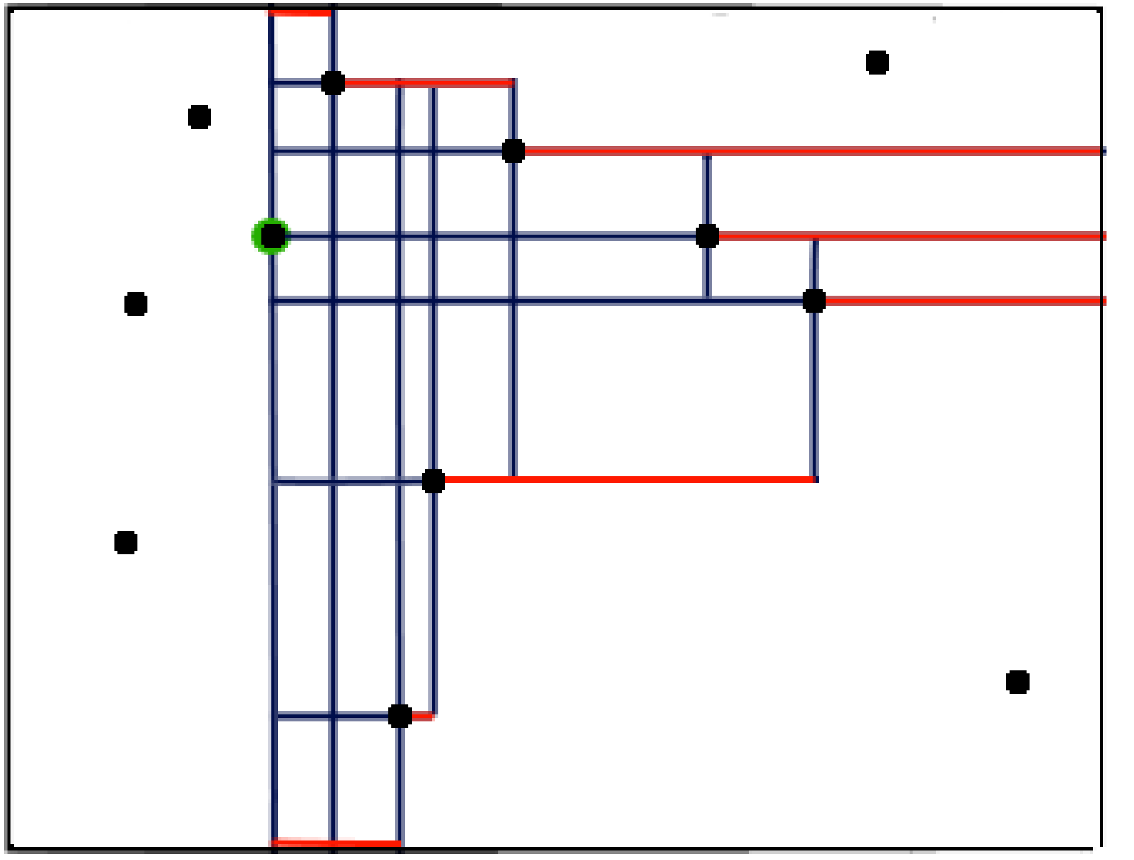
Barrido > Mayor rectángulo sin puntos adentro

- Cada lado del rectángulo tiene un punto o está sobre el borde
- En particular el lado izquierdo



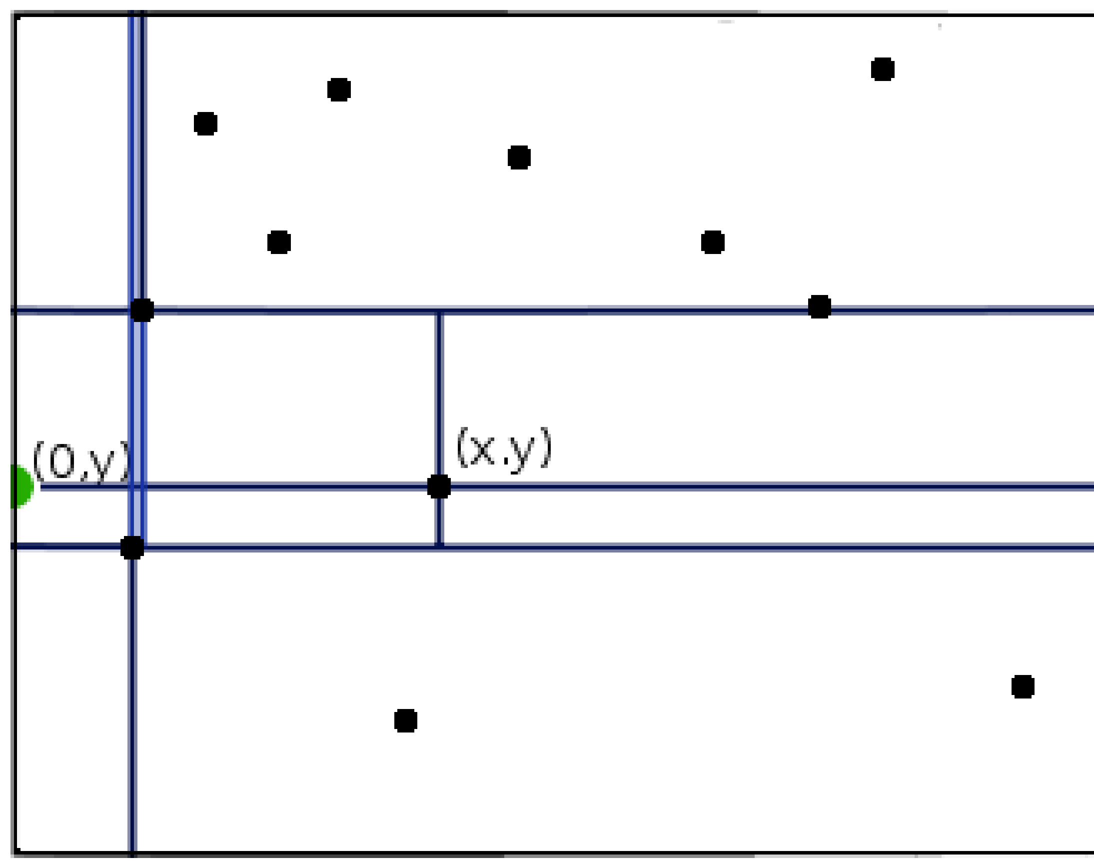
Barrido > Mayor rectángulo sin puntos adentro

- Para cada punto se busca el mejor rectángulo cuyo lado izquierdo esté sobre él.
- El lado derecho puede estar en otro punto a la derecha o sobre el borde derecho
- Para cada punto, se iteran los otros puntos de izquierda a derecha
- Se calculan “techo” y “piso” (en rojo)
- Un punto a la misma altura “parte” la búsqueda en dos



Barrido > Mayor rectángulo sin puntos adentro

- También se busca el mejor rectángulo que tenga el lado izquierdo sobre el borde izquierdo
- Si la cantidad de puntos no es cero, algún otro lado del rectángulo tiene que contener un punto (x, y), entonces se puede encontrar con el procedimiento anterior sobre (0, y)
- Si no hay puntos entonces el mejor rectángulo es todo el borde
- En total es **O(n²)**



Barrido > Mayor rectángulo sin puntos adentro

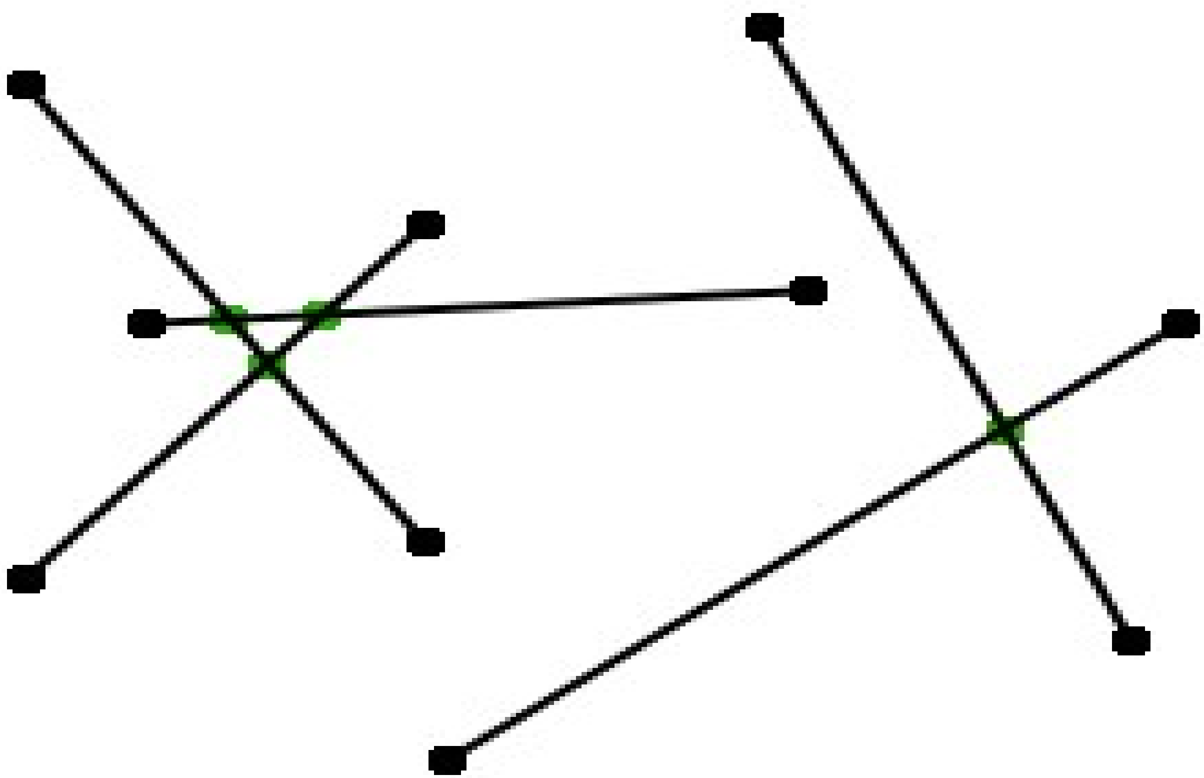
- Para practicar
  - Secure Region(sam03, live archive: 2882)

    http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2882
  - Chainsaw massacre (eur-sw00, live archive: 2204)

    http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2204

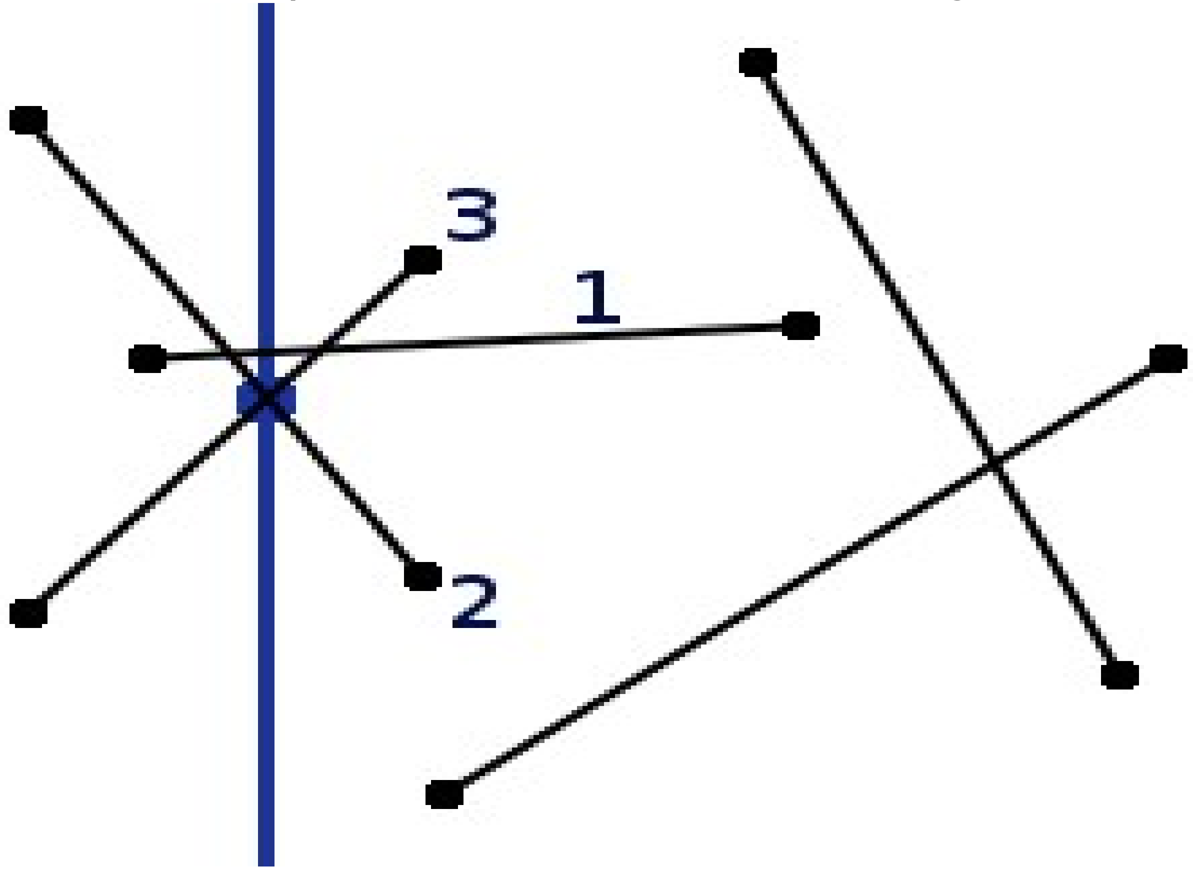
Barrido > Intersección de segmentos

- Entrada: Conjunto de segmentos
- Salida: Todas las intersecciones de segmentos
- Algoritmo de Bentley–Ottman: **O(n.log(n+i))**



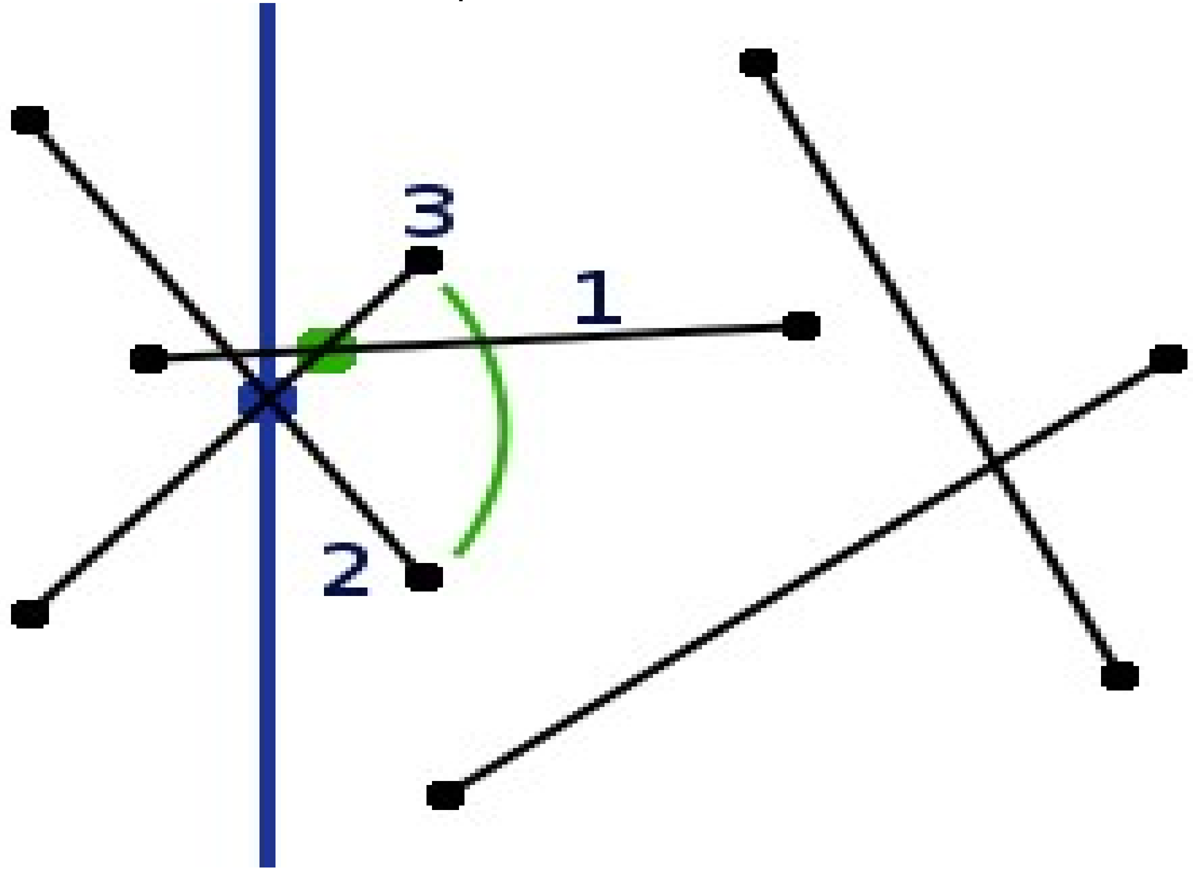
Barrido > Intersección de segmentos

- Barrido: Cola de eventos en “x”: vértices e intersecciones
- Los eventos de intersección se van agregando (priority queue)
- Se mantiene la lista de segmentos que cruzan el barrido ordenados verticalmente, esta lista se actualiza en cada evento
- Se necesita una función que calcule intersección de segmentos



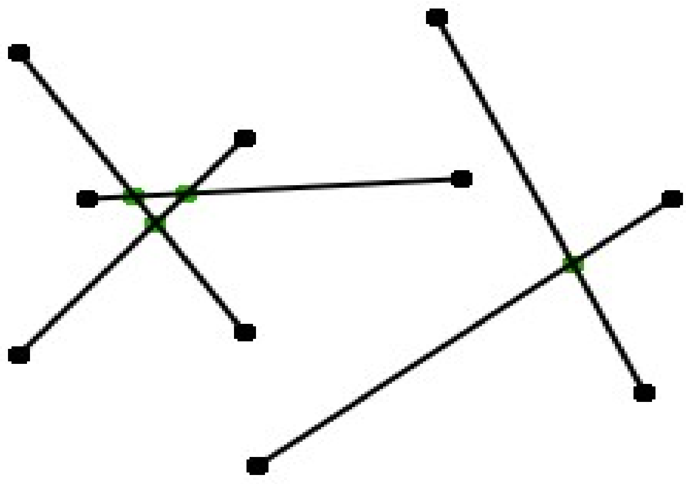
Barrido > Intersección de segmentos

- En los vértices que comienzan: Se agrega el segmento en la lista. Considerando la posición “y” de cada segmento en la lista sobre la “x” del evento. Se necesita una estructura eficiente que soporte esto.
- En los vértices que terminan: Se remueve el segmento de la lista
- En las intersecciones: Se intercambian los segmentos de lugar en la lista
- En todos los casos se chequan por nuevas intersecciones entre “nuevos vecinos” en la lista y se agregan a la cola de eventos si no están presentes



Barrido > Intersección de segmentos

- **Complicaciones** a tener en cuenta
  - Segmentos que se superponen: Considerarlos juntos en la lista de segmentos
  - Vértices sobre otros segmentos: Agregar el vértice y la interseccion a la cola de eventos, para que luego queden en el orden correcto
  - Segmentos verticales: puede ser un evento especial
  - Multiples intersecciones en un mismo punto: hay que diferenciarlas en la cola de eventos



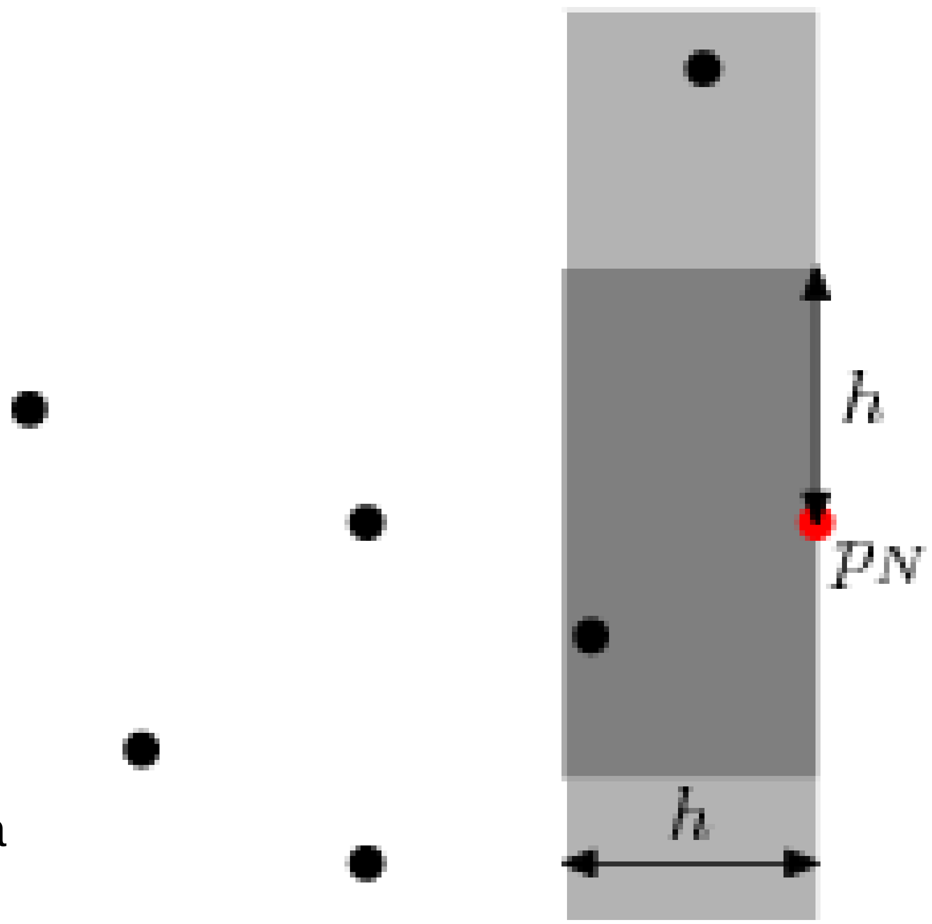
Barrido > Intersección de segmentos

- Para practicar
  - Painter (wf09, live archive: 4125)

    http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=4125

Barrido > Par de puntos más cercano

- Se barren los puntos en sentido horizontal
- Durante el barrido se mantiene:
  - El par de puntos más cercano encontrado y su distancia “h”
  - Todos los puntos a distancia horizontal menor a “h” del barrido ordenados verticalmente
- Por cada punto:
  - se agrega a la franja en orden vertical **O(log(n))**
  - se chequea para abajo y arriba contra los otros puntos sobre la franja a distancia vertical menor a “h”. Es **O(1)** como en el D&C
  - se sacan los puntos que quedaron fuera de la nueva franja: **O(n.log(n))** total



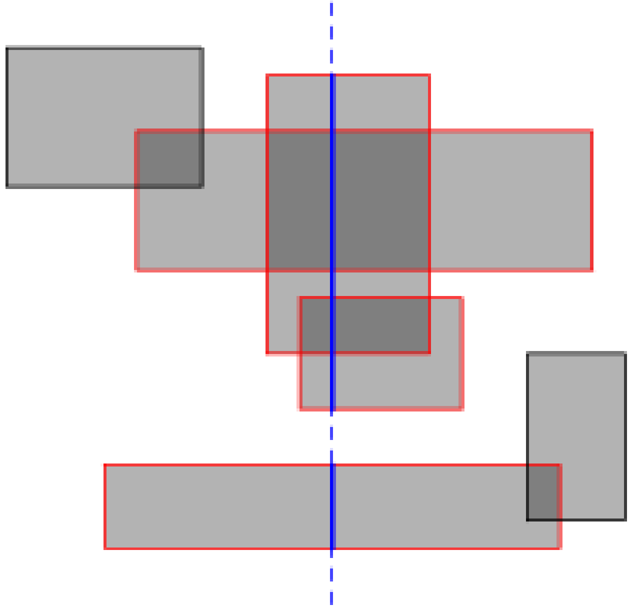
Barrido > Unión de intervalos

- Unión de intervalos
  - Dada una lista de intervalos de la recta real [a,b]
  - Calcular el tamaño de su unión
- Se recorren los extremos de los intervalos de izquierda a derecha calculando la cantidad de intervalos abiertos

```
int length(int n, double[] a, double[] b){
  vector<pair<double, int> > v;
  for(int i = 0; i < n; i++){
    v.push_back(make_pair(a[i], 1));
    v.push_back(make_pair(b[i], -1));
  }
  sort(v);
  double result = 0.0; int open = 0;
  for(int i = 0; i < v.size(); i++){
    if(open > 0) result += v[i].first – v[i-1].first;
    open += v[i].second;
  }
  return result;
}
```

Barrido > Unión de rectángulos

- Dado un conjunto de rectángulos, calcular el area de su unión
- El barrido se hace recorriendo los lados derecho e izquierdo de los rectángulos, en sentido horizontal
- Durante el barrido se mantiene la lista de intervalos que representan a los rectángulos sobre esa linea vertical.
  - Los intervalos se mantienen ordenandos verticalmente
  - Cada lado izquierdo agrega un intervalo y su lado derecho lo saca **O(n.log(n))**
- En cada paso se calcula la unión de estos intervalos y se multiplica por la distancia entre eventos. Como están ordenados es **O(n)**
- En total es **O(n².log(n))** y se puede lograr en **O(n.log(n))**



Fin!

;-)