.UBAfiuba
FACULTAD DE INGENIERÍA

División y Conquista: Puntos más cercanos en el plano

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Problema

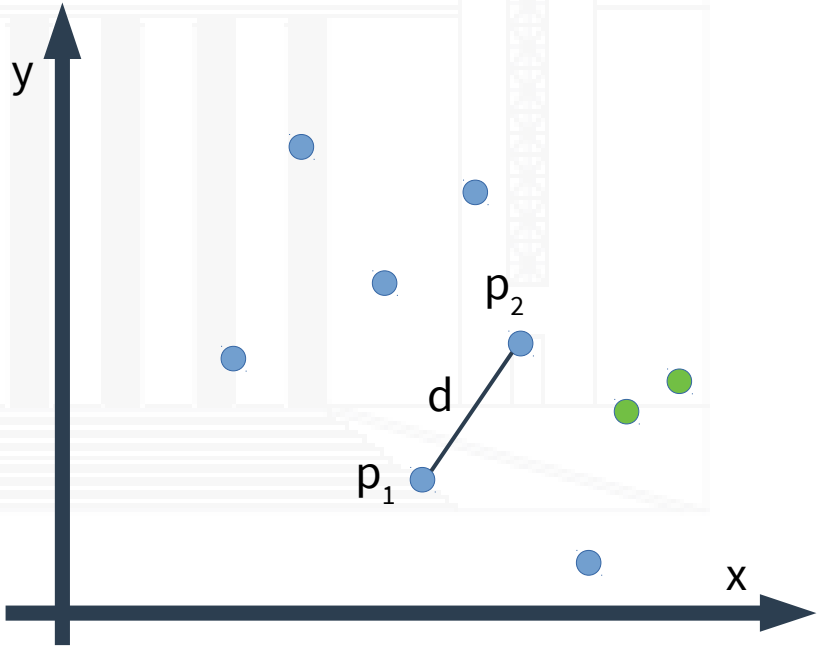
Sea

P un conjuntos de “n” puntos en el plano

$d(p_1,p_2)$ la función distancia entre $p_1=(x_1,y_1)$, $p_2=(x_2,y_2) \in P$

Queremos

Encontrar los puntos más cercanos en P



Solución por fuerza bruta

Para buscar los puntos más cercanos

Tengo que calcular las distancia entre cada punto y el resto de ellos

Quedarme con aquellos 2 de menor distancia

Teniendo n puntos

Para el primer punto calculo n-1 distancias

Para el segundo calculo n-2 distancias

...

Para el punto n calculo 0 distancias

Realizo en total n*(n-1)/2

Complejidad $O(n^2)$

¿Lo puedo hacer mejor?

Simplificación: puntos cercanos en una recta

Si reducimos

el problema a solo 1 dimensión

Podemos ordenar los puntos

De mayor a menor → O (n log n)

Luego por cada punto (iniciando por el menor)

Calcular la distancia al punto siguiente → O(n)

Recordar los puntos si es la distancia menor encontrara hasta el momento

¿Podemos hacer algo similar con los puntos en el plano?

(no directamente... pero utilizaremos el ordenamiento de los puntos como parte de la solución)

Solución utilizando División y conquista

El algoritmo utilizando división y conquista fue propuesto por

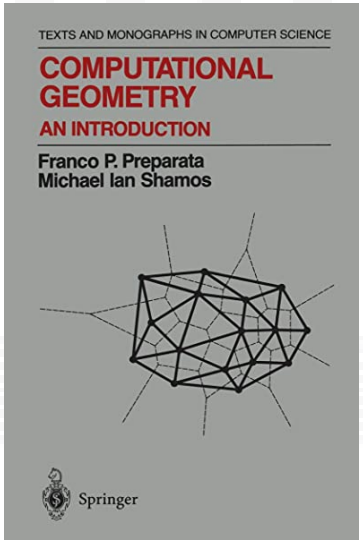
Michael. I. Shamos en 1975

Lo explica en detalle en su libro de 1985

"Computational Geometry. An introduction"

Con su coautor

Preparata, Franco P



Michael I.Shamos

Preliminares

Asumiremos que

No existe $p_1=(x_1,y_1), p_2=(x_2,y_2) \in P$ / $x_1 = x_2$ o $y_1 = y_2$

(Este requerimiento se puede eliminar modificando ligeramente el algoritmo)

Crearemos

Px: almacenando los puntos ordenados de acuerdo a la coordenada x

Py: almacenando los puntos ordenados de acuerdo a la coordenada y

Para cada punto $p \in P$

almacenaremos en que posición aparece en P_x y P_y

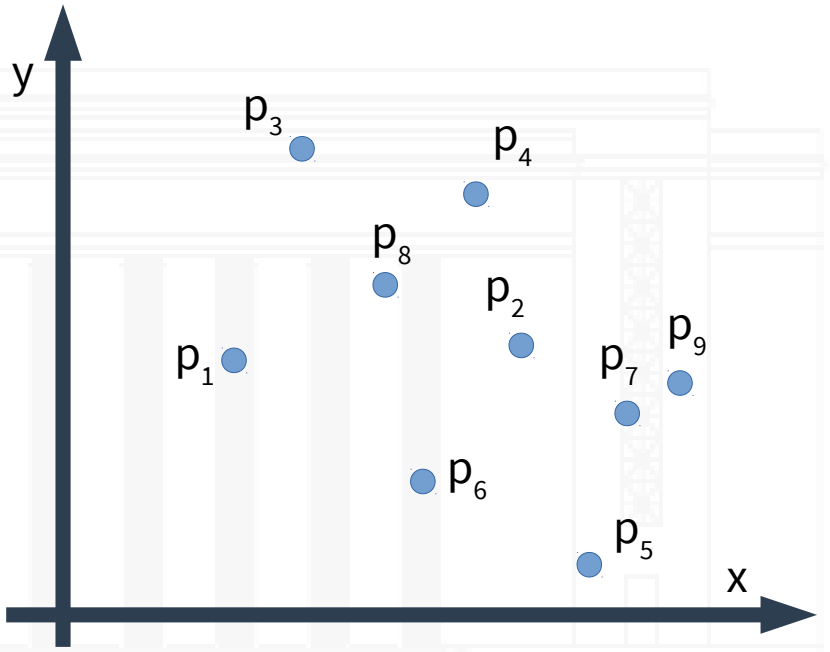
Ejemplo

En el ejemplo

P: $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$

P_x: $p_1, p_3, p_8, p_6, p_4, p_2, p_5, p_7, p_9$

P_y: $p_5, p_6, p_7, p_9, p_1, p_2, p_8, p_4, p_3$



División: Sub problemas

Llamaremos

Q al set de puntos que se encuentra en las primeras ⌈n/2⌉ posiciones de P_x

R al set de puntos que se encuentra en las ultimas ⌊n/2⌋ posiciones de P_x

Para

Q, calcularemos Qx y Qy

R, calcularemos Rx y Ry

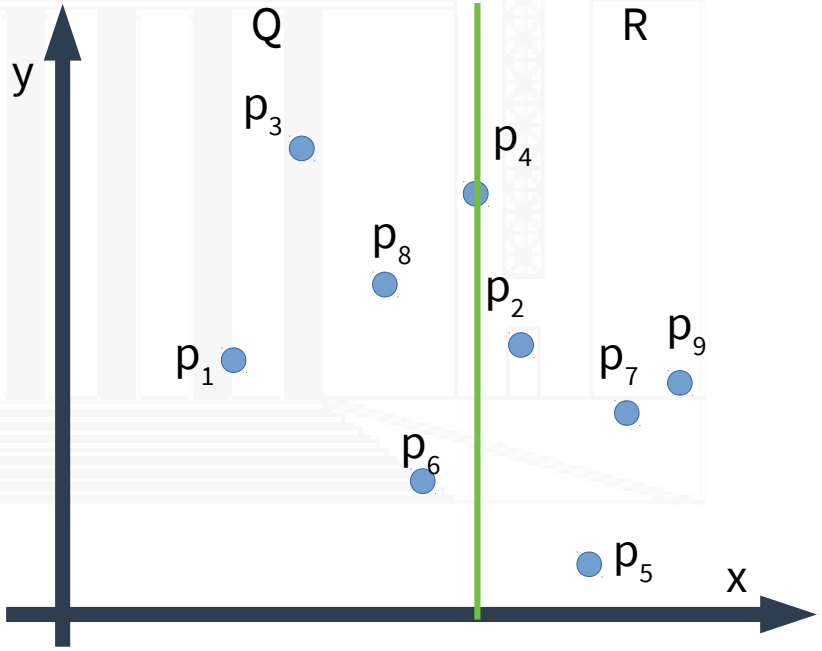
cada punto q ∈ Q

almacenaremos en que posición aparece en Q_x y Q_y

cada punto r ∈ R

almacenaremos en que posición aparece en R_x y R_y

(podemos hacerlo en O(n) utilizando P, P_x y P_y



Conquista: Sub problemas

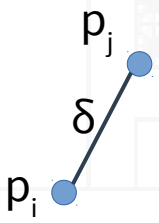
Recursivamente

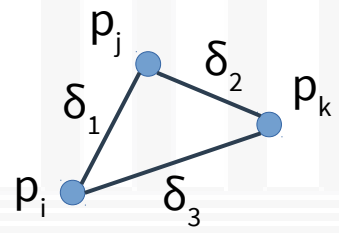
Resolveremos el problema de encontrar el par de puntos mas cercanos

Volviendo a dividir el subproblema en 2

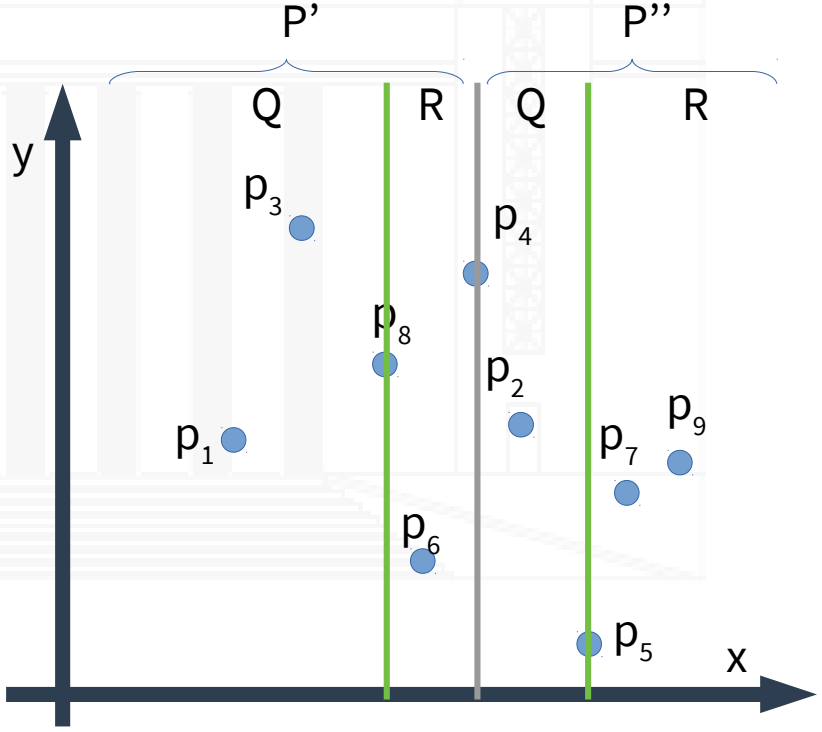
El problema base

Corresponde a un set de 2 o 3 puntos





$\delta = Min(\delta_1, \delta_2, \delta_3)$



Combinación: Sub problemas

Hasta el momento

Dividimos cada problema en 2 subproblemas de mitad de tamaño

Realizamos la división en O(n)

Para juntar los problemas

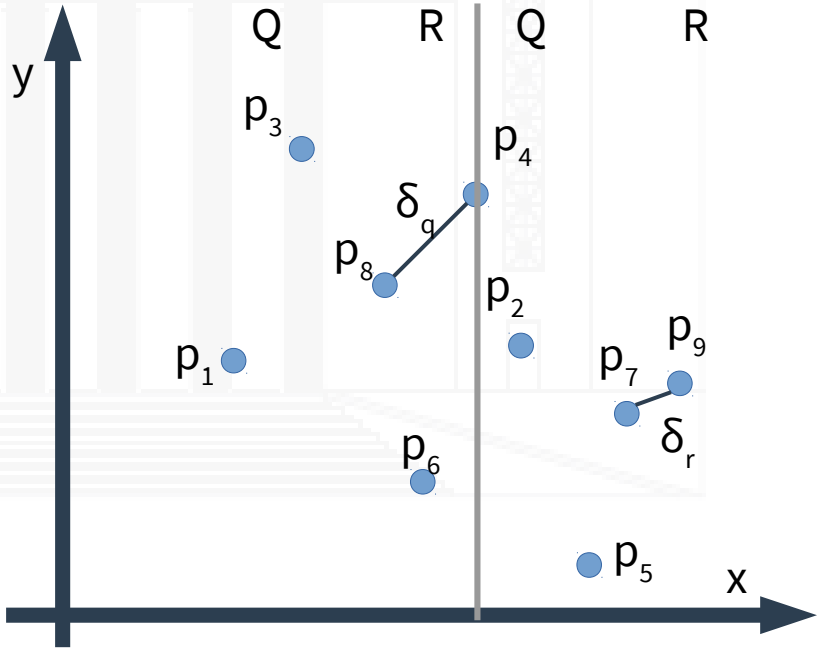
Comparamos los pares de puntos retornados por cada subproblema → $\delta = \min(\delta_q, \delta_r)$

Nos quedamos con el de menor distancia O(1)

Lo podemos expresar con la recurrencia

$T(n) = 2\,T(n/2) + O(n)$

… que por teorema maestro: O(nlogn)



Combinación: Sub problemas

Pero, pero... Hay un problema fundamental en nuestra solución.

La distancia mínima podría darse entre el punto $q \in Q$ y el punto $r \in R$

Al combinar

Debemos tener en cuenta los puntos de ambos subproblemas

Comparar los puntos de un lado con los del otro

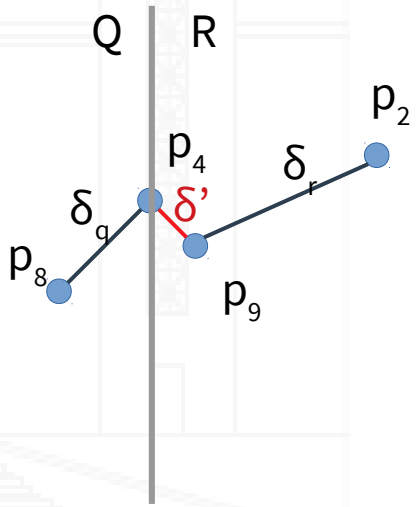
Quedando una recurrencia $T(n) = 2T(n/2) + O(n^2)$

Según teorema maestro quedando

Una complejidad $T(n)= O(n^2)$

Que indica que no es mejor que fuerza bruta

Debemos encontrar una forma más sencilla de combinar los subproblemas



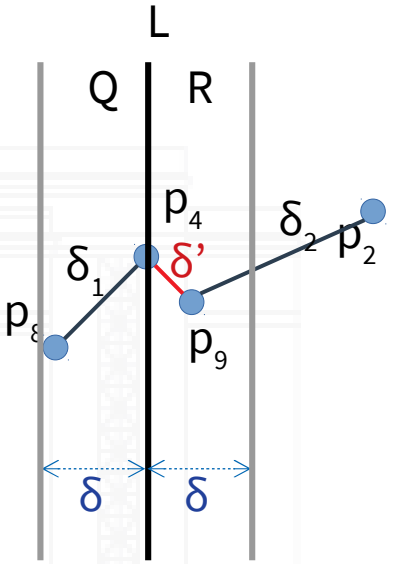
Puntos entre los subproblemas

Llamemos

x* a la coordenada del punto Q mas a la derecha

L a la linea vertical con la ecuación x=x*

(L separa Q y R)



Vemos que

si distancia mínima esta entre un punto q ∈ Q y un punto r ∈ R

(su distancia tiene que ser menor a $\delta=\min(\delta_q, \delta_r)$)

No pueden estar

a más distancia de δ de la linea L

Puntos entre los subproblemas (cont.)

Nos interesa conocer aquellos puntos S ⊆ P

Que se encuentran a una distancia menor a δ de L

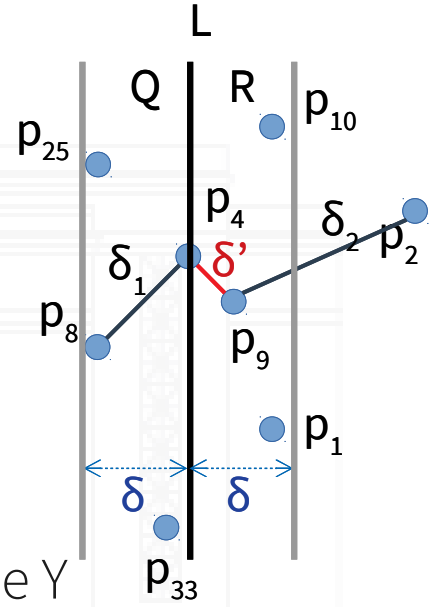
Podemos

obtenerlos en O(n) Recorriendo P

Llamaremos

S_y a la lista de elementos de S ordenadas por la coordenada de Y

(Que podemos construir en O(n) mediante P_y)



$S=p_1, p_4, p_8, p_9, p_{10}, p_{25}, p_{33}$

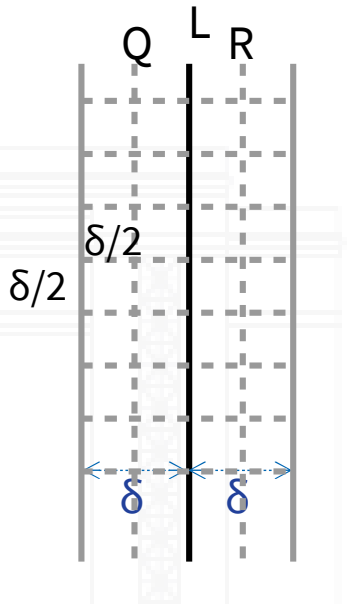
$S_y=p_{33}, p_1, p_8, p_9, p_4, p_{25}, p_{10}$

Puntos entre los subproblemas (cont.)

Podemos ver que

Si subdividimos el espacio entre x=L-δ y x=L+δ

(como se indica en la figura)



En celdas

de δ/2 x δ/2

Solo puede existir un punto P dentro de una celda

(de lo contrario la distancia mínima en Q o en R seria menor a δ)

Puntos entre los subproblemas (cont.)

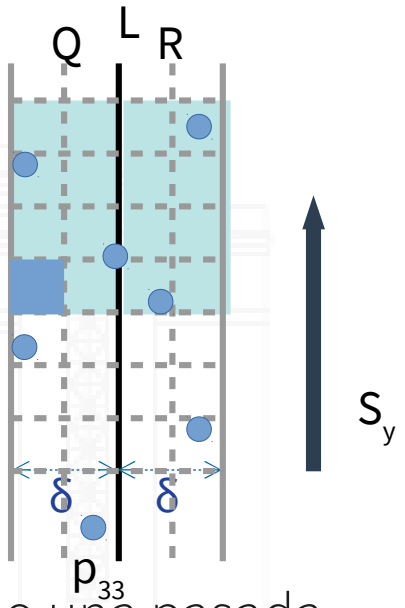
Cada punto s ∈ S estará en una celda

Debemos comparar s unicamente con otros puntos de S que estén en celdas cercanas (a no mas de δ de distancia)

Recorriendo S en el orden de S_y

Podemos ver que unicamente tenemos que comparar a lo sumo con 15* puntos siguientes (o 15 celdas siguientes)

Es decir que hacemos c=15 comparaciones por punto en S en solo una pasada



En el peor escenario tenemos 15n comparaciones

Por lo tanto combinar el resultado es O(n)

* se puede demostrar que se puede reducir de 15 a 7

Pseudocódigo

```
Puntos-cercanos(P)
    Construir Px y Py   // O(nlogn)
    (p0, p1) = puntos-cerc-rec(Px,Py)

puntos-cerc-rec(Px,Py)
    Si |Px|<=3
        retornar (p0, p1) par mas cercano comparando todos los puntos
    Sino
        Construir Qx, Qy, Rx, Rz // O(n)

        (q0, q1) = puntos-cerc-rec(Qx,Qy)
        (r0, r1) = puntos-cerc-rec(Rx,Ry)

        d = min (dist(q0,q1),dist(r0,r1))
        x' = máxima coordenada x de punto en Q
```

Pseudocódigo (cont)

```
L = {(x,y) : x=x'}
S = puntos de P a distancia d de L

Construir Sy  // O(n)
Por cada punto s de Sy          // O(n)
    computar distancia con próximos 15 puntos de Sy
    sea s, s' el par de puntos de menor distancia


Si dist(s,s')< d
    retornar (s,s')
Sino si dist(q0, q1)< dist(r0, r1)
    retornar (q0, q1)
sino
    retornar (r0, r1)
```

.UBAfiuba

FACULTAD DE INGENIERÍA

Presentación realizada en Septiembre de 2020