Problema della coppia più vicina

Dato un insieme P di n punti nel piano cartesiano, vogliamo progettare un algoritmo che restituisce la coppia di punti di P la cui distanza euclidea è minima.

Soluzione immediata: per ogni coppia di punti si calcola la distanza e si restituisce la coppia che dà luogo al minimo.

Complessità: $O(n^2)$.

Utilizzando la strategia divide et impera possiamo progettare un algoritmo di costo $O(n \log n)$.

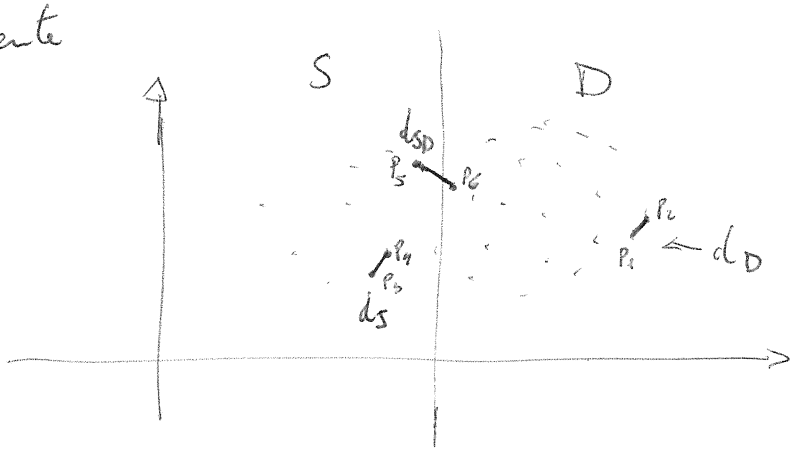
Idea:
- se $|P|$ è piccola, applico la soluzione immediata descritta in precedenza
- altrimenti, divido P in due parti uguali, S e D, per esempio con S che raccoglie i punti a sinistra di una linea verticale e D quelli a destra.

  Troviamo ricorsivamente le coppie più vicine in S e D, a distanza d_S e d_D, rispettivamente.

  Troviamo la coppia a distanza minima d_{SD} con un punto in S e uno in D.

  Diamo in output la coppia di punti a distanza $\min\{d_S, d_D, d_{SD}\}$

Graficamente



Se riusciamo in tempo $O(n)$ a calcolare la coppia a distanza minima d_{SD} (con un punto in S e uno in D) e a dare in output la coppia tale che la distanza sia

$$\min\{d_S, d_D, d_{SD}\}$$

allora l'algoritmo complessivamente costa

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq K \text{ (costante)} \\ 2T(n/2) + c \cdot n & \text{altrimenti} \end{cases}$$

Per il Master Theorem, $T(n) = O(n \log n)$

Cerchiamo di capire come implementarla!

Sia $p \in P$ un punto. Indichiamo con

$p.x$ (ascissa) e $p.y$ (ordinata)

Assumiamo che i punti abbiano ascisse distinte.

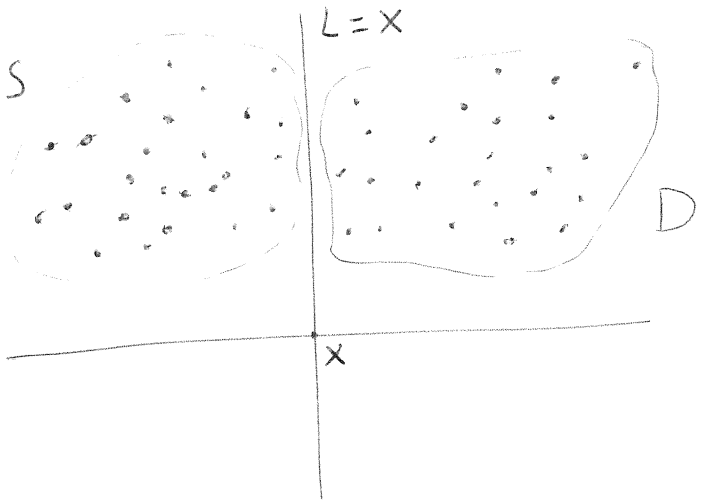
Ordiniamo i punti di P per ascissa.

Sia x l'ascissa del punto in posizione centrale nell'ordinamento

Definiamo i due insiemi:

$$S = \{ p \in P : p.x \leq x \} \qquad D = \{ p \in P : p.x > x \}$$

di punti a sinistra della retta $L = x$ e a destra



Come anticipato, l'algoritmo risolve ricorsivamente il problema per gli insiemi S e D, ottenendo coppie a distanza d_S e d_D e poi cerca la soluzione finale considerando i punti "a cavallo" della retta $L = x$.

Detta cioè $d = \min\{d_S, d_D\}$, l'algoritmo cerca $p_s \in S$ e $p_d \in D$ tali che la loro distanza sia $< d$.

Quindi la ricerca di p_s e p_d si può restringere ai sottoinsiemi:

$$S' = \{p \in S : x - d \leq p.x \leq x\} \qquad D' = \{p \in D : x \leq p.x \leq x + d\}.$$

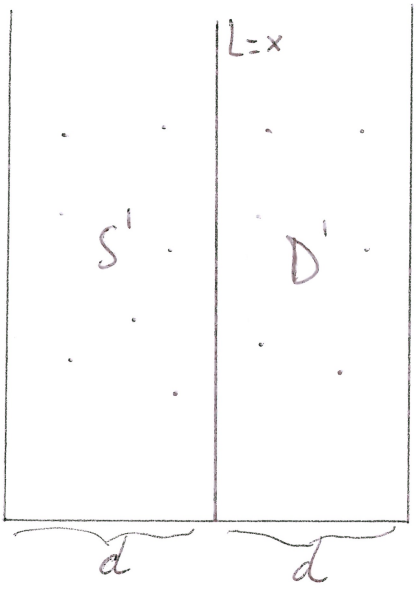
Infatti p_s e p_d non possono essere a distanza maggiore di d dalla retta $L = x$.

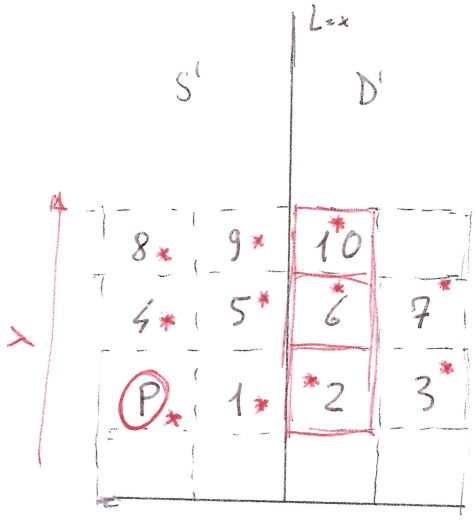
Nota che sia S' che D' potrebbero contenere molti punti, $\Omega(n)$. Ma si può dimostrare che:

> fissato un punto $p \in S'$, i punti in D' potenzialmente a distanza $< d$ sono pochi, in numero costante.

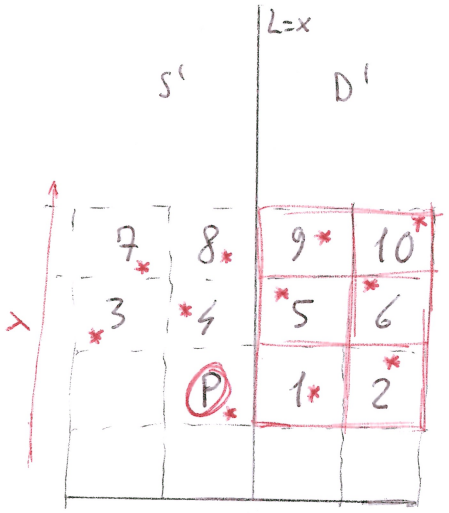
Lemma 3.1 Sia P_d la lista dei punti in $S' \cup D'$ ordinati secondo la loro ordinata $(p.y)$. Se due punti p e q in $S' \cup D'$ sono a distanza $< d$, allora occorrono come elementi in P_d in due posizioni che distano al più 10.

Dimostrazione. Dividiamo la fascia di punti a distanza d da $L=x$ attraverso una griglia composta da quadrati di lato $d/2$.





(a)



(b)

Prima osservazione: ogni quadrato contiene al più un punto.

Se ne avesse due, questi avrebbero distanza $< d/2$, contro l'ipotesi che i due punti più vicini trovati in S e in D hanno distanza d!

Calcoliamo il numero massimo di punti in $S' \cup D'$ che separano p e q in P_d. Supponiamo che p preceda q nell'ordinamento ($p.y < q.y$) in P_d e che $p \in S'$ e $q \in D'$.

Il punto p può essere in una delle non adiacente (fig (a)) o adiacente (fig (b)) alla retta $L = x$.

Il punto q può essere nelle celle 2, 6, 10 (fig (a)) o 1, 2, 5, 6, 9, 10 (fig (b)). In entrambi i casi q è al più 10 posizioni avanti a p. ∎

Pertanto, per trovare una coppia di punti in $S' \cup D'$ a distanza minore di d è sufficiente scorrere i punti nell'ordinamento P_d e, per ognuno di questi, calcolare le distanze con i 10 punti che lo seguono nell'ordinamento. Il costo di questa operazione è lineare in n, cioè $O(n)$.

Prima di eseguire l'algoritmo occorre allora produrre due ordinamenti dei punti, P_x e P_y, per ascissa e per ordinata rispettivamente. Poi si eseguono le chiamate ricorsive.

L'algoritmo prende in input P_x e P_y e procede come segue:

1. Sia n il # punti in input: se n è piccolo, soluzione immediata.
2. Altrimenti, costruisci S_x e D_x selezionando i primi $n/2$ punti in P_x e i secondi $n/2$ punti in P_x.
3. Sia p il punto mediano di P_x. Dividi P_y in S_y e D_y: S_y contiene i punti di P_y a sinistra di p, D_y quelli a destra.
4. Invoca la funzione (ricorsivamente) con input (S_x, S_y) e (D_x, D_y) ottenendo due coppie di punti a distanza rispettivamente d_S e d_D: sia $d = \min\{d_S, d_D\}$
5. Dalla sequenza ordinata P_y estrapola i punti a distanza al più d dalla retta verticale passante per p

creando la sequenza P_d (di taglia n).

6. Trova la coppia (p, q) finale usando come candidate:
   - la coppia a distanza d trovata nel passo ricorsivo
   - le coppie di punti in P_d, limitandosi a controllare, per ogni punto, le distanze con i 10 punti che seguono in P_d.

Teorema. L'algoritmo è corretto e la sua complessità è $O(n \cdot \log n)$.

Nota: nel codice che segue $Dist(p, q)$ è la funzione che calcola la distanza Euclidea tra i punti p e q. Precisamente, detti $p = (p_x, p_y)$ e $q = (q_x, q_y)$,

$$Dist(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

```
CoppiaPiuVicina ( Px , Py , n)        → taglia del problema = # punti
                                      → array degli n punti ordinati per ordinata
                                      → array degli n punti ordinati per ascisse
IF (n ≤ 3) {
   RETURN RicercaEsaustiva (Px, Py, n);        /* è la soluzione immediata
} ELSE {
   P = Px [n/2];                               /* punto mediano
      for (i=0; i < n/2; i++) {                /* decompone il problema in due sottoproblemi (Sx, Sy) e (Dx, Dy)
         Sx[i] = Px[i];  Dx[i] = Px[i + n/2];
      }
      for (i=j=k=0; i<n; i++)
        IF (Py[i].x ≤ p.x) { Sy[j] = Py[i]; j = j+1; }
          ELSE { Dy[k] = Py[i]; k = k+1; }
   (ps, qs) = CoppiaPiuVicina (Sx, Sy, n/2);   /* Invocazioni ricorsive sui sottoproblemi
   (pd, qd) = CoppiaPiuVicina (Dx, Dy, n/2);

individua                                      IF ( Dist (ps, qs) < Dist (pd, qd)) { d = Dist (ps, qs); (p, q) = (ps, qs); }
- min d tra                                      ELSE { d = Dist (pd, qd); (p, q) = (pd, qd); }
- soluzioni dei
- sottoproblemi

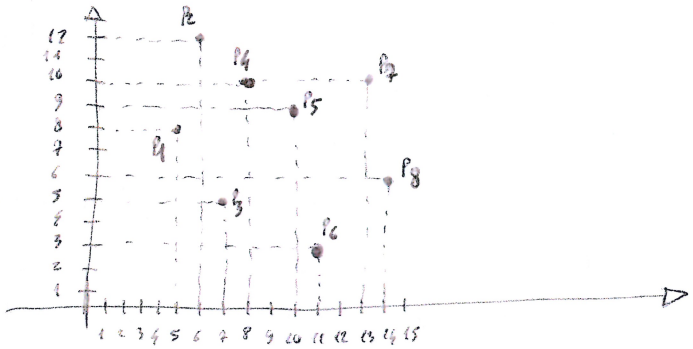
* Costruisce                                   for (i = m = 0;  i < n;  i = i+1;) {
l'insieme Pd                                     IF ( | Py[i].x - p.x | ≤ d) { Pd[m] = Py[i]; m = m+1; }
                                               }

Controlla                                      for (i=0; i < m; i = i+1) {
tra le                                           for (j = i+1;  j ≤ min {i+10, m};  j = j+1) {
coppie ammissibili                                 IF (Dist (Pd[i], Pd[j]) < d {
- Pd ce n'è                                          d = Dist (Pd[i], Pd[j]);
una migliore                                         (p, q) = (Pd[i], Pd[j]);
- quella trovata                                   }
nelle invocazioni                                }
ricorsive                                      }
   RETURN (p, q);                              /* coppia di punti più vicini
}
```

Esempio di esecuzione:

Siano dati i punti:

$P_1 = (5, 8)$ $P_2 = (6, 12)$ $P_3 = (7, 5)$ $P_4 = (8, 10)$ $P_5 = (10, 9)$ $P_6 = (11, 3)$ $P_7 = (13, 10)$ $P_8 = (14, 6)$



Chiamata iniziale

$CPV(P_X, P_Y, 8)$

dove:

$$P_X = \{ \overset{P_1}{(5,8)}, \overset{P_2}{(6,12)}, \overset{P_3}{(7,5)}, \overset{P_4}{(8,10)}, \overset{P_5}{(10,9)}, \overset{P_6}{(11,3)}, \overset{P_7}{(13,10)}, \overset{P_8}{(14,6)} \}$$

$$P_Y = \{ \overset{P_6}{(11,3)}, \overset{P_3}{(7,5)}, \overset{P_8}{(14,6)}, \overset{P_1}{(5,8)}, \overset{P_5}{(10,9)}, \overset{P_7}{(13,10)}, \overset{P_4}{(8,10)}, \overset{P_2}{(6,12)} \}.$$

$$p = P_X[n/2] = P_X[4] = (8,10) = P_4$$

$$S_X = \{ \overset{P_1}{(5,8)}, \overset{P_2}{(6,12)}, \overset{P_3}{(7,5)}, \overset{P_4}{(8,10)} \} \qquad D_X = \{ \overset{P_5}{(10,9)}, \overset{P_6}{(11,3)}, \overset{P_7}{(13,10)}, \overset{P_8}{(14,6)} \}$$

$p.x = 8$

$$S_Y = \{ \overset{P_3}{(7,5)}, \overset{P_1}{(5,8)}, \overset{P_4}{(8,10)}, \overset{P_2}{(6,12)} \} \qquad D_Y = \{ \overset{P_6}{(11,3)}, \overset{P_8}{(14,6)}, \overset{P_5}{(10,9)}, \overset{P_7}{(13,10)} \}$$

Le due chiamate ricorsive sono:

$$CPV(S_X, S_Y, 4) \quad e \quad CPV(D_X, D_Y, 4)$$

La chiamata CPV $(S_x, S_y, 4)$ restituisce $(p_s, q_s) = \underset{P_2}{(6,12)}, \underset{P_4}{(8,10)}$

La chiamata CPV $(D_x, D_y, 4)$ restituisce $(p_d, q_d) = \underset{P_5}{(10,9)}, \underset{P_7}{(13,10)}$

$Dist(P_2, P_4) = \sqrt{(6-8)^2 + (10-12)^2} = \sqrt{8}$

$Dist(P_5, P_7) = \sqrt{(10-13)^2 + (9-10)^2} = \sqrt{10}$

$\Rightarrow$ $d = \sqrt{8} = 2\sqrt{2} = 2{,}8 < 3$

$(p, q) = (P_2, P_4)$

Costruiamo l'insieme P_d. Ricordo che $p.x = 8$ e che

$$P_y = \{ \underset{P_6}{(11,3)}, \underset{P_3}{(7,5)}, \underset{P_8}{(14,6)}, \underset{P_1}{(5,8)}, \underset{P_5}{(10,9)}, \underset{P_7}{(13,10)}, \underset{P_4}{(8,10)}, \underset{P_2}{(6,12)} \}$$

Pertanto, $|P_y[i] - p.x|$ per $i = 1, \dots, 8$, dà luogo a:

$\lvert 11-8 \rvert$	$\lvert 7-8 \rvert$	$\lvert 14-8 \rvert$	$\lvert 5-8 \rvert$	$\lvert 10-8 \rvert$	$\lvert 13-8 \rvert$	$\lvert 8-8 \rvert$	$\lvert 6-8 \rvert$
NO	SI	NO	NO	SI	NO	SI	SI

$$P_d = \{ \underset{P_3}{(7,5)}, \underset{P_5}{(10,9)}, \underset{P_4}{(8,10)}, \underset{P_2}{(6,12)} \} \qquad |P_d| = 4$$

Le distanze da calcolare sono allora:

$Dist((7,5),(10,9))$ $Dist((7,5),(8,10))$ $Dist((7,5),(6,12))$

$Dist(\underset{P_5}{(10,9)}, \underset{P_4}{(8,10)})$ $Dist((10,9),(6,12))$

$Dist((8,10),(6,12))$

Poiché $Dist(P_5, P_4) = \sqrt{(10-8)^2 + (9-10)^2} = \sqrt{5} < \sqrt{8} = d$,

la coppia restituita dall'algoritmo è $(p, q) = (P_4, P_5)$.