Problem set 2
Esercizio 1

Alessandro Straziota

Esercizio 1. Si consideri il seguente problema computazionale:
INPUT: Un insieme di punti del piano $\{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\} \subseteq \mathbb{R}^2$
OUTPUT: La coppia (una coppia, se ce n'è più di una) a distanza minima. Ossia, una coppia di punti distinti p_i, p_j, tali che

$$d(p_i, p_j) = \min\{d(p_h, p_k) \, : \, h, k = 1, \ldots, n, \, h \neq k\}$$

dove $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ è la distanza Euclidea fra i due punti.

1. Osservare che esiste un algoritmo ovvio che ha running time $\mathcal{O}(n^2)$;
2. Usando la tecnica Divide et Impera, progettare un algoritmo con running time $o(n^2)$.

Un algoritmo abbastanza immediato che viene in mente è quello di dividere in due parti l'insieme di punti, calcolare ricorsivamente le coppie di di punti a distanza minima nelle due sottoistanze e poi verificare quale fra le due è minore dell'altra. Però questo metodo non è corretto perché nulla vieta il fatto che i due punti a distanza minima siano nei due "versanti" differenti. Allora a questo punto si potrebbe pensare di verificare per ogni coppia di punti (per esempio) sulla destra se esiste un punto sulla sinistra che ha distanza minima minore a quella calcolata in precedenza. Questo metodo però non porta a nessun vantaggio in termini di complessità e si può facilmente verificare con l'equazione di ricorrenza

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^2) \in O(n^2)$$

Infatti è evidente che il costo del confronto dei punti sulla destra con quelli sulla sinistra ha complessità

$$\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4} \in O(n^2)$$

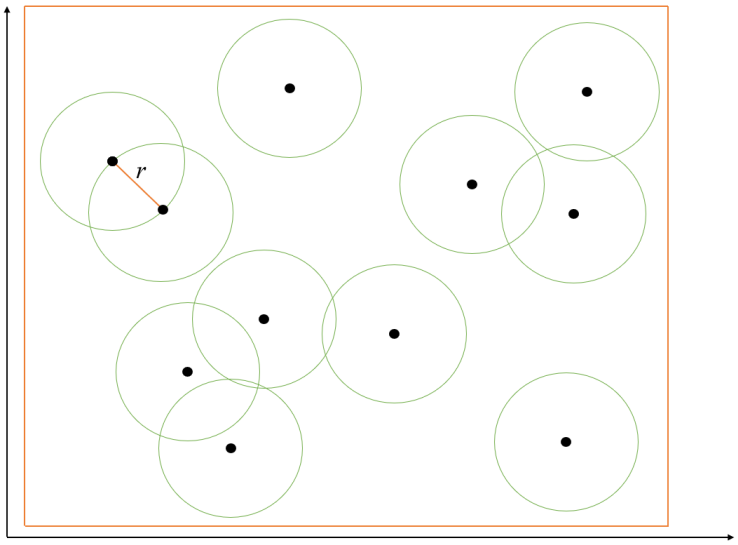


Figure 1: Esempio di distanze dei punti in un versante

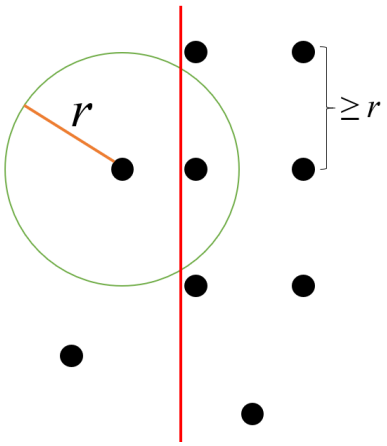
C'è però da fare un'altra piccola osservazione che potrebbe condurre a una soluzione valida:
Sia r la distanza minima trovata tra i due "versanti", *certamente* qualsiasi distanza tra due punti che appartengono a uno stesso versante sarà minore i tutt'al più uguale a r.

Quindi se proprio esiste una coppia di punti a, b appartenenti a due versanti differenti tale che la loro distanza sia *minima* certamente questa distanza deve essere minore di r. Perciò questo riduce la ricerca a tutti quei punti distanti al più r dalla "frontiera" che divide i due versanti.

Verrebbe da pensare (giustamente) che anche questo metodo non porti altri vantaggi, infatti potrebbe capitare che anche se riduciamo l'area di ricerca di un'eventuale distanza minima a cavallo dei due versanti, potrebbe capitare di considerare un numero di punti proporzionale ad n.

Però osserviamo che, se consideriamo i punti in ordine di *ordinata*, possiamo confrontare per ogni punto tutti e soli quei punti che *a)* appartengono al versante opposto e *b)* sono distanti rispetto a y (tanto so che rispetto a x non sono più distanti di $2 \cdot r$) non più di r, forti del fatto che per tutti e due

i versanti nessun punto è più vicino a un altro meno r.
Ora non resta che capire quanti punti del versante destro devo confrontare per ogni punto del versante sinistro (o viceversa). Verrebbe da chiedersi anche in questo caso "*e se ce ne sono $O(n)$?*". Beh, non sarebbe possibile, perché andando a confrontare solamente punti distanti non più di r rispetto all'asse delle y nel versante opposto, stiamo in pratica restringendo l'area di ricerca a un *quadrato* $r \times r$.
Infatti è facile notare che per ogni versante, all'interno dell'area di un quadrato $r \times r$, ci potranno essere **al più** quattro punti (sempre perché, ricordiamo, r è la distanza minima possibile tra due punti in uno stesso versante).



Perciò un algoritmo possibilmente efficiente è il seguente:

1. Divido a metà l'insieme dei punti e *ricorsivamente* trovo le coppie di punti con distanza minima nei rispettivi sottinsiemi

2. Date le due distanze minime trovate nei due sottinsiemi scelgo quella più piccola (che chiameremo r)

3. prendo il punto $p_k = (x_k, y_k)$ del versante sinistro con valore x massimo (ovvero quel punto sul "limite" destro del versante)

4. prendo tutti i punti del mio problema con valore di x compreso tra $x_k - r$ e $x_k + r$, e scarto tutti gli altri

5. quelli rimasti li ordino in senso *non crescente* rispetto all'asse delle y

6. in ordine, per ogni punto p_j che apparteneva al versante sinistro prendo tutti quei punti che erano nel versante destro con valore di y compreso tra $y_j + r$ e $y_j - r$ (tanto in base alle osservazioni precedenti o in base all'immagine so essere al più 6) e calcolo la distanza minima

7. una volta trovata la coppia di punti con distanza minima in tutto questo sottinsieme la confronto con r, se la loro distanza è minor di r allora quella sarà la coppia con distanza minima altrimenti era quella coppia la cui distanza è r

È ora doverosa un'analisi sui costi di questo algoritmo. Osserviamo che i punti compresi tra $x_k - r$ e $x_k + r$ possono essere dell'ordine di $O(n)$, perciò ordinarli mi costa $O(n \log n)$. Adesso, dato che la sequenza di punti è ordinata proprio rispetto a y, andare a prendere tutti quei punti con y compreso tra $y_j + r$ e $y_j - r$ mi costa tempo $O(\log n)$ (perché eseguirò al più 6 ricerche dicotomiche). Ancora, una volta trovati i punti interessati calcolare le distanze minime mi costa tempo costante ($O(1)$).
Perciò la fase di *impera* avrà complessità $O(n \log n)$.

Alla luce di questo risultato è possibile stimare il costo dell'algoritmo con l'equazione di ricorrenza

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + O(n \log n) \\ &= O(n \log n) \end{aligned} \tag{1}$$