Divide and Conquer

sabato 22 maggio 2021 08:36

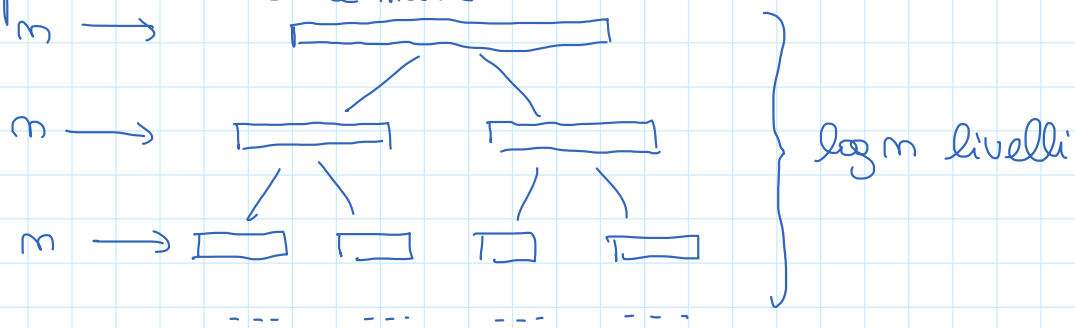
Cos'è? D&C (o Divide et Impera) è un paradigma della programmazione che consta dei seguenti passi:

di solito facili
- divido in "due" parti (solitamente, ma non sempre, a metà) lo spazio di ricerca;
- risolvo ricorsivamente il problema su ciascuna delle due parti;

difficile →
- combino le soluzioni trovate ed estendile alla soluzione globale.

Il criterio per dividere in due può anche essere diverso dallo spezzare a metà. A volte, ad esempio si spezza in corrispondenza del minimo/massimo di un array.

La complessità di questi algoritmi ha solitamente un fattore $\log(n)$ (quando si divide a metà).



ESEMPI

ESEMPIO 1 – MAX OF SUBARRAY

Dato un array di interi, determinare il sottoarray con somma massima.

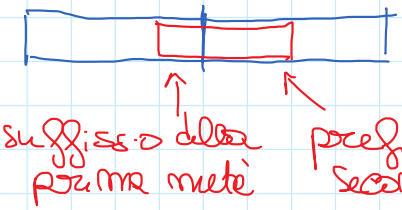
Esistono soluzioni (facili) $O(n)$, ma ora ne vediamo una $O(n \log n)$ che usa D&C.

- Step 1: divido l'array a metà
- Step 2: per ogni metà, calcolo la massima somma di un sottoarray interamente contenuto in quella metà.

- Step 3: combino ed estendo le soluzioni.

Per lo Step 3 siano S_l e S_r le due soluzioni parziali, e sia S la soluzione globale.

Allora $S = \max\{ S_l, S_r, $ max somma di un sottoarray che "attraversa" la metà $\}$



Quindi mi basta sommare la max somma di un suffisso e la max somma di un prefisso (complessità $O(n)$).

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \rightsquigarrow O(n \log n)$$

ESEMPIO 2: PONTI

Ci sono n torri, la i-esima ha altezza h_i. Posso costruire un ponte tra le torri i e j $(i < j)$ ad altezza y se:

- $h_i, h_j \geq y$;
- per $i < k < j$, $h_k < y$.

Il costo è $y + c$, dove c è una costante data

Voglio collegare la torre 0 alla torre $n-1$ con il minimo costo.

Soluzione $O(n \log n)$.

Considero le torri 0 e $n-1$. Ho due possibilità: o le connetto direttamente, o devo spezzare in corrispondenza di una delle torri interne di altezza massima.

La prima cosa si può fare solo se $h_{max} < h_0, h_{n-1}$.

Dopodiché ricorro sui due pezzi

La complessità di ricorrenza è $O(n)$, perché ogni torre viene considerata come massimo esattamente una volta.

(Caso base: ans(i, i+1) = e).

Calcolare il max costa log n.

Il problema si può risolvere anche con Dijkstra.

ESEMPIO 3: MIN DISTANCE

Dati n punti nel piano, calcolare la distanza minima tra tutte le coppie di punti.

Soluzione naive $O(n^2)$ (provo tutte le coppie).

Si può fare in $O(n \log n)$ con D&C.

- Primo step: ordino i punti per coordinata x e metto i primi $\lfloor \frac{n}{2} \rfloor$ in un gruppo e gli altri nell'altro gruppo.
- Secondo step: calcolo ricorsivamente le risposte per i due gruppi. Siano queste d_l e d_r.
- Terzo step: calcolo la distanza minima tra coppie di punti che si trovano in gruppi diversi.

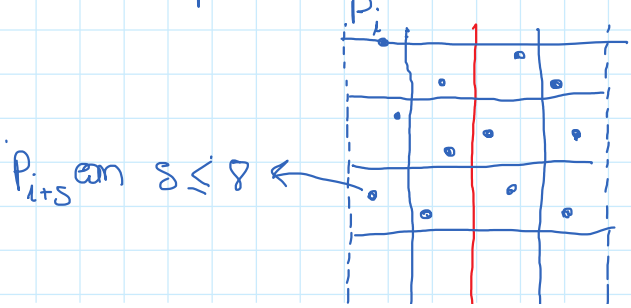
Apparentemente il terzo step è $O(n^2)$, ma si può fare in $O(n)$!

Sia $d_{min} = \min\{d_l, d_r\}$. Prendo una retta verticale che splitta i due gruppi, e considero il sottoinsieme di punti che distano $\leq d_{min}$ da tale retta.

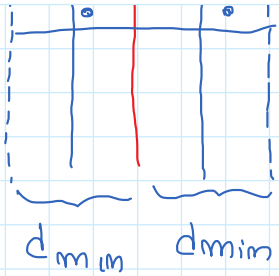
Ordino questi punti per coordinata y, siano essi $P_1, \dots, P_k$.

Allora si dimostra che per $1 \leq i \leq k$, mi basta controllare soltanto i successivi 7 punti.

Questo ha complessità $O(n)$.



WLOG P_i sta sul bordo superiore di uno dei quadrati



Osservazione chiave: ciascun quadrato contiene al più un punto. Infatti, se un quadrato contenesse almeno due punti, la loro distanza sarebbe $\leq \sqrt{2} \cdot \frac{d_{min}}{2} = \frac{d_{min}}{\sqrt{2}} < d_{min}$, assurdo perché appartengono allo stesso gruppo.

Ma ora, la distanza tra P_i e P_{i+5} (e quindi quella tra P_i e P_{i+8}) è almeno $2 \cdot \frac{d_{min}}{2} = d_{min}$, e quindi non è necessario controllarli.

Come si sorta (per y) in $O(n)$? Come in un merge sort: avrò ricorsivamente sortato le due metà, e a quel punto mi basta mergearle.

Anche in questo caso c'è una soluzione che non usa D&C.