Parlare di algoritmi

Giorgio Audrito

Online, 8 aprile 2021

Introduzione

Di cosa ci serve parlare, quando siamo interessati ad algoritmi?

1. Cosa fa un algoritmo?
2. Quanto bene lo fa?
3. Lo fa davvero?

Come farlo in modo *matematicamente preciso*?

Cosa fa un algoritmo?

Dobbiamo specificare due cose:

Cosa riceve in input

- quali oggetti e di che tipo riceve in input ⟵ segnatura
- quali condizioni devono valere su questi oggetti per consentire all'algoritmo di agire correttamente ⟵ precondizioni

Cosa produce in output

- quali oggetti e di che tipo produce in output ⟵ segnatura
- che proprietà hanno gli oggetti che sono prodotti dall'algoritmo, e in che modo sono collegati con gli input ⟵ postcondizioni

Esempi

precondizione: `v` è un vettore ordinato che contiene `e`

```
1       int ricerca_binaria(const vector<int>& v, int e); // segnatura
```

postcondizione: `return` è la posizione di `e` dentro `v` oppure -1 se non è presente

precondizione: `v` non è vuoto

```
1       void rimuovi_duplicati(vector<int>& v); // segnatura
```

postcondizione: il valore finale di `v` è una sottosequenza permutata del valore iniziale, in cui non sono presenti duplicati e in cui sono presenti tutti gli elementi che originariamente erano in `v`

Come usare queste cose?

- Per comunicare precisamente e velocemente ad altri il comportamento di una funzione che avete scritto. $\longleftarrow$ senza scriverle formalmente
- Per utilizzare correttamente una funzione. $\longleftarrow$ senza scriverle formalmente
- Per debuggare un programma. $\longleftarrow$ assert
- ~~Per dimostrare la correttezza di un programma~~ $\longleftarrow$ non alle olimpiadi.

Quanto bene lo fa?

Se abbiamo due algoritmi che risolvono lo stesso problema,
come possiamo esprimere la loro efficienza in modo da poter facilmente decidere quale dei due sia migliore in un certo contesto?

Aspetti:

- Tempo di esecuzione
- Spazio occupato in memoria
- Altro (risorse gerarchiche, consumo batteria...)

Problema:

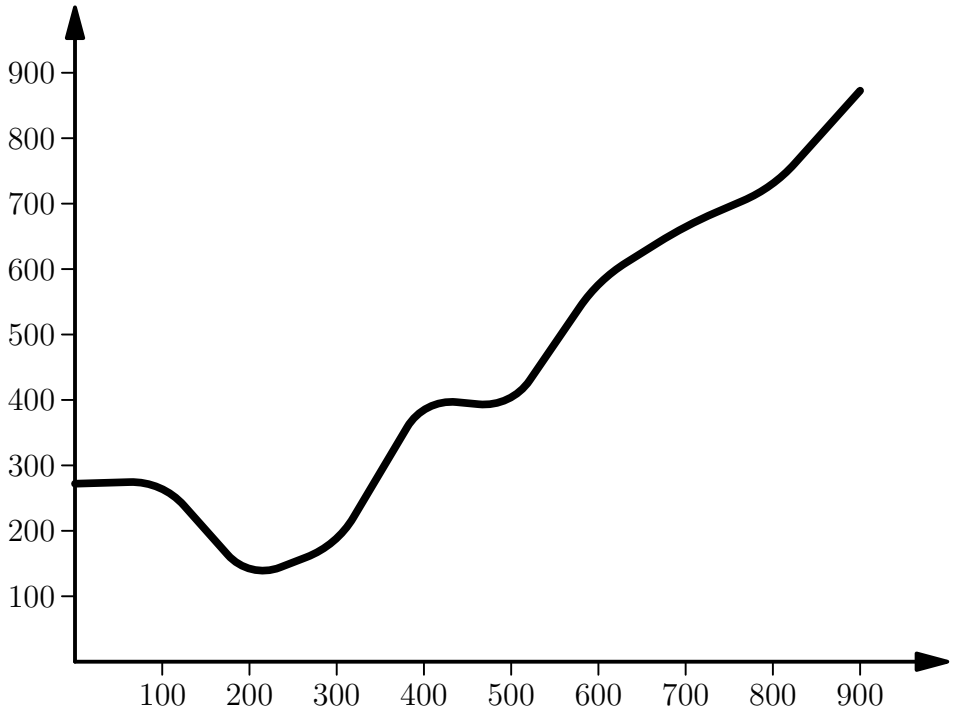
La performance di un algoritmo dipende

- dagli input (in molti modi)
- dal macchinario che lo esegue
- a volte da fattori casuali (o altro)

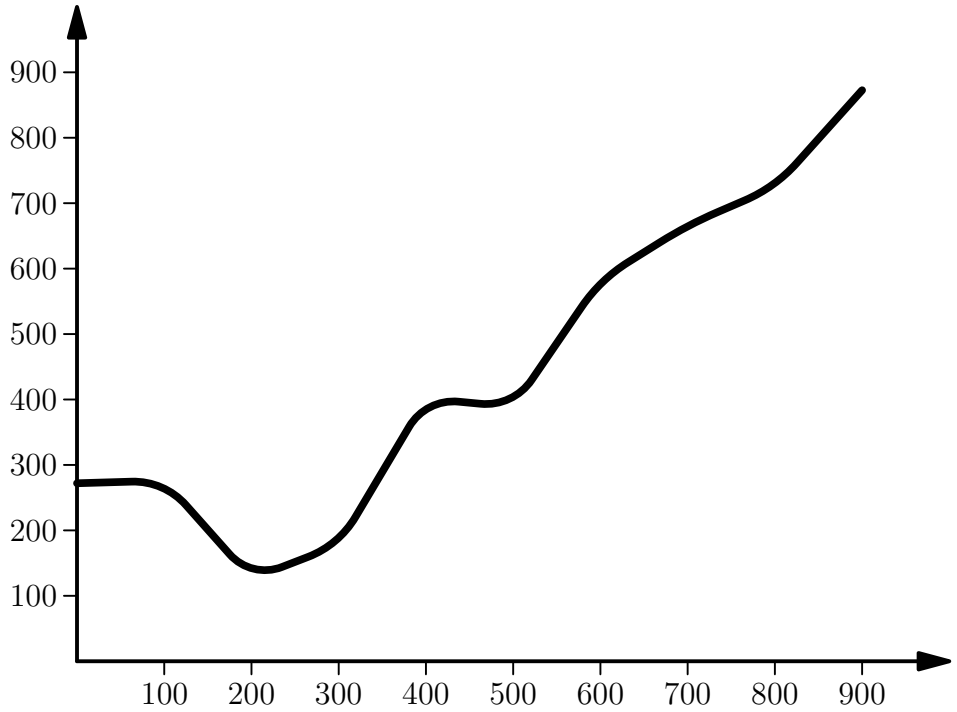
Dipendenza dagli input

Focalizziamoci su una risorsa (tempo). Dobbiamo:

- decidere quali parametri sono rilevanti negli input $\longleftarrow$ dimensione n (in bits)
- decidere quali input ci interessano a parità di parametri $\longleftarrow$ caso migliore, medio, peggiore
- ottenere un informazione *sintetica* e che *non dipenda dal calcolatore!*



Dipendenza dagli input



Proposta:

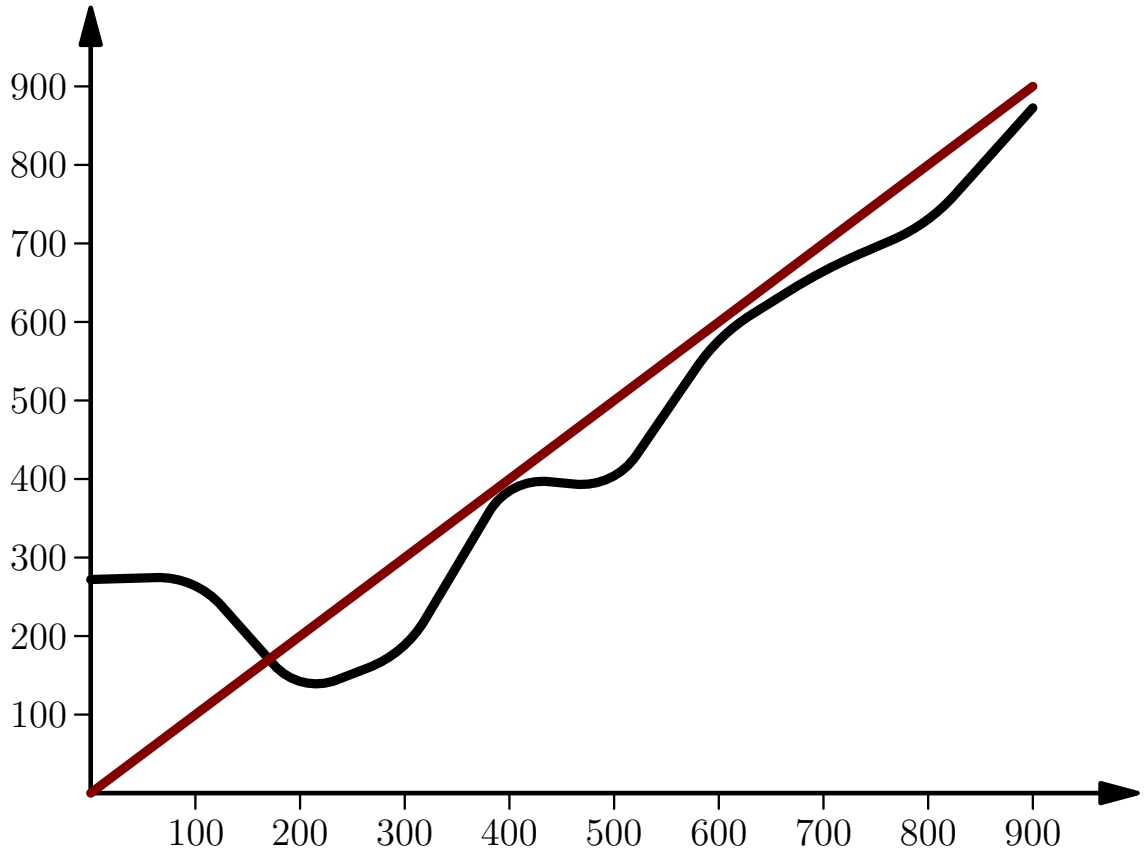
- ci focalizziamo su valori grandi dei parametri
- non consideriamo fattori moltiplicativi, perché cambiano a seconda della macchina su cui l'algoritmo è eseguito
- sostituiamo funzioni complesse (grafici) con funzioni semplici e leggibili che siano sufficientemente simili

Complessità asintotica

Come confrontare funzioni?

Diciamo che $f(n) \in \mathcal{O}(g(n))$ se f *non cresce più di* g, vale a dire se esiste una costante k e un valore n_0 tale per cui $f(n) \leq k \cdot g(n)$ per ogni $n \geq n_0$.

Per esempio, confrontiamo la funzione del grafico con $g(n) = n$.



Esempi

$$
\begin{array}{rcll}
n & \in & \mathcal{O}(n)? & \text{SI} \\
n+4 & \in & \mathcal{O}(n)? & \text{SI} \\
3n+17 & \in & \mathcal{O}(n)? & \text{SI} \\
1 & \in & \mathcal{O}(n)? & \text{SI} \\
n & \in & \mathcal{O}(1)? & \text{NO} \\
17 & \in & \mathcal{O}(1)? & \text{SI} \\
n^2+\sin(n) & \in & \mathcal{O}(n^2)? & \text{SI} \\
n^3-50n^2 & \in & \mathcal{O}(n^2)? & \text{NO} \\
2^n & \in & \mathcal{O}(4^n)? & \text{SI}
\end{array}
$$

- $\mathcal{O}(1) \longrightarrow$ costante
- $\mathcal{O}(\log n) \longrightarrow$ logaritmico
- $\mathcal{O}(n) \longrightarrow$ lineare
- $\mathcal{O}(n^2) \longrightarrow$ quadratico
- $\mathcal{O}(n^c) \longrightarrow$ polinomiale
- $\mathcal{O}(2^n) \longrightarrow$ esponenziale

Complessità asintotica

al massimo — $\mathcal{O}$

Diciamo che $f(n) \in \mathcal{O}(g(n))$ se f *non cresce più di* g, vale a dire se esiste una costante k e un valore n_0 tale per cui $f(n) \leq k \cdot g(n)$ per ogni $n \geq n_0$.

al minimo — Ω

Diciamo che $f(n) \in \Omega(g(n))$ se $g(n) \in \mathcal{O}(f(n))$.

esattamente — Θ

Diciamo che $f(n) \in \Theta(g(n))$ se $f(n) \in \mathcal{O}(g(n))$ e $f(n) \in \Omega(g(n))$.

Caso medio

Con quanto visto finora possiamo facilmente esprimere l'efficienza di un programma nel caso pessimo o nel caso migliore.

In quale senso si *considera il caso medio*?

- Assegnamo un peso a ogni istanza di una certa lunghezza, e facciamo la media pesata dei tempi/spazi impiegati. $\longleftarrow$ e se le frequenze di apparizione degli input non sono come ce le aspettavamo?
- Se l'algoritmo ha un comportamento random su ciascun input, facciamo la media dei tempi/spazi impiegati per un input nel caso peggiore $\longleftarrow$ caso medio randomizzato
- Se l'algoritmo viene eseguito più volte in sequenza, facciamo la media della sequenza di esecuzioni nel caso peggiore $\longleftarrow$ costo ammortizzato

Lo fa davvero?

Per esserne sicuri purtroppo bisogna *dimostrarlo*.

Strumenti di base

- ragionamento passo passo
- scomposizione in funzioni con pre- e post-condizioni
- e quando troviamo un ciclo?

Invarianti di ciclo

Asserzioni che sono vere all'inizio di ogni iterazione di un ciclo:

- si dimostra tramite induzione che se vale all'inizio allora vale alla fine
- devono consentire di proseguire il ragionamento passo passo
- possono essere usati in fase di debug come `assert` $\longleftarrow$ unico motivo per conoscerli alle olimpiadi