Algoritmica

Saturday, December 4, 2021 8:50 AM

D'ora in poi, algoritmo = programma in C++

FALSO!
ma è una buona approssimazione

Cosa ci interessa di un algoritmo?

- Correttezza: l'algoritmo fa quello che deve quando l'input soddisfa le "precondizioni"? ← dimostrazione matematica
- Complessità: quanto performa bene?

COMPLESSITÀ COMPUTAZIONALE

È una metrica (scala di misura) per tempo di esecuzione e memoria allocata da un algoritmo, che dipende UNICAMENTE dall'algoritmo.

La complessità è una funzione della GRANDEZZA DELL'INPUT.

Grandezza dell'input = numero di bit dell'input (nel modo più generale possibile).

Es: se l'input è un solo intero N, la gdi è $\sim \log_2(N)$

$\rightsquigarrow \Theta(N^2) \leftrightarrow O((2^{gdi})^2)$.
N

NOTAZIONI O, Ω, Θ

Sono definizioni utili per quantificare la c.c. in modo ASINTOTICO (cioè due classi di complessità coincidono a meno di moltiplicare per una costante).

O, Ω Θ sono insiemi di funzioni: (sono 1 o funzioni

$\mathcal{O}$, Ω, Θ sono insiem. di funzioni. (siano f, g funzioni $\mathbb{R}^+ \to \mathbb{R}^+$)

$\mathcal{O}$	Ω	Θ
$g \in \mathcal{O}(f)$	$g \in \Omega(f)$	$g \in \Theta(f)$
$\Longleftrightarrow$	$\Longleftrightarrow$	$\Longleftrightarrow$
$\exists\, c>0 : g(x) \le c f(x)$ $\forall x \in \mathbb{R}^+$ "upper bound"	$\exists\, c>0 \cdot g(x) \ge c f(x)$ $\forall x \in \mathbb{R}^+$ "lower bound"	$g \in \mathcal{O}(f)$ e $g \in \Omega(f)$ oppure $\exists\, c_1, c_2 > 0 \cdot$ $c_1 f(x) \le g(x) \le c_2 f(x)$ $\forall x \in \mathbb{R}^+$ "bound esatto"

Cose che valgono in generale:

- si assume che le istruzioni "base" abbiano complessità $O(1)$ (operatori base, dichiarazione/iniz. di variabili che occupano spazio costante, etc.)
- cosa complica il tutto?
  - espressioni condizionali (if)

```
if (condizione) {
    blocco1;
} else { blocco2; }
```

$O(\text{tutto}) \le \max\{O(\text{blocco1}), O(\text{blocco2})\} + O(\text{condizione})$

  - cicli (while, for, do ... while)

```
for (condizione ...) {
    blocco;
}
```

caso pessimo: $O(\text{tutto}) \le O(\text{blocco}) \cdot (\#\text{iterazioni})$

  - funzioni (ricorsione)

in generale non si può dire molto

ANALISI AMMORTIZZATA

A volte, le "regole" di sopra non producono il bound migliore possibile ma il tutto può essere minore della somma delle parti

Una COMPLESSITÀ AMMORTIZZATA è tale che la complessità di una singola operazione, nel caso pessimo, è maggiore della media su un gran numero di operazioni.

Qui bisogna ragionare in modo diverso di volta in volta.

(esempi: vector, queue, min queue, BBST, ...)

ALGORITMI RANDOMIZZATI

Sono algoritmi in cui è presente un generatore di numeri casuali (PRNG), quindi l'algoritmo dipende dal seed.

Due tipi:

- algoritmi che ritornano sempre l'ottimo, ma non è garantito che terminino;
- algoritmi che terminano sempre, ma non ritornano necessariamente l'ottimo

Quando si analizza la complessità di un algoritmo randomizzato, è importante parlare di caso medio e probabilità di avere un numero di operazioni "alto".

INDUZIONE

È uno strumento matematico che permette di provare sia la correttezza di un algoritmo, sia dei bound sulla complessità (Esercizio 5)

ENUNCIATO: Sia $P(n)$ una proprietà vera per alcuni numeri naturali. Supponiamo che, $\forall n \in \mathbb{N}$, se vale $P(k)$ $\forall\, 0 \leq k < n$, allora vale $P(n)$. Allora $P(n)$ vale per ogni $n \in \mathbb{N}$.

(Andate a vedere il problema "activity selection problem").

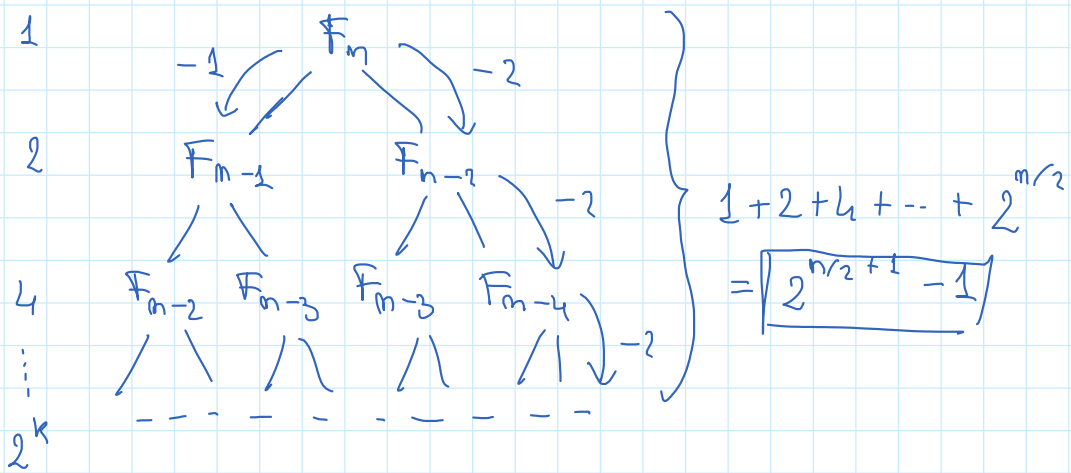
RICORSIONE

È una specie di "induzione al contrario".

Consiste nel chiamare una funzione dall'interno della stessa.

Esempio classico: Fibonacci.

$F_0 = F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n \quad \forall\, n \geq 0.$

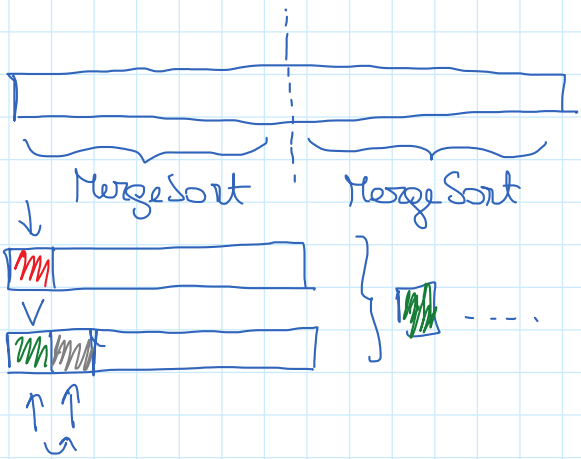


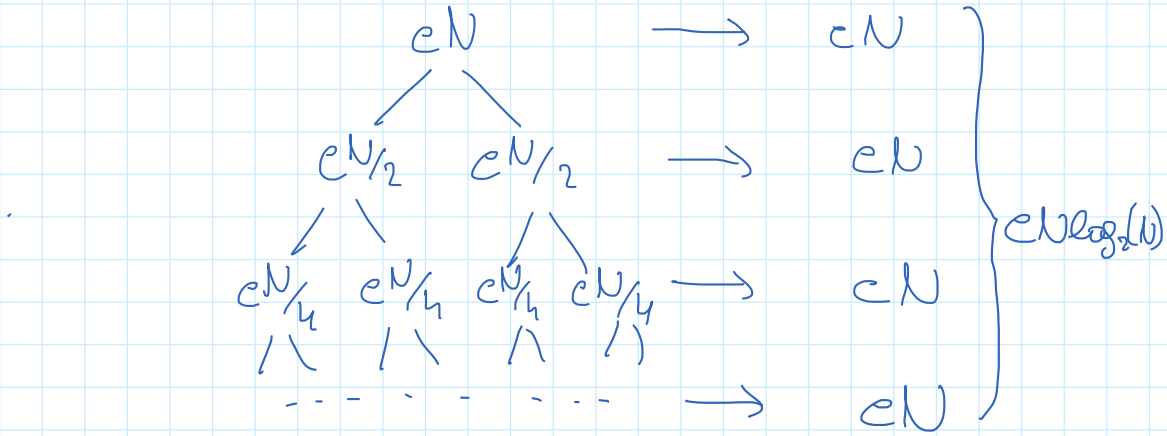
$\Rightarrow$ La complessità è esponenziale.

ALGORITMI DI ORDINAMENTO

- Quadratici → BubbleSort, InsertionSort, SelectionSort.
- Pseudolineari $(N\log(N))$ → HeapSort, QuickSort, MergeSort.
  ↓
  sort() del C++

MERGESORT





TEOREMA MASTER (CASI SPECIALI)

$T(N) = O(N) + 2T(\frac{N}{2}) = O(N\log N)$

- $T(N) = O(g(N)) + T(\alpha N)$ dove:

  $g(\alpha N) \leq \beta\, g(N)$, $\beta < 1$
  g è una funzione crescente

  $0 < \alpha < 1$

$\Rightarrow T(N) = O(g(N))$

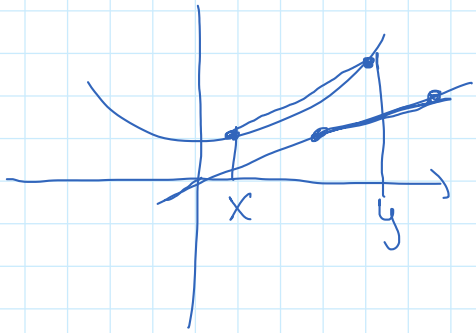
- $T(N) = O(g(N)) + T(\alpha_1 N) + T(\alpha_2 N) + \dots + T(\alpha_k N)$

dove: $f(N)$ è crescente e CONVESSA,

$K \geq 2$, $\alpha_1 + \dots + \alpha_K \leq 1$, $\alpha_i > 0$

$\Rightarrow T(N) = O(f(N) \log N)$

Cos'è la CONVESSITÀ?

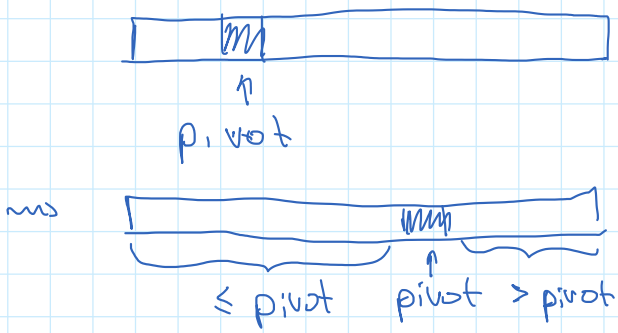


f convessa $\iff \forall\, 0 < \alpha < 1,\ f(\alpha x + (1-\alpha) y) \leq \alpha f(x) + (1-\alpha) f(y)$

$f(N) = N, N^2, N^\alpha$ con $\alpha \geq 1$ sono convesse

$f(N) = \sqrt{N}, \log N, N^\alpha$ con $\alpha < 1$ NON sono convesse.

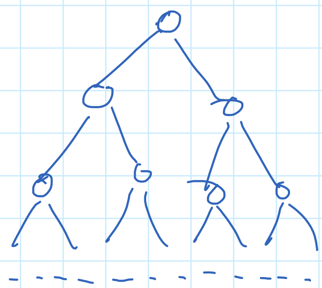
QUICKSORT



Problema: se il pivot è "molto piccolo" o "molto grande", la suddivisione è sbilanciata, e la complessità può diventare quadratica

Per sistemarlo, si può implementare il Quicksort randomizzato, in cui il pivot viene scelto a caso.

…randomizzato ma con il pivot viene scelto a caso.



Questo algoritmo è $O(N \log N)$ quasi certamente.

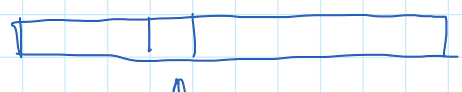
SELEZIONE: QUICKSELECT

Dato un array, trovare il K-esimo.

Si può risolvere ordinando l'array ⇝ $O(N \log N)$

Si può fare di meglio ($O(N)$).

Stesso algoritmo del QuickSort, ma ricorro solo sulla parte che contiene il K-esimo.



- $i = K$ ⇝ ho finito
- $i > K$ ⇝ parte sinistra
- $i < K$ ⇝ parte destra

$T(N) = O(N) + T(\alpha N)$ ⇝ $O(N)$

Nella STL del C++ esiste la funzione

nth_element(begin, K, end) che implementa QuickSelect.

RICERCA

Dato un array, dire se contiene X

In generale, non si può fare meglio di lineare.

In generale, non si può fare meglio di lineare. Ma se l'array è ordinato (in ordine crescente o decrescente), si può fare in $O(\log N)$.

RICERCA BINARIA (BINARY SEARCH):

- guardo l'elemento centrale: se è x, ho finito
- altrimenti:
  - se è > x, vado a sinistra;
  - se è < x, vado a destra.
- se l'intervallo diventa vuoto, x non c'è

Nella STL avete le funzioni:

lower_bound(begin, end, x); // iteratore al primo elemento $\geq x$

upper_bound(begin, end, x); // iteratore al primo elemento $> x$

RICERCA BINARIA SULLA RISPOSTA

Volete conoscere il minimo o massimo intero che soddisfa una certa proprietà.

Supponiamo che esista m tale che la proprietà è verificata $\forall x \geq m$ e non è verificata $\forall x < m$.

Se 0 = non verificata, 1 = verificata:

0 0 0 -- 0 1 1 -- 1 1 ← m.m.mo

↑ m

ESPONENZIAZIONE VELOCE (FAST EXP)

Problema: calcolare a^b $(a, b > 0)$

Si può fare meglio di lineare? Sì: $O(\log(b))$

$a^{2k} = (a^k)^2$

$a^{2k+1} = (a^k)^2 \cdot a$

Divido b per 2 (interamente), calcolo $a^{\lfloor \frac{b}{2} \rfloor} = x$,

ritorno: $\begin{cases} x \cdot x & \text{se } b \text{ è pari} \\ x \cdot x \cdot a & \text{se } b \text{ è dispari.} \end{cases}$

Passo base: $b = 0$ $(a^0 = 1)$

IMPORTANTE: a^b esplode se $a \geq 2$ e $b \gg 1$,

quindi di solito si chiede la risposta

mod m.