Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Syllabus e Introduzione

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

Di cosa tratta questo corso

- **Algoritmi**

descriveremo alcuni algoritmi “classici” per risolvere problemi di base (ricerca e ordinamento)

- **Strutture dati**

Esploreremo le strutture dati più adatte da usare

- **Efficienza**

Valuteremo l’efficienza, calcolandone il costo computazionale.

- **Problem solving**

P.S. gli algoritmi verranno descritti tramite pseudocodice, per poterne dare una versione compatta senza perdersi in dettagli implementativi…

Cos'è un algoritmo?

- Un algoritmo è una sequenza di comandi "**elementari**" ed "**univoci**" che terminano in un tempo finito ed operano su strutture dati.

possono essere interpretati in un solo modo

non possono essere scomposti in comandi più semplici

Allora:

se l'algoritmo è ben fatto, chi lo esegue NON ha bisogno di pensare, ma solo di essere preciso come un …. calcolatore!

ATTENZIONE: il calcolatore non pensa! Esegue gli algoritmi pensati da una persona

QUINDI: se il risultato finale è sbagliato, non ha sbagliato il calcolatore ma la persona che ha scritto l'algoritmo

Strutture dati

Un algoritmo è una sequenza di comandi "elementari" ed "univoci" che terminano in un tempo finito ed operano su strutture dati.

Per risolvere problemi avremo bisogno di usare dati e quindi useremo delle STRUTTURE DATI, strumenti per memorizzare e organizzare i dati e semplificarne l'accesso e la modifica.

Non esiste una struttura dati che vada bene per ogni problema, quindi dobbiamo conoscere proprietà, vantaggi e svantaggi delle principali strutture dati.

Il progetto o la scelta della struttura dati da adottare nella soluzione di un problema è un aspetto fondamentale per la risoluzione del problema stesso, al pari del progetto dell'algoritmo.

Perciò, gli algoritmi e le strutture dati fondamentali vengono sempre **studiati e illustrati assieme**.

Esempi: vettore, lista, coda, pila…

Efficienza

Un algoritmo è una sequenza di comandi "elementari" ed "univoci" che terminano in un tempo finito ed operano su strutture dati.

Affinché un algoritmo sia utilizzabile, deve concludersi e produrre il suo output entro un tempo "ragionevole".

Un aspetto fondamentale che va affrontato nello studio degli algoritmi è la loro **efficienza**, cioè la ***quantificazione delle loro esigenze in termini di tempo e di spazio***, ossia tempo di esecuzione e quantità di memoria richiesta.

Questo perché:

- I calcolatori sono molto veloci, ma non infinitamente veloci;
- La memoria è economica e abbondante, ma non è né gratuita né illimitata.

Nel corso illustreremo il concetto di **costo computazionale** degli algoritmi, in termini di numero di operazioni elementari e quantità di spazio di memoria necessari in funzione della dimensione dell'input.

Per comprendere l'importanza del costo computazionale di un algoritmo, in particolare per quanto riguarda il tempo di esecuzione richiesto, facciamo un semplice esempio:

Consideriamo due calcolatori, V (veloce) e L (lento):
calcolatore V (veloce): $10^9 istruzioni/secondo$
calcolatore L (lento): $10^7 istruzioni/secondo$

Consideriamo due algoritmi di ordinamento per una lista di n=10^6 elementi
Algoritmo IS: $2n^2$ istruzioni
Algoritmo MS: $50n \log n$ istruzioni

Domanda: Eseguiamo IS sul calcolatore V ed MS sul calcolatore L. La maggiore velocità del calcolatore V riesce a bilanciare la minore efficienza dell'algoritmo IS?

Risposta: No…

Infatti:

$$V(IS) = \frac{2 * (10^6)^2 \ istruz.}{10^9 \ istruz./sec.} = 2000 \ secondi = 33 \ minuti$$

$$L(MS) = \frac{50 * 10^6 \log 10^6 \ istruz.}{10^7 \ istruz./sec.} = 100 \ secondi = 1.5 \ minuti$$

Se la lista contiene $n = 10^7$ elementi il divario aumenta:
V(IS)=2 giorni e L(MS)=20 minuti

Questo ci fa capire come, indipendentemente dall'aumento di velocità dei calcolatori prodotto dagli avanzamenti tecnologici, l'efficienza degli algoritmi sia un fattore di importanza cruciale.

Random Access Machine

- Per studiare l'efficienza di un algoritmo dobbiamo ANALIZZARLO, cioè prevedere le risorse che esso richiede per la sua esecuzione, **senza che tale analisi sia influenzata da una specifica tecnologia** che, inevitabilmente, col tempo diviene superata.
- Per fare questo, dobbiamo avere un modello astratto di calcolatore, che sia indipendente dalla tecnologia usata.
- In questo corso consideriamo un modello astratto di calcolo detto RANDOM ACCESS MACHINE (RAM).
- La RAM è quindi una ***macchina astratta***, la cui validità e potenza concettuale risiede nel fatto che non diventa obsoleta con il progredire della tecnologia.

Nel modello RAM:

- esiste un **singolo processore**, che esegue le operazioni sequenzialmente.

- esistono delle **operazioni elementari**, l'esecuzione di ciascuna delle quali richiede per definizione un ***tempo costante***. (Es.: operazioni aritmetiche, letture, scritture, salto condizionato, ecc.)

Misure di costo

- Faremo l'ipotesi che ogni operazione aritmetica o di confronto viene eseguita in tempo costante, indipendentemente dalla dimensione degli operandi.
- Questo criterio è detto MISURA DI COSTO UNIFORME.
- Sebbene utile, questo è un modello di costo non sempre realistico perché i dati potrebbero non essere rappresentabili in una sola parola di memoria, e quindi anche le operazioni aritmetiche non richiederebbero, in realtà, tempo costante.

Criterio della misura di costo uniforme

```
def esempio(n):
    x = 1
    for i in range(n):
        x = 2*x
    print(x)
```

(ciclo, reiterato *n* volte, che calcola il valore 2^n)

Il tempo di esecuzione totale è ***proporzionale ad n***:

- si tratta di un **ciclo eseguito *n* volte**;
- ad ogni iterazione del ciclo si compiono **due operazioni**, ciascuna delle quali ha costo unitario:
  1. l'incremento del contatore
  2. il calcolo del nuovo valore di *x*.

Pseudocodice

Concludiamo con l'osservare che, affinché ogni esecutore sia in grado di comprendere un algoritmo, è necessario utilizzare una descrizione:

- il più formale possibile
- indipendente dal linguaggio che si intende usare

Pseudocodice

Problem solving

Il ***problem solving*** è un'attività che ha lo scopo di raggiungere una soluzione a partire da una situazione iniziale.

E' quindi un'attività creativa, di natura essenzialmente progettuale, ed in questo risiede la sua difficoltà.

Approccio al problem solving:

- ***analisi del problema**:* lettura approfondita della situazione iniziale, comprensione ed identificazione del problema;
- ***esplorazione degli approcci possibili**:* identificazione delle metodologie di soluzione tra i metodi noti;
- ***selezione di un approccio**:* scelta dell'approccio migliore;
- ***definizione dell'algoritmo risolutivo**:* identificazione dei dati e progettazione della sequenza di passi elementari da applicare su di essi;
- ***riflessione critica**:* a problema risolto, ripensamento delle fasi della soluzione proposta per identificare eventuali criticità e possibili migliorie.

Quali problemi?

In questo corso restringiamo la nostra attenzione ai ***problemi computazionali***, problemi cioè che richiedono di descrivere in modo automatico una specifica relazione tra un insieme di valori in ***input*** e il corrispondente insieme di valori in ***output***.

Un algoritmo è ***corretto*** se, ***per ogni istanza di un problema computazionale, termina producendo l'output corretto***.

In tal caso diremo che ***l'algoritmo risolve il problema***.

Esempio di problema computazionale:

ordinare n numeri dal più piccolo al più grande

Input: (anche detto istanza del problema)
sequenza di *n* numeri $a_1, a_2, \ldots, a_n$

Output:
permutazione $a'_1, a'_2, \ldots, a'_n$ della sequenza di input con $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Dettagli per studiare

Pagina web del corso

Alla pagina:

https://twiki.di.uniroma1.it/twiki/view/Intro_algo/PZ/WebHome

troverete:

- Il programma del corso
- Il diario delle lezioni
- Le modalità d'esame
- Un elenco di libri di testo utili
- Delle dispense e degli esercizi svolti

Libri di testo

Un qualunque manuale di algoritmi e strutture dati fondamentali va bene, ma se dovete comprarne uno…

- Cormen, Leiserson, Rivest, Stein: Introduzione agli algoritmi e strutture dati, Mc Graw Hill

oppure

- Demetrescu, Finocchi, Italiano: Algoritmi e strutture dati, Mc Graw Hill

oppure

- R. Kleinberg ed E. Tardos. Algorithm Design. Addison Wesley.

Suggerimenti

Non basta essere solo “spettatori”:

- Alla fine di (quasi) ogni lezione troverete degli esercizi che vengono proposti.
- Provate a risolverli e, se non ci riuscite, approfondite ulteriormente l’argomento trattato.

Corso di laurea in Informatica
Introduzione agli Algoritmi
Lezioni a distanza

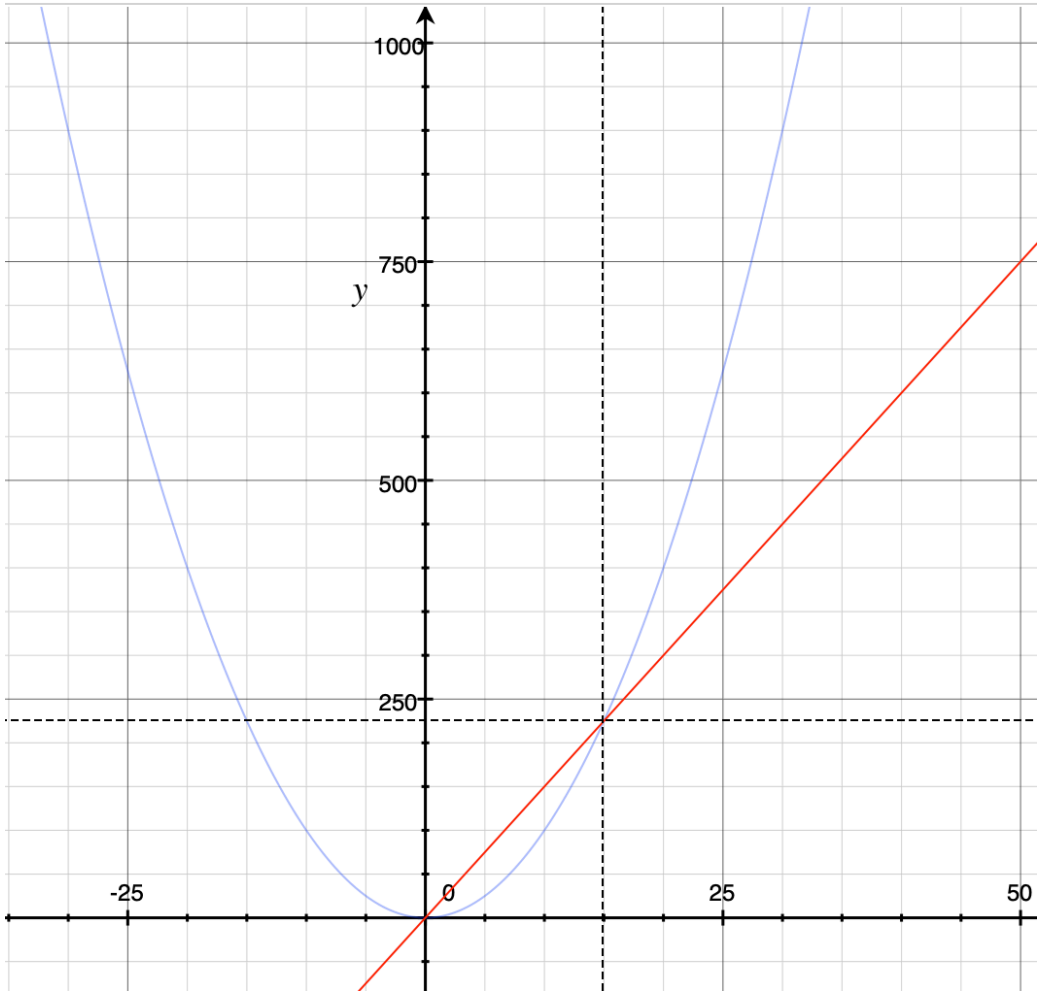
la notazione asintotica

SAPIENZA
UNIVERSITÀ DI ROMA

In matematica la notazione asintotica permette di confrontare il tasso di crescita (comportamento asintotico) di una funzione nei confronti di un'altra.

$g(n) = n^2$

$f(n) = 15n$



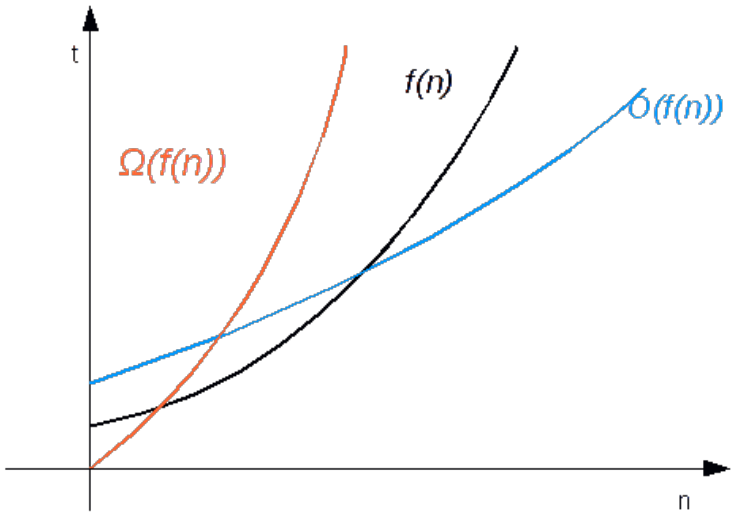
In informatica il calcolo asintotico è utilizzato per analizzare la complessità di un algoritmo.
In particolar modo, per stimare quanto aumenta il tempo al crescere della dimensione n dell'input.

Vedremo tre diverse notazioni asintotiche:

Notazione asintotica O. La notazione *O grande* è il limite superiore asintotico

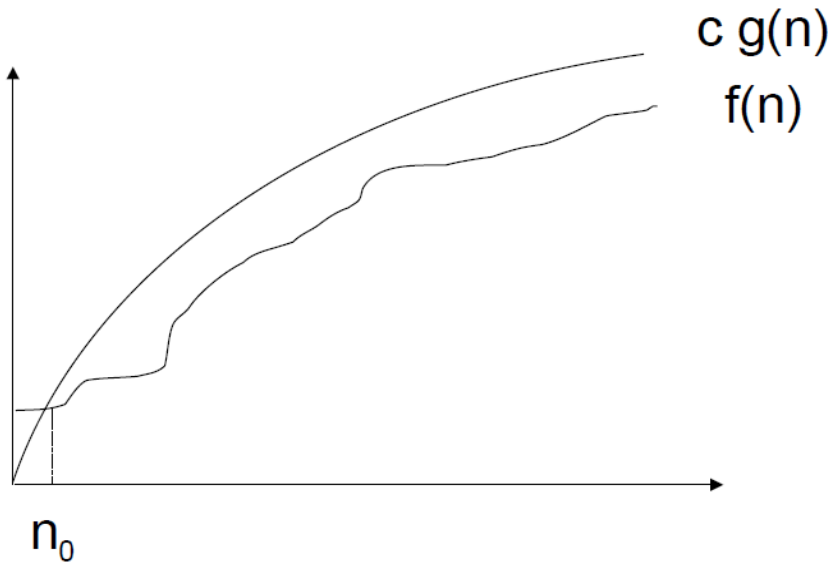
Notazione asintotica Ω**.** La notazione *Omega* è il limite inferiore asintotico

Notazione asintotica Θ**.** La notazione *Theta* è il limite asintotico stretto



Il limite superiore asintotico e la notazione *O grande*

$$O(g(n)) = \{ f(n) : \exists c > 0 \, e \, n_0 \geq 0 \;\; t.c. \; f(n) \leq c \cdot g(n) \; \forall n \geq n_0 \}$$



- $O(g(n))$ **è un insieme di funzioni**

- Con abuso di notazione spesso si scrive $f(n) = O(g(n))$ invece che $f(n) \in O(g(n))$

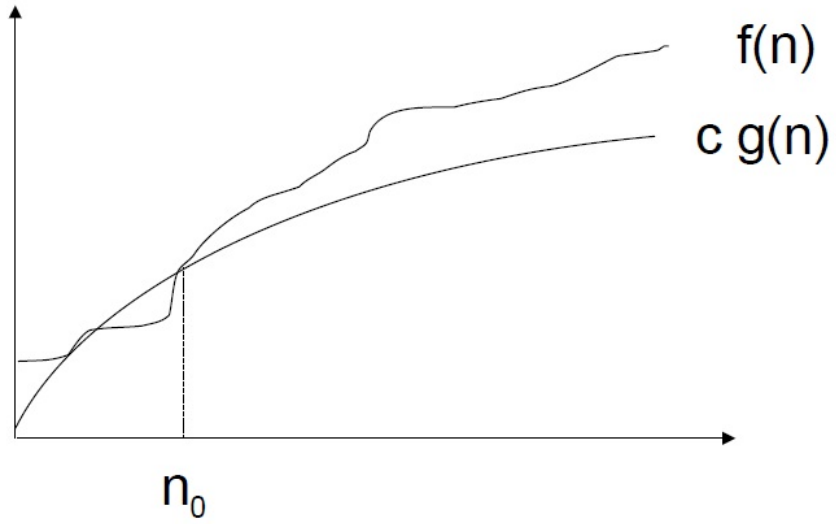
Banalmente: $n^2 = O(n^2)$ Basta prendere $c = 1$ e $n_0 = 0$ e si ha $n^2 \leq cn^2 \; \forall n \geq n_0$

Banalmente: $5n^2 = O(n^2)$ Basta prendere $c = 5$ e $n_0 = 0$ e si ha $5n^2 \leq cn^2 \; \forall n \geq n_0$

Posso astrarre dalle costanti moltiplicative

Il limite inferiore asintotico e la notazione *Omega*

$$\Omega(g(n)) = \{ f(n) : \exists c > 0 \, e \, n_0 \geq 0 \;\; t.c. \; f(n) \geq c \cdot g(n) \; \forall n \geq n_0 \}$$



- $\Omega(g(n))$ **è un insieme di funzioni**

- **Con abuso di notazione spesso si scrive** $f(n) = \Omega(g(n))$ **invece che** $f(n) \in \Omega(g(n))$

Banalmente: $n^2 = \Omega(n^2)$ **Basta prendere** $c = 1$ **e** $n_0 = 0$ **e si ha** $n^2 \geq cn^2 \; \forall n \geq n_0$

Banalmente: $5n^2 = \Omega(n^2)$ **Basta prendere** $c = 5$ **e** $n_0 = 0$ **e si ha** $5n^2 \geq cn^2 \; \forall n \geq n_0$

Posso astrarre dalle costanti moltiplicative

Il limite stretto asintotico e la notazione *Theta*

$$\Theta(g(n)) = \{ f(n) : \exists c_1 > 0, c_2 > 0 \, e \, n_0 \geq 0 \;\; t.c. \; c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n) \; \forall n \geq n_0 \}$$

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

$$1) \; f(n) = O(g(n)) \implies \exists c_1 > 0 \; e \; n_1 \geq 0 \; t.c. f(n) \leq c_1 \cdot g(n) \; \forall n \geq n_1$$

$$2) \; f(n) = \Omega(g(n)) \implies \exists c_2 > 0 \; e \; n_2 \geq 0 \; t.c. f(n) \geq c_2 \cdot g(n) \; \forall n \geq n_2$$

$$1) + 2) \implies$$

$$\exists c_1 > 0, c_2 > 0 \, e \, n_0 \geq 0 \;\; t.c. \; c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n) \; \forall n \geq n_0 = \max\{n_1, n_2\}$$

Ad esempio:

$n^2 = O(n^2)$ **e** $n^2 = \Omega(n^2)$ $\implies$ $n^2 = \Theta(n^2)$

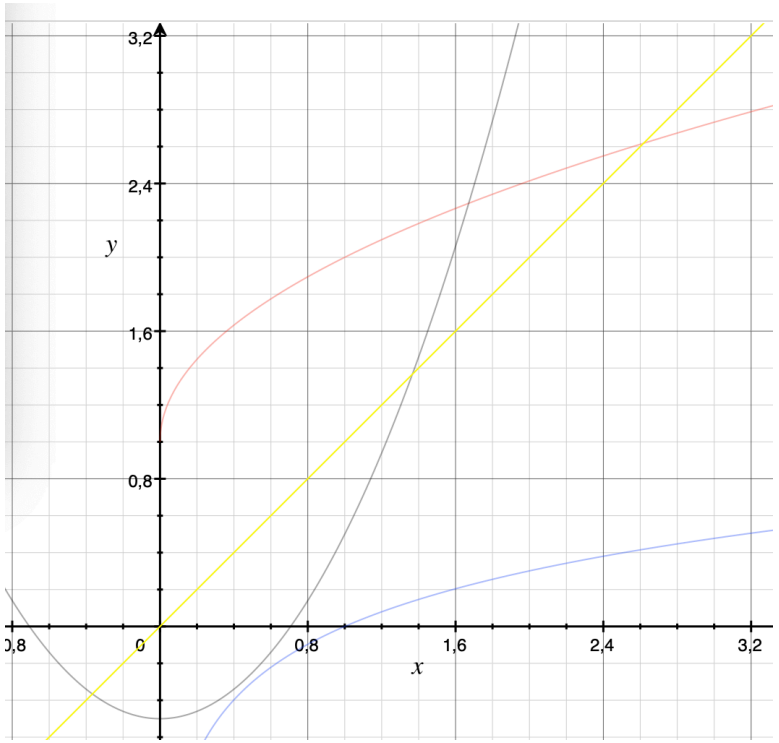
$5n^2 - 7 = O(n^2)$ **e** $5n^2 - 7 = \Omega(n^2)$ $\implies$ $5n^2 - 7 = \Theta(n^2)$

$f_1(n) = \log n$

$f_2(n) = \sqrt{n} + 1$

$f_3(n) = n$

$f_4(n) = n^2 - 1/2$



$\log n = O\left(\sqrt{n}\right)$ $\qquad$ $\sqrt{n} = O(n)$ $\qquad$ $n = O(n^2)$

più in generale:

$\log^c n = O\left(n^{\frac{1}{c'}}\right)$ per qualunque $c, c' > 0$

$n^{\frac{1}{c}} = O\left(n^{c'}\right)$ per qualunque $c, c' > 1$

$n^c = O\left(c'^n\right)$ per qualunque $c, c' > 1$

Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

ESERCIZIO:

- Classifica le seguenti funzioni per ordine di crescita, vale a dire trova un ordinamento $g_1, g_2, \ldots$ delle funzioni che soddisfi $g_1 = \Omega(g_2),\ g_2 = \Omega(g_3)\ \ldots\ .$

- Partiziona la lista in classi di equivalenza di modo che le funzioni $f(n)$ e $g(n)$ sono nella stessa classe se e solo se $f(n) = \Theta(g(n))$.

$$\left(\sqrt{2}\right)^{\lg n} \quad n^2 \quad n! \quad (\lg n)! \quad \lg n^n \quad n^3 \quad \lg^2 n$$

$$\lg(n!) \quad 2^{2^n} \quad n^{\frac{1}{\lg n}} \quad \lg\lg n \quad n \cdot 2^n \quad n^{\lg\lg n} \quad 5$$

$$\lg n \quad 2^{\lg n} \quad 4^{\lg n} \quad (\lg n)^{\lg n} \quad 2^{2^{n+1}} \quad (n+1)! \quad e^n$$

$$n \cdot \lg n \quad 2^n \quad n \quad 2^{\sqrt{2\lg n}} \quad \left(\frac{3}{2}\right)^n$$