Capitolo 2: Notazioni, convenzioni, richiami

§.1 Definizione informale di algoritmo

Informalmente un algoritmo è una ben definita procedura di calcolo che riceve una o più informazioni come input e produce una o più informazioni come output.
Un algoritmo può essere inoltre visto come uno strumento di risoluzione di uno specifico problema computazionale.
L'enunciato del problema specifica in termini generali la relazione desiderata tra input e output.
L'algoritmo descrive invece una specifica procedura computazionale finalizzata all'individuazione della relazione tra input e output.
Chiameremo istanza di un problema uno specifico input, che soddisfi le eventuali condizioni imposte dal problema, necessario al calcolo della soluzione del problema.
Diremo che un algoritmo è corretto se per ogni stanza di input esso terminerà col corretto output. Diremo, inoltre, che un algoritmo corretto risolvere un dato problema computazionale.
Un algoritmo può essere specificato come programma o come struttura hardware.
Diremo che due algoritmi sono equivalenti quando, a parità di input, producono il medesimo output. In sintesi:

- Algoritmo: **sequenza finita** di operazioni elementari che trasforma 1 o + valori in ingresso (input) in 1 o + valori in uscita (output).
- Algoritmo A; $f_A(x)$ funzione che associa ad ogni input x di A la corrispondente uscita $f_A(x)$; questa corrispondenza rappresenta il problema risolto dall'algoritmo.

§.2 Definizione formale di algoritmo

- f: D_I -> D_S (insieme delle Istanze , insieme delle Soluzioni); x appartiene a D_I; f(x) appartiene a D_S.
- Un algoritmo A **risolve il problema** f se $f(x) = f_A(x)$ per ogni istanza di x.

§.3 Analizzare un algoritmo

Analizzare un algoritmo vuol dire sostanzialmente fare una previsione sulle risorse che saranno necessarie per eseguire l'algoritmo.
La quantità che solitamente si vuole misurare è il tempo di elaborazione e ciò al fine di individuare tra algoritmi equivalenti, quelli più efficienti.
L'analisi di un algoritmo non può prescindere dall'architettura hardware utilizzata. In tutta la trattazione assumeremo di avere a disposizione un elaboratore dotato di un unico processore quindi sarà escluso il parallelismo computazionale.
Assumeremo invece che il nostro elaboratore di riferimento abbia la capacità di svolgere le comuni istruzioni aritmetiche (somma, sottrazione, moltiplicazione, divisione, resto, parte intera), di movimento dei dati (load, store, copy) e di controllo (conditional and unconditional branch, subroutine call and return).
Ognuna di queste istruzioni richiede un tempo di elaborazione unitario costante.

Oltre a quelle appena citate, vi sono ovviamente altre istruzioni disponibili e per le quali bisognerà porre attenzione nel computo dei tempi di elaborazione.
Ad esempio l'elevamento a potenza non è in generale un'istruzione eseguita in un tempo unitario infatti per eseguire x^y ci vogliono parecchie istruzioni se $x, y \in \mathbb{R}$.
Ma in alcuni casi l'elevamento a potenza è un'istruzione unitaria: vedremo, ad esempio, che 2^k in binario corrisponde a "shiftare a sinistra" di k posizioni i bit di input.
Dunque analizzare un algoritmo può non essere così immediato. Gli strumenti matematici necessari allo scopo includono il comportamento asintotico delle funzioni, il calcolo combinatorio, la teoria delle probabilità e l'abilità di identificare gli elementi significativi in una formula.

§.4 Studio di algoritmi:

- **Sintesi** (progetto): dato un problema f, costruire A per risolvere f, cioè tale che f = f_A.
- **Analisi**: dati A e f, dimostrare che A risolve f, cioè che f = f_A (correttezza) e valutazione risorse consumate da A (complessità concreta).
- **Classificazione** (complessità strutturale) data una quantità T di risorse trovare la classe di problemi risolubili da algoritmi che usano al più tale quantità (P, NP, NP-completi).

Valutazione del **consumo di risorse** (spazio, tempo, numero e varietà dei dispositivi di calcolo), che può pregiudicare la possibilità di utilizzo dell'algoritmo stesso.
Si fissa un **modello di calcolo** preciso e si definisce in base ad esso la nozione di algoritmo e di risorse consumate (es: macchina RAM).

§.5 Complessità

Nel seguito indicheremo i tempi di elaborazione (running time) di un algoritmo A in funzione della dimensione n dell'input con $T_A(n)$. Poichè possiamo ipotizzare che per eseguire una singola istruzione elementare sia necessario un tempo unitario, confonderemo i tempi di elaborazione col numero di operazioni elementari eseguite.

Chiameremo, inoltre, $S_A(n)$ il numero di celle di memoria utilizzate anch'esso in funzione della dimensione dell'input.

E' evidente che entrambe sono funzioni intere di variabile intera.

E' preferibile studiare la dipendenza dalle dimensioni $n = |x|$ dell'input piuttosto che dall'input x stesso perchè, ovviamente, input differenti possono avere la medesima dimensione.
La funzione **dimensione** associa ad ogni ingresso un numero naturale che rappresenta **intuitivamente** la quantità di informazione contenuta nel dato considerato. Ad esempio:

- se $x \in \mathbb{R} \Rightarrow n = |x| = 1 + \lfloor \log_2 x \rfloor$
- se $x \in \mathbb{R}^k \Rightarrow n = k$
- se x è un grafo $\Rightarrow n$ coincide col numero dei suoi nodi

In generale però, a parità di dimensione dell'input, il medesimo algoritmo richiederà tempi di elaborazione differenti. Si pensi, ad esempio, a un algoritmo di ordinamento rispetto a un vettore di elementi disordinati e a un vettore di elementi già ordinati.

Come definire allora la complessità in funzione della sola dimensione dell'input?
Si può considerare

- **il caso peggiore** $T_A^P(n): \mathbb{N} \to \mathbb{N}$ tale che $\forall n \in \mathbb{N} \quad T_A^P(n) = \max_x \{T_A(x), |x| = n\}$
- **il caso medio** $T_A^M(n): \mathbb{N} \to \mathbb{R}$ tale che $\forall n \in \mathbb{N} \quad T_A^M(n) = \frac{1}{|I_n|} \sum_{|x|=n} T_A(x)$ dove I_n è l'insieme delle possibili istanze x di dimensione n accettabili in input da A.

Spesso il caso peggiore dà una valutazione **troppo pessimistica** mentre il caso medio assume una **distribuzione uniforme** sulle istanze, ipotesi discutibile in molte applicazioni.

Riportiamo di seguito la tabella con i tempi di elaborazione, in funzione della complessità dell'algoritmo, per alcuni valori della dimensione dell'input.
Si assuma che ogni operazione elementare richieda un tempo di elaborazione pari a un microsecondo.

Complessità	$n=10$	$n=20$	$n=50$	$n=100$	$n=10^3$	$n=10^4$	$n=10^5$	$n=10^6$
n	$10ms$	$20ms$	$50ms$	$0,1ms$	$1ms$	$10ms$	$0,1s$	$1s$
$n\log_2 n$	$33,20ms$	$86,40ms$	$0,28ms$	$0,6ms$	$9,9ms$	$0,1s$	$1,6s$	$19,9s$
n^2	$0,1ms$	$0,4ms$	$2,5ms$	$10ms$	$1s$	$100s$	$2,7h$	$11,5g$
n^3	$1ms$	$8ms$	$125ms$	$1s$	$16,6m$	$11,5g$	$31,7a$	$\sim 300c$
2^n	$1ms$	$1s$	$35,7a$	$\sim 10^{14}c$	∞	∞	∞	∞
3^n	$59ms$	$\sim 1h$	$\sim 10^8 c$	∞	∞	∞	∞	∞

In questa seconda tabella riassumiamo invece le dimensioni massime dell'input elaborabile in un minuto in funzione della complessità dell'algoritmo.

Complessità	Input max
n	6×10^7
$n\log_2 n$	28×10^5
n^2	77×10^2
n^3	390
2^n	25

Si potrebbe pensare che le valutazioni generali sopra riportate dipendano dall'attuale livello tecnologico e siano destinate ad essere superate con l'avvento di una tecnologia più sofisticata che permetta di introdurre strumenti di calcolo sensibilmente più veloci. Questa opinione può essere confutata facilmente considerando l'incremento, dovuto ad

una maggior rapidità nell'esecuzione delle operazioni fondamentali, delle dimensioni massime di input trattabili in un tempo fissato.

Supponiamo di disporre di due calcolatori che chiamiamo rispettivamente C_1 e C_2 e assumiamo che C_2 sia M volte più veloce di C_1 dove $M > 1$. Quindi se $C1$ esegue un certo calcolo in un tempo t, allora C_2 eseguirà lo stesso calcolo in un tempo t / M.

Nella seguente tabella si mostra come cresce, passando da C_1 a C_2, la massima dimensione di ingresso trattabile in un tempo fissato da algoritmi dotati di differenti complessità temporali.

Complessità	input max C_1	input max C_2
n	d_1	$M \cdot d_1$
$n \log_2 n$	d_2	$\sim M \cdot d_2$
n^2	d_3	$\sqrt{M} \cdot d_3$
2^n	d_4	$d_4 + \log_2 M$

Come si evince dalla tabella, algoritmi lineari o quasi lineari traggono pieno vantaggio dal passaggio alla tecnologia più potente; negli algoritmi polinomiali il vantaggio è ancora evidente ma smorzato mentre negli algoritmi esponenziali il cambiamento tecnologico è quasi ininfluente.

§.6 Confronto locale. Simboli di Landau

La notazione che utilizzeremo per descrivere il comportamento locale di una funzione sarà quella classicamente presentata nei corsi di analisi e che viene attribuita a Edmund Landau (non il Lev Landau fisico). In realtà E. Landau usò tale notazione nei suoi lavori riconoscendone però l'effettiva paternità a Paul Bachmann.

Nello studio del comportamento di una funzione vicino a un punto x_0 di accumulazione, dopo aver visto se la funzione ammette limite (finito, nullo o infinito) per $x \to x_0$, può interessare come la funzione tende a tale valore.
Inoltre, se si studiano due funzioni f e g, entrambe definite in un intorno di x_0 escluso al più il punto stesso, ha interesse studiare se c'è relazione tra i loro limiti (ammesso che esistano), per $x \to x_0$. Le definizioni che seguono precisano proprio questa indagine.

Uguaglianza asintotica
Diremo che $f(x)$ è asintotico a $g(x)$ per $x \to x_0$ e scriveremo $f \sim g$ in $U(x_0)$ se:

(2) $$\lim_{x \to x_0} \frac{f(x)}{g(x)} = 1$$

Esempi:

1. $3x+\sqrt{x}\sim\sqrt{x}$ in $U(0)$
2. $x^3\sin x\sim x^3$ in $U(\pm\infty)$
3. $\log x\sim(x-1)$ in $U(1)$

Ugual ordine di grandezza

Diremo che $f(x)$ è dello stesso ordine di grandezza di $g(x)$ per $x\to x_0$ e scriveremo $f=\Theta(g)$ in $U(x_0)$ se:

(3) $$\lim_{x\to x_0}\frac{f(x)}{g(x)}=l\neq 0$$

oppure, in modo equivalente, se $\exists h,k>0: hg(x)\leq|f(x)|\leq kg(x)$ in $U(x_0)$

Esempi:

1. $1-\cos x=\Theta\left(x^2\right)$ in $U(0)$ (si osservi che $\cos x=1-2\sin^2\frac{x}{2}$)
2. $\sin 2x=\Theta(x)$ in $U(0)$
3. $\sqrt{2x^2+x}=\Theta(|x|)$ in $U(\pm\infty)$

Si osservi che se f e g sono entrambe infinitesime o infinite in $U(x_0)$ non è detto che siano asintotiche nè abbiano il medesimo ordine di grandezza.

Si considerino, come esempio, le due funzioni $f(x)=x$ e $g(x)=x^2$ in $U(0)$ oppure in $U(+\infty)$.

Rapporto infinitesimo

Diremo che $f(x)$ è trascurabile rispetto a ("o-piccolo" di) $g(x)$ per $x\to x_0$ e scriveremo $f=o(g)$ in $U(x_0)$ se:

(4) $$\lim_{x\to x_0}\frac{f(x)}{g(x)}=0$$

Alternativamente potremo anche dire che $g(x)$ è "omega-piccolo" di $f(x)$ e scrivere che $g=w(f)$ in $U(x_0)$.

Esempi:

1. $x^2=o(x)$ in $U(0)$
2. $x=o(x^2)$ in $U(+\infty)$
3. $x^n=o(e^x)\quad\forall n>0$ in $U(+\infty)$

Rapporto limitato

Diremo che $f(x)$ è "o-grande" di $g(x)$ per $x\to x_0$ e scriveremo $f=O(g)$ in $U(x_0)$ se:

(5) $$\exists k>0:|f(x)|\leq kg(x) \text{ in } U(x_0)$$

Equivalentemente, diremo che $g(x)$ è "omega-grande" di $f(x)$ per $x \to x_0$ e scriveremo $g = \Omega(f)$ in $U(x_0)$ se:

(6) $$\exists k > 0 : kf(x) \leq |g(x)| \text{ in } U(x_0)$$

Esempi:

1. $x = O(x^2)$ in $U(+\infty)$
2. $ax^2 + bx + c = O(x^2)$ in $U(+\infty)$

Le relazioni di uguaglianza asintotica e di ugual ordine di grandezza sono relazioni di equivalenza. Le altre sono relazioni d'ordine.

Tra le relazioni presentate intercorrono le seguenti implicazioni:

(7) $$\sim \Rightarrow \Theta \Rightarrow O \Leftarrow o$$

Ordine di infinitesimo e di infinito

Se $f(x)$ e $g(x)$ sono due funzioni entrambe infinitesime in $U(x_0)$ allora diremo che:

- f è un infinitesimo di ordine superiore a g se $f = o(g)$
- f e g sono infinitesimi dello stesso ordine se $f = \Theta(g)$
- f è un infinitesimo di ordine inferiore a g se $f = w(g)$

La classificazione per l'ordine di infinito è analoga.

L'esponenziale ha ordine di infinito superiore a qualsiasi potenza di x in $U(+\infty)$

L'esponenziale ha ordine di infinitesimo superiore a qualsiasi potenza di $1/|x|$ in $U(-\infty)$

Il logaritmo ha ordine di infinito inferiore a qualunque potenza di x in $U(+\infty)$

Il logaritmo ha ordine di infinito inferiore a qualunque potenza di $1/x$ in $U(0^+)$

§.7 Efficienza degli algoritmi

Usando la notazione O-grande potremo spesso descrivere i tempi di elaborazione di un algoritmo limitandoci semplicemente ad un'analisi superficiale della struttura complessiva dell'algoritmo stesso.

Ad esempio nell'algoritmo INSERTION SORT visto a lezione il doppio ciclo (loop) in esso contenuto suggerisce immediatamente $O(n^2)$ come limite superiore dei tempi di elaborazione nel caso peggiore.

Infatti il costo di ogni ciclo interno è limitato superiormente da $O(1)$ (ovvero è costante), gli indici i e j sono entrambi al più n e il ciclo interno viene eseguito al più una volta per ognuna delle n^2 coppie di valori i, j.

Poichè la notazione O-grande descrive un limite superiore, quando verrà usata per porre un limite al caso peggiore, quello che si otterrà sarà una maggiorazione dei tempi di elaborazione dell'algoritmo in esame valida per qualsiasi istanza di input.

Invece la limitazione del caso peggiore con $\Theta(n^2)$ non sarà in generale applicabile ad ogni istanza di input. Ad esempio, sempre nel caso dell'algoritmo INSERTION-SORT, è facile verificare che se l'input è già ordinato, $T_A(n) = \Theta(n)$

Pertanto, quando diremo che i tempi di elaborazione di un algoritmo sono $O(n^2)$, dovremo intendere che $\exists f(n) = O(n^2)$ tale che per ogni istanza di input che abbia dimensione n, i tempi di elaborazione sono limitati superiormente dal valore di $f(n)$. In altri termini diremo che il caso peggiore ha tempi di elaborazione che sono $O(n^2)$.

In maniera del tutto equivalente, l'uso della notazione Ω-grande significa porre un limite inferiore al caso migliore. Ad esempio abbiamo appena visto che il caso migliore dell'algoritmo insertion-sort è $\Omega(n)$ dunque complessivamente i tempi di elaborazione cadono tra $\Omega(n)$ e $O(n^2)$ ovvero che in ogni caso saranno compresi tra una qualsiasi funzione lineare e una qualsiasi funzione quadratica di n.

Esercizio

Sia $P(n) = \sum_{j=0}^{d} a_j n^j$ un polinomio di grado d e sia k una costante. Si provino le seguenti affermazioni:

1. se $k \geq d$ allora $P(n) = O(n^k)$
2. se $k \leq d$ allora $P(n) = \Omega(n^k)$
3. se $k = d$ allora $P(n) = \Theta(n^k)$
4. se $k > d$ allora $P(n) = o(n^k)$
5. se $k < d$ allora $P(n) = w(n^k)$

§.8 Pseudocodice

Al fine di non legare l'esposizione di un algoritmo ad un particolare linguaggio di programmazione usato per l'implementazione, gli algoritmi vengono comunemente presentati sotto forma di pseudocodice.
Non esiste una sintassi rigorosa per la scrittura in pseudocodice dunque con pseudocodice intenderemo qualsiasi espressione che specifichi chiaramente il funzionamento di un dato algoritmo.
E' convenzione comune l'uso della lingua inglese. Inoltre:

1. ogni blocco di istruzioni (ovvero ogni gruppo di istruzioni che possiedono un senso solo se raggruppate) viene "rientrato" rispetto al margine sinistro originale
2. le istruzioni per eseguire dei cicli (for, next, do, while, until, return), così come i passi condizionati (if, then, else) hanno la stessa interpretazione data nei principali linguaggi di programmazione
3. il simbolo $\triangleright$ indica che quanto scritto alla sua destra non fa parte delle istruzioni ma è solo a commento

4. il simbolo $\leftarrow$ indica assegnazione di un valore ad una variabile; è possibile effettuare assegnazioni multiple che verranno indicate, ad esempio, con $i \leftarrow j \leftarrow x$ intendendo con tale istruzione che il valore x viene assegnato alla variabile j e, successivamente, che il valore assunto dalla variabile j (che ovviamente ora sarà pari a x) viene assegnato alla variabile i
5. le variabili sono locali; per renderle globali è necessaria una esplicita dichiarazione
6. l'accesso alle componenti di un vettore è effettuato esplicitando tra parentesi quadre l'indice della componente alla quale si vuole accedere. Ad esempio $A[i]$ indicherà l'i-esima componente del vettore A
7. dati complessi verranno tipicamente organizzati in oggetti dotati di attributi; l'accesso ad un attributo verrà indicato scrivendo il nome dell'attributo seguito dal nome dell'oggetto racchiuso tra parentesi quadre; ad esempio il numero di elementi di un vettore A verrà indicato con lenght[A]
8. i parametri vengono passati per valore (by val) quindi se la procedura che riceve il parametro ne modifica il valore tale modifica non viene recepita all'esterno della procedura chiamata; viceversa gli oggetti vengono passati per riferimento (by ref) e quindi modifiche ai suoi attributi sono visibili anche all'esterno della procedura chiamata
9. gli argomenti degli operatori booleani AND e OR vengono valutati in sequenza dunque se in $a \,\text{AND}\, b$ a è FALSO l'operatore restituisce immmediatamente FALSO senza valutare b; viceversa se in $a \,\text{OR}\, b$ a è VERO l'operatore restituisce immediatamente VERO senza valutare b

§.9 Rappresentazione dei dati

In questo paragrafo introdurremo i vari formati utilizzati dagli elaboratori per rappresentare gli interi, i decimali e i caratteri di testo. Inoltre verranno esposti alcuni esempi implementati in linguaggio C++.

Innanzi tutto osserviamo che un numero reale può essere rappresentato dall'accostamento di una sequenza infinita di cifre in cui la parte intera è distinta da quella frazionaria da un simbolo (in genere il punto o la virgola) detto "separatore decimale". Più precisamente un numero $x \in \mathbb{R}$ sarà rappresentato da una coppia di successioni $\{a_k\}_{k=0}^{n-1}$ e $\{b_h\}_{h=0}^{+\infty}$.

Poichè, però, su ogni elaboratore la dimensione massima della memoria disponibile è sempre finita, noi potremo considerare solo numeri la cui successione della parte frazionaria è finita.
La fondamentale conseguenza di ciò è che gli elaboratori possono rappresentare solo i numeri razionali (e nemmeno tutti ma solo quelli la cui parte decimale non eccede il massimo numero di cifre significative rese disponibili dall'elaboratore stesso).

Dunque, nella nostra trattazione avremo che:

(8) $$x \equiv a_{n-1} \cdots a_0 . b_{m-1} \cdots b_0$$