Notazione asintotica e complessità

Notazione asintotica

La notazione asintotica (parte dello studio di funzione) **è uno strumento per lo studio del comportamento di una funzione agli estremi** del suo dominio, in particolare nella parte positiva e per la loro classificazione in base ad esso.

Alcune funzioni sono **dominanti** su altre, ma è possibile anche che due funzioni siano **incommensurabili**, cioè che nessuna domina sull'altra, in quanto si intersecano continuamente.

Contenimento: $\log n < n < n^2 < n^3 < \cdots < 2^n$

Notazione O-grande

$\mathcal{O}(g(n))$ è l'insieme delle funzioni limitate superiormente dalla funzione $g(n)$.

$$f(n) \in \mathcal{O}(g(n)) \leftrightarrow \exists\, c, n_0 | \forall n \geq n_0, 0 \leq f(n) \leq cg(n)$$

In questo insieme sono valide le seguenti affermazioni

- $per\ c_1 > 0\ e\ c_2 > 0 \rightarrow c_1 n^k - c_2 n^{k-1} \in \mathcal{O}(n^k)$
- $\boldsymbol{n^k \in \mathcal{O}(n^{k+1})}$
- $\boldsymbol{n^{k+1} \notin \mathcal{O}(n^k)}$

Valgono inoltre, le seguenti proprietà

- proprietà riflessiva $g(n) \in \mathcal{O}(g(n))$
- proprietà transitiva $f(n) \in \mathcal{O}(h(n))\ per\ h(n) \in \mathcal{O}(g(n))$
- regola dei fattori costanti positivi $f(n) = dh(n)\ per\ h(n) \in \mathcal{O}(g(n))$
- regola della somma $f(n) = h(n) + k(n)\ per\ h(n), k(n) \in \mathcal{O}(g(n))$

Contenimento: $\log n \in \mathcal{O}(n) \in \mathcal{O}(n^2) \in \mathcal{O}(n^3) \in \cdots \in \mathcal{O}(2^n)$ ...

Notazione Omega

$\Omega(g(n))$ è l'insieme delle funzioni limitate inferiormente dalla funzione $g(n)$.

$$f(n) \in \Omega(g(n)) \leftrightarrow \exists\, c, n_0 | \forall n \geq n_0, 0 \leq cg(n) \leq f(n)$$

In questo insieme sono valide le seguenti affermazioni

- $\boldsymbol{n^{k+1} \in \Omega(n^k)}$
- $\boldsymbol{n^k \notin \Omega(n^{k+1})}$

Valgono inoltre, le seguenti proprietà

- proprietà riflessiva $g(n) \in \Omega(g(n))$
- proprietà transitiva $f(n) \in \Omega(h(n))\ per\ h(n) \in \Omega(g(n))$
- regola dei fattori costanti positivi $f(n) = dh(n)\ per\ h(n) \in \Omega(g(n))$
- regola della somma $f(n) = h(n) + k(n)\ per\ h(n), k(n) \in \Omega(g(n))$

Contenimento: ... $2^n \in \cdots \in \Omega(n^3) \in \Omega(n^2) \in \Omega(n) \in \Omega(\log n)$

Notazione Theta

$\vartheta(g(n))$ è l'insieme delle funzioni limitate inferiormente e superiormente dalla funzione $g(n)$.

$$f(n) \in \vartheta\big(g(n)\big) \leftrightarrow \exists\, c_1, c_2, n_0 | \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

In questo insieme vengono messe in relazione le due notazioni precedenti:

$$f(n) \in \vartheta\big(g(n)\big) \leftrightarrow f(n) \in \mathcal{O}\big(g(n)\big) \wedge f(n) \in \Omega(g(n))$$

Valgono inoltre, le seguenti proprietà

- proprietà simmetrica $f(n) \in \vartheta\big(g(n)\big) \leftrightarrow g(n) \in \vartheta(f(n))$
- proprietà riflessiva $g(n) \in \vartheta(g(n))$
- proprietà transitiva $f(n) \in \vartheta\big(h(n)\big)\ per\ h(n) \in \vartheta(g(n))$
- regola dei fattori costanti positivi $f(n) = dh(n)\ per\ h(n) \in \vartheta\big(g(n)\big)$
- regola della somma $f(n) = h(n) + k(n)\ per\ h(n), k(n) \in \vartheta\big(g(n)\big)$

Essendo valide le prime tre proprietà, tale notazione rappresenta una relazione di equivalenza, dunque permette di classificare le funzioni in classi di equivalenza, ottenendo così una ***gerarchia delle funzioni***:

$$\vartheta(1), \vartheta(\log n), \vartheta(n), \vartheta(n \log n), \vartheta(n^2), \vartheta(n^3) \ldots \vartheta(2^n) \ldots$$

Approssimazione di Stirling

L'approssimazione di Stirling stabilisce che **la base del logaritmo è indifferente** nel calcolo asintotico.

È attraverso l'analisi asintotica che è possibile definire il costo di un algoritmo *e quindi di un programma.*

Costo economico

Ogni programma, cioè l'algoritmo implementato dalla pseudocodifica ad un linguaggio di programmazione, viene eseguito su una piattaforma e questa sua esecuzione ha un **costo economico**, che viene **espresso in termini delle risorse di calcolo** (tempo, memoria, traffico dati …). Tra queste la più critica è il ***tempo di calcolo***, direttamente proporzionale alla dimensione dell'input, che dipende dal problema ed è influenzato da diversi fattori (hardware, linguaggio, compilatore…).

Analisi degli algoritmi

Si procede dunque, con un'analisi degli algoritmi, **volta a prevedere il tempo di esecuzione attraverso l'analisi asintotica delle funzioni**. Il tempo di calcolo però non è una funzione, perciò si decide di eseguire il calcolo asintotico sulle tre funzioni che rappresentano il ***caso peggiore, migliore, medio*** (generalmente si fa riferimento al caso peggiore).

Stima del tempo di calcolo

Il tempo di calcolo $T(n)$ nel caso peggiore, si può determinare attraverso una **stima a partire dallo pseudocodice**. La strategia più veloce è quella di calcolare il costo per porzioni del codice e comporli per ottenere il comportamento asintotico, senza mai calcolarlo esplicitamente.
La regola della somma permette di considerare l'andamento dei singoli termini per determinare quello dell'intera somma.

Complessità degli algoritmi

Un algoritmo ha complessità temporale

- $\mathcal{O}(f(n))\ se\ T(n) \in \mathcal{O}(f(n))$ tempo sufficiente
- $\Omega(f(n))\ se\ T(n) \in \Omega(f(n))$ tempo necessario
- $\vartheta(f(n))\ se\ T(n) \in \mathcal{O}(f(n)) \wedge \Omega(f(n))$ tempo sufficiente e necessario

L'analisi della complessità degli algoritmi permette di determinare la complessità di un problema.

Complessità dei problemi

L'analisi della complessità di un problema è volta a classificare i problemi in base alla loro difficoltà di soluzione, cioè in termini di quantità di risorse.
Sono però infiniti gli algoritmi che possono risolvere un problema, alcuni più efficienti di altri, ma non è possibile considerarli tutti.

Un problema ha complessità temporale

- $\mathcal{O}(f(n))\ se\ \exists\ A \in \mathcal{O}(f(n))$ risorse sufficienti
- $\Omega(f(n))\ se\ \exists\ A \in \Omega(f(n))$ risorse necessarie
- $\vartheta(f(n))\ se\ \exists\ A \in \mathcal{O}(f(n)) \wedge \Omega(f(n))$ risorse sufficienti e necessarie

L'algoritmo asintoticamente ottimo è quello in cui limite superiore (upper bound) e limite inferiore (lower bound) coincidono.

Complessità ignota

Alcuni problemi hanno complessità ignota, in quanto presentano limiti incommensurabili, prendono perciò il nome di **NP-completi**. Questi sono tutti **equivalenti** e quindi risolvibili con lo stesso algoritmo polinomiale, se questo esiste. (esempio: commesso viaggiatore)

Lower bound di problemi comuni

Algoritmi di ricerca

Gli algoritmi di ricerca basati su confronti hanno, nel caso peggiore, complessità lineare o logaritmica a seconda della struttura di dati usata.
L'esecuzione di questi algoritmi può essere rappresentata da un albero di

decisione, cioè un albero binario in cui i nodi sono le condizioni, le foglie sono le possibili soluzioni e l'altezza è pari al numero di confronti. Risulta quindi un cammino $\Omega(\log n)$.

Algoritmi di ordinamento

Gli algoritmi di ordinamento basati su confronti hanno, nel caso peggiore, tutti complessità $\Omega(n \log n)$. L'esecuzione di questi algoritmi può essere rappresentata da un albero di decisione, in cui i nodi sono le condizioni, le foglie sono le permutazioni degli elementi da ordinare $n!$ e nel caso peggiore, il numero di confronti è $\Omega(n \ln n)$. Questo però dipende anche dai valori di input e risulta avere altezza $\Omega(\log_2 n!)$.

Costi delle istruzioni

Le istruzioni (descritte per la pseudocodifica) hanno i seguenti costi, che si indicano con la notazione theta per una stima in eccesso e in difetto.

- le istruzioni semplici hanno costo costante $\vartheta(1)$
- una sequenza di istruzioni semplici ha costo costante $\vartheta(1)$
- una sequenza di istruzioni gerarchiche ha costo pari alla somma dei costi delle singole istruzioni
- le istruzioni condizionali hanno costo pari alla somma del costo della condizione e del maggiore tra quello delle operazioni di `then` e `else`
- le istruzioni ripetitive hanno costo pari alla somma del costo della condizione e delle ripetizioni
- i cicli annidati hanno costo quadratico dato dal prodotto delle somme delle ripetizioni
- le funzioni che al loro interno richiamano una procedura di cui è noto il tempo di esecuzione, hanno costo pari al costo di questa, dipendente dall'input. Se l'input della procedura interna non è legato all'input principale, essa ha costo costante. Una chiamata a funzione all'interno di un'altra comporta la creazione di record di attivazione diversi per ognuna delle funzioni, quindi in caso di ricorsività, uno per ogni ricorsione. Il costo si esprime perciò come $T(n) = T(n-1) + \vartheta(1)$.

Formula di ricorrenza

La formula di ricorrenza è la coppia di equazioni o disequazioni che, basandosi su casi base e induttivi, **descrive una funzione** in termini del suo valore su input minori. Le soluzioni per queste formule non sono però, sempre facili da trovare.

Un problema con formula di ricorrenza $T(n) = \begin{cases} a \\ T(n-1) + g(n) \end{cases}$ ha come soluzione, quindi complessità la somma dei contributi.

$$\boldsymbol{T(n) = a + \sum_{k=1}^{n} g(k)}$$

Master Theorem

Il teorema dell'esperto definisce la soluzione alla formula di ricorrenza di alcuni problemi.

Un problema con formula di ricorrenza $\boldsymbol{T(n) = \begin{cases} \vartheta(1) \\ aT(n/b) + \mathcal{O}(n^k) \end{cases}}$, in cui a indica il numero di sottoproblemi e n/b la dimensione di ognuno dei problemi, ha come soluzione la somma dei costi dei livelli dell'albero di ricorrenza su un'istanza di dimensione n, che presenta altezza pari al numero di archi $h = \log_b n$ e profondità pari al numero di nodi $n/b^n = 1$.

$$T(n) = \begin{cases} \boldsymbol{\vartheta(n^k)\ se\ a < b^k} \text{ (primo livello dominante)} \\ \boldsymbol{\vartheta(n^k \log n)\ se\ a = b^k} \text{ (nessun livello dominante)} \\ \boldsymbol{\vartheta(n^{\log_b a})\ se\ a > b^k} \text{ (ultimo livello dominante)} \end{cases}$$

Complessità ammortizzata

La complessità ammortizzata è la complessità di un'operazione intesa come parte di una sequenza di operazioni.

Questa si riscontra solitamente quando si fa una gestione telescopica della memoria.

Gestione telescopica della memoria

La gestione telescopica della memoria è una strategia che si utilizza per contrastare i limiti sulla dimensione di strutture di dati come le pile.

Essa **consiste nel raddoppiare la dimensione della struttura ogni volta che questa risulta piena** (anziché generare un errore).

Il costo di questa copia raddoppia ogni volta, ma lo **si può distribuire sui costi delle operazioni precedenti** ("ora costano tutte uguali, ma il pagamento viene fatto dopo). In questo modo la complessità risulta ammortizzata.

Sommario

Notazione asintotica 1
Notazione O-grande 1
Notazione Omega 1
Notazione Theta 2
Approssimazione di Stirling 2
Costo economico 2
Analisi degli algoritmi 2
Stima del tempo di calcolo 3
Complessità degli algoritmi 3
Complessità dei problemi 3
Complessità ignota 3
Lower bound di problemi comuni 3
Algoritmi di ricerca 3
Algoritmi di ordinamento 4
Costi delle istruzioni 4
Formula di ricorrenza 4
Master Theorem 5
Complessità ammortizzata 5
Gestione telescopica della memoria 5