Università degli Studi di Verona

CORSO DI LAUREA IN INFORMATICA

Algoritmi

Federico Brutti
federico.brutti@studenti.univr.it

Inserire citazione inerente alla materia

Indice

Capitolo 1

Fondamenti

Il corso di Algoritmi ha l'obiettivo di fornire gli strumenti teorici e metodologici per la progettazione, descrizione e analisi formale di algoritmi. In particolare, studieremo come specificare in modo rigoroso una procedura di calcolo, come dimostrarne la correttezza e valutarne l'efficienza.

Definiamo **algoritmo** come una procedura di calcolo ben definita che prende uno o più valori in input per generarne ulteriori in output con un numero finito di passi. Di ogni algoritmo analizzeremo la **complessità**, intesa come funzione che misura le risorse necessarie alla sua esecuzione, intese tipicamente come tempo e spazio, in relazione alla dimensione dell'input. Questo ci permetterà di confrontare soluzioni alternative e scegliere quella più efficiente rispetto al problema considerato.

1.1 Complessità e notazione asintotica

La **complessità** di un algoritmo è una funzione che descrive la quantità di risorse necessarie alla sua esecuzione in relazione alla dimensione dell'input. Tipicamente la consideriamo in termini di **complessità temporale**, che riguarda il tempo di esecuzione, e di **complessità spaziale**, per la memoria utilizzata.

Non si tratta di un valore fisso, ma una funzione $T(n)$, dove n rappresenta la dimensione dell'input. Al variare di n, varia anche il numero di operazioni eseguite dall'algoritmo. Poiché siamo interessati al comportamento dell'algoritmo per input di grandi dimensioni, utilizziamo l'**analisi asintotica**, che permette di descrivere l'ordine di crescita della funzione trascurando costanti moltiplicative e termini di ordine inferiore. In particolare presenta le seguenti notazioni:

- **Limite asintotico superiore**: $O(f(n))$

Date due funzioni $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, diciamo che $f(n) \in O(g(n))$ se esistono una costante c e un valore $\overline{n}$ entrambi positivi tali che:

$$0 \leq f(n) \leq cg(n) \quad \forall n \geq \overline{n}$$

In tal caso g fornisce un limite superiore asintotico per f. Intuitivamente, per un n sufficientemente grande, la funzione f cresce al più come g, a meno di una costante moltiplicativa. Formalmente:

$$f(n) \in O(g(n)) \iff \exists c > 0, \exists \overline{n} : \forall n \geq \overline{n}, f(n) \leq cg(n)$$

- **Limite asintotico inferiore**: $\Omega(f(n))$

  Date due funzioni $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, diciamo che $f(n) \in \Omega(g(n))$ se esistono una costante c e un valore $\overline{n}$ entrambi positivi tali che:

  $$0 \leq cg(n) \leq f(n) \quad \forall n \geq \overline{n}$$

  In questo caso g fornisce un limite inferiore asintotico per f. Formalmente:

  $$f(n) \in \Omega(g(n)) \iff \exists c > 0, \exists \overline{n} : \forall n \geq \overline{n}, f(n) \geq cg(n)$$

- **Limite asintotico stretto**: $\Theta(f(n))$

  Date due funzioni $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, diciamo che $f(n) \in \Theta(g(n))$ se per i valori costanti positivi $c_1, c_2, \overline{n}$ risulta:

  $$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq \overline{n}$$

  In questo caso g descrive l'ordine di crescita preciso di f, poiché il valore di quest'ultima sarà sempre compreso fra le altre due. Formalmente:

  $$f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \land f(n) \in \Omega(g(n))$$

Esempio 1. ***Ricerca di un elemento in un array***
La complessità della ricerca di un elemento in un array è data da una funzione lineare, consentendoci di modellarla come:

$$T(n) = cn + a$$

Dove c rappresenta il costo del confronto, n è la dimensione dell'input, mentre a rappresenta i costi iniziali, espressi in una costante. Lavorando con l'analisi asintotica possiamo rimuovere i termini costanti e considerare esclusivamente il valore variabile n, ottenendo:

- *Caso ottimo:* $T(n) \in \Theta(1)$, *perché prende il primo elemento dell'array.*
- *Caso pessimo:* $T(n) \in \Theta(n)$, *perché scorre l'intero array fino all'elemento cercato.*
- *Caso medio:* $T(n) \in \Theta(n)$, *perché al netto di costanti, la complessità dipende dalla dimensione.*

Esempio 2. ***Dimostrazione di appartenenza a O(n)***
Sia la funzione $T(n) = 5n + 7 \in O(2n)$. *Dobbiamo trovare un* c *tale per cui valga l'equazione*

$$5n + 7 \leq c(2n)$$

Per far rispettare la relazione e quindi trovare un limite valido possiamo scegliere la costante $c = 3$, *con la quale, svolgendo le operazioni, otteniamo:*

$$5n + 7 \leq 6n$$

Risolvendo la disequazione infine otteniamo il valore $\overline{n} = 7$, *dimostrando la correttezza della relazione.*

Questi erano esempi concreti con funzioni ben definite, ma è possibile ragionare in termini più generali grazie alle seguenti proprietà:

- **Proprietà transitiva**

$$\begin{aligned} f(n) \in \Theta(g(n)) \land g(n) \in \Theta(h(n)) &\implies f(n) \in \Theta(h(n)) \\ f(n) \in O(g(n)) \land g(n) \in O(h(n)) &\implies f(n) \in O(h(n)) \\ f(n) \in \Omega(g(n)) \land g(n) \in \Omega(h(n)) &\implies f(n) \in \Omega(h(n)) \end{aligned} \tag{1.1}$$

- **Proprietà riflessiva**

$$\begin{aligned} f(n) &\in \Theta(f(n)) \\ f(n) &\in O(f(n)) \\ f(n) &\in \Omega(f(n)) \end{aligned} \tag{1.2}$$

- **Proprietà simmetrica**

$$f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n))$$

- **Simmetria trasposta**

$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n))$$

Poiché queste proprietà valgono per le notazioni asintotiche, è possibile trarre un'analogia concettuale fra il confronto asintotico di funzioni e numeri reali. In particolare:

$$\begin{aligned} f(n) &\in O(g(n)) \equiv a \leq b \\ f(n) &\in \Omega(g(n)) \equiv a \geq b \\ f(n) &\in \Theta(g(n)) \equiv a = b \end{aligned} \tag{1.3}$$

Per la dimostrazione delle relazioni nei diversi ordini di grandezza ci serviremo del metodo di **sostituzione**, con lo scopo di ricavare la tesi tramite passaggi logici. Una buona prassi è data da:

1. Scrivere i dati in base alla loro definizione.

2. Identificare ciò che è necessario per la dimostrazione.

3. Verificare la tesi.

Esempio 3. ***Dimostrazione limite asintotico superiore***
Supponiamo che la seguente formula sia vera. La si dimostri per confermare la supposizione:

$$f_1 \in O(g_1), f_2 \in O(g_2) \implies f_1 + f_2 \in O(g_1 + g_2)$$

Procediamo con i passaggi precedentemente menzionati:

1. *Riduciamo gli elementi alla loro definizione formale, ottenendo:*

$$f_1 \leq c_1 g_1(n); \quad f_2 \leq c_2 g_2(n)$$

2. *Per dimostrare la relazione è necessario trovare due valori c e $\overline{n}$ positivi. Ogni funzione ha i propri, quindi la c sarà data dalla loro somma, mentre $\overline{n}$ dal massimo di entrambi. Infatti:*

$$c = c_1 + c_2; \quad \overline{n} = max(\overline{n_1}, \overline{n_2})$$

3. *Procediamo con la verifica della tesi sostituendo agli elementi le relative definizioni:*

$$\begin{aligned} T(n) &= f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \\ &= f_1(n) + f_2(n) \leq c(g_1(n) + g_2(n)) \\ &= f_1(n) + f_2(n) \leq c(g_1 + g_2)(n) \implies f_1(n) + f_2(n) \in O(g_1 + g_2) \end{aligned} \tag{1.4}$$

Naturalmente per dare una visione completa della complessità di un algoritmo bisogna considerare anche il caso ottimo, quindi il suo limite asintotico inferiore. I passaggi di dimostrazione sono fondamentalmente gli stessi di prima.

Esempio 4. *Dimostrazione limite asintotico inferiore*
Dimostrare la seguente formula per confermare il limite inferiore dell'algoritmo:

$$f \in O(g(n)) \iff g(n) \in \Omega(f(n))$$

Stiamo asserendo che f è nell'ordine di grandezza di g se e solo se quest'ultima è il limite asintotico inferiore della prima. Procediamo:

1. *Definizione formale degli elementi:*

$$\exists c > 0, \exists \overline{n} : \forall n \geq \overline{n}, \quad f(n) \leq cg(n)$$

2. *Elementi necessari alla dimostrazione: I soliti valori $c, \overline{n}$.*

3. *Siccome la costante è positiva, effettuiamo il passaggio:*

$$cg(n) \geq f(n) \implies g(n) \geq \frac{f(n)}{c}$$

Infine diciamo che $\frac{1}{c} = c'$, che ci fa ottenere la definizione del limite inferiore:

$$g(n) \geq c'f(n) \implies g(n) \in \Omega(f(n))$$

Per riassumere, se vogliamo determinare l'ordine esatto di crescita è necessario dimostrare sia il **limite superiore** $O(f(n))$ che il **limite inferiore** $\Omega(f(n))$ rispetto alla stessa funzione, il quale è descritto da:

$$\theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

1.2 Equazioni di ricorrenza

Nella programmazione esistono due metodi principali per la costruzione di algoritmi: **ricorsione** e **iterazione**; in questa sezione studieremo entrambe le idee, con particolare attenzione agli algoritmi ricorsivi, rappresentati tramite le **equazioni di ricorrenza**. Una forma generale è data da:

$$T(n) = \begin{cases} \Theta(1) & n \leq \overline{n} \\ aT(n/b) + f(n) & n > \overline{n} \end{cases}$$

Dove, in particolare:

- **Numero dei sottoproblemi**: a.

- **Dimensione di ogni sottoproblema**: n/b.
- **Costo del lavoro svolto fuori dalle chiamate ricorsive**: $f(n)$.

Essendo che il caso base influisce esclusivamente su costanti additive e non sull'ordine asintotico, possiamo considerare solo il passo ricorsivo. Per risolvere queste equazioni ci serviremo del metodo di **sostituzione**, dove si suppone una stima asintotica della funzione, per poi provare a dimostrarla tramite induzione matematica, e degli **alberi di ricorrenza**, i quali la rappresentano con dei nodi la cui somma nel singolo livello è il costo della ricorsione in quel passo.

Esempio 5. ***Metodo di sostituzione***
Supponiamo l'equazione di ricorrenza $T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$. Vogliamo dimostrare che il suo limite superiore è:

$$T(n) \in O(nlogn)$$

Assumiamo $T(n) \leq c(nlogn)$ per ogni $n \geq \overline{n}$ come ipotesi induttiva per trovare i vincoli da rispettare. Stabilire la verità di questa ipotesi è sufficiente a dimostrare la tesi.

Se il limite vale per $n \geq \overline{n}$, allora l'ipotesi varrà per $n = \lfloor n/2 \rfloor$, da cui, per definizione, possiamo dire:

$$T(n) \leq c(nlogn) \implies T(\lfloor n/2 \rfloor) \leq 2(c \cdot \lfloor n/2 \rfloor log(\lfloor n/2 \rfloor) + \Theta(n)$$

Da questa forma possiamo procedere con la risoluzione dell'equazione di ricorrenza:

$$\begin{aligned} T(n) &\leq 2(c \cdot \lfloor n/2 \rfloor log(\lfloor n/2 \rfloor) + \Theta(n) \\ &\leq c \cdot nlog(n/2) + \Theta(n) \\ &\leq c \cdot nlog(n) - c \cdot nlog(2) + \Theta(n) \\ &\leq c \cdot nlog(n) - cn + \Theta(n) \end{aligned} \tag{1.5}$$

Nell'ultimo passaggio abbiamo rimosso le costanti moltiplicative. Logicamente, questo è un valore minore dell'ipotesi, quindi possiamo dimostrare la veridicità della tesi affermando che:

$$c(nlogn) \geq cnlog(n) - cn + \Theta(n)$$

Esempio 6. ***Albero di ricorsione***
Supponiamo l'equazione di ricorrenza $T(n) = 3T(n/4) + \Theta(n^2)$, definiamo il suo albero di ricorsione. Ad una prima occhiata è evidente che abbiamo 3 sottoproblemi e 4 come fattore di riduzione. Ragioniamo quindi per livelli:

- ***Livello 0 - Radice***

  Nodi: 1 *Costo:* n^2

- ***Livello 1***

  Nodi: 3 *Dimensione nodo:* $n/4$ *Costo nodo:* $(n/4)^2$ *Costo totale:* $\frac{3}{16}n^2$

- ***Livello i***

  Nodi: 3^i *Dimensione nodo:* $n/4^i$ *Costo nodo:* $(n/4^i)^2$ *Costo totale:* $\left(\frac{3}{16}\right)^i n^2$

Infine determiniamo ***altezza*** *dell'albero e* ***costo totale****. La prima è legata al totale di livelli della ricorsione. Siccome questa termina quando la somma dei nodi è uguale a 1, la ricaviamo con:*

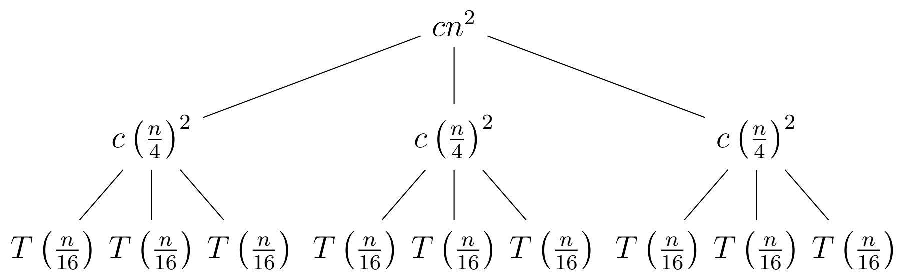
$$n/4^h = 1 \implies 4^h = n \implies h = log_4(n)$$

Il costo totale si ottiene invece facendo la somma di tutti i nodi di ogni livello, quindi si descrive con una sommatoria che va da 0 *al valore dell'altezza* $log_4(n)$*, dunque:*

$$\sum_{i=0}^{log_4(n)} cn^2 \left(\frac{3}{16}\right)^i$$

Questa è allo stesso tempo una serie geometrica con ragione $3/16 < 1$*. Questa converge, quindi il costo è dato dal primo termine:*

$$T(n) \in \Theta(n^2)$$



Per quanto riguarda la **modalità iterativa**, usiamo la notazione $f^i(n)$ per denotare la funzione f applicata iterativamente i volte a un valore di entrata n. Formalmente, sia una funzione $f(n)$ definita sui reali; per ogni intero non negativo i definiamo ricorsivamente:

$$f^i(n) = \begin{cases} n & i = 0 \\ f(f^{i-1}(n)) & i > 0 \end{cases}$$

Semplicemente, $f(n) = 2n \implies f^i(n) = 2^i n$. Un esempio più complesso è il seguente.

Esempio 7. *Metodo iterativo*
Sia la funzione $T(n) = 1 + T(n-1)$. Per dimostrare $T(n) \in \Theta(n)$ è necessario esaurire tutte le iterazioni richieste.

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + 1 + T(n-2) \\ &= 1 + 1 + 1 + T(n-3) \\ &= 1 + ... + 1 + T(n-i) \\ &= 1 + ... + 1 + T(n-n) \equiv 1 + ... + 1 + T(1) \end{aligned} \tag{1.6}$$

Otteniamo dunque $T(n) = n \implies T(n) \in \Theta(n)$

1.3 Teorema dell'esperto

Il **Teorema dell'esperto**, master theorem o metodo principale, è un enunciato utile per imporre una dominazione stretta su una funzione $f(n)$ tramite un polinomio di mezzo, definito come n^ϵ. Pone una condizione sufficiente ma non necessaria per le dimostrazioni e consente di risolvere più facilmente le ricorrenze in forma:

$$T(n) = aT(n/b) + f(n)$$

Poniamo una **funzione spartiacque** $n^{log_b a}$, usata per effettuare un confronto. Se una funzione è dominata da un'altra che sta sotto alla spartiacque, avremo la garanzia che la prima si troverà nell'ordine di grandezza di quest'ultima. Il teorema si definisce formalmente come:

Teorema 1. *Teorema dell'esperto*
Siano $a > 0, b > 1$ delle costanti e $f(n)$ una funzione forzante definita e non negativa per tutti i reali maggiori di una certa soglia. Sia inoltre $T(n)$ la ricorrenza definita per $n \in \mathbb{N}$ da:

$$T(n) = aT(n/b) + f(n)$$

Dove l'espressione $aT(n/b) = a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ per opportune costanti $a', a'' \geq 0 : a = a' + a''$. Allora il comportamento asintotico della ricorrenza può essere caratterizzato nel seguente modo:

1. *Se esiste una costante $\epsilon > 0 : f(n) \in O(n^{log_b(a-\epsilon)})$, allora $T(n) \in \Theta(n^{log_b a})$.*

2. *Se esiste una costante $k \geq 0 : f(n) \in \Theta(n^{log_b a} \cdot log^k n)$, allora $T(n) \in \Theta(n^{log_b a} \cdot log^{k+1} n)$.*

3. *Se esiste una costante $\epsilon > 0 : f(n) \in \Omega(n^{log_b(a+\epsilon)})$ e inoltre $f(n)$ soddisfa la seguente* ***condizione di regolarità****:*

$$af(n/b) \leq cf(n) \forall n \text{ abbastanza grandi e una } c < 1$$

Allora $T(n) \in \Theta(f(n))$.

Il teorema, tuttavia, non si può usare in ogni istanza. I casi da evitare sono:

- La ricorrenza non è della forma $aT(n/b) + f(n)$.
- C'è una divisione irregolare, come $T(n-1)$.
- $f(n)$ non è confrontabile con un polinomio.
- Manca la condizione di regolarità nel caso 3.

Esempio 8. *Caso 1 del teorema*
Sia $T(n) = 9T(n/3) + n$, abbiamo $a = 9, b = 3 \implies n^{log_3 9} = n^2$. Abbiamo inoltre $f(n) = n$; confrontandola con l'ϵ risulta:

$$n \in O(n^{2-\epsilon})$$

Ci troviamo quindi nel primo caso e confermiamo che $T(n) \in \Theta(n^2)$.

Esempio 9. *Caso 2 del teorema*
Sia $T(n) = T(2/3n) + 1$, abbiamo $a = 1, b = 3/2, f(n) = 1$, quindi:

$$n^{log_b a} = n^{log_{3/2} 1} = n^0 = 1$$

Siccome $f(n) = 1 \implies f(n) \in \Theta(1)$ ci troviamo nel caso 2. Ogni qualvolta si riporta il problema a costanti, ci troviamo in ordine di $log(n)$, dato che rappresenta il numero dei livelli dell'albero. Confermiamo che $T(n) \in \Theta(log(n))$.

Esempio 10. *Caso 3 del teorema*
Sia $T(n) = 3T(n/4) + nlog(n)$, abbiamo $a = 3, b = 4, f(n) = nlog(n)$, quindi:

$$n^{log_b a} = n^{log_4 3}$$

Notiamo che $nlog(n)$ cresce più rapidamente della spartiacque $n^{log_4 3}$, dunque ci troviamo nel caso 3. Verifichiamo prima la condizione di regolarità, prendendo per esempio $c = 3/4$:

$$3(n/4)log(n/4) \leq c \cdot nlog(n)$$

La condizione è vera per n grande con $c < 1$. Dunque $T(n) = \Theta(nlog(n))$

Esempio 11. *Teorema non applicabile*
Sia $T(n) = 2T(n/2) + nlog(n)$, abbiamo $a = 2, b = 2, f(n) = nlog(n)$, quindi:

$$n^{log_b a} = n^1 = n$$

Notiamo che non ci troviamo in nessuno dei tre casi precedenti, perché la funzione $f(n)$ cresce più rapidamente di qualunque potenza positiva, quindi il teorema dell'esperto non è applicabile.

1.4 Esercizi svolti

Capitolo 2

Ordinamenti e selezioni

2.1 Insertion Sort

2.2 Merge Sort

2.3 Heap Sort

2.4 Quick Sort classico e probabilistico

2.5 Counting Sort

2.6 Radix Sort

2.7 Bucket Sort

2.8 Problema della selezione

Capitolo 3

Strutture dati

3.1 Heap

3.2 Alberi binari

3.2.1 RB-alberi

3.2.2 B-alberi binomiali

3.3 Tabelle hash

3.4 Code con priorità

3.5 Insiemi disgiunti

3.6 Estensione di strutture dati

3.7 Grafi

Capitolo 4

Progetto e analisi di algoritmi

4.1 Divide et impera

4.2 Programmazione greedy

4.3 Programmazione dinamica

4.4 Ricerca locale

4.5 Backtracking

4.6 Branch and bound

Capitolo 5

Algoritmi fondamentali

5.1 Alberi di copertura di costo minimo

Prim, Kruskal

5.2 Programmazione lineare

Simplesso, algoritmo fondamentale polinomiale basato sugli ellissoidi

5.3 Cammini minimi

5.3.1 Sorgente singola

Dijkstra, Bellman-Ford

5.3.2 Sorgente multipla

Floyd-Warshall, Johnson

5.4 Flusso massimo

Ford-Fulkerson, Karp

5.5 Matching massimale su grafo bipartito