Appunti di Algoritmi e Strutture dati
A.A. 2022/2023

1 Fondamenti

1.1 Algoritmi e loro rappresentazione

Un **problema** é un quesito che richiede la determinazione o la costruzione di uno o piú enti matematici che soddisfino le condizioni specificate nell'enunciato. Con *problema* si denota l'enunciato generale, con *istanza* si denota un caso particolare del problema, ovvero un insieme specifico di dati per il quale si vuole ottenere una soluzione.

Un **algoritmo** é una sequenza di azioni non ambigue che risolve un problema utilizzando un insieme di azioni elementari, eseguibili da un opportuno esecutore. Con **programma** si denota la rappresentazione di un algoritmo utilizzando un linguaggio (con opportune "traduzioni") direttamente comprensibile da un elaboratore.
L'algoritmo é specificato da un insieme ben definito di dati in input e in output, deve essere eseguibile in un numero finito di passi, fornire il risultato corretto per ogni possibile input ed essere abbastanza generale da essere applicabile a un'intera classe di problemi.

Lo **pseudocodice** é un linguaggio astratto ed informale, inteso per uso umano, utilizzato per descrivere un algoritmo. Utilizza la struttura di un linguaggio di programmazione normale ma non é vincolato nella sintassi ed é integrabile con linguaggio naturale o notazioni matematiche compatte.

1.2 Confronto di algoritmi

Nel confrontare gli algoritmi, ci si astrae da tutti gli aspetti dipendenti dall'implementazione.
La risorsa principale su cui ci si basa é il **tempo di esecuzione**, quantificato non in secondo ma in accessi alla RAM.
L'algoritmo viene visto come una funzione $f(n)$ (dove n é l'input fornito) di cui si considera solamente il termine dominante (solitamente si trascura anche il coefficiente). Per confrontare due algoritmi, si confrontano le due rispettive funzioni per $n \to \infty$.

1.2.1 Notazione O-grande

La crescita asintotica delle funzioni viene descritta con diverse notazioni, la cui maggiormente usata é la notazione O grande (limite asintotico **superiore**), dove O sta per ordine di grandezza. L'**ordine di grandezza** é la piú piccola funzione maggiorante (siccome ne esistono infinite).

Definizione: siano $f : \mathbb{R} \to \mathbb{R}, g : \mathbb{R} \to \mathbb{R}$, $f(x) = O(g(x))$ se $\exists c, x_0 : \forall x > x_0$ si ha $f(x) \leq c \cdot g(x)$.

Esempi

- Se si vuole dimostrare $f(x) = x^2 + 2x + 1$ é $O(x^2) \to x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2 \iff 3x^2 - 2x - 1 \geq 0 \iff x \geq 1$ ($\frac{2 \pm \sqrt{16}}{6} \to \frac{2+4}{6} = 1$). Ho dimostrato che la definizione vale per $n_0 = 1, c = 4$.
- Per dimostrare invece che $f(n) = \frac{n(n+1)}{2}$ non é $O(n) \to \frac{1}{2}(n^2 + n) \leq Cn \iff n^2 + n \leq 2cn \iff n^2 \leq n(2c - 1) \iff n \leq 2C - 1$. Essendo c costante, $\forall c \in R, \exists n \in R : n > 2c - 1$, quindi $f(n)$ non é $O(n)$.

Alcuni ordini di grandezza ben noti in ordine crescente: **costante** ($O(1)$), **logaritmico** ($O(\log n)$), **lineare** ($O(n)$), **log lineare** ($O(n \log n)$), **polinomiale** ($O(n^k)$ con k costante), **esponenziale** ($O(c^n)$ con c costante).

1.2.2 Altre notazioni

Altre notazioni utilizzate nell'analisi asintotica sono Ω (Omega grande, limite asintotico **inferiore**) e Θ (Theta grande).

Definizione di Ω: siano $f : \mathbb{R} \to \mathbb{R}, g : \mathbb{R} \to \mathbb{R}, f(n) = \Omega(g(n))$ se $\exists c, n_0 : f(n) \geq c \cdot g(n) \ \forall n \geq n_0$.
É utilizzata nei limiti inferiori di complessitá e per l'analisi del tempo di esecuzione nel caso ottimo.

Definizione di Θ: siano $f : \mathbb{R} \to \mathbb{R}, g : \mathbb{R} \to \mathbb{R}, f(n) = \Theta(g(n))$ se $\exists c_1, c_2, n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \ \forall n \geq n_0$. Da notare che $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge \Omega(g(n))$.
$O(fn))$ é spesso usato erroneamente al posto di $\Theta(f(n))$.

Esistono degli analoghi di O e Ω che sono o (o piccolo) e ω (omega piccolo), dove al posto della disuguaglianza ($\leq$) si ha la disuguaglianza stretta ($<$).

Equivalentemente:

$$\text{se} \lim_{n \to \infty} \frac{f(n)}{g(n)} \to c, \text{ allora } f(n) = \Theta(g(n))$$

$$\text{se} \lim_{n \to \infty} \frac{f(n)}{g(n)} \to 0, \text{ allora } f(n) = o(g(n))$$

se $\lim_{n \to \infty} \frac{f(n)}{g(n)} \to \infty$, allora $f(n) = \omega(g(n))$

1.2.3 Caso ottimo, medio, pessimo

Nell'analisi degli algoritmi si possono analizzare 3 casi: caso ottimo (migliore possibile), medio e pessimo (peggiore possibile). Principalmente si analizza il caso pessimo e il caso medio, anche se l'analisi di quest'ultimo é spesso molto piú complicata delle altre due perché richiede un'analisi statistica.

Esempio: Ricerca sequenziale.
Problema: dato un array v e un valore x, restituire l'indice della prima occorrenza di x in v, o -1 se x non é presente.

1. Caso ottimo: l'elemento é all'inizio della lista ($O(1)$)
2. Caso pessimo: l'elemento é in fondo o non é presente, quindi si itera su tutti gli elementi ($O(n)$)
3. Caso medio: per ipotesi, l'elemento é sempre presente e la probabilitá p_i che l'elemento si trovi alla posizione i sia la stessa per ogni i, quindi $p_i = \frac{1}{n}$. Se l'elemento é nella prima posizione, devo controllare una sola volta, nella seconda due volte, nella terza tre volte e cosí via fino a n: questo lo posso esprimere con la somma dei primi n numeri, ovvero con la serie geometrica $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Moltiplico il tutto per la probabilitá p_i, che é la stessa per ogni elemento e ottengo $\frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{1}{2}(n+1)$, ovvero $O(n)$.

2 Analisi asintotica

Un algoritmo A ha **costo di esecuzione** $O(f(n))$ rispetto ad una risorsa di calcolo, su instanze di ingresso di dimensione n se la quantitá $r(n)$ di risorsa sufficiente per eseguire A su una **qualunque istanza di dimensione** n verifica la relazione $r(n) = O(f(n))$.

Dato lo pseudocodice, é possibile ottenere il costo di esecuzione analizzando la sua struttura. Ad esempio, data una serie di istruzioni, $t(\text{istruzione 1}) + \cdots + t(\text{istruzione } n)$, negli if-else $max(t(\text{ sequenza 1}), t(\text{ sequenza 2}))$, nei for e nei while bisogna vedere se il ciclo viene eseguito un numero di volte funzione di n o meno.

2.1 Algoritmi ricorsivi

Negli algoritmi ricorsivi, il tempo di esecuzione dell'algoritmo puó essere descritto come la somma dei tempi di esecuzione di $f(n_1), \ldots, f(n_k)$ con $n_i < n$, ovvero richiede di calcolare tutti i termini precedenti, fino al caso base.

2.1.1 Ricerca binaria

La ricerca binaria ha classe di complessitá $O(logn)$.
Il tempo di esecuzione puú essere espresso come:

$$\begin{cases} c_1 \text{ se } n = 1 \text{ (caso base)} \\ T(\lfloor \frac{n}{2} \rfloor) + c_2 \text{ se } n > 1 \end{cases} \tag{1}$$

É dimostrabile con diversi metodi, tra cui:

1. Metodo iterativo. Partendo da $T(n)$, devo arrivare al caso base $T(1) = T(\frac{n}{n})$ dimezzando n ad ogni passo. $T(n) = T(\frac{n}{2}) + c_2 = T(\frac{n}{4}) + 2c_2 = T(\frac{n}{8}) + 3c_2 = \ldots$
   Mi fermo quando $n = 2^k$, quindi dopo $k = log_2 n$ chiamate ricorsive, per un totale di ck chiamate ($c = c_1 + kc_2$). Quindi,

   $$T(n) = c \cdot log_2 n = O(logn)$$

2. Dimostrazione per induzione. Prima di dimostrare, si "indovina" (grazie anche all'esperienza) la soluzione: $T(n) \leq c \cdot log_2 n (T(n) = O(logn))$.
   Assumendo $T(n') \leq c \cdot log_2 n' \; \forall n > n'$ (**ipotesi induttiva**), voglio dimostrare che $T(n) \leq c \cdot log_2 n$.

   $T(n) = T(\dfrac{n}{2}) + 1$ (costante qualsiasi) $\leq c \cdot log_2(\dfrac{n}{2}) + 1$ (ipotesi induttiva ($\dfrac{n}{2}$ é n'))
   $= c \cdot log_2 n - c \cdot log_2 2$ (proprietá del logaritmo) $+1$
   $= c \cdot log_2 n - c \cdot 1 + 1$.
   Se $c \geq 1$, allora

   $$c \cdot log_2 n + c - 1 \leq c \cdot log_2 n \rightarrow T(n) \leq c \cdot log_2 n$$

3 Ordinamenti e grafi

3.1 Grafo

Un grafo $G = (V, E)$ é composto da un insieme di *vertici* V e un insieme di *archi* $E \subset V \times V$ (non $\subseteq$ perché non si ha un arco tra un vertice e se stesso) che connettono i vertici.
In un grafo *non orientato*, due vertici u, v sono **adiacenti** se $\{u, v\} \in E$.
In un grafo *orientato*, se $(u, v) \in E \rightarrow v$ adiacente a u e (u, v) é **incidente** in v.
Il *grado* di un vertice é il numero di vertici adiacenti ad esso.

Il *cammino* é una sequenza di vertici $v_1, \ldots, v_n$ tale che per ogni coppia di vertici consecutivi v_i, v_{i+1}, v_{i+1} é adiacente a v_i. Un cammino é detto *elementare* se non ci sono vertici ripetuti. Un *ciclo* é un cammino elementare in cui il primo vertice coincide con l'ultimo (torna all'inizio).
É detto *grafo connesso* qualsiasi coppia di vertici unita da almeno un cammino. Un *sottografo* é un sottoinsieme di vertici e archi di un grafo dato. Una *componente connessa* é un sottografo connesso massimale (non si possono aggiungere altri vertici o archi)