Algorithms and Data Structures - 2024/2025

Homework set #1

Giuseppe Maglione, Annarita Fabiano

Chapter 1

Soluzioni esercizi

1.1 Notazione asintotica

- L'appartenenza $f(n) = O(f(n)^2)$ è a volte vera.

Per dimostrare riportiamo due esempi, uno mostra un caso in cui l'appartenenza non è verificata mentre l'altro mostra un caso in cui si.

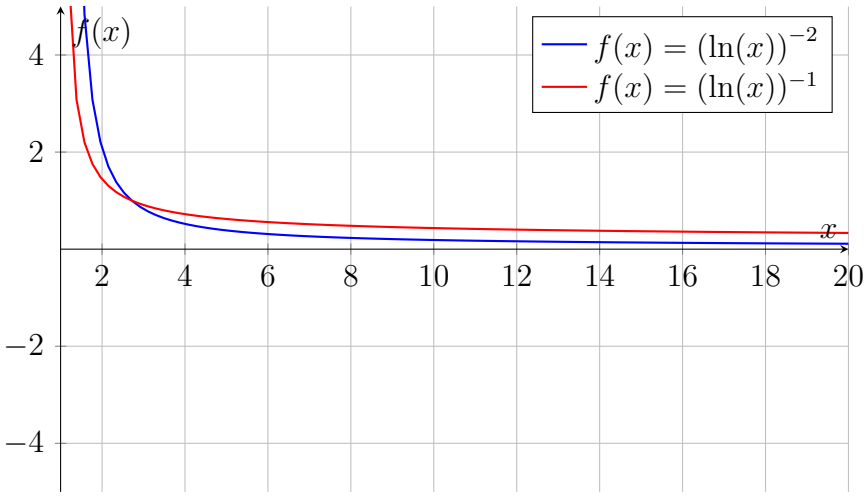


Figure 1.1: L'appartenenza non è verificata.

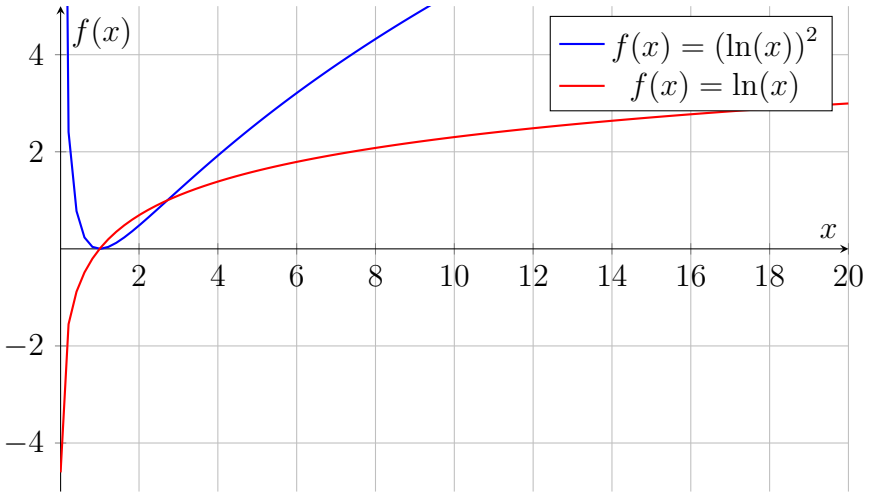


Figure 1.2: L'appartenenza è verificata.

- L'appartenenza $f(n) + O(f(n)) = \Theta(f(n))$ è sempre vera.

Per dimostrare, iniziamo osservando che $f(n) + O(f(n))$ restituisce sempre una funzione ancora $O(f(n))$, questo intuitivamente perché sommando a $f(n)$ una funzione $O(f(n))$ il termine dominante continuerà ad essere $O(f(n))$. A questo punto, per concludere la dimostrazione ci basta ricordare che la notazione Θ è più forte della notazione O, ossia $\Theta(f(n)) \subseteq O(f(n))$. In conclusione l'appartenenza è sempre verificata.

- L'appartenenza $f(n) = \Omega(g(n))$ e $f(n) = o(g(n))$ è mai vera.

Per dimostrare, ricordiamo che la definizione della notazione Ω richiede che esistano delle costanti positive c e n_0 tale che $0 \leq cg(n) \leq f(n)$ per ogni $n \geq n_0$. La notazione o, invece, richiede che per qualsiasi costante c positiva esista un $n_0 > 0$ tale che $0 \leq f(n) > cg(n)$ per ogni $n \geq n_0$. Dalle definizioni appare evidente il conflitto tra le due per l'appartenenza proposta, la quale non può quindi mai verificarsi.

1.2 Complessità

- L'appartenenza $2^{n+1} = O(2^n)$ è vera.

Sfruttando le proprietà algebriche degli esponenziali, il primo termine può riscriversi come $2 \cdot 2^n$. L'appartenenza diventa quindi $2 \cdot 2^n = O(2^n)$, che è chiaramente verificata esistendo

sicuramente almeno una costante positiva c che per n sufficientemente grande soddisfi la condizione $0 \leq 2 \cdot 2^n \leq c \cdot 2^n$.

- L'appartenenza $2^{2n} = O(2^n)$ è falsa.

Sfruttando ancora le proprietà algebriche degli esponenziali, il primo termine può riscriversi come $2^n \cdot 2^n$. L'appartenenza si riduce quindi a $2^n \cdot 2^n = O(2^n)$, da cui appare evidente che non esistano costanti positive c che per n sufficientemente grande soddisfino la condizione richiesta da O precedentemente descritta.

1.3 Ricorrenze

- Fornire il limite inferiore per $T(n) = 2T(\frac{n}{3}) + nlogn$.

Proviamo a risolvere la ricorrenza applicando il teorema dell'esperto, utilizzando i valori $a = 2$, $b = 3$ e $f(n) = nlogn$. Mostriamo che la ricorrenza assegnata ricade nel terzo caso del teorema, ossia si verifica che:

1. $f(n) = \Omega(n^{log_3 2+\epsilon})$ per qualche costante ϵ positiva.

2. $af(\frac{n}{b}) \leq cf(n)$ per qualche costante $c < 1$ e per n sufficientemente grande.

Per la 1. è possibile osservare che la scelta di $\epsilon = 1 - log_3 2$ riconduce a $nlogn = \Omega(n)$, che è chiaramente verificata esistendo sicuramente una costante c' positiva e un n_0 che soddisfi la definizione $0 \leq nlogn \geq c'n$ per $n \geq n_0$. Per la 2. invece, notiamo che la disuguaglianza è verificata scegliendo, ad esempio, la costante $c = \frac{2}{3}$. Possiamo quindi concludere, in virtù del teorema dell'esperto, che $T(n) = \Theta(f(n)) = \Theta(nlogn)$.

- Fornire il limite inferiore per $T(n) = 3T(\frac{n}{5}) + log^2 n$.

Proviamo a risolvere la ricorrenza applicando il metodo di sostituzione, ipotizzando come soluzione $T(n) = O(n)$.

1. Caso base ($k = 1$): $T(1) = O(1) \rightarrow 10 \leq c$, con c costante positiva.

2. Caso induttivo ($k = n - 1$): $T(k) \leq ck$.

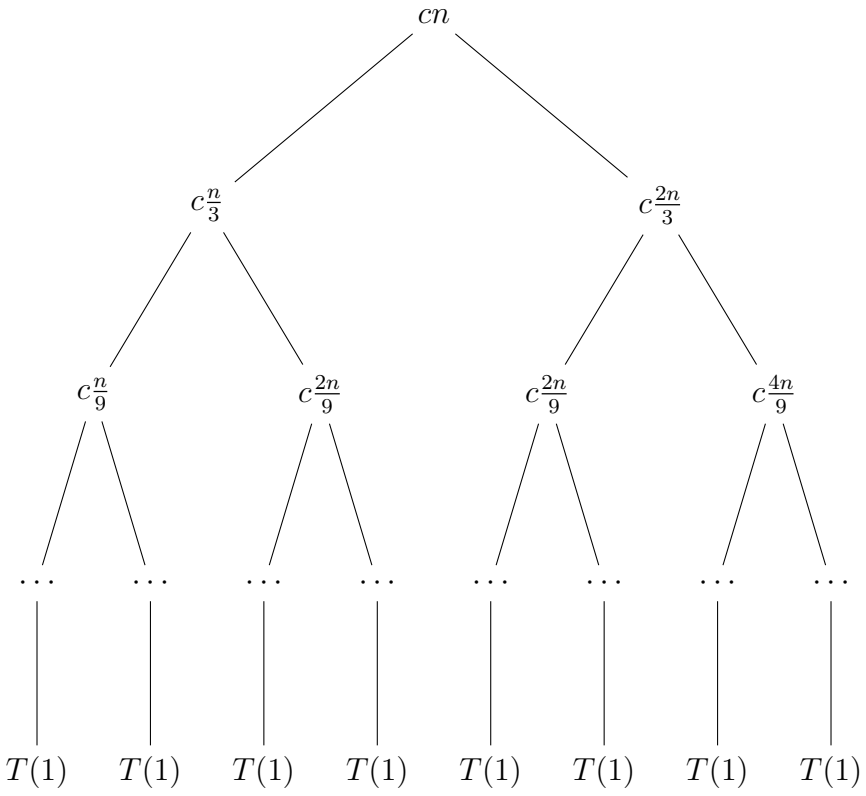
3. Caso da dimostrare ($k = n$): $T(n) = O(n)$.

Per dimostrare la 3. osserviamo che la ricorrenza diventa $T(n) = 3T(\frac{n}{5}) + log^2 n$. A questo punto, sfruttando il caso induttivo otteniamo $T(n) \leq \frac{3}{5}cn + log^2 n$. Affinché l'ipotesi induttiva sia valida, dobbiamo avere $cn \geq \frac{3}{5}cn + log^2 n \rightarrow cn(1 - \frac{3}{5}) \geq log^2 n \rightarrow \frac{2}{5}c \geq \frac{log^2 n}{n}$, ma per n sufficientemente grande il termine $log^2 n$ cresce molto più lentamente rispetto a n, ossia $\frac{log^2 n}{n} \rightarrow 0$. In base a quanto dimostrato, la costante c può essere scelta arbitrariamente grande in modo da soddisfare sia la disuguaglianza precedente che la condizione di contorno. In conclusione, $T(n) = O(n)$.

1.4 Ricorrenze

- Dimostrare che la ricorrenza $T(n) = T(\frac{n}{3}) + 2T(\frac{2n}{3}) + cn$ ha soluzione $T(n) = \Theta(nlogn)$ tramite l'albero delle ricorrenze.

Iniziamo rappresentando graficamente l'albero.



Possiamo osservare che la dimensione del sotto-problema al livello i è $\frac{n}{2^i}$, il costo $T(1)$ è raggiunto quando la dimensione è pari a $\frac{n}{2^i} = 1 \rightarrow i = log_2 n$. Concludiamo che il numero di

livelli nell'albero, e quindi la sua altezza, è $logn_2n + 1$. Osservando l'albero delle ricorrenze, notiamo che il costo ad ogni livello è costante e pari a cn, per cui il costo totale della ricorrenza può essere calcolato come il prodotto tra questa quantità e l'altezza dell'albero, quindi $T(n) = cn(log_2n + 1) = cnlog_2n + cn$. Secondo la definizione della notazione, affinché $T(n) = \Omega(nlogn)$ deve esistere una costante $d > 0$ per cui $0 \leq dnlog \leq T(n)$, che è in accordo con il risultato trovato prima per $T(n)$. In conclusione $T(n) = \Omega(nlogn)$.

Chapter 2

Soluzioni problemi

2.1 Array unimodale

- Si fornisca un algoritmo per calcolare il massimo elemento in un array unimodale $A[1, \ldots, n]$ che esegua in $O(logn)$. Si dimostri che la complessità è $O(logn)$.

Iniziamo riportando lo pseudo-codice per l'algoritmo.

find_max(A, n)

$first \leftarrow 0, last \leftarrow n - 1$

while $first \leq last$:

$mid \leftarrow (first + last)/2$

if $A[mid - 1] < A[mid]$ AND $A[mid + 1] < A[mid]$:

return $A[mid]$

else if $A[mid - 1] < A[mid]$ AND $A[mid + 1] > A[mid]$:

$first \leftarrow mid + 1$

else:

$last \leftarrow mid - 1$

return $high$:

Il funzionamento dell'algoritmo sfrutta le sequenze crescenti e decrescenti presenti nell'array unimodale in ingresso, procedendo quindi ricorsivamente a verificare se l'elemento massimo è situato nella posizione mid. In caso contrario, sulla base di controlli effettuati sui valori

presenti agli indici precedenti e successivi a *mid*, la procedura viene richiamata sulla metà sinistra o destra dell'array fino a trovare l'indice che soddisfi la condizione di massimo.
Per l'analisi della complessità, osserviamo che l'algoritmo divide il problema in sotto-problemi di dimensione $n/2$. Inoltre, la complessità per suddividere l'algoritmo è costante, in quanto consiste nel calcolo dell'indice medio e semplici controlli sugli elementi di un array indicizzato. Per quanto detto, la ricorsione diventa $T(n) = T(\frac{n}{2}) + O(1)$, che per il secondo caso del teorema dell'esperto ha soluzione $T(n) = \Theta(nlogn)$. Alternativamente, potevamo giungere alla stessa conclusione osservando che l'algoritmo è basato su una ricerca binaria con qualche modifica sui controlli effettuati.

2.2 Prefisso comune

- Dato un insieme di stringhe, si implementi un algoritmo divide et impera per trovare il prefisso in comune più lungo.

Vedi allegati per l'implementazione e i casi di test.
Il funzionamento dell'algoritmo prevede di suddividere ricorsivamente l'insieme delle stringhe in ingresso in sottoinsiemi di dimensione dimezzata, procedendo fino al caso in cui le stringhe da analizzare sono due. Una volta fatto ciò, sulla coppia di stringhe viene richiamata la procedura che calcola il prefisso in comune tra le due, restituendolo e facendo proseguire la comparazione con i prefissi ottenuti dalle altre coppie di stringhe.
Per l'analisi della complessità, iniziamo osservando che l'algoritmo scompone ricorsivamente il problema in 2 sotto-problemi di dimensione $\frac{n}{2}$, fino ad arrivare al caso in cui le stringhe da confrontare sono 2. A quel punto, per ciascuna delle coppie individuate viene calcolato il prefisso in comune, poi confrontato ricorsivamente con i restanti prefissi trovati. Per quanto detto, è possibile esprimere la ricorrenza dell'algoritmo come $T(n) = 2T(\frac{n}{2}) + f(n)$, dove n è il numero di stringhe nell'insieme di partenza e $f(n)$ è il costo associato alla funzione che si occupa del calcolo del prefisso comune in una coppia di stringhe. Quest'ultima, nel peggiore dei casi, effettua un controllo su un totale di m simboli, dove m è la lunghezza della stringa più piccola nella coppia analizzata. Per cui, la complessità associata alla funzione è $O(m)$

e dunque la ricorrenza dell'algoritmo diventa $T(n) = 2T(\frac{n}{2}) + O(m)$. Il secondo caso del teorema dell'esperto ci permette di concludere che $T(n) = O(nlogn)$.

2.3 Treap

- Si implementi un algoritmo di inserimento per un treap.

Vedi allegati per l'implementazione.

Il funzionamento dell'algoritmo prevede di inserire il nuovo nodo in base al valore della sua *key*, trattando l'inserimento come una classica procedura per BST e generando una *priority* casuale. Una volta inserito il nuovo nodo, per mantenere equilibrato l'albero, vengono richiamate delle operazioni di rotazione sulla base di controlli effettuati sfruttando la priorità del nodo appena inserto.