Homeworks Algorithms and Data Strucutres

Rocco Lo Russo
roc.lorusso@studenti.unina.it

Università di Napoli Federico II - DIETI — 26/08/2025

Introduzione

In questo documento verranno sviluppati i due set di Homeworks assegnati per sostenere l'esame.

1 Homework set 1

1.1 Esercizio 1.1

Traccia:
Per ognuna delle seguenti affermazioni, si dica se essa è sempre vera, mai vera, o a volte vera, per funzioni asintoticamente non-negative. Se la si considera sempre vera o mai vera, si spieghi il perché. Se è a volte vera, si dia un esempio per cui è vera e uno per cui è falsa.

- $f(n) = O(f(n)^2)$;
- $f(n) + O(f(n)) = \Theta(f(n))$;
- $f(n) = \Omega(g(n))$ e $f(n) = o(g(n))$

Soluzione:
$f(n) = O(f(n)^2)$ è un'affermazione vera a volte: infatti $f(n) = O(f(n)^2) \iff f(n) \leq cf(n)^2$ per qualche costante $c > 0$ e per qualche $n > n_0$. Poichè la funzione f(n) è definita asintoticamente *non negativa*, possiamo assumere che per n sufficientemente grande la funzione assumerà soltanto valori o positivi o nulli.
Nel caso in cui $f(n) = 0$, la disuguaglianza è verificata perchè $0 \leq c \cdot 0^2$ è vero sempre.
Nel caso in cui $f(n) > 0$:

$$f(n) \leq cf(n)^2 \implies f(n) \geq \frac{1}{c}, \forall n \geq n_0 \tag{1}$$

La disuguaglianza nel secondo caso è verificata solo se la funzione $f(n)$ risulta, per n sufficientemente grande, maggiore di $\frac{1}{c}$, ovvero se risulta limitata inferiormente da una costante strettamente maggiore di 0, come riportato nel passaggio (1). Un esempio di funzione asintoticamente non negativa che soddisfa la definizione è $f(n) = n$, mentre un esempio di funzione asintoticamente non negativa che non soddisfa la definizione è $f(n) = \frac{1}{n}$.

Per quanto riguarda la seconda affermazione, $f(n) + O(f(n)) = \Theta(f(n))$, dimostriamo che è sempre vera: il generico elemento $h(n) \in \{f(n) + O(n)\}$ possiamo scriverlo come $h(n) = f(n) + g(n)$, con $g(n) \in O(n)$; I passaggi illustrati in (2) dimostrano che $f(n) + O(f(n)) = O(f(n))\ \forall n \geq n_0$, mentre i passaggi illustrati in (3) dimostrano che $f(n) + O(f(n)) = \Omega(f(n))\ \forall n \geq n_0$.

$$\begin{aligned} g(n) \leq cf(n) \implies h(n) &= f(n) + g(n) \\ &\leq f(n) + cf(n) \\ &\leq (1+c)f(n) \end{aligned} \tag{2}$$

$$g(n) \geq 0 \implies f(n) + g(n) \geq f(n) \tag{3}$$

Possiamo concludere che il generico elemento $h(n) \in \{f(n) + O(f(n))\}$ è sia un elemento di $O(f(n))$ che un elemento di $\Omega(f(n))$, ragione per cui $\{f(n) + O(f(n))\} \subseteq \Theta(f(n)) \implies f(n) + O(f(n)) = \Theta(f(n))$.

Per quanto riguarda la terza affermazione, $f(n) = \Omega(g(n))$ e $f(n) = o(g(n))$, è sempre falsa. Intuitivamente, dire che $f(n) = \Omega(g(n))$ significa che $f(n)$ asintoticamente cresce almeno come cresce $g(n)$, mentre dire che $f(n) = o(g(n))$ significa che $f(n)$ è asintoticamente trascurabile rispetto a $g(n)$, e questo è chiaramente in contraddizione. Verifichiamolo tramite definizioni, assumendo $f(n)$ definitivamente non negativa e $g(n)$ definitivamente positiva:

$$\begin{aligned} f(n) = \Omega(g(n)) &\implies \exists c > 0, n > n_0 \mid f(n) \geq cg(n) \; \forall n > n_0 \\ f(n) = o(g(n)) &\implies \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \leftrightarrow \forall \epsilon > 0 \; \exists \; n_1 \in \mathbb{N} \text{ t.c. } \frac{f(n)}{g(n)} < \epsilon \; \forall \; n > n_1 \end{aligned} \tag{4}$$

Se dalla definizione del limite scegliamo $\epsilon = \frac{c}{2}$, risulta, per $n > \max\{n_0, n_1\}$, contemporaneamente $f(x) \geq cg(n)$ e $f(x) < \frac{c}{2}g(n)$, e questo è impossibile.

1.2 Esercizio 1.2

Traccia:
Per ognuna delle seguenti coppie di funzioni $f(n)$ e $g(n)$, trovare una appropriata costante positiva c tale che $f(n) \leq cg(n)$ per tutti i valori di $n > 1$.

- $f(n) = n^2 + n + 1, \quad g(n) = 2n^3;$
- $f(n) = n\sqrt{n} + n^2, \quad g(n) = n^2;$
- $f(n) = n^2 - n + 1, \quad g(n) = n^2/2$

Soluzione:
La prima costante c che soddisfi $f(n) = n^2 + n + 1 \leq cg(n) = 2cn^3$ può essere facilmente trovata tramite i passaggi presentati in (5), nei quali osserviamo che $\frac{1}{n} + \frac{1}{n^2} + \frac{1}{n^3}$ è una funzione definitivamente decrescente $\forall n > 1$, e in particolare assume il valore massimo proprio in $n = 1$.

$$\begin{aligned} n^2 + n + 1 \leq 2cn^3 \implies \frac{n^2 + n + 1}{n^3} \leq 2c &\implies \frac{1}{n} + \frac{1}{n^2} + \frac{1}{n^3} \leq 2c \\ 3 \leq 2c &\implies c \geq \frac{3}{2} \end{aligned} \tag{5}$$

Per questo motivo, anche se stiamo assumendo $n > 1$ strettamente, una costante opportuna potrebbe essere proprio $c = \frac{3}{2}$.

La seconda costante c che soddisfi $f(n) = n\sqrt{n} + n^2 \leq cg(n) = cn^2$ può essere trovata tramite lo stesso metodo presentato sopra, ovvero osservando nei passaggi (6) che la funzione $\frac{\sqrt{n}}{n} + 1 = \frac{1}{\sqrt{n}} + 1$ è una funzione decrescente $\forall n > 1$, e che assume valore massimo pari a 2 proprio in $n = 1$.

$$\begin{aligned} n\sqrt{n} + n^2 \leq cn^2 &\implies n\sqrt{n} \leq cn^2 - n^2 = n^2(c-1) \\ &\implies c \geq \frac{1}{\sqrt{n}} + 1 \end{aligned} \tag{6}$$

Per questo motivo, anche se stiamo assumendo $n > 1$, una costante opportuna potrebbe essere proprio $c = 2$.

La terza costante c che soddisfi $f(n) = n^2 - n + 1 \leq cg(n) = cn^2/2$ può essere trovata ancora tramite lo stesso metodo, ovvero osservando nei passaggi (7) che la funzione $1 - \frac{1}{n} + \frac{1}{n^2}$ è una funzione che per $n \to \infty$ viene limitata dal valore 1.

$$
\begin{aligned}
n^2 - n + 1 \leq \frac{cn^2}{2} &\implies 1 - n \leq \frac{cn^2}{2} - n^2 \\
&\implies 1 - n \leq n^2(\frac{c}{2} - 1) \\
&\implies \frac{c}{2} \geq 1 - \frac{1}{n} + \frac{1}{n^2}
\end{aligned} \tag{7}
$$

Per questo motivo, anche se stiamo assumendo $n > 1$, una costante opportuna potrebbe essere proprio $c = 2$, e in particolare c è la costante più piccola che garantisca la diseguaglianza $\forall n > 1$.

1.3 Esercizio 1.3

Traccia:
Dimostrare che per qualsiasi costante reale a e b, con b > 0, $(n + a)^b = \Theta(n^b)$.

Soluzione:

$$
(n + a)^b = n^b \left(1 + \frac{a}{n}\right)^b \tag{8}
$$

Osserviamo che se scegliamo $n_0 = 2|a|$ ad esempio, è soddisfatta la disuguaglianza $\frac{|a|}{n} < \frac{1}{2}\ \forall n > 2|a|$. Possiamo continuare con la (9):

$$
\frac{1}{2} \leq 1 - \frac{|a|}{n} \leq 1 + \frac{a}{n} \leq 1 + \frac{|a|}{n} \leq \frac{3}{2} \implies \frac{1}{2} \leq 1 + \frac{a}{n} \leq \frac{3}{2} \tag{9}
$$

Dato che b>0, possiamo elevare tutti i membri della catena di disequazioni (9) alla b senza alterare i segni, così come possiamo moltiplicare per n^b.

$$
\left(\frac{1}{2}\right)^b n^b \leq \left(1 + \frac{a}{n}\right)^b n^b \leq \left(\frac{3}{2}\right)^b n^b \implies \left(\frac{1}{2}\right)^b n^b \leq (n + a)^b \leq \left(\frac{3}{2}\right)^b n^b \tag{10}
$$

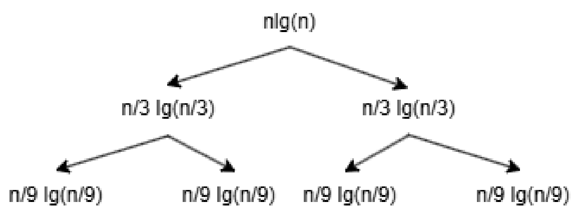
La dimostrazione risulta praticamente conclusa, infatti abbiamo dimostrato che la diseguaglianza finale in (10) è valida $\forall n > 2|a|$. Osserviamo che vale $\forall n > |a|$, e la scelta è stata dettata dalla semplicità espositiva.

1.4 Esercizio 1.4

Traccia:
Fornire il limite inferiore e superiore per $T(n)$ nella seguente ricorrenza, usando il metodo dell'albero delle ricorrenze ed il teorema dell'esperto se applicabile. Si fornisca il limite più stretto possibile giustificando la risposta.

- $T(n) = 2T(\frac{n}{3}) + n \lg n$;
- $T(n) = 3T(\frac{n}{5}) + \lg^2 n$;

Soluzione:
Per quanto riguarda la ricorrenza $T(n) = 2T(\frac{n}{3}) + n \lg n$, possiamo utilizzare l'albero delle ricorrenze presentato in figura (1.4) per ipotizzare una possibile forma chiusa.



Notiamo che il livello i-esimo dell'albero presenta un costo di $\left(\frac{2}{3}\right)^i n \lg\left(\frac{n}{3^i}\right)$. Con buona approssimazione possiamo ipotizzare che l'albero abbia profondita massima $\log_3 n$, dove vi sono $2^(log_3 n)$ foglie, il cui costo è T(1). Quindi, il costo totale dell'ultimo livello è $T(1)2^{\log_3 n} = T(1)n^{log_3 2} = \Theta(n^{log_3 2})$.

$$\begin{aligned}
T(n) &= \sum_{i=0}^{log_3 n-1} \left(\frac{2}{3}\right)^i n \lg \frac{n}{3^i} + \Theta(n^{log_3 2}) \\
&= n \lg n \sum_{i=0}^{log_3 n-1} \left(\frac{2}{3}\right)^i - n \lg n \sum_{i=0}^{log_3 n-1} \left(\frac{2}{3}\right)^i \lg 3^i + \Theta(n^{log_3 2}) \\
&= n \lg n \sum_{i=0}^{log_3 n-1} \left(\frac{2}{3}\right)^i - n \lg n \sum_{i=0}^{log_3 n-1} i \left(\frac{2}{3}\right)^i \lg 3 + \Theta(n^{log_3 2}) \\
&\leq n \lg n \sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^i + \Theta(n^{log_3 2}) \\
&\leq 3n \lg n + \Theta(n^{log_3 2})
\end{aligned} \tag{11}$$

Per questo motivo è possibile ipotizzare $T(n) = O(n \lg n)$. Osservando poi che la prima chiamata costa proprio $n \lg n$, possiamo restringere l'ipotesi a $T(n) = \Theta(n \lg n)$. L'ipotesi può essere verificata mediante il terzo caso del teorema dell'esperto:

$$n \lg n = \Omega(n^{\log_3 2+\epsilon});\ \text{se scegliamo } \epsilon = 0.27, \text{la condizione risulta verificata.} \tag{12}$$

$$\begin{aligned}
2\frac{n}{3} \lg \frac{n}{3} &\leq cn \lg n \\
\frac{2}{3} n \lg n - \frac{2}{3} n \lg 3 &\leq cn \lg n \\
\left(\frac{2}{3} - c\right) \lg n &\leq \frac{2}{3} \lg 3
\end{aligned} \tag{13}$$

Tramite (12) abbiamo verificato la prima ipotesi, mentre tramite (13) verifichiamo la seconda scegliendo come costante $0 < c = \frac{2}{3} < 1$ per n sufficientemente grande. Verificate le due ipotesi del terzo caso del teorema dell'esperto, possiamo concludere che l'ipotesi era corretta, ovvero $T(n) = \Theta(nlgn)$.

Per quanto riguarda la ricorrenza $T(n) = 3T(\frac{n}{5}) + \lg^2 n$, possiamo utilizzare il teorema dell'esperto per stabilire una forma chiusa per $T(n)$.

$$\lim_{n\to\infty} \frac{lg^k n}{n^\alpha}, \text{per } x = lgx \to \lim_{x\to\infty} \frac{x^k}{2^{\alpha k}} = 0 \quad \forall\alpha > 0,\ \forall k > 0. \tag{14}$$

Per quanto stabilito in 14, possiamo affermare che $\lg^k n = O(n^\alpha) \forall n > n_0$ e $\forall \alpha > 0, k > 0$. Nel nostro caso $\lg^2 n = O(n^\alpha)\ \forall\alpha > 0$. Poichè $log_5 3 \approx 0.68$, rientriamo nel primo caso del teorema dell'esperto, le cui ipotesi sono verificate in (15).

$$\lg^2 n = O(n^{0.68-\epsilon}) \text{ scegliendo ad esempio } \epsilon = 0.1 > 0 \to \lg^2 n = O(n^{0.58}) \tag{15}$$

La (15) risulta verificata per quanto asserito in (14). In conclusione, per il teorema dell'esperto risulta $T(n) = \Theta(n^{\log_5 3})$.

1.5 Problema 1.1

Traccia:
Si implementi un algoritmo di ordinamento che sfrutta l'inserimento e la visita in un albero binario di ricerca. Dato un vettore di n numeri interi in input, l'algoritmo procede prima ad inserire i numeri in un albero binario di ricerca (usando ripetutamente TREE-INSERT per inserire i numeri uno alla volta), e poi stampa i numeri in ordine con un attraversamento in ordine simmetrico dell'albero. Si analizzi la complessità nel caso peggiore e nel caso migliore per questo algoritmo di ordinamento.

Soluzione:
La soluzione presentata di seguito, così come gli altri problemi, è pensata in linguaggio c. Il file sorgente editabile "*problem1_1.c*" richiesto dalla consegna è presente nella directory *problems* presente nel corrente archivio. Nel file sorgente sono presenti anche le implementazioni delle procedure di cui qui espongo solo la firma e le strutture dati utilizzate.

```c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void tree_insert(Tree*, Node *);
5 void tree_init(Tree*);
6 void in_order_tree_walk(Node *);
7 void delete_tree(Tree *);
8
9 int main(int argc, char ** argv){
10     int n = atoi(argv[1]);
11     Tree * tree = (Tree*)malloc(sizeof(Tree));
12     tree_init(tree);
13
14     for(int i=0; i<n; i++){
15         Node * new_node = (Node*)malloc(sizeof(Node));
16         new_node->key=atoi(argv[i+2]);
17         new_node->left = NULL;
18         new_node->right = NULL;
19         tree_insert(tree, new_node);
20     }
21     in_order_tree_walk(tree->root);
22     delete_tree(tree);
23     free(tree);
24     return 0;
25 }
```

L'input viene passato tramite linea di comando, e consiste in un intero che indica la dimensione dell'array da ordinare, seguito dagli interi costituenti l'array.

Command Line

```
$ gcc problem1_1.c -o problem1_1.exe
$ ./problem1_1.exe 7 1 2 3 4 5 6 7
1 2 3 4 5 6 7

$ ./problem1_1.exe 7 12 4 3 0 1 2 45
0 1 2 3 4 12 45

$ ./problem1_1.exe 1 2
2

$ ./problem1_1.exe 0

$ ./problem1_1.exe 10 -1 -22 40 515 23 21 34 -541 99 0
-541 -22 -1 0 21 23 34 40 99 515
```

La complessità del ciclo for (linee 14-20) è determinata da n esecuzioni di tree_insert, che ha complessità O(h) dove h è l'altezza dell'albero. La complessità $T(n) = O(nh)$. Nel caso migliore, l'albero è bilanciato quindi $h = \lg n$, e la complessità risulta $T(n) = O(n \lg n)$. Nel caso peggiore, l'input è costituito da dati già ordinati (caso di test 1) e questo comporta un'albero la cui altezza è $h = n$, e la complessità risulta $T(n) = O(n^2)$. La procedura in_order_tree_walk ha complessità O(n) poichè visita tutti i nodi una sola volta, quindi la complessità totale è dominata dal ciclo for (linee 14-20).

2 Homework set 2

2.1 Esercizio 2.1

prova prova

2.2 Esercizio 2.2

prova prova