# Elementi di Algoritmi e Strutture Dati
## Testo della prova scritta del 22 luglio 2008
## docente: Luciano Gualà

Cognome:..... Nome:..... Matr.:..... Corso di Laurea:.....

**Esercizio 1 [8 punti]** Siano $f_1(n), f_2(n), g_1(n), g_2(n) : \mathbb{N} \to \mathbb{R}^+$ quattro funzioni. Dimostrare o confutare la seguente relazione

$$\max\{f_1(n), f_2(n)\} = \Theta(g_1(n) + g_2(n))$$

quando

**(a)** $f_1(n) = \Theta(g_1(n))$ e $f_2(n) = \Theta(g_2(n))$.

**(b)** $f_1(n) = \omega(g_1(n))$ e $f_2(n) = o(g_1(n))$ e $g_2(n) = \omega(f_2(n))$.

**Soluzione esercizio 1**

**(a)** Vera. Per ipotesi abbiamo che esistono sei costanti positive $c_1', c_2', n_0', c_1'', c_2'', n_0''$ tale che:

$$c_1' g_1(n) \leq f_1(n) \leq c_2' g_1(n) \quad \forall n \geq n_0';$$
$$c_1'' g_2(n) \leq f_2(n) \leq c_2'' g_2(n) \quad \forall n \geq n_0''.$$

Vogliamo trovare tre costanti positive $c_1, c_2, n_0$ tale che risulti

$$c_1(g_1(n) + g_2(n)) \leq \max\{f_1(n), f_2(n)\} \leq c_2(g_1(n) + g_2(n))$$

per ogni $n \geq n_0$. Si noti che per ogni $n \geq \max\{n_0', n_0''\}$, valgono le seguenti due catene di disuguaglianze:

$$\max\{f_1(n), f_2(n)\} \leq f_1(n)+f_2(n) \leq c_2' g_1(n)+c_2'' g_2(n) \leq \max\{c_2', c_2''\}(g_1(n)+g_2(n));$$

$$\max\{f_1(n), f_2(n)\} \geq \frac{1}{2}(f_1(n)+f_2(n)) \geq \frac{1}{2}(c_1' g_1(n)+c_1'' g_2(n)) \geq \frac{1}{2}\min\{c_1', c_1''\}(g_1(n)+g_2(n)).$$

Da cui segue la tesi scegliendo $n_0 = \max\{n_0', n_0''\}, c_1 = \frac{1}{2}\min\{c_1', c_1''\}, c_2 = \max\{c_2', c_2''\}$.

**(b)** Falsa. Un controesempio è il seguente insieme di funzioni: $f_1(n) = n^2, f_2(n) = 1, g_1(n) = n, g_2(n) = n^3$. Si noti che le relazioni date in $(b)$ sono rispettate; inoltre $\max\{f_1(n), f_2(n)\} = \max\{n^2, 1\} = n^2 \neq \Theta(n + n^3) = \Theta(g_1(n) + g_2(n))$.

**Esercizio 2 [8 punti]** Sia $A$ una matrice $m \times n$ nella quale gli elementi di ogni riga sono ordinati (in ordine crescente) da sinistra verso destra e gli elementi di ogni colonna sono ordinati (in ordine crescente) dall'alto verso il basso. Un esempio per $m = 4$ e $n = 3$ è il seguente:

$$A = \begin{pmatrix} 2 & 10 & 20 \\ 5 & 14 & 22 \\ 8 & 21 & 25 \\ 15 & 23 & 30 \end{pmatrix}$$

Realizzare un algoritmo che, preso in input una tale matrice $A$ e un valore $x$, verifichi in tempo $o(nm)$ se $x$ è presente in $A$ o meno. Si studi la complessità dell'algoritmo anche quando $m = \Theta(n)$. *Attenzione*: l'esercizio sarà valutato solo se corredato da adeguata descrizione del funzionamento dell'algoritmo, in base ai seguenti parametri: correttezza, efficienza e analisi di complessità.

**Soluzione esercizio 2** Di seguito discutiamo due soluzioni al problema. Per ognuna delle soluzioni daremo l'idea dell'algoritmo ad alto livello e ne discuteremo la complessità temporale. Gli pseudocodici sono lasciati per esercizio allo studente.

L'idea della prima soluzione è quella di sfruttare la proprietà di ordinamento delle righe e delle colonne della matrice per effettuare delle opportune ricerche binarie. Per esempio, un semplice algoritmo è il seguente: si considerano tutte le righe e in ogni riga si cerca l'elemento $x$ effettuando una ricerca binaria. Questa soluzione ha complessità $O(n \log m)$ perché per ognuna delle $n$ righe della matrice viene effettuata una ricerca binaria di costo $O(\log m)$. Un approccio del tutto simmetrico è quello di considerare le $m$ colonne della matrice e di effettuare una ricerca binaria su ogni colonna per verificare la presenza di $x$. Chiaramente la complessità temporale di questo approccio è $O(m \log n)$. Combinando i due algoritmi appena discussi, è possibile ottenerne un terzo che se $n \leq m$ esegue il primo algoritmo (ricerca binaria sulle righe), invece se $m < n$ esegue il secondo (ricerca binaria sulle colonne). La complessità di questo algoritmo è $O(\min\{n \log m, m \log n\})$. Si noti infine che tale soluzione ha complessità $O(n \log n)$ quando $m = \Theta(n)$.

La seconda soluzione usa la proprietà di ordinamento degli elementi in modo diverso. Prima di definire formalmente l'algoritmo, ne diamo l'idea attraverso un esempio. Si assuma di voler cercare nella matrice $A$ riportata sopra come esempio il valore 21. Partiamo dall'elemento in alto a destra, ovvero l'elemento di valore 20. Poiché $21 > 20$ e visto che la prima riga è ordinata in ordine crescente, sappiamo che l'elemento cercato non può essere presente nella prima riga della matrice. Allora possiamo ridurre il nostro problema di ricerca a quello di cercare l'elemento 21 in una nuova matrice che è ottenuta da quella vecchia eliminando la prima riga. A questo punto reiteriamo il procedimento e consideriamo l'elemento in alto a destra della nuova matrice, ovvero 22. Poiché $21 < 22$ e gli elementi delle colonne sono ordinati, siamo sicuri che l'elemento cercato non può essere nell'ultima colonna e quindi possiamo

ulteriormente ridurre la matrice eliminando l'ultima colonna. Procedendo in questo modo i successivi elementi della matrice che saranno considerati sono l'elemento 14 e successivamente l'elemento 21 (con cui la ricerca termina con successo).

Formalemente, l'algoritmo è il seguente. A ogni passo si considera l'elemento $A[i, j]$ (all'inizio $i = 1$ e $j = m$). Se $A[i, j] = x$ l'elemento è stato trovato e vengono ritornati i valori di $i$ e di $j$. Se, invece, $A[i, j] < x$ si decrementa $j$ di 1. Altrimenti, cioè quando $A[i, j] > x$, si incrementa $i$ di 1. Se $j$ diventa 0 o $i$ supera $n$, l'algoritmo si arresta e segnala che l'elemento $x$ non è presente nella matrice. Per mostrare la correttezza dell'algoritmo può essere dimostrato formalmente che ad ogni passo, se $A[i, j] \neq x$, allora $A[i', j'] \neq x$ per ogni $i' \leq i$ e $j' \geq j$ (la dimostrazione è lasciata come esercizio allo studente). Per quanto riguarda la complessità dell'algoritmo, si noti che ad ogni passo viene considerato un elemento, e o si incrementa $i$ o si decrementa $j$. Quindi non possono essere fatti più di $n$ incrementi e più di $m$ decrementi. Quindi la complessità è $O(m + n)$. Si noti che quando $m = \Theta(n)$ questa seconda soluzione ha complessità $O(n)$ ed è preferibile alla prima.

**Esercizio 3 [8 punti]**

**(a)** A partire da un albero AVL vuoto, si mostrino le modifiche apportate all'albero in seguito ai seguenti inserimenti: 20, 10, 15, 30, 40, 25. Si cancelli poi il nodo con chiave 20.

**(b)** Si risolva la seguente equazione di ricorrenza: $T(n) = 2T(n/2)+n \log n$, $T(1) = 1$.

**Soluzione esercizio 3** Per il punto $(a)$ si veda la figura. Per quanto riguarda il punto $(b)$ si noti come prima cosa che non è possibile applicare il teorema Master. Infatti la funzione $f(n) = n \log n = \omega(n^{\log_2 2})$, ma $f(n) \neq \Omega(n^{\log_2 2+\epsilon})$, per ogni $\epsilon > 0$. Risolviamo la relazione di ricorrenza utilizzando il metodo dell'albero della ricorsione. E' facile vedere che ogni nodo a livello $i$ ha dimensione $n/2^i$ e che tali nodi sono esattamente $2^i$. Poiché per risolvere ognuno di tali nodi spendiamo $n/2^i \log(n/2^i)$, ogni livello ci costa $n \log(n/2^i) \leq n \log n$. Dato che i livelli sono $O(\log_2 n) = O(\log n)$, possiamo affermare che $T(n) = O(n \log^2 n)$.

**Esercizio 4 [8 punti]** Si descriva in modo sintetito e preciso l'algoritmo `Fibonacci6` che, preso in input un intero $n$, calcola l'$n$-esimo numero di Fibonacci in tempo $O(\log n)$. Si analizzi in particolare la sua correttezza e si derivi la sua complessità temporale.

**Soluzione esercizio 4** Si consulti il libro di testo.

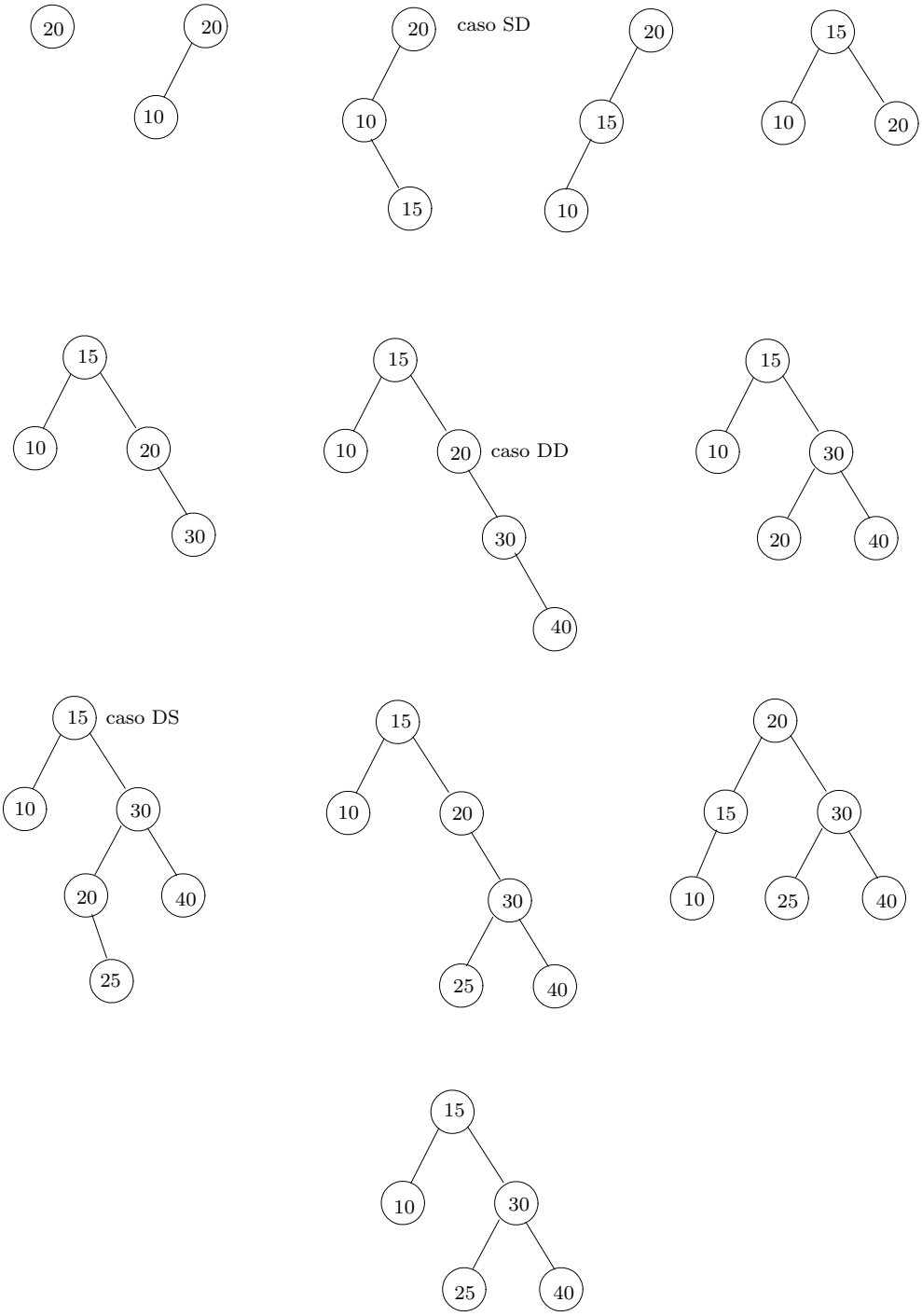


Figura 1: I primi 11 alberi mostrano le operazioni di inserimento. Per ogni inserimento, qualora l'albero risultasse sbilanciato, è indicato vicino al nodo critico il caso che si verifica e le corrispondenti rotazioni singole o doppie. L'ultimo albero, invece, mostra la struttura dati dopo l'eliminazione della chiave 20 (è stato eliminato fisicamente il predecessore di 20).