Prof. L. Margara **Scritto del 2/2/2024**

Algoritmi e Strutture Dati

Esercizio 1.[11 punti]
Risolvere in ordine di grandezza la seguente equazione ricorsiva

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1, \\ T(n/5) + 7^n & \text{if } n > 1. \end{cases}$$

Traccia della soluzione dell'esercizio 1.
$T(n)$ è banalmente $\Omega(7^n)$.
Per induzione è possibile anche dimostrare che $T(n)$ è $O(7^n)$
$T(n) = T(n/5) + 7^n \leq k2^{n/5} + 7^n$
Esiste k tale per cui $k2^{n/5} + 7^n \leq k7^n$?
Risolvendo la disequazione otteniamo che ciò è vero per $k \geq \frac{7^n}{7^n - 7^{n/5}}$
Visto che

$$\lim_{n->\to\infty} \frac{7^n}{7^n - 7^{n/5}} = 1$$

ciò significa che per tutti gli n maggiori di un opportuno n_0 la quantità $\frac{7^n}{7^n - 7^{n/5}}$ sarà minore di 2. A questo punto basta prendere $k = 2$ e il gioco è fatto !

Esercizio 2.[10 punti]
Supponiamo di avere a disposizione un algoritmo A capace di sommare o moltiplicare due matrici in tempo costante. Trovare un algoritmo con costo computazionale $O(n^2)$ che, presa in input la matrice M di adiacenza di un grafo non orientato e non pesato con n nodi e m archi, restituisca il numero di coppie di nodi (u, v) tali per cui $u \neq v$ e $d(u, v) \leq 5$ (dove $d(u, v)$ è la distanza tra u e v).

Traccia della soluzione dell'esercizio 2.
Basta calcolare $S = M + M^2 + M^3 + M^4 + M^5$ (costo computazionale costante usando A) e poi restituire il numero di elementi di S maggiori di zero (costo n^2).

Esercizio 3.[11 punti]
Fornire un algoritmo (quello con il costo computazionale più basso possibile nel caso pessimo) che dati in input un albero binario di ricerca T e due numeri interi positivi a e b con $a \leq b$ ritorni il numero di elementi compresi strettamente tra a e b.
Discutere le correttezza e il costo computazionale dell'algoritmo proposto.

Traccia della soluzione dell'esercizio 3.
Qual è il caso pessimo?
Se ogni nodo avesse solo il figlio sinistro (tranne l'unica foglia che non ha figli) avremmo il massimo memorizzato nella radice e il minimo memorizzato nell'unica foglia. Se il massimo fosse b e il minimo fosse a, qualunque algoritmo per calcolare la soluzione dovrebbe (partendo dalla radice) scorrere tutti gli elementi fino ad arrivare alla foglia. Quindi il tempo necessario sarebbe almeno pari al numero dei nodi dell'albero.
L'algoritmo che ordina il BST (costo lineare) e che poi calcola gli elementi compresi tra a e b (costo lineare) è quindi ottimo nel caso pessimo.