Quarta Esercitazione di Algoritmi 1

Beniamino Accattoli

8 ottobre 2007

1 Strumenti per l'esercitazione

Teorema fondamentale delle ricorrenze: la relazione di ricorrenza

$$T(n) = \begin{cases} a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \\ 1 & n = 1 \end{cases}$$

dove a e b sono costanti positive, ed f è una funzione sempre positiva, ha soluzione

1. $T(n) = \Theta\left(n^{\log_b a}\right)$, se $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ per $\epsilon > 0$
2. $T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$, se $f(n) = \Theta\left(n^{\log_b a}\right)$
3. $T(n) = \Theta\left(f(n)\right)$, se $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ per $\epsilon > 0$ e vale $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ per una costante $c < 1$ positiva e per n sufficentemente grande

2 Esercizi

1. Risolvere le seguenti ricorrenze utilizzando il teorema fondamentale:
   (a) $T(n) = 4T\left(\frac{n}{2}\right) + n^2$
   Risposta: vale $\log_b a = \log_2 4 = 2$ bisogna ora capire quale caso del teorema applicare, ovvero bisogna capire se $f(n) = n^2 = O\left(n^{2-\epsilon}\right)$, se $f(n) = \Theta\left(n^2\right)$ o se $f(n) = \Omega\left(n^{2+\epsilon}\right)$, con $\epsilon > 0$. Chiaramente vale $n^2 = \Theta\left(n^2\right)$, quindi è il secondo caso, e dunque $T(n) = \Theta\left(n^2 \cdot \log n\right)$
   (b) $T(n) = 4T\left(\frac{n}{2}\right) + n$
   Risposta: qui si ha $n = O\left(n^{2-\epsilon}\right)$ per ogni ϵ tale che $0 < \epsilon \leq 1$. Quindi è il primo caso e si ha $T(n) = \Theta\left(n^2\right)$
   (c) $T(n) = 3T\left(\frac{n}{3}\right) + n$
   Risposta: vale $\log_b a = 1$ e $n = \Theta(n)$, quindi è il primo caso del teorema e si ha $T(n) = \Theta(n \cdot \log n)$
   (d) $T(n) = 4T\left(\frac{n}{2}\right) + n^3$
   Risposta: si ha $n^3 = \Omega(n^{2+\epsilon})$ per $0 < \epsilon \leq 1$. Quindi può essere il terzo caso. Bisogna verificare anche che $4\left(\frac{n}{2}\right)^3 \leq c \cdot n^3$ per una costante c tale che $0 < c < 1$. Poiché $4\left(\frac{n}{2}\right)^3 = 4\frac{n^3}{8} = \frac{n^3}{2}$ allora prendendo $c = 1/2$ si soddisfa la disequazione. Quindi è il terzo caso e vale $T(n) = \Theta\left(n^3\right)$

(e) $T(n) = 4T\left(\frac{n}{2}\right) + n \cdot \log n$
Risposta: si ha $n \cdot \log n = O\left(n^{2-\epsilon}\right)$ per $0 < \epsilon < 1$ (attenzione: $\epsilon = 1$ non va bene). Cioè è il primo caso, ovvero $T(n) = \Theta\left(n^2\right)$

(f) $T(n) = T\left(\frac{n}{3}\right) + n \cdot \log n$
Risposta: $\log_b a = 0$ e $n \cdot \log n = \Omega\left(n^{0+\epsilon}\right)$ per $0 < \epsilon \leq 1$. Possibilità del terzo caso: bisogna verificare che esiste $0 < c < 1$ tale che $\frac{n}{3} \cdot \log \frac{n}{3} \leq c \cdot n \cdot \log n$. Basta prendere $c = 1/3$. Dunque siamo nel terzo caso e vale $T(n) = \Theta(n \cdot \log n)$

(g) $T(n) = 2T\left(\frac{n}{2}\right) + n \cdot \log n$
Risposta: $\log_b a = 1$ in questo caso il teorema non si può applicare, infatti $n \cdot \log n$ non è un $O\left(n^{1-\epsilon}\right)$ per nessun ϵ positivo, perché difatti vale il contrario cioè $n \cdot \log n = \Omega\left(n^{1-\epsilon}\right)$ per $0 < \epsilon \leq 1$. Non è neanche vero che $n \cdot \log n = \Theta(n)$ perché il fattore logaritmo causa $n \cdot \log n \neq O(n)$. Infine non è neanche $n \cdot \log n = \Omega\left(n^{1+\epsilon}\right)$, perché anche qui vale il contrario, cioè $n \cdot \log n = O\left(n^{1+\epsilon}\right)$

(h) Per la ricorrenza $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n$ analizzare l'albero della ricorsione
Risposta: si deve disegnare un albero binario alla cui radice si pone n, la dimensione della chiamata iniziale. I suoi due figli hanno etichetta $1/4$ e $3/4$, ovvero il costo dato dalle prime due chiamate. I figli di $1/4$ saranno i costi delle due nuove chiamate generate in quel ramo della ricorsione, ovvero $1/4 \cdot 1/4 = 1/16$ e $1/4 \cdot 3/4 = 3/16$. Mentre i figli di $3/4$ saranno $3/4 \cdot 1/4 = 3/16$ e $3/4 \cdot 3/4 = 9/16$. E così via per le altre chiamate. Bisogna notare che ad ogni livello la somma dei costi sui nodi fa n. Intuitivamente la complessità è data dal prodotto della profondità dell'albero, ovvero il numero dei livelli, per n, perché ad ogni livello si spende n. Questo però non è del tutto vero: rami diversi hanno profondità diverse. Tuttavia è facile vedere che il ramo più corto è quello in cui si sceglie sempre la chiamata che moltiplica per $1/4$. Quindi la complessità è minorata dalla profondità di tale ramo, ovvero $\log_4 n$, per n, e quindi si ha $T(n) = \Omega(n \cdot \log_4 n)$. Invece il ramo più lungo è quello che sceglie ripetutamente la chiamata che moltiplica per $3/4$. Anche questo ha profondità logaritmica, esattamente $\log_{4/3} n$. Quindi otteniamo $T(n) = O\left(n \cdot \log_{4/3} n\right)$. Ora sappiamo che la complessità logaritmica è indipendente dalla base del logaritmo (vedi esercizio sulla prima esercitazione) e quindi le due complessità coincidono e si ottiene$T(n) = \Theta\left(n \cdot \log n\right)$. Si noti che questa stima è stata possibile perché la somma dei costi per un livello è indipendente dal livello. Se si cerca di calcolare la ricorrenza $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n^2$, per la quale questa proprietà non vale, ci si rende presto conto che è molto più complicato.

2. (a) Considerare la seguente sequenza di numeri:

$$1 \quad 3 \quad 15 \quad 32 \quad 80 \quad 42 \quad 22 \quad 40 \quad 60 \quad 93 \quad 81 \quad 20$$

Assumendo che siano memorizzati in un array secondo la rappresentazione posizionale, discutere, motivando la risposta, se la sequenza rappresenta un heap minimo (ovvero con il minimo alla radice e tale

che i figli di un nodo sono maggiori o uguali al nodo stesso).
Risposta: no non è un heap minimo, infatti 20 è figlio di 42 ma gli è minore.

(b) Sia data un'istruzione $espandi(v)$ che prende un vettore v e ne aumenta la dimensione di 1, introducendo una locazione in più alla fine, in tempo costante. Sia H un heap massimo nella sua rappresentazione posizionale (= vettoriale). Implementare una nuova operazione $insert(H, x)$ che prende un heap H, un valore x, ed inserisce correttamente x in H.
Risposta:

Insert(H,x)
1. Espandi(H)
2. $i \leftarrow dim(H)$
3. $H(i) \leftarrow x$
4. **while** $(H(\lfloor i/2 \rfloor) < H(i))$ AND $(i > 1)$ **do**
5. scambia $H(\lfloor i/2 \rfloor)$ e $H(i)$
6. $i \leftarrow \lfloor i/2 \rfloor$

(c) Qual'è la complessità nel caso peggiore dell'operazione appena implementata? Supporre di costruire un heap H inserendo ripetutamente gli elementi con l'operazione $insert(H, x)$. Qual'è la complessità nel caso migliore di tale costruzione? Cercare di stimare l'andamento asintotico per il caso peggiore.
Risposta: il caso peggiore è quando si vuole inserire un numero maggiore di tutti gli elementi dell'heap, ovvero un massimo. In quel caso dovendo partire dalla fine e arrivare all'inizio dell'array dividendo ogni volta per due il costo è $\Theta(\log n)$. Nel costruire l'heap il caso migliore è quello in cui l'ordine di inserimento è proprio quello degli elementi nell'heap, quindi per ogni elemento si fa un lavoro costante, ed essendoci n elementi $\Theta(n)$. Il caso peggiore è invece quando ogni elemento che inseriamo è maggiroe di tutti quelli precedentemenete inseriti. In tal caso la complessità è data da $\sum_{i=1}^{n} \log_2 i$. Tale sommatoria è maggiorata da $\sum_{i=1}^{n} \log_2 n = \log_2 n \cdot \sum_{i=1}^{n} 1 = n \cdot \log_2 n$. Quindi la complessità è $O(n \cdot \log n)$. Si può essere più precisi infatti

$$\sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 \frac{n}{2} =$$

$$= \log_2 \frac{n}{2} \cdot \sum_{i=n/2}^{n} 1 = \frac{n}{2} \cdot \log_2 \frac{n}{2} = \frac{n}{2} \cdot \log_2 n - \frac{n}{2}$$

e quindi la complessità è anche $\Omega(n \cdot \log n)$, e quindi è $\Theta(n \cdot \log n)$

3. Il seguente algoritmo:

alg. elemInCom(intero dim, array A e B, interi i_{sin} e i_{des})$\rightarrow$ *booleano*
1. **if** $(i_{sin} = i_{des})$ **then**
2. **for** $k = 1$ **to** dim **do**

3. **if** $(A[k] = B[i_{sin}])$ **then return** true
4. **return** false
5. **else**
6. $i_{med} \leftarrow \left\lfloor \frac{i_{sin}+i_{des}}{2} \right\rfloor$
7. **if** (elemInCom$(dim, A, B, i_{sin}, i_{med})$) **then return** true
8. **else return** elemInCom$(dim, A, B, i_{med}+1, i_{des})$

prende in input due array A e B di n elementi e restituisce *true* se A e B hanno un elemento in comune, $false$ altrimenti. Si assuma che la prima posizione degli array abbia indice 1. La chiamata iniziale è $elemInCom(n, A, B, 1, n)$.

(a) Per capire come funziona l'algoritmo eseguirlo sui due array $A = [1, 2, 3, 4]$ e $B = [0, 7, 2, 10]$

(b) Qual'è il tempo d'esecuzione del passo base nel caso peggiore? E nel caso migliore? Supporre $n > 1$ e descrivere il caso migliore totale (cioè non limitato al passo base), poi scrivere la ricorrenza (per il caso migliore) e risolverla
Risposta: il passo base nel caso peggiore costa $\Theta(n)$, che è quando $A[dim] = B[i_{sin}]$. Invece il caso migliore del passo base è quando $A[1] = B[i_{sin}]$ è in quel caso costa $\Theta(1)$, ovvero ha costo costante. Il caso migliore totale è quando $A[1] = B[1]$. Infatti in quel caso la chiamata alla linea 8 non viene mai fatta, quindi non si effettuano mai due chiamate ricorsive all'interno della stessa procedura, e il caso base costa $\Theta(1)$. Per scrivere la ricorrenza bisogna notare che ad ogni chiamata ricorsiva la dimensione della finestra da analizzare, ovvero $i_{des} - i_{sin} + 1$, dimezza, ed inizialmente è pari ad n. E il caso base è quando tale finestra è costituita da un solo indice. Quindi si può scrivere la ricorrenza

$$T(n) = \begin{cases} T(n/2) + c_1 \cdot n & n > 1 \\ c_2 & n \leq 1 \end{cases}$$

Per questa ricorrenza si può usare il teorema fondamentale: sia ha $\log_b a = 0$ e $c_1 = f(n) = \Theta\left(n^0\right) = \theta(1)$ quindi è il secondo caso e vale $T(n) = \Theta\left(n^0 \cdot \log n\right) = \Theta(\log n)$

(c) Descrivere il caso peggiore, scriverne la ricorrenza e risolverla
Risposta: il caso peggiore si ha quando l'unico elemento in comune è $A[dim] = B[dim]$. In tal caso vengono sempre effettuate due chiamate ricorsive, entrambe della stessa dimensione, ed il passo base costa $\Theta(n)$. Quindi si ha la ricorrenza

$$T(n) = \begin{cases} 2T(n/2) + c_1 \cdot n & n > 1 \\ \Theta(n) & n \leq 1 \end{cases}$$

Per questa non si può usare il teorema fondamentale perché il passo base non ha costo costante. Per iterazione si trova che la k-esima iterazione ha forma:

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + c_1 \cdot \sum_{i=0}^{k-1} 2^i$$

il caso base si raggiunge quando $k = \log_2 n$. Sostituendo questo valore, e svolgendo i calcoli, si trova che è un $\Theta\left(n^2\right)$

4. Siano $f(n)$ e $g(n)$ due funzioni sempre positive. Dimostrare, o confutare con un controesempio, le seguenti affermazioni:

   (a) se $f(n) = O\left(g\left(n\right)\right)$ allora $2^{f(n)} = O\left(2^{g}\left(n\right)\right)$
   Risposta: l'affermazione è falsa, infatti se $f(n) = 2n$ e $g(n) = n$ si ha $f(n) = O\left(g\left(n\right)\right)$, ma $2^{f(n)} = 2^{2n} = \left(2^2\right)^n = 4^n = \Omega\left(2^{g(n)}\right) = \Omega\left(2^n\right)$

   (b) $f(n) = \Theta\left(\frac{f(n)}{3}\right)$
   Risposta: l'affermazione è vera: nelle definizioni di Ω e O basta prendere $c = 1/3$ e banalmente si ha $f(n) = O\left(\frac{f(n)}{3}\right)$ e $f(n) = \Omega\left(\frac{f(n)}{3}\right)$, ovvero $f(n) = \Theta\left(\frac{f(n)}{3}\right)$.

   (c) $f(n) = \Theta\left(f\left(\frac{n}{3}\right)\right)$
   Risposta: l'affermazione è falsa. Ad esempio prendendo $f(n) = 2^n$ si ha $2^n \neq \Theta\left(2^{n/3}\right)$

   (d) $max\{f(n), g(n)\} = \Theta\left(g\left(n\right) + f(n)\right)$
   Risposta: l'affermazione è vera. Si ha che $max\{f(n), g(n)\} \leq f(n) + g(n)$ per ogni n e quindi $max\{f(n), g(n)\} = O\left(g\left(n\right) + f(n)\right)$. Invece $max\{f(n), g(n)\} \geq \frac{1}{2}\left(f(n) + g(n)\right)$ per ogni n e quindi $max\{f(n), g(n)\} = \Omega\left(g\left(n\right) + f(n)\right)$. Dunque $max\{f(n), g(n)\} = \Theta\left(g\left(n\right) + f(n)\right)$

   (e) se $f(n) \leq g(n)$ per ogni $n > 0$, allora $g(n) - f(n) = \Theta\left(g\left(n\right)\right)$
   Risposta: l'affermazione è falsa. Infatti se $f(n) = n$ e $g(n) = n + 1$ allora $f(n) \leq g(n)$ per ogni n, ma $g(n) - f(n) = n + 1 - n = 1 \neq \Theta\left(n\right) = \Theta\left(g\left(n\right)\right)$