Introduzione agli Algoritmi!
19 Giugno 2014!
Prof. Emanuela Fachini (canale 1) e Prof. Irene Finocchi (canale 2)!

Parte I

Le risposte non motivate non saranno prese in considerazione.
Negli esercizi di progettazione, prima di passare allo pseudocodice descrivete l'idea algoritmica soggiacente. Per tutti gli algoritmi progettati è necessario analizzare tempo di esecuzione e correttezza.

Esercizio 1

Si consideri la seguente funzione:

```
test (intero n)
    if n≤81 then return 1
    k = 1
    h = 1
    while k ≤ n do
        for j=1 to k do h++
        k = k+2
    return 9*h + test(n/3)
```

Scrivere la relazione di ricorrenza che descrive il tempo di esecuzione T(n) della funzione test e la si risolva.

Soluzione:

Il codice all'esterno della chiamata ricorsiva consiste, al di là di operazioni di costo costante, di due cicli annidati; l'operazione di costo costante su h viene eseguita 1 volta, poi 3, poi 5, poi 7, dobbiamo quindi sommare i dispari da 1 a n, ottenendo $(n^2-1)/4+(n-1)/2 +1$. Le istruzioni al di fuori della chiamata hanno una complessità asintotica $O(n^2)$. C'è una sola chiamata ricorsiva su 1/3 degli elementi, quindi la relazione di ricorrenza è:
$T(n) = T(n/3) + O(n^2)$.
Per risolverla si aggiunge il caso base e si elimina la notazione asintotica, introducendo due costanti positive a e b; a dà conto del costo del controllo su n e b dà conto del costo delle operazioni all'esterno della chiamata:

$T(n) = a$ se $n \leq 81$
$T(n) = T(n/3) + bn^2$ se $n > 81$

Qualche passo di iterazione può suggerire la soluzione:
$T(n) = T(n/3^2) + b(n/3)^2 + bn^2 = T(n/3^3) + b(n/3^2)^2 + b(n/3)^2 + bn^2$

Si può dedurre dunque che all'i-simo passo:
$T(n) = T(n/3^i) + b(n/3^{i-1})^2 + \ldots + b(n/3)^2 + bn^2$
Poiché la relazione vale fino a $n>81$, preso $n = 3^h$ e imponendo che $3^h/3^i = 3^4$ si ottiene che $T(n) = T(3^{h-4}) = a$, quindi $T(n) = a + b(n/3^{h-5})^2 + \ldots + b(n/3)^2 + bn^2$. La sommatoria $b(n/3^{h-5})^2 + \ldots + b(n/3)^2 + bn^2$ si può scrivere come segue: $bn^2[(1/3^2)^{h-5} + \ldots + (1/3)^2 + 1] = O(n^2)$. L'ultimo passaggio è giustificato dal fatto che la serie geometrica di ragione 1/9 converge a una costante.
Volendo verificare la correttezza del risultato, si procede per induzione. Si tratta di dimostrare che esistono due costanti positive n_0 e d tali che $T(n) \leq dn^2$ per $n \geq n_0$.
Passo induttivo: supponiamo che $T(m) \leq dm^2$, per $m < n$ e cerchiamo di determinare la costante positiva d. Si applica l'ipotesi induttiva a T(n/3):
$T(n) = T(n/3) + bn^2 \leq d(n/3)^2 + bn^2 \leq d/9n^2 + bn^2$
Si vuole determinare d in modo tale che $d/9n^2+bn^2 \leq dn^2$ semplificando e portando tutti i termini con d a secondo membro si ha
$9b \leq 9d - d$ da cui $d \geq 9/8b$
Quindi un qualsiasi valore di d maggiore o uguale di 9/8b va bene.
Prendendo $d > a$ si ha che $T(n) = a \leq dn^2$ per $1 \leq n \leq 81$, e quindi si potrebbe prendere $d = a+2b$ e $n_0=1$e concludere la dimostrazione. Oppure si può osservare che per $n = 81$ si ha il caso base, quindi si considera $T(3*81) = T(81) + b\,(3*81)^2 = a + b\,(3*81)^2$.
Si vuole che $T(3*81) = a + b\,(3*81)^2 \leq d(3*81)^2$.
Questa disuguaglianza è vera per $d \geq a/(3*81)^2 + b$ e quindi si può prendere $d = a+2b$ e $n_0 = 3*81$.

Esercizio 2

Si progetti un algoritmo che, preso un array di interi, ne ridistribuisca i valori in modo tale che ogni elemento risulti maggiore dei suoi immediati vicini sinistro e destro o minore di essi.
Si dimostri la correttezza dell'algoritmo proposto motivando le scelte progettuali e se ne analizzi il tempo di esecuzione nel caso migliore e nel caso peggiore. L'algoritmo dovrebbe avere tempo di esecuzione $\Theta(n \log n)$ nel caso peggiore.
E' possibile risolvere il problema in tempo asintoticamente inferiore a $\Theta(n \log n)$? Motivare la risposta.

Soluzione (solo idea):
Per risolvere il problema si potrebbe ordinare l'array con un algoritmo che nel

caso peggiore garantisca la complessità richiesta, come il mergesort, e poi scambiare elementi di posto pari con quelli di posto dispari. Questo ha una complessità nel caso peggiore dominata da quella dell'ordinamento. Osservando che gli elementi nell'array devono essere o in sequenze crescenti o decrescenti o già soddisfare l'arrangiamento richiesto, si può semplicemente leggere gli elementi da sinistra verso destra e fare gli opportuni scambi per ottenere la proprietà richiesta tra elementi vicini. Questa soluzione ha un tempo lineare.

Parte II

Esercizio 1

A. Qual'è il minimo numero di nodi che può avere un albero AVL di altezza h? Si motivi la risposta.
B. Qual'è il massimo numero di nodi che può avere un albero AVL di altezza h? Si motivi la risposta.
C. E' vero che in un AVL il minimo si trova in una foglia o nel penultimo livello? Si motivi la risposta.
D. Si dimostri che un AVL ha altezza logaritmica.

Soluzione: per A e D si vedano appunti o i libri consigliati. Per B è chiaro che l'albero completo di altezza h, che è l'albero binario di altezza h con il massimo numero di nodi è anche un AVL e quindi il suo numero di nodi e il massimo anche per gli AVL.
Per C si consideri un AVL di 7 nodi con due nel sottoalbero sinistro e 4 nel destro in cui il nipote della radice nel sottoalbero sinistro sia figlio destro. In questo caso il minimo si trova in un nodo interno che non è al penultimo livello.

Esercizio 2

L'elemento mediano di un insieme di n elementi è l'elemento che occuperebbe la posizione n/2 se gli elementi fossero ordinati.

La struttura dati heap-a-clessidra mantiene un max-heap H_{max} e un min-heap H_{min} disposti "a clessidra" .

Assumete per semplicità che tutti gli elementi siano distinti. In un heap a clessidra valgono le seguenti proprietà:
- Il nodo di congiungimento dei due heap contiene l'elemento mediano;
- H_{min} è un min-heap che contiene tutti gli elementi maggiori del mediano, oltre al mediano stesso;
- H_{max} è un max-heap che contiene tutti gli elementi minori del mediano, oltre al mediano stesso.

Poiché la radice del min-heap, che è anche la radice del max-heap, deve essere il mediano dell'insieme degli elementi nei due heap, la seguente proprietà P deve essere sempre vera:
$|H_{max}| = |H_{min}|$ oppure $|H_{min}| = |H_{max}|+1$

(dove $|A|$ è la cardinalità dell'insieme A) a seconda che il numero totale di elementi distinti sia dispari ($|H_{max}| = |H_{min}|$) o pari ($|H_{min}| = |H_{max}|+1$).

I due heap H_{max} e H_{min} siano rappresentati in memoria tramite due strutture dati separate (l'elemento mediano è quindi ripetuto, mentre ogni altro elemento compare in uno solo dei due heap).

1. Come implementereste l'operazione di cancellazione del mediano in un heap a clessidra? Conviene utilizzare come subroutine le procedure viste a lezione per gli heap (max-heap e min-heap).

2. Analizzate il tempo di esecuzione, che dovrebbe essere in O(log n).

3. Analizzate inoltre la correttezza: dopo l'estrazione del mediano, la radice dei due heap deve contenere il nuovo mediano dell'insieme, che ora avrà un elemento in meno (in altre parole, la proprietà P deve rimanere vera).

Soluzione (idea):
Gi elementi del min-heap sono tutti maggiori di quelli del maxheap, tolta la radice che è un valore comune. Si estrae il minimo, y, dal min-heap, quando $|H_{min}| = |H_{max}|+1$, e si copia il nuovo minimo, x, nella radice del

max-heap per ottenere così di nuovo il mediano alla radice dei due heap, per i quali varrà $|H_{min}| = |H_{max}|$. Infatti x risulta per definizione minore dei restanti elementi nel min-Heap, ma anche maggiore dei restanti elementi del maxheap, visto x è maggiore di y .

Esempio:

H_{min} = 8 10 20 30 40 50
H_{max} = 8 3 5 2 1

Qui $|H_{min}| = |H_{max}| + 1$ estraendo 8 da H_{min} si ottiene
H_{min} = 10 30 20 50 40
e copiando 10 nel massimo di H_{max} :
H_{max} = 10 3 5 2 1

Ora $|H_{min}| = |H_{max}|$ e 10 è il nuovo mediano dell'insieme.

Altrimenti cioè se $|H_{min}| = |H_{max}|$, si deve estrarre il massimo, y, da H_{max} e copiare il nuovo massimo, x, nella radice di H_{min} per ottenere così di nuovo il mediano alla radice dei due heap, per i quali varrà $|H_{min}| = |H_{max}|+1$. Infatti x risulta per definizione maggiore dei restanti elementi nel max-Heap, ma anche minore dei restanti elementi del minheap, visto x è minore di y .

Esempio:
H_{min} = 10 20 30 40 50
H_{max} = 10 3 5 2 1
Qui $|H_{min}| = |H_{max}|$ estraendo 10 da H_{max} si ottiene
H_{max} = 5 3 1 2
e copiando 5 nel minimo di H_{min}:
H_{min} = 5 20 30 40 50
Ora $|H_{min}| = |H_{max}| + 1$ e 5 è il nuovo mediano dell'insieme.

La complessità è O(lg n) visto che si applicano operazioni con questo costo in funzione di un confronto sulle dimensioni dei max/min-heap più un assegnamento.