Introduzione agli algoritmi
31/5/2018
T. Calamoneri - E. Fachini - R. Petreschi

Le soluzioni degli esercizi scritte in modo illeggibile o in cui compaiano solo conti o pseudocodice senza commenti e risposte non motivate saranno valutati 0. Prima di descrivere un algoritmo in pseudocodice si deve delineare l'idea algoritmica. Inoltre deve essere precisato l'output atteso da eventuali singole funzioni utilizzate, oltre agli eventuali vincoli sul loro input (precondizioni).

1. Si risponda ad uno dei seguenti due quesiti:
   a) Si definiscano gli alberi di Fibonacci e si dimostri che la loro altezza è logaritmica nel numero dei nodi.

   b) Si dimostri che l'altezza di un albero rosso-nero è logaritmica nel numero dei nodi.

In aula si è chiarito che l'opzione b) riguarda gli studenti del corso di laurea in teledidattica. Per la risposta si consulti il libro di testo.

2. Si consideri la seguente funzione ricorsiva, si imposti la relazione di ricorrenza che ne esprime il tempo di esecuzione e la si risolva con il metodo della sostituzione.

```
MagicFunction (V, primo, ultimo)
input: V è un array di interi, primo e ultimo due interi che delimitano la porzione dell'array su cui la chiamata agisce, inizialmente primo =0 e ultimo = n-1, se gli elementi sono n.
ultimo - primo +1 = n
if n ≤ 9  return
for i = primo to  ultimo
   j = primo
   while j ≤ ultimo   do
         stampa (V [i] + V [j]);
         j = j+2
secondo = primo+n/3 -1;
terzo = secondo + n/3;
MagicFunction(V, primo, secondo);
MagicFunction(V, secondo+1, terzo);
MagicFunction(V, terzo+1, ultimo);
return
```

Sol 2.

Ci sono tre chiamate su circa un terzo degli elementi e il tempo di esecuzione delle istruzioni al di fuori delle chiamate è $\Theta(n^2/2)$ perchè si tratta di un ciclo eseguito n volte con al suo interno un ciclo eseguito n/2 volte, in tutti i casi.

$T(n) = 3T(n/3) + \Theta(n^2/2)$

Per risolverla esplicitiamo le costanti:
$T(n) = d$ se $n \leq 1$
$T(n) = 3T(n/3) + cn^2$

L'albero della ricorsione è un albero ternario completo se $n = 3^h$, e il costo di un nodo al livello i è $c(n/3^i)^2$, poichè i nodi sono 3^i il costo computazionale relativo al livello i è $3^i\, c(n/3^i)^2 = cn^2/3^i$. In conclusione
$T(n) = 3^hT(1) + \Sigma_{i=0\ldots h} cn^2/3^i =$
$nT(1) + \Sigma_{i=0\ldots h} cn^2(1/3)^i \leq nT(1) + \Sigma_{i=0\ldots\infty} cn^2(1/3)^i = O(n^2)$

Prova induttiva:
Si vuole dimostrare che esistono k e $n_0 \geq 0$ tali che $T(n) \leq kn^2$, per ogni $n \geq n_0$.
Supponiamo vera la tesi per ogni $m < n$ e consideriamo T(n). Poichè $n/3 < n$ possiamo applicare l'ipotesi induttiva a T(n/3) e quindi:
$T(n) \leq k(n/3)^2 + cn^2$ imponiamo che questo valore sia $\leq kn^2$ così otteniamo
$3k(n/3)^2 + cn^2 \leq kn^2 \Leftrightarrow k/3n^2 + cn^2 \leq kn^2 \Leftrightarrow 2kn^2 - 3cn^2 \geq 0$ questo è vero per $k = 2c$ e per ogni $n \geq 1$. Trascuriamo il caso base.

3. Dato un albero binario di ricerca T, a chiavi intere, e un intero k, si delinei un algoritmo che dà in output l'albero T privato dei nodi di chiave minore di k.

Sol 3.

Una prima soluzione utilizza la funzione Delete(T,x), che ha come risultato di cancellare il nodo x da T, inserita in una visita dell'albero, per cancellare i nodi di chiave minore di k. Se tutte le chiavi sono minori di k queste verranno cancellate a una a una con un tempo di esecuzione O(nh), questo è il caso peggiore. Nel caso migliore invece, sono tutte maggiori di k e quindi si ha il tempo della visita O(n). Ma in questa soluzione non si sfruttano appieno le proprietà degli ABR.
Innanzi tutto si potrebbe osservare che la cancellazione si può eseguire sempre come se il nodo avesse solo un figlio. Infatti, se x è un nodo di chiave minore di k allora anche tutti i nodi nel suo sotto albero sinistro hanno chiave minore di k e dunque si può direttamente rendere il figlio destro di x figlio del

padre di x.
Se la chiave del nodo in considerazione è minore di k, dopo aver cancellato il nodo, bisogna controllare se nel sotto albero destro ci sono nodi di chiave minore di k. Se invece la chiave del nodo è maggiore di k si deve scendere nel sotto albero sinistro alla ricerca di nodi con chiave minore di k.
Definiamo una funzione di cancellazione modificata di un nodo, visto che anche se il nodo da cancellare ha due figli, noi dobbiamo in ogni caso sostituire il nodo con il suo figlio destro.
Osserviamo che poiché cancelliamo i nodi scendendo dalla radice il nodo da cancellare è necessariamente un figlio sinistro, infatti o è il primo che si cancella e allora suo padre ha una chiave ≥ k e il nodo non può che essere suo figlio sinistro oppure deve diventare figlio di un nodo che ha rimpiazzato uno cancellato e che è un figlio sinistro.

```
DeleteMod(T,x)
input: un puntatore a un nodo di un albero binario T
prec: x è non  nullo
output: il figlio destro di x, dopo aver cancellato il nodo x
z = x.right
if x == T then T = z
          else  x.p.left = z
return z
```

```
CN(T,k)
input: un puntatore alla radice di  un albero binario T a chiavi intere e un intero k
output: l'albero T privato dei nodi con chiave minore di k
if  T== NULL then return NULL
if T.key == k then T.left = NULL return T
if T.key < k then T = T.right return T
z = T
while z ≠ NULL do
        if z.key < k then z = DeleteMod(z)
        if z.key == k then z.left = NULL return T  (uscendo dal ciclo)
        if z.key > k then z = z.left
return T
```

Qui si scende di padre in figlio in ogni esecuzione del ciclo, inoltre la funzione di cancellazione è eseguita in tempo costante, quindi il tempo di esecuzione è in O(h).