Algoritmi e Principi dell'Informatica
Seconda Prova in Itinere

13 Febbraio 2015

Avvisi importanti
Il tempo a disposizione è di **1 ora e 45 minuti.**

Esercizio 1 (punti 5/15-esimi)

Si calcoli la complessità del seguente frammento di codice che definisce l'"ossatura" di una funzione `f` applicata a un array `A`:

```
f(A):

n := A.length
i := 1
altre eventuali inizializzazioni eseguite in tempo costante
while i < n
  do_something_in_tempo_costante
  i := i*2
f(A[1..n/3])
f(A[n/3+1..(2/3)*n])
j := 2
while j < n*n
  do_something_in_tempo_costante
  j := j*j
return result
```

Esercizio 2 (punti 7/15-esimi)

Si consideri un albero rosso-nero di altezza nera *bh* e costituito da soli nodi neri.

1. Quali sono, rispettivamente, il numero minimo e massimo di nodi contenuti nell'albero? (si precisi se nel conto viene incluso il nodo T.NIL o no).
2. E' possibile ottenere da esso un nuovo albero rosso-nero costituito da soli nodi neri applicandovi *esclusivamente* operazioni di inserimento di nuovi nodi? In caso positivo in che modo? (eventualmente illustrandolo mediante un semplice esempio).
3. Qual è il numero massimo di nuovi nodi che si possono inserire in esso senza effettuare cancellazioni e senza alterare *bh*?

Si forniscano brevi ma chiare spiegazioni per le risposte date.

Esercizio 3 (punti 7/15-esimi)

Sia dato un grafo G e due sottoinsiemi dei suoi vertici, V_1 e V_2.
La distanza tra V_1 e V_2, $d(V_1, V_2)$, è la distanza minima tra un nodo appartenente a V_1 e un nodo appartenente a V_2. Se V_1 e V_2 non sono disgiunti allora la loro distanza è uguale 0.

Si specifichi un algoritmo che, a partire da un grafo G e da due suoi sottoinsiemi V_1 e V_2, calcoli la distanza tra V_1 e V_2; se ne discuta quindi la complessità. La valutazione dell'algoritmo proposto dipenderà ovviamente dalla sua complessità.
N.B.: Si può assumere che esista una funzione che in tempo costante determini l'appartenenza di un vertice a uno dei due insiemi.

Tracce di soluzioni

Esercizio 1

Lo pseudocodice ricorsivo dell'esercizio ha una funzione di complessità definita da una ricorrenza del tipo

$T(n) = 2T(n/3) + \log(n) + \log(\log(n))$

che si risolve mediante il master theorem ottenendo una funzione

$\Theta(n^{\log_3(2)})$.

Esercizio 2

Risposte

1. Posto che l'altezza nera si calcola contando il numero di nodi neri tra la radice, esclusa, e la foglia T.NIL, compresa e posto che ogni "cammino nero" dalla radice a T.NIL deve avere la stessa lunghezza, un albero rosso-nero di soli nodi neri è necessariamente perfettamente bilanciato; quindi il numero di nodi interni è $\sum_{i=0}^{bh-1} 2^i = 2^{bh} - 1$ cui va aggiunto l'unico T.NIL e il numero, sia minimo che massimo, di nodi totale è 2^{bh}.
2. Ogni nodo nuovo viene inserito con colore rosso; successivamente la procedura di FIXUP lascia in ogni caso almeno un nodo rosso. Quindi non è possibile ottenere alberi interamente neri esclusivamente mediante inserimenti (a meno che l'albero non consista nella sola radice).
3. Se l'albero è inizialmente costituito da soli nodi neri si possono inserire nuovi nodi senza alterarne la *bh*, quindi necessariamente rossi, fino al massimo a raddoppiarne l'altezza: *h = 2.bh*. Ciò può essere ottenuto anche eseguendo solo inserimenti: ad esempio l'albero di figura a può essere trasformato in quello di figura b eseguendo nell'ordine l'inserimento dei nodi 3, 7, 12, 20, 1. 4. 6. 8, 11, 13, 16, 22.

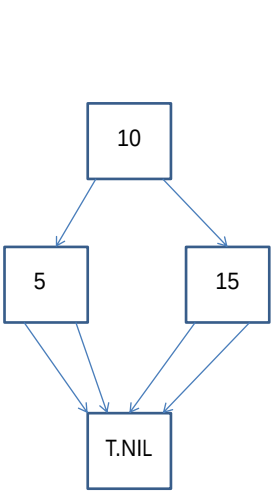

Figura a

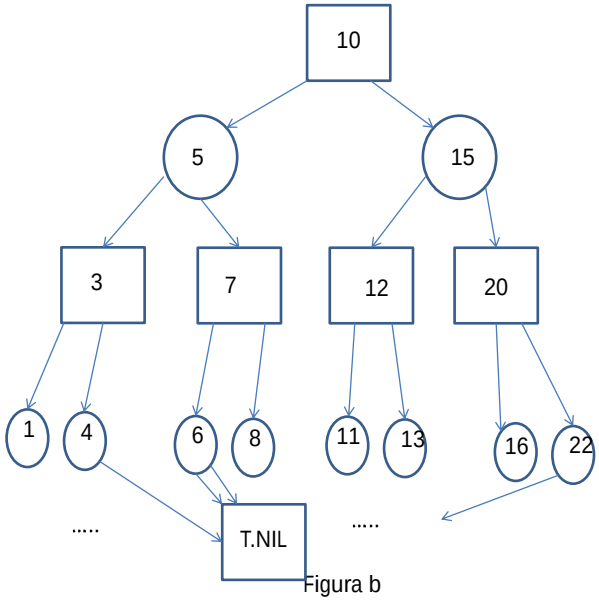

Figura b

Legenda: i nodi quadrati rappresentano nodi neri; quelli tondi nodi rossi.

Quindi il numero totale di nodi (compreso T.NIL) del nuovo albero di altezza *2.bh* è 2^{2bh} e il numero massimo di nodi inseribili è $2^{2bh} - 2^{bh}$.

Esercizio 3

Siano $n = |V|$ e $m = |E|$. È possibile effettuare una visita BFS a partire da ogni nodo in V_1. $|V_1| = O(n)$; quindi si eseguono $O(n)$ visite, ognuna di costo $O(m + n)$. Tale soluzione ha complessità $O(n(m+ n))$.

Una soluzione migliore consiste nel modificare la BFS in modo che tutti i nodi u in V_1 siano inseriti subito nella coda e la loro distanza, $u.dist$, sia posta a 0. L'algoritmo analizza quindi tutti i nodi adiacenti ai nodi in V_1 che non appartengono a V_1, li inserisce nella coda e pone la loro distanza uguale a 1. Se tra questi nodi è incluso un nodo di V_2, l'algoritmo termina restituendo 1 (la distanza calcolata). Altrimenti, si procede con l'estrazione dalla coda dei nodi a distanza 1 e con l'inserimento di tutti i nodi a distanza 2 da V_1. Vengono cosi progressivamente individuati tutti i nodi a distanza crescente dai nodi in V_1.

Per come è definita la BFS, il primo nodo in V_2 che si incontra durante la visita avrà distanza minima e tale distanza potrà essere restituita dall'algoritmo come valore di ritorno. Se la visita non trova alcun nodo dell'insieme V_2, allora V_2 non è raggiungibile da V_1; quindi l'algoritmo restituisce un valore opportuno (per es., ∞).

Essendo basto su una visita BFS, l'algoritmo ha complessità $O(m + n)$.