Introduzione agli algoritmi
Proff. T. Calamoneri – S. Caminiti - E. Fachini
11 giugno 2020

1. Si dimostri, utilizzando la definizione di *Θ, che*

$$f(n) = 50n^2 + lg^2\, n = \Theta(n^2)$$

mettendo in evidenza e commentando con chiarezza i passi seguiti.

Sol. Per dimostrare che f(n) è in $\Theta(n^2)$ dobbiamo trovare tre costanti positive c,c' e n' tali che $0 \le c\, n^2 \le 50n^2 + lg^2\, n \le c'\, n^2$ per ogni $n \ge n'$.

Trattiamo le due disuguaglianze separatamente.

1. $c\, n^2 \le 50n^2 + lg^2\, n$ è banalmente vero per $c = 1$, per ogni $n \ge 1$.

2. $50n^2 + lg^2\, n \le c'\, n^2$
Poichè $50n^2 + lg^2\, n \le 51n^2$, visto che la funzione quadratica cresce più in fretta della funzione logaritmica al quadrato concludiamo che l'affermazione è vera per c=1, c'= 51 e n' = 1.

2. Si imposti la relazione di ricorrenza che definisce il tempo di esecuzione della seguente funzione e la si risolva usando il metodo della sostituzione. Si commentino opportunamente i passaggi del calcolo, si disegni l'albero della ricorsione e come si giunge alla previsione sull'andamento del tempo di calcolo, si imposti l'induzione con chiarezza, sia nello scrivere quanto si vuole dimostrare sia nel formulare l'ipotesi induttiva.

```
Analizzzami(A,i,j)
input: A array di interi ,i e j interi positivi
n = j - i + 1
if n < 1 then return 0;
if n = 1 then return (A[1]);
m = n/3
for k =1 to  m do
 A[k] ← A[k] − A[1];
end for
x = Analizzami (A,i,i+m) + Analizzami(A, i + m+1,i+2m) +
     Analizzami(A,i+2m+1,j)
return(x);
```

Sol. La funzione è chiamata 3 volte su n/3 elementi e il ciclo while è eseguito n/3 volte dunque le istruzioni al di fuori della chiamata sono eseguite in tempo lineare, quindi la relazione di ricorrenza è

$T(n) = 3T(n/3) + \Theta(n)$.
Si deve quindi risolvere la ricorrenza
$T(n) = 3T(n/3) + c\,n$ se $n > 1$
$T(n) = d$ se $n \leq 1$

L'albero della ricorsione per fare una previsione sull'andamento è qui abbozzato:

$c\,n$

$c\,n/3 \qquad c\,n/3 \qquad c\,n/3$

$c\,n/3^2 \quad c\,n/3^2 \quad c\,n/3^2 \quad \ldots.$

Ponendo $n = 3^h$, l'albero ha altezza h, ogni nodo ha 3 figli e al livello i ogni nodo è etichettato con $c\,n/3^i$ Quindi, sommando su ogni livello si ha $3^i\, c\, n/3^i = cn$ da cui:
$T(n) = 3^h\, d + \sum_{i=0\ldots h-1} cn = 3^h\, d + hcn = O(n\log_3 n)$

Infatti il termine $n\log_3 n$ è dominante rispetto a nd, quindi $T(n) = O(n\log_3 n)$.

Ora verifichiamo la previsione per induzione.
Faremo vedere che esistono due costanti positive k e n' tali che
$T(n) \leq kn\log_3 n$, per ogni $n \geq n'$.
Sia vero per ipotesi induttiva per ogni $m < n$, consideriamo T(n):
$T(n) = 3T(n/3) + c\,n \leq$
$3k\,(n/3)\,\log_3 n/3 + c\,n = k\,n\,(\log_3 n - \log_3 3) + c\,n = k\,n\log_3 n - kn + c\,n.$

Vediamo se troviamo due valori per k e n' tali che
$k\,n\log_3 n - kn + c\,n \leq k\,n\log_3 n \Leftrightarrow c\,n \leq kn$ quest'ultima disuguaglianza è vera per $k \geq c$ e per ogni $n \geq 1$

Considerando il caso base abbiamo che $T(1) = d$ e $T(3) = 3d+c$ e
$T(3) \leq k3\log_3 3 = 3k$ è vero prendendo $k = d+c$.

Il termine additivo è lineare, quindi fornisce un limite inferiore minore del superiore. Verifichiamo se $T(n) = \Omega(n\log_3 n)$ per induzione.
Faremo vedere che esistono due costanti positive k' e n" tali che
$T(n) \geq k'n\log_3 n$, per ogni $n \geq n"$.
Sia vero per ipotesi induttiva per ogni $m < n$, consideriamo T(n):
$T(n) = 3T(n/3) + c\,n \geq$
$3k\,(n/3)\,\log_3 n/3 + c\,n = k\,n\,(\log_3 n - \log_3 3) + c\,n = k\,n\log_3 n - kn + c\,n.$
Vediamo se troviamo due valori per k' e n" tali che
$k'\,n\log_3 n - k'n + c\,n \geq k'\,n\log_3 n \Leftrightarrow c\,n \geq k'n$ quest'ultima disuguaglianza è vera per $k' \leq c$ e per ogni $n \geq 1$.
Considerando il caso base abbiamo che $T(1) = d$ e $T(3) = 3d+c$ e

$T(3) \geq k'3\log_3 3 = 3k'$ è vero prendendo $k' = c$, quindi $T(n) \geq cn \log_3 n$ per ogni $n \geq 3$. Quindi prendendo $k' = c$ e $n_0 = 3$ l'affermazione è vera. Concludiamo che $T(n) = \Theta(n \lg n)$.

3. Dati due numeri interi *x* ed *y*, definiamo la loro distanza come la loro differenza in valore assoluto dist(x, y) = |x − y|. Sia dato un albero binario di ricerca *T*, contenente chiavi intere. Si progetti un algoritmo il più efficiente possibile che determini le chiavi di *T* aventi distanza massima e se ne calcoli il tempo di esecuzione asintotico. Si descriva a parole l'idea algoritmica, si analizzi il tempo di esecuzione asintotico e si produca lo pseudocodice. Si discuta brevemente sul modo in cui (eventualmente) cambiano l'algoritmo ed il costo computazionale nel caso in cui *T* sia:

   - un albero bilanciato (AVL o rosso-nero);
   - un max-heap;

Sol. Sappiamo che la massima distanza in un insieme finito di interi è data dalla differenza in valore assoluto del massimo e del minimo. Poiché in un ABR è facile individuare il massimo e il minimo, rispettivamente come l'ultimo nodo senza figlio destro nel cammino più a destra e come l'ultimo nodo senza figlio sinistro nel cammino più a sinistra, si può risolvere il problema semplicemente utilizzando queste due funzioni. Il calcolo del minimo e del massimo è in O(h), infatti nel caso migliore la radice è il minimo o il massimo e in tal caso vuol dire che non ha il figlio sinistro, rispettivamente il destro e quindi il tempo di esecuzione è in Θ(1), mentre nel caso peggiore il minimo e il massimo sono due foglie di profondità massima e quindi il tempo di esecuzione è in Θ(h). Se l'albero è un AVL valgono le stesse considerazioni, ma le operazioni sono eseguite in Θ(lg n). Se invece l'albero è un max-heap allora il ragionamento cambia perchè in un max-heap il massimo è alla radice e quindi può essere determinato (non estratto) in Θ(1), mentre l'individuazione del minimo comporta Θ(n) passi, perchè il minimo può trovarsi in una qualsiasi delle foglie.

```
DistABR(T)
input: T è il puntatore a un albero binario
prec: T è un ABR
output: i puntatori alle due chiavi di T a massima distanza
p = minimum(T)
q = maximum(T)
return (p,q)
```

Le funzioni:

```
minimum(T)
input: T è il puntatore a un albero binario non nullo
prec: T è un ABR
output: il puntatore al nodo di chiave minima in  T
p = T
while p.left ≠ NIL do
        p = p.left
return p
```

```
e maximum(T)
input: T è il puntatore a un albero binario non nullo
prec: T è un ABR
output: il puntatore al nodo di chiave massima in  T
p = T
while p.right ≠ NIL do
        p = p.right
return p
```

Invece nel caso del max-heap si ha:

```
DistMax-heap(A)
input: A è un array di interi
prec: A[1..A.heapsize] è un max-heap
output:  le entrate nell'array del minimo e del massimo di A
n = A.heapsize
if n ≥ 1 then max = A[1] else return "max-heap vuoto"
min = n/2 +1
for i = n/2 + 2 to n do
    if A[i] < min  then min = i
return (min,max)
```