Introduzione agli algoritmi
Prof.sse T. Calamoneri - E. Fachini - R. Petreschi
18 Settembre 2019

1. Si consideri il problema di determinare gli *m* elementi più grandi in un array di *n* elementi e si confrontino dal punto di vista del tempo di esecuzione asintotico i due algoritmi qui sotto proposti:

   Algoritmo A. Si ordinano gli elementi e si prendono i primi *m*;

   Algoritmo B. Si costruisce un max-heap dall'array e poi si esegue per *m* volte l'estrazione del massimo.

   Si diano i tempi di esecuzione dei due algoritmi, dettagliando i tempi di esecuzione dei passi intermedi dei due algoritmi e motivando i risultati. Si consideri il caso in cui m=O(lg n).

Sol.

L'algoritmo A può essere realizzato nel caso peggiore in Θ(nlgn). Si potrebbe usare il mergesort o anche l'heapsort, che hanno tempo di esecuzione in Θ(nlgn) nel caso peggiore, e poi copiare in un array o stampare a video i primi m (o gli ultimi se è ordinato in ordine crescente), in tempo Θ(m). Poiché m≤n e quindi m≤nlgn, si può concludere che l'intero algoritmo lavora in Θ(nlgn). Nel caso in cui m = O(lg n) il risultato non cambia perchè l'operazione dominante è l'ordinamento di tutto l'array.

L'algoritmo B prevede come primo passo la costruzione di un maxheap a partire da un array qualunque, operazione che si può eseguire in O(n). L'estrazione del massimo poi costa O(lg n) e poiché si esegue m volte si ha un tempo complessivo O(mlgn). In conclusione si ha un tempo di esecuzione O(n) + O(mlgn). Sapere che m≤n porterebbe a concludere che anche in questo caso il tempo è O(nlgn), ma se m fosse in O(lg n) il termine dominante nella somma diventerebbe O(n), perchè $lg^2(n) = O(n)$. Infatti esistono k e $n_0 \geq 0$ tali che $lg^2(n) \leq kn$, per ogni $n \geq n_0$, anche per k=1, basta prendere $n_0 = 16$. Possiamo concludere che se m è abbastanza piccolo, in O(lg n), questo secondo algoritmo è lineare nel numero degli elementi dell'array dato e quindi preferibile al primo.

2. Scrivere e risolvere con il metodo della sostituzione l'equazione di ricorrenza che esprime il tempo di esecuzione della funzione **Analizzami**, dove *A* è un array di *n* interi:

```
Analizzami (A, n)
if n ≤ 2 then return A[1];
j = n;
while j ≥ 1 do
      {A[j] = A[j] + A[j-1];
      j= j – 4;}
```

Analizzami (*A*, n/4);

Sol.

Il ciclo while viene eseguito Θ(n/4) volte, le operazioni al suo interno sono eseguite in tempo costante quindi tutto il ciclo lavora in Θ(n). Anche le operazioni al di fuori del ciclo e della chiamata sono eseguite in tempo costante. Essendoci un'unica chiamata ricorsiva la relazione di ricorrenza che esprime il tempo di esecuzione dell'algoritmo Analizzami è

$T(n)= T(n/4) + \Theta(n)$

Per risolverla esplicitiamo le costanti e introduciamo il caso base:

$T(n) = T(n/4) + cn$ se $n > 4$, dove c e $d>0$

$T(n) = d$ altrimenti

Sviluppando la ricorsione, con l'ipotesi semplificative che $n= 4^h$, si ottiene:

$T(n)= T(n/4) + cn = T(n/4^2) + cn/4 + cn = T(n/4^3) + cn/4^2 + cn/4 + cn =$

$\ldots T(1) + cn/4^{h-1} + \ldots + cn/4 + cn.$

La sommatoria $\Sigma_{i=0\ldots h-1}\ cn/4^i = cn\Sigma_{i=0\ldots h-1}\ 1/4^i = cnO(1)$, visto che la somma si può maggiorare con la serie geometrica che converge a una costante.

Facciamo dunque la previsione che l'andamento sia lineare e dimostriamo che si tratta di una previsione corretta.

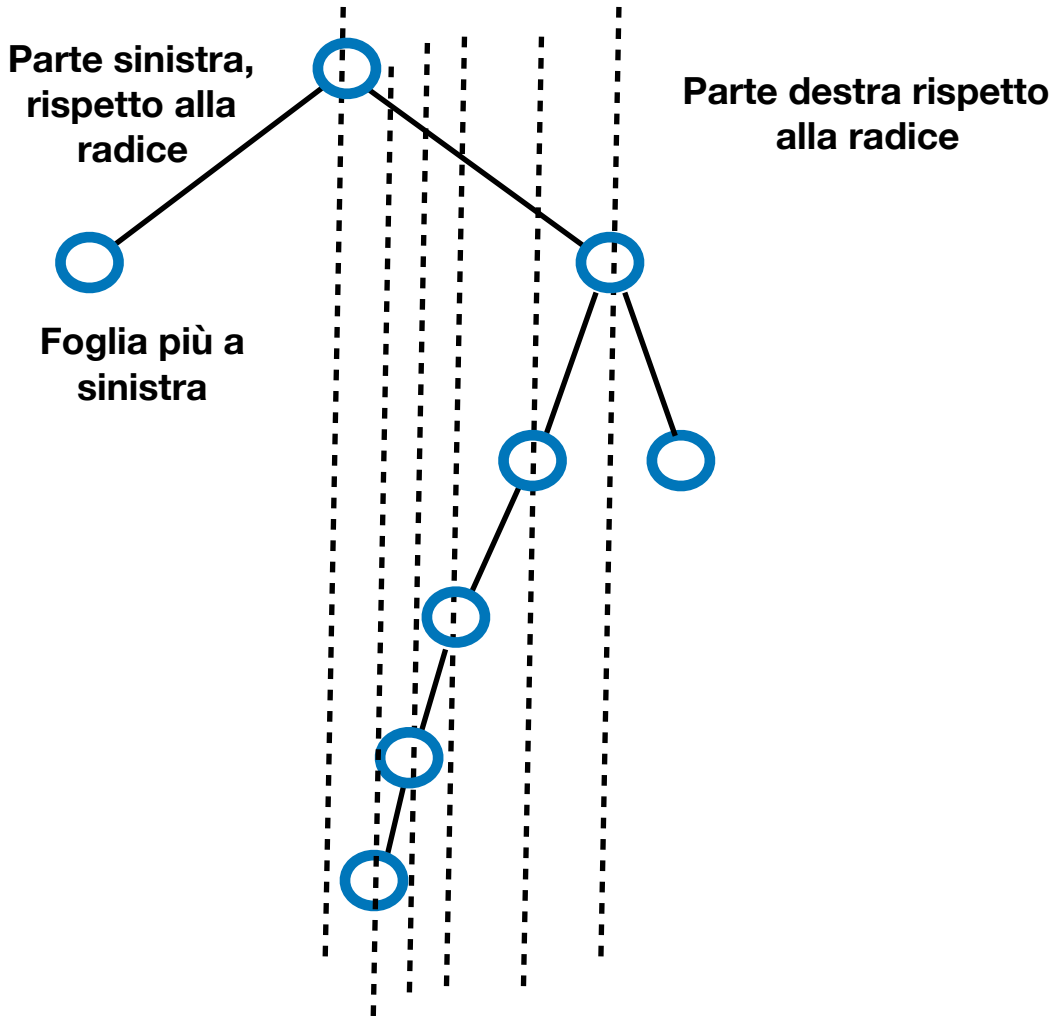
Prova induttiva:
Si vuole dimostrare che esistono k e $n_0 \geq 0$ tali che $T(n) \leq kn$, per ogni $n \geq n_0$. Supponiamo vera la tesi per ogni $m < n$ e consideriamo T(n). Poichè $n/4 < n$ possiamo applicare l'ipotesi induttiva a T(n/4) e quindi:
$T(n) = T(n/4) + cn \leq kn/4 + cn$, imponiamo che questo valore sia $\leq kn$ così otteniamo
$kn/4 + cn \leq kn \Leftrightarrow kn + 4cn \leq 4kn \Leftrightarrow 4cn \leq 3kn$ questo è vero per $k = 2c$ e per ogni $n \geq 1$. Per il caso base notiamo che $T(4) = T(1) + 4c = d + 4c$ e cerchiamo un valore di k per cui $d + 4c \leq 4k$. Basta prendere $k = 2(d+c)$ per soddisfare la disuguaglianza, possiamo concludere che $T(n) \leq 2(d+c)n$, per ogni $n \geq 4$.

3. Si definisca una funzione ricorsiva che prende in input un albero binario e restituisce il livello della foglia più a sinistra dell'albero. L'albero è rappresentato in memoria mediante strutture con tre campi: il puntatore al figlio sinistro, il puntatore al figlio destro e il campo chiave.

Dell'algoritmo presentato:

a. si dia la spiegazione a parole
b. si scriva lo pseudocodice
c. si dimostri che il tempo di esecuzione asintotico è lineare nel numero dei nodi dell'albero.

L'algoritmo prevede una visita in post order modificata nella quale si privilegia la chiamata sul figlio sinistro. Solo quando un nodo non ha figlio sinistro si esegue la chiamata al figlio destro. La prima foglia incontrata è necessariamente quella più a sinistra. Per calcolare il livello incrementiamo di 1 il risultato restituito risalendo di chiamata in chiamata, dando in output 0 visitando la foglia. Il tempo di esecuzione è in O(h), dove h è l'altezza dell'albero perchè ogni chiamata comporta la discesa di un livello, ma alla prima foglia trovata ci si ferma. Nel caso l'albero sia degenere è O(n). Per capire bene perché la prima trovata scendendo nell'albero come illustrato sopra individua la foglia giusta, si immagini che il piano sia diviso in due da ogni radice individuando così la porzione di sinistra e quella di destra, queste porzioni diventano sempre più "piccole" mano mano che si scende, ma in ogni caso tutte le porzioni di piano ottenute dalle radici di sotto alberi del sotto albero destro sono nella porzione a destra della radice. Per esempio nel disegno qui sotto:



```
LeftmostLeaf(T)
Input: T è un puntatore alla radica di un albero binario
Output: il livello della foglia più a sinistra dell'albero
   if T== NULL then return -1
   if T.left ≠ NULL then lev = LeftmostLeaf(T.left) +1
                    else
                     if T.right ≠ NULL then lev = LeftmostLeaf(T.right)+1
                                       else return 0
   return lev
```