Introduzione agli algoritmi
Proff. S. Caminiti - E. Fachini
16 Settembre 2020

1. 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.

```
fun (array A, int i, int f) {
   n = f-i+1
   t = n^2
   m=n/4
   if (n ≤ 1) then return 1
   while t≥1 do t ← t/2;
   return fun (A, i, i+m-1) + fun (A, i+m+1, i+2m-1) + fun (A, i+2m+1,i+3m-1) +
   fun (A, i+3m+1,f);
}
```

Sol. Il ciclo while viene eseguito circa lg n volte, le operazioni al suo interno sono eseguite in tempo costante quindi tutto il ciclo lavora in $\Theta(\lg n)$. Anche le operazioni al di fuori del ciclo e della chiamata sono eseguite in tempo costante. Essendoci quattro chiamate ricorsive su circa un quarto degli elementi la relazione di ricorrenza che esprime il tempo di esecuzione dell'algoritmo Analizzami è
$T(n)= 4T(n/4) + \Theta(\lg n)$

Per risolverla esplicitiamo le costanti e introduciamo il caso base:

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

$T(n) = d$ altrimenti

L'albero della ricorsione, che ci consente di semplificare i calcoli per giungere a una previsione dell'andamento di T è quaternario, visto che ci sono quattro chiamate, e sotto l'ipotesi semplificatrice che $n = 4^h$, ha altezza h e ogni nodo sul livello i, $0 \le i < h$, è etichettato con $c \lg n/4^i$. Sommando sul livello i si ottiene dunque $4^i\, c(\lg n/4^i)$. Al livello delle foglie invece ogni nodo corrisponde al costo per n = 1 quindi a d.

Sommando anche i livelli si ottiene $4^h d + \Sigma_{i=0\ldots h-1}\, 4^i\, c(\lg n/4^i) = 4^h d + \Sigma_{i=0\ldots h-1}\, 4^i\, c(\lg n - \lg 4^i) = 4^h d + \Sigma_{i=0\ldots h-1}\, (4^i\, c\lg n - 4^i * 2i\, c) = 4^i d + c\lg n\, (\Sigma_{i=0\ldots h-1}\, 4^i - 4^i * 2i\, c) = O(n \lg n)$, trascurando il termine negativo, perché $\Sigma_{i=0\ldots h-1}\, 4^i = O(4^h)$

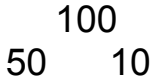
Facciamo dunque la previsione che l'andamento sia in $O(n \lg n)$ e dimostriamo che si tratta di una previsione corretta.

Prova induttiva:

Si vuole dimostrare che esistono k e $n_0 \geq 0$ tali che $T(n) \leq kn \lg n$, 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:
$T(n) = 4T(n/4) + cn \lg n \leq 4kn/4 \lg n/4 + c \lg n = kn (\lg n - \lg 4) + c \lg n = k n \lg n - 2kn + c \lg n$, imponiamo che questo valore sia $\leq kn$ così otteniamo
$k n \lg n - 2kn + c \lg n \leq kn \lg n$ sviluppando,
$-2kn + c \lg n \leq 0 \Leftrightarrow c \lg n \leq 2kn$ e questo è vero per k = 1, e per ogni valore di k più grande, e per ogni $n \geq 1$. Per il caso base notiamo che $T(4) = 4T(1) + 2c = 4d + 2c$ e cerchiamo un valore di k per cui $4d + 2c \leq k\, 4 \lg 4 = 8k$. Basta prendere $k = d+c$ per soddisfare la disuguaglianza, possiamo concludere che $T(n) \leq kn \lg n$, per ogni $n \geq 4$.

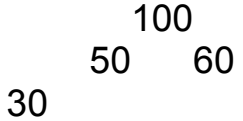
2. Dato un maxHeap H si dimostri in generale o si confuti, presentando un controesempio, che
1. dati due elementi x e y di H, dette h_x e h_y le altezze dei sottoalberi radicati in x e y se x>y allora $h_x > h_y$,
2. la somma delle chiavi dei nodi di un livello è maggiore o uguale alla somma delle chiavi dei nodi del livello successivo.

Sol. 1 è falsa infatti il seguente contro esempio lo prova:



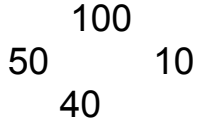
50 maggiore di 10 ma i due sotto alberi hanno la stessa altezza

oppure, volendo un esempio più articolato



prendendo x = 60 e y = 50 si ha che x > y, ma $h_x < h_y$

2 è anche falsa infatti il seguente controesempio lo prova:



La somma delle chiavi dei nodi del livello 1 è 60, mentre quella dei nodi nel livello 2 è 70.

3. Dato un ABR T, con chiavi intere positive, e un intero positivo k si progetti un algoritmo ricorsivo che preso in input T e k dia in output la più piccola differenza non negativa tra la chiave di un nodo di T e k. Nel caso in cui T sia nullo o tutti i nodi avessero chiavi minori di k la risposta deve essere -1. Si descriva l'idea algoritmica in modo da convincere della sua correttezza, si

analizzi il tempo di esecuzione e solo successivamente si scriva lo pseudo-codice. Non è previsto l'uso di memoria aggiuntiva.

Sol.

Consideriamo un albero binario T con chiave della radice c.

Se c = k allora possiamo concludere che il nodo in questione è quello cercato.

Se c > k può darsi che scendendo a sinistra si trovi una chiave che differisce, non negativamente, di meno. Si tratta quindi di visitare il sotto albero sinistro, la differenza c-k viene calcolata per confrontarla con le altre differenze tra le chiavi e k risalendo nell'albero. Come per il calcolo del minimo si considera come minimo corrente la prima differenza positiva e poi si rimpiazza tale differenza con una più piccola, se la si trova.

Se invece c < k bisognerà andare a vedere se nel sotto albero destro ci sono delle chiavi maggiori di k, sempre confrontando la più piccola differenza incontrata fino a quel momento con quella della radice del sotto albero in questione.

Di seguito un possibile pseudocodice:

```
DiffMin(T,k)

input: T è il puntatore alla radice di un albero binario e k un intero

precond:  T è un ABR, a chiavi intere positive  e k >0

output:  la più piccola differenza  non negativa tra la chiave di un
nodo e  k

if T== NIL  then return -1

if T.key  = k  then return 0

if T.key  > k  then Dm= DiffM(T.left,k,)
                 if Dm >0 then return min(Dm,T.key-k)
                              else return T.key-k
        else
             return DiffM(T.right,k)
```

Il tempo di esecuzione nel caso peggiore della procedura è in Θ(h), dove h è l'altezza dell'albero, perchè potrebbe esaminare un cammino di lunghezza massima dalla radice a una foglia. Nel caso migliore lavora in O(1), perchè k potrebbe coincidere con la chiave della radice. In generale la procedura termina in O(h).