Introduzione agli Algoritmi

Esame Scritto a canali unificati con idee per la soluzione

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
17 Gennaio 2023

Esercizio 1 (10 punti): Per la soluzione di un certo problema disponiamo di un algoritmo iterativo con costo computazionale $\Theta(n^3)$. Ci viene proposto in alternativa un algoritmo ricorsivo il cui costo è catturato dalla seguente ricorrenza:

$T(n) = a \cdot T\left(\frac{n}{2}\right) + \Theta(\sqrt{n})$ per $n \geq 2$

$T(n) = \Theta(1)$ altrimenti

dove a è una certa costante intera positiva con $a \geq 2$.

Determinare quale sia il valore massimo che la costante intera a può avere perché l'algoritmo ricorsivo risulti asintoticamente più efficiente dell'algoritmo iterativo di cui disponiamo. **Motivare bene la risposta.**

La risposta è 7. Per motivare questo valore, cominciamo col risolvere la ricorrenza. Applicando ad esempio il metodo principale abbiamo $f(n) = \Theta(\sqrt{n})$ mentre $n^{\log_2 a} \geq n^{log_2 2} = n$ si ha quindi $f(n) = O(n^{1-\epsilon})$. Siamo pertanto nel primo caso del metodo e la soluzione della ricorrenza è $\Theta\left(n^{\log_2 a}\right)$.
Ora, se $a = 8$ la ricorrenza ha soluzione $\Theta\left(n^{\log_2 8}\right) = \Theta\left(n^3\right)$ quindi, affinché l'algoritmo ricorsivo abbia costo inferiore a quello dell'algoritmo iterativo, deve aversi $a \leq 7$.

Esercizio 2 (10 punti):

Dati due arrays A e B, rispettivamente di n ed m interi distinti, con $m < n$, si vuole sapere se l'array A contenga l'array B come sottoarray.
Ad esempio, se $A = [5, 9, 1, 3, 4, 8, 2]$, per $B = [3, 4, 8]$ la risposta è *SI* mentre per $B = [3, 8, 2]$ o $B = [9, 6, 8]$ la risposta è *NO*.

Progettare un algoritmo che, dati gli arrays A e B, restituisca 1 se la risposta al problema è *SI*, 0 altrimenti. Il costo computazionale dell'algoritmo deve essere $O(n)$.

Dell'algoritmo proposto:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale.

a) **Osserviamo preliminarmente che condizione necessaria perché B sia sottoarray di A è che $B[0]$, sia presente in A; inoltre, poiché A contiene interi distinti, può esserci un'unica posizione in cui compare $B[0]$.**
L'algoritmo quindi comincia scorrendo A alla ricerca dell'elemento $B[0]$. Se $B[0]$ non appartiene ad A allora l'algoritmo termina restituendo 0. In caso contrario, l'algoritmo controlla che tutti gli m elementi di B siano presenti in A ordinatamente e consecutivamente a partire dalla posizione in cui è stato individuato $B[0]$; se questo risulta vero allora l'algoritmo restituisce 1, in caso contrario restituisce 0.

b)
```
def es2(A, B):
    n, m = len(A), len(B)
    i = 0
    while i < n and A[i] != B[0]:
        i += 1
    if i == n:  return 0
    j = 0
    while i < n and j < m and A[i] == B[j]:
        i += 1
        j += 1
    if j == m:  return 1
    return 0
```

c) **Il codice presenta due cicli *while* in sequenza. Il primo *while* ha costo $O(n)$, il secondo *while* ha costo $O(\min(n, m))$ e tutto il resto ha costo cotante $\Theta(1)$. Ne segue un costo complessivo di $O(n)$.**

Esercizio 3 (10 punti): Sia dato un albero binario T, in cui ogni nodo p ha tre campi: il campo valore $p.val$, il campo col puntatore al figlio sinistro $p.sx$ e il campo col puntatore al figlio destro $p.dx$, in mancanza di figlio il puntatore vale $None$.
Progettare un algoritmo *ricorsivo* che, dato il puntatore p alla radice dell'albero binario T, restituisca 1 se tutti i nodi dell'albero hanno lo stesso valore, 0 altrimenti. Il costo computazionale dell'algoritmo deve essere $O(n)$, dove n è il numero di nodi dell'albero.
Dell'algoritmo proposto:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale.

a) **L'algoritmo richiesto deve restituire 1 se il sottoalbero sinistro ed il sottoalbero destro hanno tutti i valori uguali a quello del nodo radice, restituire 0 in caso contrario.**
Il modo più semplice per implementare questo procedimento è tramite una visita in postorder dell'albero, in modo che ciascun nodo possa ricevere dai figli l'informazione sullo stato dei sottoalberi sinistro e destro.

b)
```
def es3(p):
    if not p return 1
    if not es3(p.sx):  return 0
    if not es3(p.dx):  return 0
    if (not p.sx or p.sx.val == p.val) and (not p.dx or p.dx.val == p.val):
        return 1
    return 0
```

c) **Nel corso della visita dell'albero, se l'algoritmo rileva un sottoalbero che non contiene tutti i nodi uguali, termina senza che tutti i nodi vengano visitati. Il costo computazionale dell'algoritmo è dunque limitato dal costo della visita di un albero con n nodi. L'equazione di ricorrenza relativa alla visita completa dell'albero è:**

$$T(n) = T(k) + T(n-1-k) + \Theta(1)$$
$$T(0) = \Theta(1)$$

che si può risolvere con il metodo di sostituzione dando come soluzione $\Theta(n)$.
Di conseguenza il costo dell'algoritmo è, come richiesto, $O(n)$.