Introduzione agli Algoritmi
Esame Scritto a canali unificati

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
21 Ottobre 2021

Esercizio 1 (10 punti):

Si consideri la seguente funzione:

funzione Exam(n):
    $tot \leftarrow n$;
    if $n <= 1$: return tot;
    $tot \leftarrow tot$+Exam($n-1$);
    $b \leftarrow n-1$;
    $j \leftarrow n$;
    while $j >= 0$ do:
        $tot \leftarrow tot + j$;
        $j \leftarrow j - 2$;
    return tot+Exam(b)

a) Si imposti la relazione di ricorrenza che ne definisce il tempo di esecuzione giustificando dettagliatamente l'equazione ottenuta.

b) Si risolva la ricorrenza usando il **metodo di sostituzione** e si dimostri così che la soluzione è $\mathcal{O}(n \cdot 2^n)$, commentando opportunamente i passaggi.

a) la ricorrenza è $T(1) = \Theta(1)$ **e** $T(n) = 2T(n-1) + \Theta(n)$**.**

b) Eliminiamo l'asintotica dall'equazione e otteniamo:
$T(1) = a$ **e** $T(n) = 2T(n-1) + b \cdot n$ **per due opportune costanti** a **e** b**.**

Ipotizziamo la soluzione suggerita $T(n) = \mathcal{O}(n2^n)$ ossia $T(n) \leq cn2^n$ dove c è una costante che va determinata.

Infine, sostituiamo: nel caso base, dovrebbe aversi $T(1) = a \leq 2c$ e la diseguaglianza è effettivamente soddisfatta prendendo ad esempio $c \geq a$; nella formulazione ricorsiva, dovrebbe aversi:
$T(n) \leq 2c \cdot (n-1) \cdot 2^{n-1} + b \cdot n = c \cdot n \cdot 2^n - c \cdot 2^n + b \cdot n \leq c \cdot n \cdot 2^n$ **dove l'ultima diseguaglianza risulta vera per tutti gli n se prendiamo ad esempio $c \geq b$.**

Esercizio 2 (10 punti):
Sia dato un array A **ordinato** di n interi distinti ed un intero x; si vuole trovare l'indice in A del più piccolo intero maggiore di x. Progettare un **algoritmo iterativo** efficiente che risolva il problema. Se l'array contiene solo elementi minori o uguali ad x, l'algoritmo deve restituire -1.

Ad esempio: per $A = [1, 2, 8, 10, 11, 12, 19]$, assumendo che le posizioni dell'array partano da 0, per $x = 7$ l'algoritmo deve restituire 2 (cioè l'indice dell'elemento 8), per $x = 30$ l'algoritmo deve restituire -1.
Dell'algoritmo proposto

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale.

a) modifichiamo leggermente la ricerca binaria andando a ricercare ad ogni passo in un sottovettore di dimensione dimezzata.

- **se il sottovettore contiene un solo elemento allora controllo se l'elemento è maggiore di x o meno e restituisco la posizione dell'elemento nel primo caso , restituisco -1 altrimenti.**
- **sia m l'elemento centrale del sottovettore:**
  - **se $A[m] > x$ allora vado a ricercare nel sottovettore di sinistra che arriva fino ad m**
  - **in caso contrario vado a ricercare nel sottovettore di destra che parte da $m + 1$**

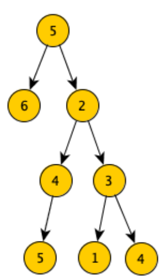
b) `def esercizio2(`A, x`):`
   $i, j = 0, len(A) - 1$
   `while` $i < j$:
      $m = (i + j)//2$
      `if` $A[m] > x$: $j = m$
      `else:` $i = m + 1$
   `if` $A[i] > x$: `return` i
   `return` -1

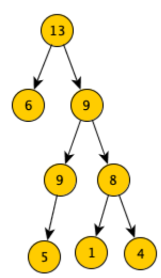
c) **il costo computazionale dell'algoritmo dipende dal numero di iterazioni del *while*. Ad ogni iterazione la dimensione del sottovettore che va da a a b si dimezza; questo significa che il numero di iterazioni prima che uno degli indici diventi negativo è al più ordine di $\log n$. Ogni iterazione del *while* richiede tempo costante, la complessità dell'algoritmo è dunque $\Theta(\log n)$.**

Esercizio 3 (10 punti):

Si consideri un albero binario radicato T, i cui nodi hanno un campo `val` contenente un intero e i campi `left` e `right` con i puntatori ai figli.

Bisogna modificare il campo `val` di ciascun nodo in modo che il nuovo risulti la somma del valore originario incrementata dal valore originario degli eventuali figli. Si consideri ad esempio l'albero T in figura a sinistra, a destra viene riportato il risultato della modifica di T.





Progettare un **algoritmo ricorsivo** che, dato il puntatore r alla radice di T memorizzato tramite record e puntatori, effettui l'operazione di modifica in tempo $\mathcal{O}(n)$ dove n è il numero di nodi presenti nell'albero.
Dell'algoritmo proposto

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale.

a) Basta implementare una modifica della visita in pre-order dell'albero, in cui si modifica il valore di ciascun nodo e poi si passa ai suoi figli:

b) `def modifica(`r`):`

  `if` $r.left! = None : r.val+ = r.left.val$

  `if` $r.right! = None : r.val+ = r.right.val$

  `if` $r.left! = None$: `modifica(`$r.left$`)`

  `if` $r.right! = None$: `modifica(`$r.right$`)`

c) la complessità dell'algoritmo è quella di una semplice visita (ed i calcoli vanno riportati sul compito).