Introduzione agli Algoritmi
Esame Scritto a canali unificati con idee per la soluzione

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
8 Settembre 2021

Esercizio 1 (10 punti):
Si consideri la seguente funzione:

```
funzione Exam(n):
    tot ← n;
    if n <= 4:  return tot;
    b ← n//4;
    tot ← tot+Exam(b);
    j ← 1;
    while j * j <= n do:
        tot ← tot + j;
        j ← j + 1;
    return tot+Exam(b).
```

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

b) Si risolva l'equazione usando il **metodo dell'albero**, dettagliando i passaggi del calcolo e giustificando ogni affermazione.

- **la ricorrenza è:** $2T\left(\frac{n}{4}\right) + \Theta(\sqrt{n})$ **se** $n > 4$ **e** $T(n) = \Theta(1)$ **altrimenti.**
- **In ogni livello** i **dell'albero tutti i** 2^i **nodi danno un contributo di** $\sqrt{n}/2^i$ **per cui, sommando i contributi livello per livello, ogni livello** i **contribuisce per** $\sqrt{n}$**. L'albero che si ottiene è binario completo e pertanto la sua altezza è logaritmica in** n**. Ne consegue che il costo totale è** $\Theta(\sqrt{n}\log n)$**.**

Esercizio 2 (10 punti):

Progettare un algoritmo che, dati tre array A, B e C **ordinati** e contenenti ciascuno n interi **distinti**, stampi in tempo $O(n)$ gli interi che compaiono nell'intersezione dei tre array. L'algoritmo proposto deve utilizzare spazio di lavoro $\Theta(1)$.
Ad esempio: per $A = [1, 2, 3, 4, 5, 6]$, $B = [1, 4, 5, 6, 8, 9]$ e $C = [2, 4, 6, 7, 8, 9]$ l'algoritmo deve stampare gli elementi 4 e 6.

Dell'algoritmo proposto:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi formalmente il costo computazionale.

d) si dia un'idea di quello che accadrebbe al costo computazionale se si volesse generalizzarlo a $\Theta(n)$ array.

a) Assumiamo che le posizioni degli elementi nei vettori vadano da 0 a $n-1$ (ma analogamente si potrebbe procedere se andassero da 1 ad n) e scorriamo da destra verso sinistra i tre array utilizzando tre indici a, b e c che vengono decrementati in base alla seguente regola:

- **se $A[a] = B[b] = C[c]$ basta stampare uno qualunque dei tre valori e decrementiamo tutti e tre gli indici.**
- **altrimenti: calcoliamo $m = \max(A[a], B[b], C[c])$**
  - **se $m = A[a]$ decrementiamo l'indice a in quanto $A[a]$ non può essere un elemento comune.**
  - **se $m = B[b]$ decrementiamo l'indice b in quanto $B[b]$ non può essere un elemento comune.**
  - **se $m = C[c]$ decrementiamo l'indice c in quanto $C[c]$ non può essere un elemento comune.**
  - **se l'indice decrementato diventa negativo l'algoritmo termina**

È immediato modificare il precedente algoritmo in modo che funzioni scorrendo gli array da sinistra a destra e che calcoli il minimo anziché il massimo.

b)
```
def esercizio2(A, B, C):
    a = b = c = len(A) - 1
    while a >= 0 and b >= 0 and c >= 0:
        if A[a] == B[b] == C[c]:
            print(A[a])
        m = max(A[a], B[b], C[c])
        if m == A[a] : a -= 1
        if m == B[b] : b -= 1
        if m == C[c] : c -= 1
```

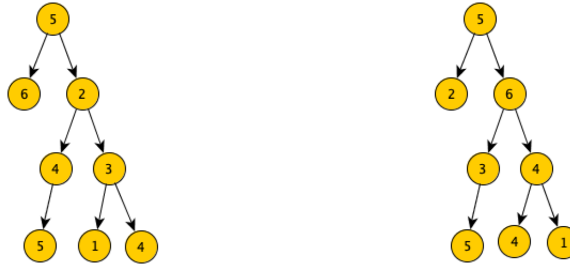
c) **il costo computazionale dell'algoritmo dipende dal numero di iterazioni del** *while*. **Ad ogni iterazione almeno uno dei tre indici** a, b **e** c **si decrementa; questo significa che il numero di iterazioni prima che uno degli indici diventi negativo è almeno** n **ed al più** $3n-3$. **Ogni iterazione del** *while* **richiede tempo costante, la complessità dell'algoritmo è dunque** $\Theta(n)$. **Questo ragionamento può essere scritto in modo più formale sommando i contributi delle singole istruzioni.**

d) **Ad ogni iterazione almeno uno dei** $\Theta(n)$ **indici che tengono traccia della posizione all'interno dei vari file si decrementa; questo significa che il numero di iterazioni prima che uno degli indici diventi negativo è almeno** n **ed al più** $n\Theta(n)$. **Ogni iterazione del** *while* **richiede tempo** $\Theta(n)$ **(il tempo richiesto per il calcolo dell'indice posizionato sull'elemento di valore minimo). La complessità dell'algoritmo proposto in precedenza è dunque** $\Theta(n^3)$. **Questo ovviamente non vuol dire che non ci siano algoritmi più efficienti.**

Esercizio 3 (10 punti):

Si consideri un albero binario radicato T, i cui nodi hanno un campo valore contenente un intero. Bisogna modificare l'albero in modo che i nodi fratelli scambino tra loro il valore. Si consideri ad esempio l'albero T in figura a sinistra, a destra viene riportato il risultato della modifica di T.

Progettare un algoritmo che, dato il puntatore r alla radice di T memorizzato tramite record e puntatori, effettui l'operazione di modifica in tempo $O(n)$ dove n è il numero di nodi presenti nell'albero. Ogni nodo dell'albero è memorizzato in un record contenente il campo *val* con il valore del nodo e i



campi *left* e *right* con i puntatori ai figli di sinistra e destra, rispettivamente. Dell'algoritmo proposto:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi formalmente il costo computazionale.

a) Per risolvere il problema utilizzo un'algorito ricorsivo. L'algoritmo parte dal nodo radice dell'albero e richiama se stesso sugli eventuali sottoalberi sinistro e destro della radice . Fatto questo, se il nodo radice ha entrambi i figli, viene effettuato uno scambio del loro valore.

b) Il codice python dell'algoritmo è il seguente:

```
def modifica(r):
    if r.left:
        modifica(r.left)
    if r.right:
        modifica(r.right)
    if r.left and r.right:
        # scambio il contenuto dei nodi fratelli
        r.left.val, r.right.val = r.right.val, r.left.val
```

c) la complessità dell'algoritmo proposto è $\Theta(n)$ in quanto il lavoro effettuato consiste in una semplice visita (in postorder) dell'albero.