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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
24 Ottobre 2024

Esercizio 1 (10 punti): Siano date le tre seguenti equazioni di ricorrenza:

$T(n) = 4T(\frac{n}{2}) + \Theta(n)$	$T(n) = 4T(\frac{n}{2}) + \Theta(n^2)$	$T(n) = 4T(\frac{n}{2}) + \Theta(n^3)$
$T(n) = \Theta(1)$	$T(n) = \Theta(1)$	$T(n) = \Theta(1)$

Si risolvano con il teorema principale, dettagliando il ragionamento usato ed evidenziando le differenze.

Le tre equaioni rientrano nei tre diversi casi del teorema principale, nell'ordine, pertanto le loro soluzioni sono $\Theta(n^2)$, $\Theta(n^2 \log n)$ e $\Theta(n^3)$.
Si ricordi che i passaggi vanno dettagliati formalmente (tramite la corretta notazione asintotica e l'uso degli ϵ) e che il terzo caso richiede la dimostrazione della condizione aggiuntiva.

Esercizio 2 (10 punti): Si scriva lo pseudocodice, opportunamente commentato, di una funzione **iterativa** che, preso in input un array A di interi, trovi la lunghezza massima delle sequenze crescenti presenti nell'array.
Ad esempio:

- per $A = [3, 1, 5, 2, 6, 8, 7, 1]$ la risposta è 3, che è la lunghezza della sequenza $2, 6, 8$;

- per $A = [10, 7, 6, 1]$ la risposta è 1 perché non ci sono in A sequenze crescenti di lunghezza maggiore ad 1.

La funzione deve avere costo computazionale $\mathcal{O}(n)$, dove n è il numero di elementi presenti nell'array. Il costo in termini di spazio oltre l'array A deve essere $\Theta(1)$ (in pratica non si può far uso di array di appoggio).

Il costo computazionale dell'algoritmo proposto va dettagliato valutando il contributo di ogni istruzione.

Il codice Python di una possibile soluzione dell'esercizio è il seguente:

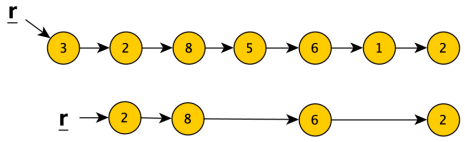
```
def esercizio2(A):
    inizio, massimo = 0, 1
    for i in range(1,len(A)):
        if A[i] > A[i-1]:
            massimo=max(massimo,i-inizio+1)
        else:
            inizio = i
    return massimo
```

Usiamo due indici i e $inizio$ per scorrere l'array, entrambi si spostano da sinistra verso destra, il primo indica l'elemento di A che via via consideriamo mentre il secondo indica la posizione in cui inizia la sequenza crescente che include $A[i]$. Se $A[i-1] < A[i]$ allora vuol dire che è stata trovata una sequenza più lunga di quella correntemente considerata, e si incrementa il solo indice i mentre, in caso contrario, si è trovata la fine della sequenza precedente e quindi si va alla ricerca della nuova incrementando anche l'indice $inizio$, che assume il valore i ad indicare che siamo in resenza di una nuova sequenza crescente al momento lunga 1.

Per quanto riguarda il costo computazionale, esso dipende essenzialmente dal numero delle iterazioni del ciclo for, perché il costo di tutte le altre operazioni, così come pure di ogni iterazione del for, è costante. Il costo è dunque

$\Theta(n)$**, ma la sua dimostrazione richiede una dimostrazione più rigorosa di quella accennata qui.**

Esercizio 3 (10 punti): Si consideri una lista concatenata, i cui nodi abbiano un campo *key* contenente un intero ed un campo *next* al nodo successivo (next vale $None$ se si tratta dell'ultimo nodo).
Si progetti una funzione **ricorsiva** che, dato il puntatore r alla testa della lista restituisca il puntatore alla testa della lista modificata in modo che vi compaiano solo i nodi contenenti gli interi pari.
Ad esempio, data la lista concatenata r della figura, la funzione deve restituire il puntatore alla nuova lista concatenata contenente i soli nodi con valore pari.



La funzione deve avere costo computazionale $\mathcal{O}(n)$ dove n è il numero di nodi presenti nella lista originaria e l'algoritmo non può generare nuovi nodi nè utilizzare array di appoggio.
Della funzione proposta:
a) si dia la descrizione a parole,
b) si scriva lo pseudocodice,
c) si giustifichi formalmente il costo computazionale dando la ricorrenza che lo caratterizza e poi la sua soluzione.
NOTA BENE: nello pseudocodice della funzione ricorsiva **non** si deve far uso di variabili globali.

L'algoritmo prevede di visitare la lista puntata da r. Se la lista è vuota allora viene restituito $None$. In caso contrario viene risolto ricorsivamente il problema sulla lista puntata da $r.next$ e se $r.key$ è dispari la risposta è il puntatore ricevuto, altrimenti il nodo puntato da r deve diventare la testa della lista ricevuta e viene restituito r.

Il codice python della procedura appena descritta è il seguente:

```
def es3(r):
    if r == None:
        return r
    if r.key%2 != 0:
        return es3(r.next)
    r.next = es3(r.next)
    return r
```

L'algoritmo richiama se stesso su di una lista che ha un nodo in meno. La ricorsione che cattura questo comportamento è $T(n) = T(n-1) + \Theta(1)$ **con** $T(0) = \Theta(1)$ **che risolta ad esempio col metodo iterativo dà** $\Theta(n)$**.**