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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
18 Settembre 2024

Esercizio 1 (10 punti): Si consideri la seguente funzione, in cui i valori a e b in input sono due interi maggiori di 1:

```
def Es1(n, a, b):
    if n < 1:
        return 1
    c, t, s = n//b, a, 0
    while t > 0:
        for j in range(a):
            s += Es1(c, a, b)
        t-= 1:
    for i in range(n):
        for j in range(n):
            s += i + j
    return s
```

a) Si imposti la relazione di ricorrenza che ne definisce il tempo di esecuzione in termini di n, a e b giustificando dettagliatamente l'equazione ottenuta.

b) Si risolva la ricorrenza utilizzando uno dei metodi studiati, nei tre casi possibili $a > b$, $a = b$ e $a < b$.

a) **Il caso base si ha quando $n < 1$, ed ha costo costante. Il ciclo $while$ viene iterato a volte così come il primo for; questo significa che la funzione $Es1$ viene ricorsivamente richiamata a^2 volte su un valore $c = n//b$. Ciascuno dei due for annidati finali viene iterato n volte per un costo totale di $\Theta(n^2)$. Per quanto detto, la relazione di ricorrenza che definisce il costo della funzione è:**

   - $T(n) = a^2 T\left(\frac{n}{b}\right) + \Theta\left(n^2\right), \;\; n \geq 1$
   - $T(n) = \Theta(1), \;\; n < 1.$

b) **Per risolvere la ricorrenza possiamo utilizzare il teorema principale. Dobbiamo confrontare tra loro i due valori $n^{\log_b a^2} = n^{2\log_b a}$ con $f(n) = \Theta(n^2)$ e il costo computazionale è governato dal maggiore dei 2. Consideriamo i tre casi:**

   - $a > b$: **In questo caso $\log_b a > 1$ ed il più grande dei due oggetti da confrontare è $n^{2\log_b a}$. Siamo nel caso 1 del teorema e la soluzione è $\Theta\left(n^{2\log_b a}\right)$.**
   - $a = b$: **In questo caso $\log_b a = 1$ e i due oggetti da confrontare sono uguali. Siamo nel caso 2 del teorema e la soluzione è $\Theta(n^2 \log n)$**
   - $a < b$: **In questo caso si ha $\log_b a < 1$ e il più grande dei due oggetti da confrontare è $f(n)$. Siamo nel caso 3 del teorema e la soluzione è $\Theta(n^2)$**

Esercizio 2 (10 punti):

Diciamo che l'elemento di indice i in un array è un *massimo parziale* se è strettamente maggiore di tutti gli elementi che lo precedono nell'array, cioè di tutti gli elementi di indice tra 0 ed $i - 1$.

Dato un array A di n interi, si vuole determinare il numero di elementi di A che siano massimi parziali.

Ad esempio per $A = [3, 4, 2, 5, 1]$ la risposta è 3 (i massimi parziali sono infatti 3, 4 e 5).

Progettare un algoritmo che, dato A, in tempo $\Theta(n)$ risolva il problema. Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si giustifichi il costo computazionale.

a) **Basta scorrere l'array da sinistra verso destra mantenendo una variabile per il massimo corrente. Se il massimo corrente coincide con l'elemento appena considerato, esso sarà necessariamente un massimo parziale e si incrementerà un contatore.**

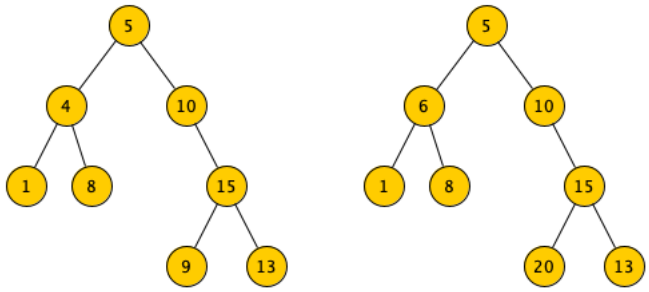
Segue un possibile codice Python dell'idea proposta.

```
def es2(A):
    c, m = 1, A[0]
    for i in range (2, len(A)):
        if A[i] > m:
            m = A[i]
            c+ = 1
    return c
```

b) **Il costo dell'algoritmo è dato principalmente dal contributo del ciclo *for* che è $\Theta(n)$. E' necessario scrivere i dettagli.**

Esercizio 3 (10 punti): Sia dato un albero binario T non vuoto contenente interi, e questo sia memorizzato tramite puntatori e record di tre campi: il campo key contenente il valore ed i campi $left$ e $right$ contenenti i puntatori al figlio sinistro e al figlio destro, rispettivamente (questi puntatori valgono $None$ in mancanza del figlio).
Dato il puntatore r al nodo radice di T, progettare un algoritmo *ricorsivo* che in tempo $\Theta(n)$ restituisca $True$ se in T esiste un cammino radice-foglia dove gli interi dei nodi che si incontrano lungo il cammino formano una sequenza strettamente crescente, $False$ altrimenti.

Ad esempio, per l'albero a sinistra in figura, l'algoritmo deve rispondere $False$ perché nessuna delle sequenze di valori che si incontrano nei cammini radice-foglia dell'albero risulta crescente. Per l'albero a destra in figura, invece, l'algoritmo deve rispondere $True$, infatti esistono ben due cammini radice-foglia i cui valori formano sequenze strettamente crescenti (le sequenze sono $5,\ 6,\ 8$ e $5,\ 10,\ 15\ 20$).



Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) Si ricavi il costo computazionale dell'algoritmo producendo la relazione di ricorrenza che cattura il tempo di calcolo dell'algoritmo e risolvendola.

NOTA BENE: nello pseudocodice dell'algoritmo ricorsivo **non** si deve far uso di variabili globali.

a) **Si utilizza una visita dell'albero in pre-ordine. Infatti, un nodo x viene visitato solo se il cammino radice-x forma una sequenza di valori crescenti.**
Giunti a visitare il nodo x, si verifica se questo è foglia (ed in questo caso l'algoritmo termina la visita restituendo il valore $True$).
Se, al contrario, x ha un figlio sinistro y e la chiave di y è maggiore della chiave di x, allora l'algoritmo termina restituendo $True$ se nel sottoalbero radicato in y esiste un cammino radice foglia crescente.

Se questo controllo fallisce, si procede in modo analogo sull'eventuale figlio destro: se x ha un figlio destro z e la chiave di z è maggiore della chiave di x, l'algoritmo termina restituendo $True$ se nel sottoalbero radicato in z esiste un cammino radice-foglia crescente.
Se entrambi i controlli non hanno dato esito positivo la visita termina restituendo $False$.

Ecco un possibile codice Python dell'algoritmo che non utilizza variabili globali:

```
def es3(r):
    if r.left == r.right == None:
        return True
    if r.left and r.left.key > r.key and es3(r.left):
        return True
    if r.right and r.right.key > r.key and es3(r.right):
        return True
    return False
```

b) **Il costo computazionale è quello della visita di un albero con n nodi. L'equazione di ricorrenza relativa alla visita è:**

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

dove k, è il numero di nodi presenti nel sottoalbero sinistro, $0 \leq k < n$. L'equazione si può risolvere con il metodo di sostituzione (che va esplicitamente svolto) dando come soluzione $\Theta(n)$.
Ogni passaggio va dettagliato.