Algoritmi e Strutture Dati
26 Giugno 2020

Note

1. La leggibilità è un prerequisito: parti difficili da leggere potranno essere ignorate.
2. Quando si presenta un algoritmo è fondamentale spiegare l'idea e motivarne la correttezza.
3. L'efficienza e l'aderenza alla traccia sono criteri di valutazione delle soluzioni proposte.
4. Si consegna la scansione dei fogli di bella copia, e come ultima pagina un documento di identità.

Domande

Domanda A (7 punti) Definire formalmente la classe $O(f(n))$. Dimostrare che la seguente ricorrenza ha soluzione $T(n) = O(n)$

$$T(n) = \frac{1}{3}T(n-1) + 2n + 1$$

Soluzione: Per la definizione di $O(f(n))$, consultare il libro.

Si deve provare che asintoticamente, per un'opportuna costante $c > 0$

$$T(n) \leq cn$$

Si procede per induzione:

$$\begin{array}{rcll} T(n) & = & \frac{1}{3}T(n-1) + 2n + 1 & \text{[per definizione della ricorrenza]} \\ & \leq & \frac{1}{3}c(n-1) + 2n + 1 & \text{[per ipotesi induttiva } T(n-1) \leq c(n-1)\text{]]} \\ & = & (\frac{1}{3}c + 2)n - (\frac{1}{3}c - 1) & \text{[assumendo } c \geq 3\text{]} \\ & \leq & (\frac{1}{3}c + 2)n & \\ & \leq & cn & \end{array}$$

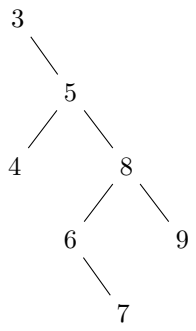
dove l'ultima disuguaglianza vale quando $c \geq 3$, consistentemente con l'assunzione. Risulta dunque dimostrata la tesi.

Domanda B (7 punti) Dare la definizione di albero binario di ricerca. Specificare l'albero ottenuto inserendo, con la procedura vista a lezione, a partire da un albero vuoto, i nodi aventi le seguenti chiavi:

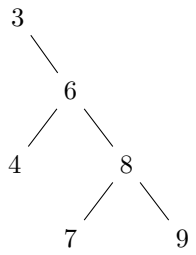
$$3, 5, 8, 6, 9, 7, 4$$

Dall'albero così ottenuto si cancelli il nodo con chiave 5 e si indichi l'albero ottenuto. Sia per gli inserimenti che per la cancellazione, motivare sinteticamente il risultato ottenuto.

Soluzione: Per la definizione di albero binario di ricerca e la descrizione delle procedure di inserimento e cancellazione si consulti il libro. Sullo specifico esempio si ottiene, dopo gli inserimenti:



La cancellazione di 5 quindi produce:



Esercizi

Esercizio 1 (9 punti) Realizzare una funzione `avgTree(T)` che dato un albero binario `T` con chiavi numeriche, verifica se, per ogni nodo che abbia discendenti, la chiave del nodo è maggiore o uguale della media delle chiavi dei discendenti e ritorna conseguentemente un valore booleano (la radice dell'albero è `T.root` e ogni nodo `x` ha i campi `x.left` `x.right` e `x.key`).

Soluzione: L'idea è quella di effettuare una visita dell'albero, in modo ricorsivo, raccogliendo per ogni sottoalbero le seguenti informazioni:

- se è soddisfatta la condizione sulla media richiesta
- la somma delle chiavi
- il numero dei nodi.

Quano mi trovo su di un nodo, analizzo i sottoalberi sinistro e destro. Quindi, avendo a disposizione la somma delle chiavi ed il numero dei nodi per i due sottoalberi posso verificare la condizione sulla media per il nodo in esame. Lo pseudocodice della soluzione può essere il seguente:

```
avgTree(T)
   v, n, s = avgTreeRec(T.root)
   return v


# avgTreeRec(x):
# verifica se il sottoalbero radicato in x e' un avgTree e ritorna tre valori:
# - un booleano,
# - il numero dei discendenti
# - la somma delle loro chiavi

avgTreeRec(x)

if x = nil
   return true, 0, 0
else
   # ispeziono i sottoalberi sinistro e destro
   vl, nl, sl = avgTreeRec(x.left)
   vr, nr, sr = avgTreeRec(x.right)

   # se non ci sono discendenti (nl+nr =0) oppure la chiave del nodo in
   # esame e' maggiore o uguale della media delle chiavi dei discendenti
   # la condizione e' soddisfatta
   v =   (nl+nr =0) or (x.key >= (sl+sr)/(nl+nr))

   # il numero di nodi dell'albero radicato in x e' dato dalla somma del
   # del numero di nodi nel sottoalbero dx e nel sottoalbero sx piu' uno
   # (il nodo x stesso)
   n =   nl + nr + 1

   # la somma delle chiavi dell'albero radicato in x e' dato dalla somma del
   # delle chiavi nel sottoalbero dx e in quello sx piu' x.key
   s =   sl + sr + x.key

   return v, n, s
```

Si tratta di una visita e quindi la complessità è $\Theta(n)$.

Esercizio 2 (9 punti) Per $n > 0$, siano dati due vettori a componenti intere $\mathbf{a}, \mathbf{b} \in \mathbf{Z}^n$. Si consideri la quantità $c(i, j)$, con $0 \leq i \leq j \leq n-1$, definita come segue:

$$c(i,j) = \begin{cases} a_i & \text{se } 0 < i \leq n-1 \text{ e } j = n-1, \\ b_j & \text{se } i = 0 \text{ e } 0 \leq j \leq n-1, \\ c(i-1, j-1) + c(i, j+1) & 0 < i \leq j < n-1. \end{cases}$$

Si vuole calcolare la quantità $m = \min\{c(i,j) : 0 \leq i \leq j \leq n-1\}$.

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di m.
2. Si valuti la complessità esatta dell'algoritmo, associando costo unitario alle somme tra numeri interi (escluse quelle tra indici) e costo nullo a tutte le altre operazioni.

Soluzione:

1. Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione in cui calcoliamo gli elementi in ordine crescente di indice di riga e, per ogni riga, in ordine decrescente di indice di colonna. Il codice è il seguente.

```
COMPUTE(a,b)
n <- length(a)
m = +infinito
for i=1 to n-1 do
   C[i,n-1] <- a_i
   m <- MIN(m,C[i,n-1])
for j=0 to n-1 do
   C[0,j] <- b_j
   m <- MIN(m,C[0,j])
for i=1 to n-2 do
   for j=n-2 downto i do
      C[i,j] <- C[i-1,j-1] + C[i,j+1]
      m <- MIN(m,C[i,j])
return m
```

2. $$T(n) = \sum_{i=1}^{n-2} \sum_{j=i}^{n-2} 1 = \sum_{i=1}^{n-2} n - 1 - i = \sum_{k=1}^{n-2} k = (n-1)(n-2)/2.$$