Dati e Algoritmi I (Pietracaprina)

Compitino 18/11/2019 (Seconda Parte)

SOLUZIONI

Problema. Sia $S = S[1], S[2], \dots, S[n]$ una sequenza di $n \geq 1$ bit. Il seguente algoritmo determina la lunghezza del più lungo segmento continuo di S con bit uguali (ad es., per $S = 10100001100$ restituisce 4 e per $S = 00110111110$ restituisce 5).

```
max_len ← 1; curr_len ← 1;
for i ← 2 to n do {
  if (S[i]=S[i-1]) then {
    curr_len ← curr_len+1;
    max_len ← max{max_len,curr_len};
  }
  else curr_len ← 1;
}
return max_len;
```

- Identificare un opportuno invariante per il ciclo che serva per provare la correttezza dell'algoritmo.
- Dimostrare che se l'invariante vale alla fine dell'ultima iterazione, allora l'algoritmo è corretto.

Soluzione.

- Per $i \geq 1$, l'invariante che vale alla fine dell'iterazione i del for è il seguente (la fine dell'iterazione 1 corrisponde all'inizio del ciclo):
 - max-len è la lunghezza del più lungo segmento continuo di bit uguali in $S[1 \dots i]$.
 - Il più lungo segmento continuo di bit uguali che termina in $S[i]$ ha lunghezza curr-len .
- Alla fine dell'ultima iterazione ($i = n$) la prima proprietà dell'invariante afferma che max-len è la lunghezza del più lungo segmento continuo di S con bit uguali.

□

Problema. Sia T un albero binario proprio dove ogni nodo $v \in T$ memorizza nel campo $v.\text{val}$ un valore numerico. Un nodo $v \in T$ si dice *dominante* se il valore $v.\text{val}$ è maggiore o uguale a tutti i valori memorizzati nei nodi del sottoalbero T_v . Si vuole progettare un algoritmo *ricorsivo* `countDominant` per contare il numero di nodi dominanti in T . Detta `countDominant(T, v)` la generica invocazione su un nodo $v \in T$, per risolvere il problema si eseguirà `countDominant(T, T.root())`.

- a. Descrivere tramite pseudocodice `countDominant(T,v)`, specificandone con attenzione l'input e l'output.
- b. Analizzare la complessità di `countDominant(T,T.root())` in funzione del numero n di nodi in T .

Soluzione.

- a. È opportuno che oltre al numero di nodi dominanti in T_v , `countDominant(T,v)` restituisca anche il massimo valore in T_v . In questo modo, dalle invocazioni fatte sui figli di v si può stabilire se v è dominante. Lo pseudocodice è il seguente:

```
Algoritmo countDominant(T,v)
input Albero T, nodo v ∈ T
output Numero di nodi dominanti in Tv, massimo valore in Tv

if (T.isExternal(v)) then return (1,v.val)
(ndL,maxL) ← countDominant(T,T.left(v))
(ndR,maxR) ← countDominant(T,T.right(v))
M ← max{maxL,maxR}
if (v.val ≥ M) then return (ndL+ndR+1,v.val)
else return (ndL+ndR,M)
```

- b. Se invocato con $v=T.root()$, l'algoritmo esegue una visita in postorder di T , dove la visita di un nodo richiede $\Theta(1)$ operazioni. La complessità è quindi $\Theta(n)$.

□