Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

19 giugno 2023

Informatica teorica

Esercizio 1 (8 punti)
Si consideri il linguaggio $L = \{xy \mid x, y \in \{a, b\}^* \text{ e } \#a(x) = \#b(y)\}$, dove $\#a(x)$ indica il numero di occorrenze del carattere a nella stringa x (e analogamente per $\#b(y)$).

1. Utilizzare un formalismo a potenza minima (tra tutti quelli visti a lezione) che caratterizzi il linguaggio L.

2. Cambierebbe il formalismo a potenza minima se il linguaggio in questione fosse $L' = \{xcy \mid x, y \in \{a, b\}^* \text{ e } \#a(x) = \#b(y)\}$? Barrare la casella opportuna e motivare la propria risposta.

Soluzione

1. Il linguaggio L è il linguaggio $\{a, b\}^*$. Si consideri, infatti, una qualunque suddivisione di una generica stringa w (di lunghezza n) in due parti x (di lunghezza i) e y (di lunghezza $n - i$) tali che $w = xy$. Sia $d_i(w) = \#a(x) - \#b(y)$. Se $i = 0$, abbiamo $d_0(w) = -\#b(w) \leq 0$. Se invece $i = n$, abbiamo $d_n(w) = \#a(w) \geq 0$. Si osservi infine che $d_{i+1}(w) = d_i(w) + 1$ per ogni $i = 0, 1, \ldots, n - 1$, perché se l'i-esimo carattere è una a allora aumenta di 1 il numero di a e il numero di b resta invariato, altrimenti diminuisce di 1 il numero di b e il numero di a resta invariato. Poiché $d_0(w) \leq 0$, $d_n(w) \geq 0$ e $d_{i+1}(w) = d_i(w) + 1$, deve necessariamente esistere un valore i tale per cui $d_i(w) = 0$ e quindi $w \in L$. La stringa w era generica, quindi $L = \{a, b\}^*$.

   Per caratterizzare tale linguaggio, il formalismo a potenza minima è allora quello della logica monadica del prim'ordine (MFO). Una formula MFO che lo esprima è la seguente: $\forall x(a(x) \vee b(x))$.

2. Per L' occorre la capacità di contare e servirebbe quindi un automa a pila deterministico.

Esercizio 2 (8 punti)

Si considerino le seguenti funzioni e si dica se sono calcolabili, motivando esaurientemente la risposta:

1. $f(x) = f_y(x) - 2$, con $y = x + 2$ (nota: se $f_y(x) = \perp$, chiaramente anche $f(x) = \perp$);

2. $g(x, y) = f_{x+y}(y) + 1$, se M_{x+y} calcola una funzione totale da $\mathbb{N}$ ad $\mathbb{N}$, $g(x, y) = 0$ altrimenti.

Soluzione

1. Calcolabile: basta costruire M_y con l'enumerazione $\mathcal{E}$ e poi simularla. Se termina, si restituisce il valore calcolato -2.

2. Non è calcolabile: si può ridurre il problema della totalità di una funzione al calcolo di g: se volessimo sapere se k è indice di una MT che calcola una funzione totale, potremmo prendere $x = k - 1$ e $y = 1$: se $g(x, y)$ è positivo, allora f_k è totale, se $g(x, y) = 0$ allora f_k è parziale.

Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

19 giugno 2023

Algoritmi e strutture dati

Esercizio 3 (8 punti)

Si forniscano limiti asintotici superiori per le seguenti equazioni alle ricorrenze, giustificando brevemente la risposta:

1. $$T(n) = 9\ T\left(\frac{n}{\sqrt[3]{3}}\right) + 14n^6$$
2. $$T(n) = 8\ T\left(\frac{n}{64}\right) + \sqrt[3]{n}$$
3. $$T(n) = 25\ T\left(\frac{n}{5}\right) + 30n^3$$

Soluzione

1. Si tratta di una ricorrenza che rispetta le ipotesi del Master theorem, in particolare del secondo caso. Abbiamo infatti che che $n^x, x = \log_{\sqrt[3]{3}}(9) = 3\log_3(9) = 6$ e $f(n) = 14n^6 \in \Theta(n^6)$. La ricorrenza è quindi $\Theta(n^6 \log(n))$.
2. Si tratta di una ricorrenza che rispetta le ipotesi del Master theorem (primo caso), in particolare abbiamo che $n^x, x = \log_{64}(8) = \frac{1}{2}$ e $f(n) = \sqrt[3]{n}$, dunque $f(n) \in \mathcal{O}(n^{\frac{1}{2}-\epsilon})$. La ricorrenza è quindi $\Theta(\sqrt{n})$.
3. Si tratta di una ricorrenza che rispetta le ipotesi del Master theorem, in (terzo caso). Abbiamo infatti che che $n^x, x = \log_5(25) = 2$ e $f(n) = 30n^3 \in \Omega(n^{2+\epsilon})$. La condizione di regolarità è automaticamente soddisfatta da $f(n) = n^3$ in quanto è un polinomio; la ricorrenza è quindi $\Theta(n^3)$.

Esercizio 4 (8 punti)

1. Si considerino alberi binari (dunque senza particolare criterio d'ordine tra le chiavi i cui valori sono irrilevanti e possono essere trascurati). Si indichi lo pseudocodice di una procedura che, dato un riferimento alla radice dell'albero, ne rimuove tutte e sole le foglie e restituisce il riferimento alla radice dell'albero da cui sono state rimosse. Si analizzi la complessità temporale e spaziale della procedura. La procedura deve essere a complessità temporale e spaziale minima.

Soluzione

La rimozione delle foglie viene effettuata tramite una procedura ricorsiva senza necessità di variabili ausiliarie (costo spaziale costante). La procedura visita l'albero fino alle foglie (chiamate ricorsive alle linee 5–6). Ciascuna foglia non viene ricollegata all'albero (linee 3–4), restituendo NIL al posto del riferimento alla foglia stessa. La procedura ha complessità temporale $\Theta(n)$, con n numero di nodi dell'albero.

```
Autunno(T)
1  if (T = NIL)
2        return T // L'albero è vuoto
3  if (T.left = NIL) ∧ (T.right = NIL)
4        return NIL // Il sottoalbero considerato è una foglia
5  T.left := Autunno(T.left)
6  T.right := Autunno(T.right)
7  return T
```

2. Si realizzi una procedura che, dato il riferimento alla radice di un albero binario, lo renda un albero *perfetto*, ovvero aggiunga ad esso tutti i nodi per ottenere un albero completo avente tutte le foglie possibili, con altezza pari a quella dell'albero ricevuto in ingresso. Si analizzi la complessità temporale e spaziale della procedura. La procedura deve essere a complessità temporale e spaziale minima.

SOLUZIONE

Per maggior intelligibilità, la procedura che risolve l'esercizio è suddivisa in tre sotto-procedure: la procedura principale RIEMPI invoca due sotto-procedure CALCOLAALT e RIEMPIRIC. CALCOLAALT calcola l'altezza dell'albero considerando, dalle foglie alla radice, l'altezza massima del sottoalbero del nodo corrente, e restituendola incrementata di 1. La funzione assegna convenzionalmente un'altezza pari a -1 ad un albero vuoto. La procedura ha complessità temporale lineare nel numero di nodi (effettua sostanzialmente una visita dell'albero) e complessità spaziale pari all'altezza dell'albero (se si considerano le variabili contenute nei record di attivazione al momento della chiamata più profonda).

RIEMPIRIC aggiunge all'albero i nodi mancanti, partendo dalla radice e creando i figli del nodo correntemente visitato nel caso il valore di h, altezza dell'albero radicato nel nodo T, sia maggiore di zero. I nodi vengono aggiunti solamente nel caso in cui l'albero non possieda già un nodo nella posizione considerata. La complessità temporale di RIEMPIRIC è sempre $\mathcal{O}(2^h)$, dove h è l'altezza dell'albero perfetto, così come quella spaziale, dominata dall'allocazione dello stesso. Nel caso pessimo (l'albero in ingresso è in realtà una lista di n nodi, quindi $h = n$) l'algoritmo ha quindi complessità $\mathcal{O}(2^n)$.

```
CalcolaAlt(T)
1  if (T = NIL)
2        return −1
3  m := Max (CalcolaAlt(T.left), CalcolaAlt(T.right))
4  return m + 1
```

```
RiempiRic(T, h)
1  if (h ≤ 0)
2        return
3  if (T.left = NIL)
4        T.left := NewNode() // NewNode crea un nuovo nodo con entrambi i figli impostati a NIL
5  if (T.right = NIL)
6        T.right := NewNode() // NewNode crea un nuovo nodo con entrambi i figli impostati a NIL
7  RiempiRic(T.left, h − 1)
8  RiempiRic(T.right, h − 1)
9  return
```

```
Riempi(T)
1  h := CalcolaAlt(T)
2  RiempiRic(T, h)
3  return
```