Algoritmi e Principi dell'Informatica

Appello del 25 Febbraio 2016

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1 e 2 in 1 ora e 15 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3 e 4 in 1 ora e 15 minuti.

NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (Punti 9)

1) Si definisca formalmente il seguente modello di automa, chiamato *macchina di Turing monodirezionale ad un nastro* (MTM1).

Una MTM1 è un accettore nondeterministico (non effettua quindi traduzioni) dotato di un nastro di ingresso e un nastro di memoria. Il funzionamento è analogo a quello di una MT tradizionale, a parte il movimento delle due testine dei nastri: esso può essere solamente S (stop) o R (right).

2) Si confronti la capacità espressiva delle MTM1 con i modelli visti a lezione.

3) Si consideri una variante "a nastro singolo" e deterministica della MTM1 e se ne valuti l'eventuale differenza di capacità espressiva.

Esercizio 2 (Punti 8)

Si consideri il seguente insieme.

S = { i | La macchina di Turing M_i termina la sua esecuzione in meno di 100 passi per qualche ingresso }

S è ricorsivamente enumerabile? S è ricorsivo? Giustificare brevemente le risposte.

Esercizio 3 (Punti 7)

Sia dato un insieme di n interi distinti. Si descriva un algoritmo che restituisce il k-esimo elemento più piccolo e se ne valuti la complessità; ovviamente la valutazione della soluzione proposta terrà conto della complessità ottenuta. Nella specifica della soluzione è possibile far riferimento ad algoritmi noti, specificandone con precisione funzionalità e complessità.

Esercizio 4 (Punti 9)

Sia T un albero binario. Sia t un suo nodo e siano t.left e t.right il sottoalbero sinistro e destro di t. Denotiamo con nodi(t) il numero di nodi del sottoalbero radicato in t.
Un albero binario si dice perfettamente bilanciato quando per ogni suo nodo t la differenza tra il numero dei nodi dei due sottoalberi di t è in valore assoluto al più 1:

$$|\,\text{nodi(t.left)} - \text{nodi(t.right)}\,| \leq 1.$$

Si definisca un algoritmo che, dato in input un nodo t, verifica se l'albero radicato in t è perfettamente bilanciato. Si discuta la complessità dell'algoritmo definito.

Tracce di soluzioni

Esercizio 1
1)

$A = <Q, \Sigma, \Gamma, q_0, F, \delta, Z_0>$, stato iniziale $q_0 \in Q$, stati finali $F \subseteq Q$

Funzione di transizione δ: $(Q \setminus F) \times \Sigma \times \Gamma \rightarrow \wp (Q \times \Gamma \times \{S, R\}^2)$

Configurazione: per comodità usiamo come seconda componente la stringa in ingresso ancora da leggere (cioè la parte destra della stringa, testina inclusa); per la terza componente, che rappresenta il nastro, indichiamo la parte scritta e visitata dalla macchina (cioè la parte sinistra, testina inclusa).

$<q, a.x, y.b> \vdash <q', a'.x, y.b'.c>$, se $(q', b', m_1, m_2) \in \delta(q,a,b)$ dove

$a' = \varepsilon$ se $m_1 = R$ altrimenti $a' = a$

$c = \varepsilon$ se $m_2 = S$ altrimenti $c = \_$ (blank)

$x \in L(A)$ sse $<q_0, x, Z_0> \vdash^* <q_F, x', y>$, $q_F \in F$.

2) Una MTM1 accede solo ad una cella di memoria (quella corrente) e non può in alcun modo ritornare sulla parte sinistra del nastro di memoria. Questa informazione è finita ($\in \Gamma$) e può essere facilmente codificata nello stato dell'organo di controllo. Per questo motivo le MTM1 definiscono tutti e soli i linguaggi Regolari.

3) L'uso di un unico nastro e del determinismo non cambia nulla.

Esercizio 2

S_1 è ricorsivo (e quindi anche ricorsivamente enumerabile). Infatti basta verificare se la macchina si arresta entro 99 passi per qualche stringa di lunghezza <100. Tali stringhe sono in numero finito, quindi la verifica avviene in un numero finito di passi. Non occorre considerare stringhe più lunghe perché, se anche ci fosse una stringa più lunga di 100 caratteri tale per cui la macchina si arresti in meno di 100 mosse, e quindi avendo letto un prefisso di lunghezza minore di 100, necessariamente vi sarebbe anche una stringa con lo stesso prefisso ma lunga meno di 100 che farebbe arrestare la macchina nello stesso numero di mosse.

Si noti che in questo caso non è applicabile il teorema di Rice, in quanto la proprietà delle macchine di Turing considerate per l'insieme S_1 (terminare in meno di 100 passi per qualche ingresso) non è una proprietà della funzione calcolata, bensì una proprietà strutturale della macchina.

Esercizio 3
Si crea un coda di priorità. Si invoca k volte la removeMin(). Si restituisce in output il valore trovato con l'ultima chiamata. Se la coda di priorità è implementata tramite un min-heap, allora la costruzione dello heap ha complessità O(n). Il costo delle k invocazioni di removeMin() è O(k log n). Si ottiene quindi la complessità desiderata.

Esercizio 4
Ci sono diverse soluzioni con complessità $\Theta(n)$ (n essendo il numero di nodi nel sottoalbero considerato), tutte basate sulla visita in post-ordine del sottoalbero. Riportiamo di seguito una soluzione "intuitiva", che ha complessità temporale $\Theta(n)$ e che: *i)* fa uso di memoria aggiuntiva per

memorizzare in ogni nodo il numero di nodi del suo albero; *ii)* visita l'albero due volte: una prima volta per calcolare il conteggio dei nodi (funzione *count(t)*) e una seconda volta per verificare il bilanciamento. È possibile definire algoritmi più "compatti", che attraversano l'albero una sola volta, restituendo a ogni chiamata ricorsiva il conteggio dei nodi del sottoalbero visitato e il risultato della verifica di bilanciamento per il sottoalbero. Per tale classe di soluzioni è necessario fare attenzione al conteggio dei nodi nei vari sottoalberi, per evitare di conteggiare più volte gli stessi nodi. Questo, infatti, comporterebbe un numero maggiore di chiamate ricorsive e il conseguente incremento della complessità.

```
boolean checkBalance(TREE t) {
 count(t);
 return checkBalanceRic(t);

}

integer count(TREE t) {

  if t = nil
     return 0
  t.count = count(t.left) + count(t.right) + 1
  return t.count
}

boolean checkBalanceRic(TREE t) {

 integer nodiL = 0, nodiR = 0
 boolean balancedL = true, balancedR = true

 if t.left != nil {
            nodiL = t.left.count
            balancedL = checkBalanceRic(t.left)
 }

 if t.right != nil {
           nodiR = t.right.count
           balancedR = checkBalanceRic(t.right)
 }

  return (balancedL AND balancedR AND (|nodiL – nodiR| <= 1)
 }
```