Algoritmi e Principi dell'Informatica

Appello del 20 Febbraio 2012

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

Chi deve sostenere solo il modulo di Informatica teorica deve svolgere le parti a), b) e c) dell'Esercizio 1 e l'Esercizio 2 in 1h.

Chi deve sostenere solo il modulo di Informatica 3 deve svolgere la parte d) dell'esercizo 1 e l'esercizio 3 in 1h e 30'.

Esercizio 1 (punti 12/30-esimi)

Si consideri il seguente linguaggio:

L_1 = { ab, abab, ababab, ...,

aabb, aabbaabb, aabbaabbaabb, …,

…,

$a^n b^n$, $a^n b^n a^n b^n$, $a^n b^n a^n b^n a^n b^n$, …,

…}

a) Fornire una specifica logica del linguaggio L_1.

b) Sintetizzare un automa a *potenza minima* (tra FSA, PDA, TM) in grado di riconoscere L_1.

c) Si consideri il linguaggio L_2={ $a^n b^n$ }$^+$. I linguaggi L_1 e L_2 sono equivalenti?

d) Si valutino le seguenti complessità relative al riconoscimento di L_1:

1. Complessità spaziale minima ottenibile mediante una macchina di Turing a k nastri
2. Complessità temporale minima ottenibile mediante una macchina di Turing a k nastri
3. Complessità spaziale minima ottenibile mediante una RAM, a criterio di costo logaritmico
4. Complessità temporale minima ottenibile mediante una RAM, a criterio di costo logaritmico

Si giustifichino brevemente le risposte date

Esercizio 2 (punti 8/30-esimi)

Si considerino le seguenti funzioni:

(si rammenta che il simbolo $\lfloor y \rfloor$ denota il massimo intero $\leq y$ e $\lceil y \rceil$ il minimo intero $\geq y$; $|y|$ denota il valore assoluto di y; Z denota l'insieme dei numeri interi.)

$f_1: Z \rightarrow Z,\ f_1(x) = \lfloor e^{\lfloor |x| \rfloor} \rfloor$

$f_2: Z \rightarrow Z,\ f_2(x) = \lceil e^{\lfloor x \rfloor} \rceil$

$f_3: Z \rightarrow Z,\ f_3(x) = \lfloor x^2 \rfloor + 1$

$f_4: Z \rightarrow Z,\ f_4(x) = \lceil \log (\lceil |x| \rceil) \rceil$

Siano poi $\mathcal{T}$ l'insieme delle funzioni totali $Z \rightarrow Z$, e

$\mathcal{H} \equiv \{f: Z \rightarrow Z \mid \exists x(f(x) = 0)\}$

Dire, giustificando la risposta, se sia decidibile il problema di stabilire se, dato un generico programma C o Java, esso computi una delle funzioni appartenenti all'insieme

$\mathcal{F} = \{f_1, f_2, f_3, f_4\} \cap \mathcal{T} \cap \mathcal{H}$

Esercizio 3 (punti 10/30-esimi)

Si abbiano due alberi rosso-neri, rispettivamente di cardinalità *m* e *n*, con $m << n$. Si definisca, mediante opportuno pseudocodice un algoritmo che costruisca una lista (monodirezionale) ordinata in ordine crescente contenente tutti e soli gli elementi appartenenti ad entrambi gli alberi, cercando di ottenerne la miglior complessità temporale possibile. Si valuti tale complessità in funzione dei due parametri *m* e *n*.

Tracce di Soluzioni

Esercizio 1

a)

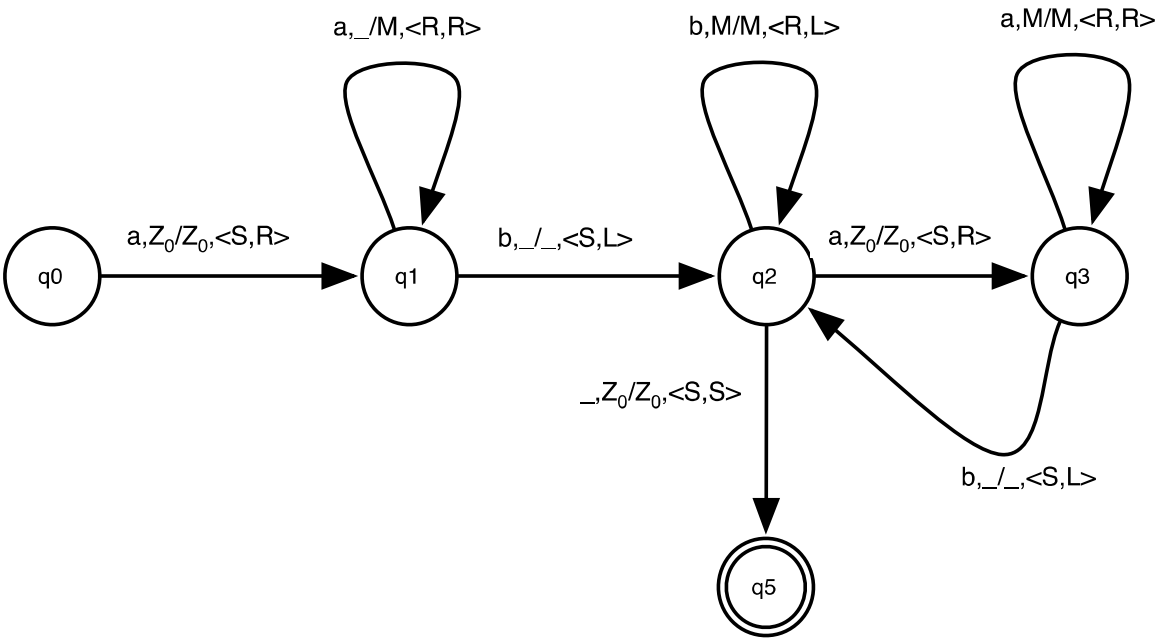
Indichiamo con L(k) il sottolinguaggio { $a^k b^k$, $a^k b^k a^k b^k$, $a^k b^k a^k b^k a^k b^k$, …}

Specifica di L(k): $\forall x \forall k\ (x \in L(k) \leftrightarrow x = a^k b^k \vee \exists y (y \in L(k) \wedge x = a^k b^k . y))$
(si sottintende la definizione di w^k per una generica stringa w e un generico intero k)
Specifica di L_1: $\forall x (x \in L_1 \leftrightarrow \exists k (x \in L(k)))$

b)

Il linguaggio non può essere riconosciuto da un automa a pila, in quanto richiede la capacità di ricontare lo stesso numero più volte.

Una macchina di Turing che riconosce L_1 è definita come segue:



c)

Il linguaggio L_1 non è equivalente al linguaggio L_2, bensì ne è un sottoinsieme proprio. Un esempio di stringa di L_2 che non appartiene a L_1 è abaabb.

d.1) Una MT può riconoscere L_1 con complessità spaziale O(log(n)), n essendo la lunghezza della prima sequenza di a, e quindi ≤ della lunghezza dell'intera stringa di ingresso. Essa può infatti memorizzare il valore di n in binario (ad esempio su due nastri); ad ogni successiva scansione di una serie di a o di b usa questo contatore decrementandolo di un'unità per ogni carattere letto; al passaggio alla sequenza successiva lo risetta al valore originario (copiandolo dall'altro nastro.

d.2) La MT di cui al precedente punto d.1) ha una complessità O(m.log(n)), dove m è la lunghezza dell'intera stringa. Un'analisi più raffinata potrebbe evidenziare che ogni conteggio di n a o b può essere fatto in O(n) (perché l'aggiornamento del contatore richiede:

$$O(n \sum_{h=0}^{\lfloor \log(n) \rfloor} \frac{h}{2^h}) = O(n)$$

); posto poi k = m/n sono necessarie k copiature di log(n) = log(m/k) bit; in ogni caso la complessità temporale totale sarebbe O(m).

Una tale analisi è però superflua perché è sufficiente constatare che la semplice MT di cui al punto b) ha evidentemente complessità temporale (e spaziale) O(m).

d.3 e d.4) Una RAM che riconosca L_1 deve necssariament memorizzare il valore di n in un contatore (meglio in due) e poi procedere in modo analogo alla precedente MT; ogni aggiornamento del contatore però costa alla RAM O(log(n)). Quindi nel caso pessimo la complessità spaziale è O(log(n)) e quella temporale O(m.log(m)) (quando n è O(m)).

Esercizio 2

L'insieme $\mathcal{F}$ è vuoto. Infatti l'insieme immagine di f_1, f_2 e f_3 non contiene lo 0 (i loro valori sono tutti > 0) e f_4 non è totale. Quindi nessun algoritmo può computare una funzione di $\mathcal{F}$ e il problema posto è banalmente decidibile.

Esercizio 3

Uno schema di algoritmo per risolvere il problema posto è il seguente:

- Siano T_m e T_n rispettivamente i due alberi di cardinalità m e n;
- Si individui il massimo elemento di T_m (Questa operazione, per un albero rosso-nero costa O(log(m)));
- Si verifichi se questo elemento si trova in T_n (costo O(log(n)));
    - Se l'elemento corrente si trova anche in T_n lo si inserisca in testa alla lista, inizialmente vuota (costo O(1));
- Si ripeta il ciclo precedente per tutti i successivi nodi di T_m in ordine decrescente. Si noti che per fare ciò occorre tenere traccia del nodo esaminato nella precedente iterazione del ciclo: indicando con N tale nodo, occorre quindi individuare il nodo a valore massimo a sinistra di N (assumendo che i nodi a valore minore siano a sinistra), ricerca che può comunque essere eseguite in tempo O(log(m)).

La complessità totale è quindi O(m.(log(m)+log(n))), che, essendo m < n è anche O(m.log(n)).

Un altro algoritmo a complessità sostanzialmente equivalente potrebbe "linearizzare" T_m in una lista ordinata in ordine crescente (a costo O(m)) e successivamente, per ogni lemento della lista verifcare se si trova in T_n (costo (O(log(n)) e in caso negativo eliminarlo dalla lista con costo O(1).

Si noti che la scelta di un tale algoritmo è giustificata dall'ipotesi che m << n. In caso contrario, ad esempio se n ed m fossero sostanzialmente vicini, converrebbe linearizzare entrambi gli alberi in O(n) e poi calcolarne l'intersezione sempre in O(n). Tuttavia, se, ad esempio, m fosse O(log(n)), una complessità $O(\log^2(n))$ sarebbe nettamente preferibile.

Si noti anche che in caso di ripetizioni entrambi gli algoritmi manterrebbero nella lista risultante il numero di occorrenze in T_m, non quello delle occorrenze in T_n.