Algoritmi e Principi dell'Informatica

Appello del 21 Settembre 2015

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 7)

Si consideri il linguaggio seguente:

$L_1 = \{a,b,c\}^* - a^*.b^*$

dove il – indica la differenza insiemistica.

Si costruiscano una macchina astratta che riconosca L_1 e una grammatica che lo generi; è fortemente raccomandato che entrambi i formalisimi siano a "potenza minima" ossia appartenenti alla categoria di automi/grammatiche a minor potenza riconoscitiva/generativa possibile.

Esercizio 2 (Punti 8)

Si consideri, oltre al linguaggio L_1 dell'esercizio 1, anche il linguaggio L_2 seguente:

$L_2 = \{a,b\}^* - \{a^n b^n \mid n > 0\}$

Si dica, motivando brevemente la risposta, quali dei seguenti problemi sono decidibili e quali no:

1. Stabilire se un generico automa a stati finiti riconosce L_1
2. Stabilire se un generico automa a stati finiti riconosce L_2
3. Stabilire se una generica macchina di Turing riconosce L_1
4. Stabilire se una generica macchina di Turing riconosce L_2

Esercizio 3 (Punti 7)

Si consideri il linguaggio $L = \{wcwcw \mid w \in \{a,b\}^+\}$. Si descriva una macchina RAM che riconosca L, minimizzando la complessità spaziale sia a criterio di costo costante che logaritmico.

Esercizio 4 (Punti 8)

Sia dato un array A di n interi che assumono valori nell'intervallo [0, …, n+1]. L'array non contiene valori duplicati; i valori in esso contenuti saranno quindi tutti quelli compresi nell'intervallo considerato, tranne due. A non è ordinato.

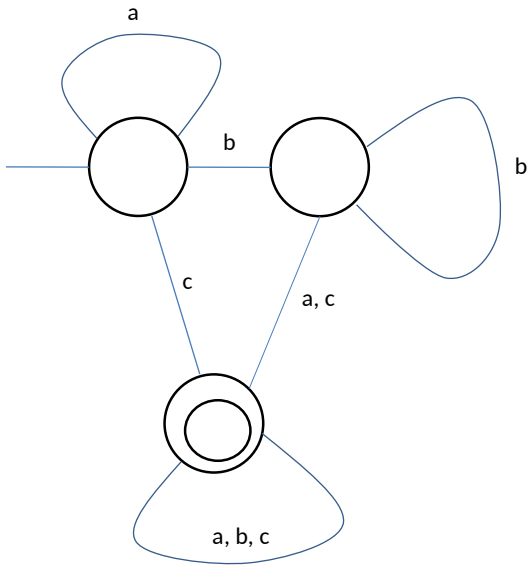
Si specifichi un algoritmo che individui e produca in output i due valori mancanti in A e se ne discuta la complessità. La valutazione dell'algoritmo proposto dipenderà dalla sua complessità.

Tracce di soluzioni

Esercizio 1

L_1 è la differenza tra due linguaggi regolari (di cui il primo è il monoide libero sull'alfabeto {a,b,c}. Siccome i linguaggi regolari sono chiusi rispetto alla differenza insiemistica, e in particolare rispetto al complemento, L_1 è pure regolare.

Un automa a stati finiti e una G regolare che riconosca e generi L_1, rispettivamente sono i seguenti:



$S \rightarrow aS \mid bB \mid cC \mid c$
$B \rightarrow bB \mid aC \mid a \mid cC \mid c$
$C \rightarrow aC \mid bC \mid cC \mid a \mid b \mid c$

Esercizio 2

1. Il problema è decidibile: in generale è decidibile l'equivalenza tra due automi a stati finiti ($L(A_1) = L(A_2)$ se e solo se $L(A_1) \subseteq L(A_2)$ e $L(A_2) \subseteq L(A_1)$; $L(A_1) \subseteq L(A_2)$ se e solo se $L(A_1) \cap (\neg L(A_2)) = \varnothing$, dove $\neg L(A_2)$ indica il complemento di $L(A_2)$).
   Quindi in particolare è decidibile se un generico A riconosce L_1.
2. Il problema è non solo decidibile ma anche banalmente deciso sulla base del fatto che L2 è (deterministico) non-contestuale ma non regolare; quindi nessun automa a stati finiti può riconoscerlo.
3. Il problema non è decidibile in base al teorema di Rice; infatti l'insieme delle MT che riconoscono L_1 (risolvono il problema rappresentato da L_1) palesemente non è né l'insieme vuoto né l'insieme universo.
4. Lo stesso ragionamento vale per L_2.

Osservazione a latere: essendo decidibile il problema dell'equivalenza tra automi a pila deterministici (risultato relativamente recente e di molto complessa dimostrazione!) il problema di stabilire se un generico automa a pila deterministico riconosca L_1 o L_2 è decidibile poiché entrambi i linguaggi sono deterministici noncontestuali.

Esercizio 3
Basta vedere i fattori w come numeri binari, per comodità con le cifre meno significative a sinistra, dove ad es a=0 e b=1. Si utilizza una cella di memoria per codificare w (es. M[1]), moltiplicando per 2 tutte le volte che si legge una nuova cifra, e sommando 1 se essa è b. Alla lettura della prima c si codifica in maniera analoga il secondo fattore w in M[2]. Alla seconda c si controlla che M[1] sia uguale ad M[2], poi si procede a codificare il fattore w finale in M[1]; l'ultimo controllo M[1]=M[2] avviene a fine stringa.
La complessità spaziale risulta pari a $\Theta(1)$ a criterio costante e $\Theta(n)$ a criterio logaritmico, con n lunghezza della stringa in ingresso.

Esercizio 4

E' possibile ispirarsi all'algoritmo del counting sort e utilizzare un array ausiliario B di n+2 valori booleani (0 e 1), tale che B[i] = 1 se i è presente nell'array; altrimenti B[i] = 0. L'algoritmo risultante (riportato di seguito) ha complessità $\Theta(n)$.

```
void mancanti(int A[], int n)
{
 int i, trovati = 0;
 int B[n+2];

   for (i=0; i< n+2; i++)
      B[i] = 0;
   for (i=0; i < n; i++)
      B[A[i]] = 1;
   i=0;

   while (i < n+2 && trovati < 2) {
       if (B[i] == 0) {
            print(i);
             trovato++;
                   }
   }
}
```