Algoritmi e Principi dell'Informatica
Appello del 14 settembre 2018

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore.
Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1 e 2 in 1 ora.
Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3 e 4 in 1 ora.
NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (9 punti)
Come di consueto, si indichi con f_i la funzione calcolata dalla *i*-esima macchina di Turing.
Si definisce il numero reale *t* nel modo seguente:

- La parte intera di *t* è 0
- L'espansione decimale di *t* è tale che la *i*-esima cifra dopo la virgola vale:
  - 0 se la funzione f_i non è computabile;
  - altrimenti, 1 se f_i non è totale;
  - altrimenti, 2 se f_i non è la funzione che calcola il quadrato del suo ingresso;
  - altrimenti 3.

Si indichi con la notazione $t[n]$ la sequenza delle prime *n* cifre nell'espansione decimale di *t*.
Si considerino quindi le funzioni *g* e *h* definite come segue:

- $g(x) = t[100]$ se x=0, altrimenti $\perp$;
- $h(x) = t[x]$.

Sono calcolabili le funzioni *g* e *h*? Motivare opportunamente la risposta.

Esercizio 2 (7 punti)
Si consideri la seguente grammatica G (con le consuete convenzioni di notazione):

$S \rightarrow aABb$
$A \rightarrow cC$
$B \rightarrow Cc$
$C \rightarrow cC \mid Cc \mid c$
$CC \rightarrow D$
$D \rightarrow dD \mid d$

Si costruisca un automa appartenente a una famiglia a potenza riconoscitiva minima che riconosca L(G).

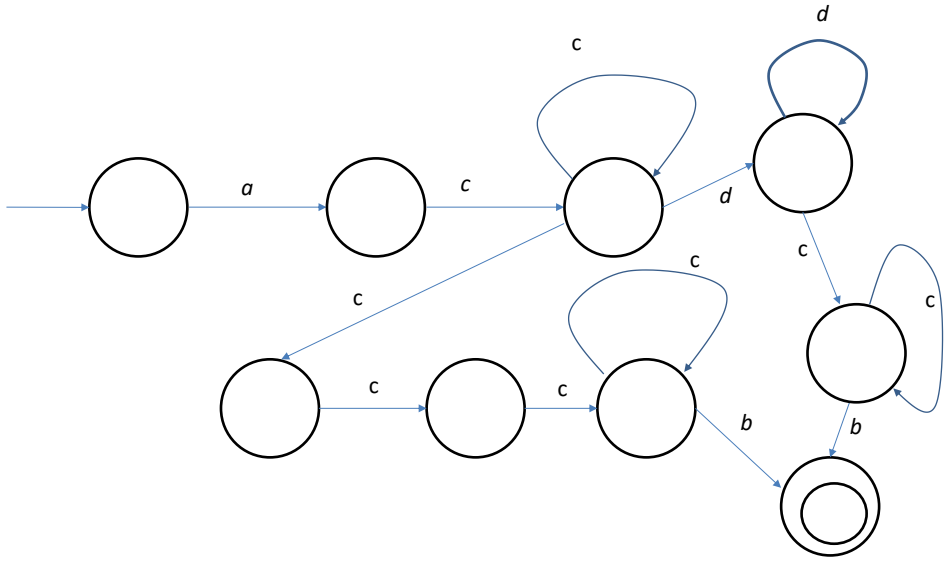
Esercizio 3 (9 punti)
Si vuole realizzare un software per la ricerca anagrammi: si fornisce una parola in ingresso e si identificano gli anagrammi di tale parola all'interno di un elenco di *N* parole. La lunghezza di ciascuna parola è di al più *M* caratteri. Si descriva un algoritmo che risolva il problema in oggetto e se ne valuti la complessità temporale fornendo un opportuno limite asintotico superiore (in funzione di *N* e *M*).

Esercizio 4 (7 punti)
Si consideri il linguaggio su alfabeto {0,1} fatto dalle stringhe x.y, dove |x| = 5, codifica in binario la posizione in y di un carattere che deve essere 1.
Si descrivano esaurientemente una macchina di Turing a nastro singolo e una macchina RAM che accettano il linguaggio, valutandone le complessità spaziali e temporali con tutti i criteri di costo disponibili.

Tracce delle soluzioni

Esercizio 1
Notiamo intanto che ogni funzione f_i è computabile (per definizione!), quindi non ci sono 0 nell'espansione decimale di t.
La funzione g è certamente calcolabile, in quanto la condizione x=0 è decidibile e $t[100]$ è una costante (numero di 100 cifre comprese tra 1 e 3). Per ognuna delle 3^{100} possibili sequenze di 100 cifre comprese tra 1 e 3, è senz'altro possibile costruire una macchina di Turing che restituisca tale sequenza quando x=0 (e che non si arresti altrimenti). Quindi, anche se non sappiamo necessariamente quale sequenza corrisponda a $t[100]$, esiste comunque una macchina di Turing che effettua il calcolo desiderato.
La funzione h non è invece calcolabile. Se lo fosse, sarebbe possibile decidere il problema della totalità di una funzione, che è notoriamente indecidibile. Per decidere se una generica funzione f_i sia totale, basterebbe infatti calcolare $h(i)$ e osservare se l'ultima cifra del risultato sia 1 (nel qual caso la funzione non è totale) o maggiore di 1 (nel qual caso la funzione è totale).

Esercizio 2
L(G) è il linguaggio regolare $\{accccc^*b\} \cup \{ac^+d^+c^+b\}$. Quindi esso può essere riconosciuto da un automa a stati finiti, ad esempio il seguente (nondeterministico).



Esercizio 3
Una soluzione particolarmente inefficiente della ricerca di anagrammi consiste nell'enumerare le $O(M!)$ permutazioni dei caratteri della parola in ingresso e verificare, per ogni permutazione, se essa corrisponda a una parola tra le N in elenco. Questo porta a una complessità di $O(M!\ N\ M)$, ipotizzando che il confronto di uguaglianza tra due parole costi $O(M)$.
Alternativamente, un semplice modo per verificare se due parole siano l'una un anagramma dell'altra consiste nell'ordinare alfabeticamente i caratteri di entrambe le parole e verificare che i risultati siano identici. Questo ha una complessità di $O(M \log M + M) = O(M \log M)$. Si può anche utilizzare un *counting sort*, con complessità di $O(M + k)$ per un alfabeto di k caratteri; tale complessità è $O(M)$ se k è costante, come è ragionevole supporre ad esempio nel caso dei caratteri dell'alfabeto inglese (k=26) o dei codici ASCII (k=128) o Unicode (k=17*65536).

L'algoritmo si può quindi implementare verificando, per ciascuna delle *N* parole in elenco, se sia un anagramma della parola in ingresso, con una complessità di O(*N M* log *M*) (oppure O(*N M*) se si utilizza il *counting sort* con alfabeto fisso).

Esercizio 4

La stringa x ha lunghezza fissata, dunque la posizione significativa più a destra nella stringa di ingresso sarà 11111 in binario, cioè 31 in decimale. Il numero di stringhe corrette, considerando i soli prefissi di lunghezza 5+31, è finito, quindi basta un automa a stati finiti per accettare il linguaggio, che si fermerà al più dopo 36 mosse. Il funzionamento sarà praticamente analogo sia per MT che per la RAM.

MT a nastro singolo: $T(n) = \Theta(1)$, $S(n) = \Theta(n)$ (la stringa in ingresso sarà comunque interamente sul nastro)

RAM, costo costante e logaritmico: $T(n) = \Theta(1)$, $S(n) = \Theta(1)$.