Algoritmi e Principi dell'Informatica

Soluzioni al Tema d'esame
15 Luglio 2020

1 Informatica teorica

Esercizio 1

a) Scrivere una grammatica a potenza generativa minima che generi il linguaggio L_1 di tutte le stringhe, sull'alfabeto $A = \{0, 1\}$, aventi la seguente proprietà: la stringa è non vuota e, se letta da sinistra a destra, è il complemento della stringa letta da destra a sinistra. Per complemento di una stringa si intende la stringa ottenuta scambiando gli 0 con gli 1.

b) Scrivere una grammatica a potenza generativa minima che generi il linguaggio L_2 di tutte le stringhe, sull'alfabeto $A = \{0, 1\}$, che non sono palindrome. Una stringa è palindroma se non cambia leggendola da sinistra a destra e da destra a sinistra.

c) Con riferimento alle proprietà di chiusura dei vari tipi di linguaggi rispetto alle operazioni insiemistiche, stabilire di che tipo è la grammatica a potenza generativa minima che genera l'intersezione di L_1 e L_2.

Soluzione

a) *Per L_1:* $S \to 0S1 \mid 1S0 \mid 01 \mid 10$

b) *Per $L2$:* $\begin{array}{l} S \to 1X0 \mid 0X1 \mid 0S0 \mid 1S1 \\ X \to 1X \mid 0X \mid \epsilon \end{array}$

c) *L'intersezione tra L_1 e L_2 è ancora L_1 (nessuna stringa di L_1 è palindroma), quindi basta una grammatica non contestuale. Naturalmente questo è compatibile con il fatto che non ci sia chiusura rispetto all'intersezione tra linguaggi contestuali, il che indica semplicemente che esistono due linguaggi NC la cui intersezione non è NC, ma non vuol dire che l'intersezione di due linguaggi NC non possa essere NC.*

Esercizio 2

a) Data una generica macchina di Turing non deterministica M e una generica stringa x, è decidibile stabilire se esistono almeno due cammini distinti di computazione effettuati da M che portano all'accettazione della stringa x?

b) Il problema è semidecidibile?

c) Come cambiano le risposte alle domande precedenti se M è un automa a stati finiti?

Soluzione

a) *Possiamo esibire una riduzione dal problema dell'arresto al problema in questione, che quindi è indecidibile. Sia M' una generica macchina non deterministica. Costruiamo una nuova macchina M'' uguale a M' ma con l'aggiunta di un cammino non deterministico che, dallo stato iniziale, accetta x in modo banale, ossia compiendo un passo verso un nuovo stato per ogni carattere di x. Se potessi decidere se M'' ammette almeno due cammini che accettano x potrei allora decidere se la (generica) macchina M' accetta la (generica) stringa x.*

*b) Il problema è semidecidibile perché, se esistono due cammini che accettano x, prima o poi li trovo provando tutti i possibili cammini con tecnica diagonale (*dovetailing*).*

c) Con un FSA il problema diventa decidibile perché posso, ad esempio, enumerare tutti i (finiti) cammini di lunghezza $|x|$ sull'automa a partire dallo stato iniziale e vedere se almeno due accettano x.

2 Algoritmi e strutture dati

Esercizio 1
Si consideri il problema di convertire valori codificati in binario in valori codificati in ottale (cioè in base 8).

1. Definire una MT a k nastri che legge da input un valore naturale codificato in binario (scritto partendo dal bit MENO significativo) e scrive in output lo stesso valore, ma codificato in ottale (scritto sempre a partire dalla cifra MENO significativa). Dire quali sono le complessità temporale e spaziale della MT a k nastri ideata.

2. Definire una MT a nastro singolo che esegue la trasformazione analoga a quella descritta al punto 1 (alla fine sul nastro deve rimanere il valore iniziale, ma codificato in ottale, partendo dalla cifra meno significativa). Dire quali sono le complessità temporale e spaziale della MT a nastro singolo ideata.

NB: Le definizioni delle MT si possono dare a parole, ma in modo sufficientemente preciso per potere valutare la complessità delle MT ideate.
Soluzione

1. *Per ottenere la prima trasformazione è sufficiente leggere i simboli in input a 3 a 3, e trasformare ogni terna nella corrispondente cifra ottale (per cui 000 diventa 0, 001 diventa 1, 010 diventa 2, ecc.; attenzione che la lettura viene fatta dal bit meno significativo). La complessità spaziale è costante, perché nulla viene memorizzato nei nastri di memoria. La complessità temporale è chiaramente $\Theta(n)$, con n lunghezza della stringa in input.*

2. *La seconda trasformazione invece richiede tempo n^2 (con n sempre la lunghezza della stringa in input). Infatti, non solo ogni terna di cifre binarie in input deve essere convertita in una cifra ottale (in questo caso viene fatta una vera e propria sostituzione, per cui si può pensare che l'ultima cifra di ogni terna venga trasformata nella cifra ottale, e le altre 2 vengano invece cancellate, riportandole a blank), ma, alla fine della conversione delle terne, occorre eliminare gli spazi che si sono creati tra le cifre ottali, e questa operazione richiede un tempo $\Theta(n^2)$. La complessità spaziale è invece n, in quanto al massimo sul nastro si trovano n simboli.*

Esercizio 2
Si consideri la struttura dati *heap*, vista con l'algoritmo *heap-sort*, e se ne progetti una generalizzazione (si chiami essa *3-heap*) con heap ternari anziché binari. Si implementi *BUILD-MAX-HEAP* per *3-heap* e se ne valuti la complessità, confrontandola con la versione tradizionale, binaria, degli heap.
Soluzione

Per comodità si usi un array con indicizzazione da 0: in questo caso i tre figli di un nodo i sono $3i+1$ (left), $3i+2$ (center) e $3i+3$ (right), mentre il padre è $\lfloor (i-1)/3 \rfloor$.

MAX-HEAPIFY *risulta quasi identica; va aggiunto* `c := CENTER(i)` *e confrontato con gli altri per calcolare il massimo, inoltre va tenuto conto dell'indice 0, quindi un indice è valido solo se è strettamente minore di* `heap-size`*. La complesità risultante è* $O(log_3(n))$.

In BUILD-MAX-HEAP *si cambia il ciclo per i che va da* `A.length/3` *a 0. La valutazione della complessità è molto simile: altezza* $h = log_3(n)$*; il numero di nodi ad altezza* h *è* $\lceil 2h/3^{h+1} \rceil$*. Si ottiene quindi* $O(n)$*, con costanti moltiplicative più basse (si noti che* $\sum_{h=0}^{\infty} 2h/3^h = 3/2$*).*