Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

7 settembre 2023

Informatica teorica

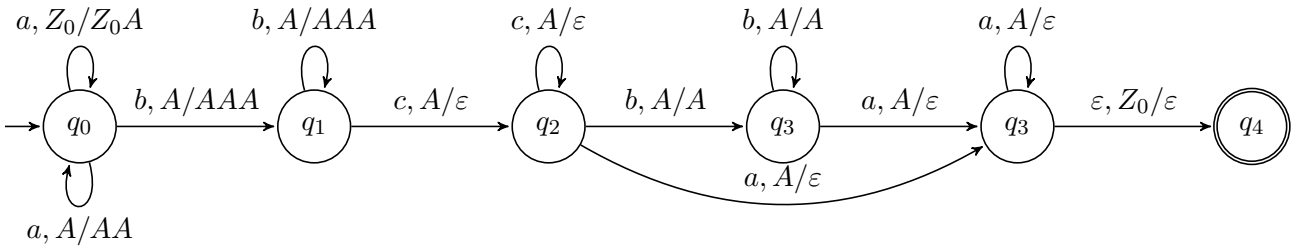
Esercizio 1 (8 punti)

Si costruisca un automa o una grammatica a potenza espressiva minima per il linguaggio:

$$L = \{a^x b^y c^z b^* a^w \mid x + 2y = z + w; x, y, z, w > 0\}.$$

Soluzione

Basta un automa a pila deterministico:



Esercizio 2 (8 punti)

Sia L un linguaggio decidibile definito sull'alfabeto A e $\bar{L}$ il suo complemento. Barrare le caselle opportune e motivare la propria risposta con esempi o dimostrazioni che la supportino.

a) L è infinito. Necessariamente☐ Possibilmente☐ Mai☐

b) $\bar{L}$ è infinito. Necessariamente☐ Possibilmente☐ Mai☐

c) Se L è fissato, il problema di stabilire se L è infinito è deciso. Vero☐ Falso☐ Dipende da L☐

Soluzione

a) Possibilmente. Esempi: A^* è infinito e decidibile; $\emptyset$ è finito e decidibile.

b) Possibilmente. Si prendano ad esempio i complementi dei linguaggi del punto precedente.

c) Dipende da L. Sia, ad esempio, L l'insieme dei numeri naturali oltre i quali esiste una coppia di numeri primi gemelli. Allo stato attuale non sappiamo se L sia finito o meno, ma è certamente decidibile, perché contiene necessariamente o tutti i numeri naturali oppure tutti quelli minori di un certo numero naturale h.

Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

7 settembre 2023

Algoritmi e strutture dati

Esercizio 3 (8 punti)
Si consideri la seguente traduzione dal linguaggio $L_1 = \{a^n b^n, n \in \mathbb{N}\}$ definito sull'alfabeto $A = \{a, b\}$ al linguaggio $L_2 = \{c^n b^n, n \in \mathbb{N}\} \cup \{a^n c^n, n \in \mathbb{N}\}$ definito sull'alfabeto $A = \{a, b, c\}$, tale per cui

$$\tau(a^n b^n) = \begin{cases} c^n b^n & \text{se } n = 2k, k \in \mathbb{N} \\ a^n c^n & \text{se } n = 2k+1, k \in \mathbb{N} \end{cases}$$

Si descriva con sufficiente precisione un algoritmo che effettui la traduzione suindicata con una MT a $k = 1$ nastro e con una MT a nastro singolo, analizzandone la complessità temporale e spaziale.

Soluzione

1. MT a $k = 1$ nastri: la MT legge dall'ingresso la stringa e scrive sul nastro di memoria un simbolo per ogni a consecutiva trovata. Scorre il nastro in direzione opposta a mano a mano che legge le b e verifica che siano in ugual numero. Scorre ulteriormente il nastro di memoria per determinare se i simboli sono in numero pari o dispari (memorizzando la parità negli stati dell'organo di controllo). Emette $c^n b^n$ o $a^n c^n$ a seconda della parità, scorrendo il nastro di memoria per tener conto di quante lettere vanno emesse. Complessità temporale e spaziale $\mathcal{O}(n)$. È possibile migliorare la complessità spaziale di questo algoritmo sostituendo il contatore unario sul nastro di memoria con un contatore binario, riducendo quindi la complessità spaziale a $\mathcal{O}(\log(n))$. Il processo di traduzione, una volta incrementato fino a n il contatore, ne crea un'ulteriore copia (a costo $\mathcal{O}(\log^2 n)$), e quindi decrementa fino a 0 la prima copia per emettere i primi n simboli e procede poi con la seconda copia per i secondi n simboli. Ricordando che, con opportuni accorgimenti, il costo complessivo di n operazioni di incremento/decremento di un contatore binario è $\mathcal{O}(n)$ la soluzione resta a complessità temporale lineare.

2. MT a nastro singolo: la MT sostituisce la prima a del nastro (se presente, altrimenti termina con errore) con $\bar{a}$, procede fino alla prima b a patto di incontrare solo a nel mentre. Sostituisce la b con $\bar{b}$ e scorre in verso opposto fino a trovare $\bar{a}$. Scorre in avanti di una cella e, se trova una a, la sostituisce con $\bar{a}$, riprendendo la marcatura. una volta terminata la marcatura delle a, oppure, nel caso in cui non siano disponibili b da marcare in corrispondenza, la MT effettua una scansione del nastro controllando se ci sono simboli non marcati. Se essi sono presenti si arresta con un errore. La parità del numero di a viene memorizzata nell'organo di controllo, aggiornandola ad ogni sostituzione di a effettuata. Se non sono presenti simboli non marcati, la MT procede, in una sola passata sul nastro, a sostituire le $\bar{a}$ o le $\bar{b}$ con c a seconda della parità memorizzata, e le rimanenti lettere con la loro versione non marcata. Complessità temporale $\mathcal{O}(n^2)$, complessità spaziale $\mathcal{O}(n)$.

Esercizio 4 (8 punti)

Definire mediante pseudocodice un algoritmo che, dato in input un array A di numeri interi, se esiste in A almeno un valore che compare più di una volta, restituisce le posizioni della coppia di valori uguali la cui distanza è massima. Se ci sono più coppie la cui distanza è massima, il programma restituisce quella corrispondente al valore più grande. Se A non contiene nessun elemento ripetuto, il programma solleva un errore.

Dare la complessità temporale dell'algoritmo ideato.

Soluzione

Un algoritmo banale che risolve il problema può sfruttare due cicli innestati tra di loro, i quali scorrono tutti i sottoarray alla ricerca di quello di lunghezza maggiore che è iniziato e terminato dagli stessi numeri. Un tale algoritmo ha chiaramente complessità temporale $\Theta(n^2)$, con n lunghezza dell'array A.

Un algoritmo più efficiente è il seguente. Si costruisce innanzi tutto un nuovo array, B, in cui ogni elemento $B[i]$ è una coppia di valori $\langle A[i], i\rangle$, (cioè il primo elemento di $B[i]$ è il valore di $A[i]$, e il secondo elemento è l'indice i stesso). A quel punto si ordina l'array B usando come definizione di ordinamento tra coppie la seguente: $\langle A[i], i\rangle < \langle A[j], j\rangle \Leftrightarrow A[i] < A[j] \vee (A[i] = A[j] \wedge i < j)$

Dopo averlo ordinato, B contiene una sequenza di coppie in cui tutte le coppie che hanno lo stesso primo valore sono adiacenti, e sono ordinate secondo la loro posizione in A. Per esempio, se

$$A = [0, -5, -3, 8, -3, -3, 8, 10]$$

dopo essere stato ordinato B conterrà (assumendo che gli indici inizino da 1)

$$[\langle -5, 2\rangle, \langle -3, 3\rangle, \langle -3, 5\rangle, \langle -3, 6\rangle, \langle 0, 1\rangle, \langle 8, 4\rangle, \langle 8, 7\rangle, \langle 10, 8\rangle].$$

Per trovare la coppia desiderata, l'algoritmo scorre B da destra a sinistra, controllando, per ogni elemento ripetuto, il valore più a destra, e quello più a sinistra, tenendo traccia della coppia con distanza massima trovata fino a quel momento (scorrendo da destra a sinistra, la coppia va aggiornata solo se se ne trova una a distanza strettamente maggiore di quella memorizzata). Se non ci sono valori ripetuti, l'algoritmo segnala un errore.

La complessità di tale algoritmo è dominata dall'ordinamento, quindi $\Theta(n \log(n))$, in quanto la ricerca della coppia di elementi uguali a distanza massima, cioè la seconda parte dell'algoritmo, ha complessità lineare.

Lo pseudocodice per l'algoritmo descritto sopra è il seguente (dove `HEAPSORT-PAIRS` è una versione modificata di `HEAPSORT` che ordina le coppie come indicato sopra; ovviamente la complessità di `HEAPSORT-PAIRS` è la stessa di `HEAPSORT`).

```
FIND_PAIR(A)
1  crea array B di coppie <v1, v2>, di lunghezza A.length
2  for i := 1 to A.length
3      B[i].v1 := A[i]
4      B[i].v2 := i
5  HEAPSORT-PAIRS(B)
6  cur := B.length
7  maxpair.v1 := -1
8  maxpair.v2 := -1
9  maxdist := -1
10 for i := B.length-1 downto 1
11     if B[i].v1 != B[cur]
12         cur := i
13     elseif B[cur].v2 - B[i].v2 > maxdist
14         maxdist := cur - B[i].v2
15         maxpair.v1 := B[i].v2
16         maxpair.v2 := B[cur].v2
17 if maxdist = -1
18     error('invalid input')
19 else
20     return maxpair
```