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

8 febbraio 2022

Informatica teorica

Esercizio 1 (8 punti)
Si consideri il linguaggio sull'alfabeto $\Sigma = \{a, b, c, d\}$ composto da tutte e sole le stringhe in cui:
1. compare al più una sola c o una sola d, mai entrambe,
2. se compare una c, il numero di a a sinistra della c è uguale a quello di b a sinistra della c; idem a destra della c;
3. se compare una d, il numero di a a sinistra della d è uguale al doppio di quello delle b a sinistra della d; idem a destra della d;
4. Nel caso in cui non compaiano c o d, il numero di a e b è arbitrario.
   Si fornisca una grammatica a potenza minima che genera il linguaggio descritto.

SOLUZIONE

Una grammatica a potenza minima (libera dal contesto) che genera il linguaggio è la seguente:

$S \to A \mid B \mid C$
$A \to aA \mid bA \mid \varepsilon$
$B \to B'cB'$
$B' \to aB'bB' \mid bB'aB' \mid \varepsilon$
$C \to C'dC'$
$C' \to aC'aC'bC' \mid aC'bC'aC' \mid bC'aC'aC' \mid \varepsilon$

Esercizio 2 (8 punti) - multichance: non è richiesto il punto 2
Uno stato q di una macchina di Turing M viene detto *utile* se esiste una stringa w tale per cui q viene raggiunto durante l'esecuzione di M quando w si trova in ingresso, ovvero, detto q_0 lo stato iniziale di M, q è utile se $\exists w, x, y, \alpha_1, \dots \alpha_k, \beta_1, \dots, \beta_k \langle q_0, \uparrow w, \uparrow Z_0, \dots, \uparrow Z_0\rangle \vdash_M^* \langle q, x\uparrow y, \alpha_1\uparrow\beta_1, \dots, \alpha_k\uparrow\beta_k\rangle$, con $x \cdot y = w$.
1. È decidibile stabilire, data una macchina di Turing M e un suo stato q, se q sia utile?
2. È semidecidibile il problema di cui sopra?

SOLUZIONE

1. Il problema è indecidibile. Se fosse decidibile potremmo infatti decidere anche il problema della *emptiness* (cioè il problema di stabilire se il linguaggio accettato è vuoto) per una generica macchina M; la emptiness è notoriamente indecidibile (e se ne può ricavare immediatamente l'indecidibilità anche mediante il teorema di Rice). Infatti, basterebbe decidere se lo stato finale di M è utile (o almeno uno degli stati finali di M, nel caso ce ne sia più di uno) per stabilire se M accetta almeno una stringa o meno.
2. Il problema è semidecidibile, poiché, mediante una enumerazione diagonale (*dovetailing*) di tutte le possibili coppie $\langle w, i\rangle$ (dove w è la stringa in ingresso e i è il numero di passi di esecuzione

di M testati, ad esempio, mediante emulazione), ci si può accorgere se, prima o poi, lo stato q venga raggiunto.

Algoritmi e strutture dati

Esercizio 3 (8 punti) - multichance: non è richiesta la complessità spaziale
Si consideri il seguente linguaggio su alfabeto $\Sigma = \{a, b, c\}$:

$$L = \{w.w.c.x \mid w \in \{a, b\}^+, |w| \leq 49, x \in \Sigma^+\}.$$

Si descriva una MT a nastro singolo che lo accetti, valutandone le complessità spaziali e temporali.

Soluzione

Il linguaggio è regolare, essendo w di lunghezza limitata. La MT potrà usare l'organo di controllo per assicurarsi che i primi 98 caratteri (al più) della stringa siano compatibili con il linguaggio, dovrà poi controllare che vi sia una c seguita da un carattere qualunque per accettare. Quindi la macchina farà al più 100 passi, cioè $T(n) = \Theta(1)$; $S = \Theta(n)$, visto che comunque la memoria sarà occupata dalla stringa in ingresso.

Esercizio 4 (8 punti) - multichance: non è richiesto il punto 2
Si considerino le seguenti funzioni:

```
func1(n)
k := 0
i := 0
while i ≤ n
    k := k + 1
    i := i + k
    (*)
return k
```

```
func2(m)
if m ≤ 1
    return m
j := 1
while j ≤ m
    j := j*3
return func2(m/3) + func2(m/3)
```

Si calcoli la complessità di `func1(n)` quando al posto di `(*)` si trovano le seguenti istruzioni:
1. `func2(`10^5`);`
2. `func2(n)`.

Soluzione

Il valore di `i` all'inizio della j-esima iterazione del ciclo in `func1` non è mai influenzato dall'esecuzione della funzione `func`. Esso è pari a $1 + 2 + 3 + 4 + 5 + ... + (j - 1)$, e quindi è dell'ordine di j^2. Quindi, il ciclo viene eseguito un numero di volte pari a $\sqrt{n}$.

Calcoliamo innanzi tutto la complessità di `func2(m)`. Essa è data dalla seguente ricorsione: $T(m) = 2T(\frac{m}{3}) + \log_3(m)$. La ricorsione si risolve tramite il Master Theorem, in quanto $\log_3(m)$ è polinomialmente più piccola di $m^{log_3(2)}$. Siamo quindi nel caso 1 del Master Theorem, e la complessità di `func2(m)` è $\Theta(m^{log_3(2)})$.

Nel caso 1 l'istruzione `(*)` ha costo costante, indipendente da `n`. Quindi, in questo caso la compessità del codice è data dal numero di iterazioni del ciclo, ed è dell'ordine di $\sqrt{n}$.

Nel caso 2 a ogni iterazione del ciclo il costo dell'istruzione `(*)` è dell'ordine di $n^{log_3(2)}$, quindi il costo totale è dell'ordine di $n^{(\frac{1}{2}+log_3(2))}$.