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

10 gennaio 2023

Informatica teorica

Esercizio 1 (8 punti)
Si considerino i linguaggi seguenti:

- $L_1 = \{a^n b^n c^n \mid n > 0\}$
- $L_2 = \{b\}^*.\{a, b, c\}^+$

Per ciascuno dei linguaggi indicati di seguito, utilizzare un formalismo a potenza minima (tra tutti quelli visti a lezione) che lo caratterizzi:

1. $L = L_1 \setminus L_2$, dove il simbolo $\setminus$ indica la differenza insiemistica;
2. $L' = L_1 \cup L_2$.

Chi ha diritto alla riduzione del 30% svolga unicamente il punto 1.

Soluzione

Il linguaggio sottraendo (L_2) contiene tutte le stringhe di almeno un carattere e quindi contiene anche tutte le stringhe del linguaggio minuendo (L_1), pertanto L è il linguaggio vuoto. Per caratterizzarlo, è sufficiente una formula MFO insoddisfacibile, come la seguente φ_L:

$$\varphi_L : \exists x(a(x) \wedge \neg a(x)).$$

Per lo stesso motivo, abbiamo $L' = L_2$, che è chiaramente regolare, in quanto specificato dall'espressione regolare $(b)^*.(a|b|c)^+$. Tuttavia è possibile caratterizzare L' anche mediante una formula MFO, ad esempio con la formula $\varphi_{L'}$ seguente:

$$\varphi_{L'} : (a(0) \vee b(0) \vee c(0)) \wedge \forall x(a(x) \vee b(x) \vee c(x)).$$

Esercizio 2 (8 punti).
Si assuma che le macchine di Turing su alfabeto binario calcolino funzioni da $\mathbb{Z}$ a $\mathbb{Z}$, interpretando le stringhe sul nastro come numeri in complemento a 2 codificati con il numero minimo di bit necessari. Si considerino i seguenti insiemi e si dica se essi sono ricorsivi, motivando la risposta:

1. $S_1 = \{i \mid \exists x : f_i(x) > 0\}$
2. $S_2 = \{i \mid \exists x : f_i(x) \leq 0\}$
3. $S_3 = S_1 \cup S_2$
4. $S_4 = S_1 \cap S_2$

Soluzione

Sono tutti non ricorsivi per il teorema di Rice, come spiegato in dettaglio nel seguito.

1. S_1 contiene tutti e soli gli indici delle funzioni che hanno valore positivo per almeno un input. Questo insieme non è banale in quanto contiene, ad esempio, l'indice della funzione costante $f(x) = 1$, ma non quello della funzione costante $g(x) = 0$ (si noti che la funzione f non deve per forza essere totale).

2. S_2 contiene tutti e soli gli indici delle funzioni che hanno valore negativo o nullo per almeno un input. Questo insieme non è banale in quanto contiene, ad esempio, l'indice della funzione $g(x) = 0$, ma non quello della funzione $f(x) = 1$.

3. S_3 non è un insieme banale in quanto contiene almeno un indice (è l'unione di due insiemi non vuoti) e non contiene, ad esempio, l'indice della funzione ovunque non definita $h(x) = \perp$.

4. S_4 sono gli indici delle MT che calcolano funzioni che assumono valori sia positivi che negativi o nulli, ad es. la funzione segno. Dunque è non banale anch'esso.

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

10 gennaio 2023

Algoritmi e strutture dati

Esercizio 3 (8 punti)
Si consideri il seguente linguaggio su alfabeto $\{0, 1, c\}$:

$$L = \{xcy \mid x, y \in \{0,1\}^+ \text{ e } y = x \bmod 3, \text{ se } x \text{ e } y \text{ sono interpretate come numeri in base } 2\}.$$

Si assuma che x e y siano numeri naturali codificati (in binario) con la cifra più significativa a sinistra. Si descrivano una macchina di Turing a nastro singolo e una macchina RAM che riconoscano L e se ne valutino le complessità con tutti i criteri di costo applicabili.

Chi ha diritto alla riduzione del 30% descriva solamente la macchina di Turing a nastro singolo.

Soluzione

Il linguaggio L è regolare. Infatti, ci sono solo 3 resti possibili di una divisione per 3 e per capire a quale classe di modulo appartenga x è sufficiente una scansione lineare dei simboli di x senza bisogno di usare memoria aggiuntiva. Se una generica stringa binaria z ha un valore $val(z)$ tale per cui $val(z) \bmod 3 = p$, con $p \in \{0, 1, 2\}$, allora se alla stessa stringa si accoda il simbolo $a \in \{0, 1\}$, la sequenza za varrà $2val(z)+a$ e quindi avrà modulo $val(za) \bmod 3 = (2p+a) \bmod 3$. Per ogni valore di p e ogni simbolo a è quindi definita la nuova classe di modulo, che si può quindi determinare con un semplice automa a stati finiti. Appena viene incontrata la c, si può infine verificare che la classe di modulo così determinata corrisponda al suffisso y della stringa in ingresso.

Sia la MT sia la RAM faranno quindi una scansione lineare della stringa in ingresso ($T_{MT}(n) = T_{RAM}(n) = \Theta(n)$) senza utilizzare memoria aggiuntiva (quindi $S_{MT}(n) = \Theta(n)$ per la MT, visto che comunque la memoria sarà occupata dalla stringa in ingresso, e $S_{RAM}(n) = \Theta(1)$). Si noti che, in questo caso, non c'è differenza tra il costo della RAM calcolato a criterio di costo costante e quello calcolato a criterio di costo logaritmico.

Esercizio 4 (8 punti)
Definire un algoritmo maxPitagora (con la migliore complessità possibile) che prende in input un array A di elementi interi positivi (quindi strettamente maggiori di 0) e, se esiste in A almeno un valore tale che il suo quadrato è la somma dei quadrati di altri 2 valori anch'essi presenti in A, restituisce il massimo di tali valori; altrimenti (se nessun elemento di A è tale che il suo quadrato è la somma dei quadrati di altri 2 elementi di A) restituisce 0. Valutare la complessità dell'algoritmo ideato.

Soluzione

Un possibile algoritmo che risolve il problema desiderato è il seguente.

```
maxPitagora(A)
1  merge-sort(A)
2  allocate B of length A.length
3  for i := 1 to B.length
4      B[i] := A[i] * A[i]
5  for i := B.length downto 3
6      if exact-sum-no-sort(B[1 .. i − 1], B[i])
7          return A[i]
8  return 0
```

Si noti che l'algoritmo EXACT-SUM-NO-SORT è una variante dell'algoritmo EXACT-SUM visto a esercitazione, senza però l'ordinamento iniziale (in quanto l'array in input $B[1 \mathinner{..} i-1]$ è già ordinato). Inoltre, si noti che il ciclo delle righe 3-4 copia il contenuto di A in un altro array, B, elevando al contempo ogni valore al quadrato, per tenere traccia del valore iniziale di ogni elemento, prima dell'elevamento al quadrato (il problema chiede di restituire il valore dell'array originale, non il suo quadrato). Si poteva evitare di introdurre un nuovo array di supporto modificando ulteriormente l'algoritmo EXACT-SUM in modo che non solo non venga fatto l'ordinamento, ma il confronto venga fatto tra i quadrati dei valori presenti nell'array, invece che tra i numeri originali. Si ricorda che l'algoritmo di EXACT-SUM, nella parte di ricerca degli elementi, ha complessità $\mathcal{O}(n)$, quindi l'algoritmo maxPitagora ha complessità $\mathcal{O}(n^2)$ (la complessità dell'algoritmo è ovviamente dominata dal ciclo 5-7, in quanto MERGE-SORT ha complessità $\Theta(n \log(n))$, e il ciclo 3-4 ha complessità $\Theta(n)$).

Per completezza, riportiamo anche la versione che non richiede l'allocazione di un nuovo array (maxPitagora-no-copy) e l'algoritmo di EXACT-SUM senza ordinamento e che in più esegue il confronto tra i quadrati dei valori (EXACT-SUM-NO-SORT-SQUARES):

```
maxPitagora-no-copy(A)
merge-sort(A)
for i := A.length downto 3
    if exact-sum-no-sort-squares(A[1 .. i − 1], A[i])
        return A[i]
return 0
```

```
Exact-sum-no-sort-squares(A, x)
i := 1
while i ≤ A.length and (A[i])^2 ≤ x^2
    i := i + 1
i := i − 1
if i < 1
    return false
j := 1
while j < i
    if (A[i])^2 + (A[j])^2 = x^2
        return true
    elseif (A[i])^2 + (A[j])^2 < x^2
        j := j + 1
    else i := i − 1
return false
```