Esercitazione 4 - Linguaggi e Complessità

03-05-2019

Antonio Cruciani
antonio.cruciani@alumni.uniroma2.eu

Funzioni Time e Space constructible

Definizione:
Una funzione $f : \mathbb{N} \to \mathbb{N}$ è **time-constructible** se esiste una macchina di Turing T di tipo trasduttore che, dato in input un intero n in unario (1^n) scrive sul nastro di output $f(n)$ in unario ($1^{f(n)}$) in $dtime(T, n) \in \boldsymbol{O}(f(n))$.

Esercizi a lezione

Esercizio 1:
Dimostrare $f(n) = n^n$ è una funzione time-constructible
Esercizio 2:
Dimostrare che per ogni $k \in \mathbb{N}$ **costante**, 2^{n^k} è una funzione time-constructible.

Esercizi per casa

Esercizio 1:
Per ognuna delle seguenti affermazioni si dimostri, si confuti o si mostri che sono dei problemi aperti:

1) Se $L_1, L_2 \in \mathbf{coNP} \;\Rightarrow\; L_1 \cap L_2 \in \mathbf{coNP}$
2) Se $L \in \mathbf{NP},\; L_1 \subsetneq L, e\; L_1 \in \mathbf{coNP} \;\Rightarrow\; L - L_1 \in NP$
3) Se $L \in \mathbf{NPC} \;\Rightarrow\; \{xx \;: x \in L\} \in \mathbf{NPC}$
4 Se $L_1, L_2 \in \mathbf{NP} \cap \mathbf{coNP} \;\Rightarrow\; L_1 \oplus L_2 \in \mathbf{NP} \cap \mathbf{coNP}$. Dove $L_1 \oplus L_2 = \{x : x \textit{ è esattamente in uno dei due}\}$

Esercizio 2:
Dimostrare che per ogni $k \in \mathbb{N}$ **costante** e per ogni (k+1)-pla $\langle a_0, a_2, \ldots a_k \rangle$ di **costanti** tali che $\forall 0 \leq i \leq k [a_i \in \mathbb{N}]$, $f(n) = \sum_{i=0}^{k} a_i\, n_i$ è una funzione time-constructible.

Esercizio 3:
Sia $f(n)$ un funzione time-constructible. Si dimostri che $2^{f(n)}$ è anch'essa time-constructibile.

Soluzioni esercizi a lezione

Esercizio 1:
Claim: $f(n) = n^n$ è una funzione time-constructible.
Proof:
Forniamo una macchina di Turing che calcola $f(n)$, per comodità utilizzeremo il linguaggio *Pascal Minimo*.
Legenda:

- i è la variabile che indica il numero della fase
- h è la variabile che indica la posizione della testina sul nastro n_1
- n_j indica il nastro j
- con l'operazione $n_i \leftarrow n_j$ copia n_j su n_i
- l'operatore + tra n_i e n_j indica la concatenazione tra n_i e n_j

Forniamo, ora, lo pseudocodice dell'algoritmo.

Algorithm 1 Calcola n^n

$n_1 \leftarrow n$
$n_3 \leftarrow n_1$
$n_4 \leftarrow n_1$
$i \leftarrow 2$
while $(i \leq n_4)$ **do Begin**
    $n_2 \leftarrow n_3$
    $h \leftarrow 1$
    while $(h \leq n_4)$ **do Begin**
        $n_3 \leftarrow n_3 + n_2$
        $h \leftarrow h + 1$
    $i \leftarrow i + 1$
Output(n_3)

Mostriamo che tale macchina di Turing opera in tempo (n^n).
Calcoliamo il numero di passi di T.
Le istruzioni prima del ciclo **while** richiedono n passi (scrittura simultanea sui nastri) e ne occorrono n per riavvolgere il nastro n_4 (per eseguire l'istruzione $i \leftarrow 2$).

L'i-esima iterazione del ciclo **while** esterno richiede:

- controllo condizione del **while** in 1 passo
- $n_2 \leftarrow n_3$ richiede n^{i-1} passi
- riavvolgimento n_2 in altrettanti passi
- riavvolgimento di n_1 $(h \leftarrow 1)$ in n passi

Il ciclo **while** intero richiede (effettua per n volte):

- controllo della condizione in 1 passo
- $n_3 \leftarrow n_3 + n_2$ viene concatenata in n^{i-1} passi il valore di $n_3(n^{i-1})$ contenuto in n_2
- riavvolgimento di n_2 in n^{i-1} passi

In definitiva l'algoritmo impiega:

$$
\begin{aligned}
&2n + \sum_{i=2}^{n} [2n^{i-1} + n + n(1 + n^{i-1} + n^{i-1}) + 1] = \\
&2n + 2n(n-1) + n - 1 + 2\sum_{i=2}^{n} n^i + 2\sum_{i=2}^{n} n^{i-1} \\
&\leq 2n^2 + n + 4\sum_{i=2}^{n} n^i
\end{aligned}
$$

utilizziamo il risultato noto delle serie geometriche:

$$
\sum_{k=m}^{n} x^k = \frac{x^m - x^{n+1}}{1-x}
$$

Ottenendo:

$$
\begin{aligned}
&2n^2 + n + 4\sum_{i=2}^{n} n^i = 2n^2 + n + 4\frac{n^2 - n^{n+1}}{1-n} \\
&\leq 2n^2 + n + 4\ \frac{n^{n+1}}{\frac{n}{2}} = \\
&= 2n^2 + n + 8n^n \sim \boldsymbol{O}(n^n)
\end{aligned}
$$

E questo conclude la nostra analisi. Abbiamo dimostrato che $f(n) = n^n$ è una funzione time-constructible.

Esercizio 2:

Claim: $\forall k \in \mathbb{N}$ costante , $f(n) = 2^{n^k}$ è una funzione time-constructible.

Proof:

Forniamo una macchina di Turing che calcola $f(n)$, per comodità utilizzeremo il linguaggio *Pascal Minimo.*

Legenda:

- n_i è il nastro i-esimo, semi-infinito dove le celle di n_i sono numerate a partire dalla posizione 1
- La variabile i_j corrisponde alla posizione della testina sul nastro n_j.
- L'operatore + applicato sui nastri $(n_j + n_h)$ indica l'operazione di concatenazione.
- L'operatore + applicato alle testine $(i_j + 1)$ indica lo spostamento a destra della testina.
- La subroutine CALCOLA POTENZA K-ESIMA(n_i, n_j) scrive su n_j il valore corrispondente al valore di n_i elevato alla k.

Forniamo, ora, lo pseudocodice dell'algoritmo.

Algorithm 2 Calcola 2^{n^k}

$n_1 \leftarrow n$
CALCOLA POTENZA K-ESIMA (n_1, n_2)
$n_3 \leftarrow 2$ (in unario, ovvero 1^2)
$i_2 \leftarrow 2$
while $(i_2 \leq n_2)$ **do Begin**
    $n_4 \leftarrow n_3$
    $n_3 \leftarrow n_3 + n_4$
    $i_2 \leftarrow i_2 + 1$
Output(n_3)

Mostriamo che tale macchina di Turing opera in tempo $\boldsymbol{O}(2^{n^k})$.
La subroutine CALCOLA POTENZA K-ESIMA (n_1, n_2) richiede tempo $\boldsymbol{O}(n^k)$.
Procediamo con l'analisi del ciclo **while**: all'inizio della generica iterazione i il nastro n_3 contiene il valore 2^{i-1} il quale viene copiato su n_4 e poi concatenato al valore stesso di n_3 ottenendo così il valore 2^i. L'operazione i-esima richiede

quindi tempo $2^{i-1} + 2^i$.
Osserviamo esplicitamente che $2 \leq i \leq n^k$.

$$\Rightarrow \sum_{i=2}^{n^k} (2^{i-1} + 2^i) = \sum_{i=2}^{n^k} 2^{i-1} + \sum_{i=2}^{n^k} 2^i = \frac{1}{2} \sum_{i=2}^{n^k} 2^i + \sum_{i=2}^{n^k} 2^i = \sum_{i=2}^{n^k} 2^i (\frac{1}{2} + 1) = \sum_{i=2}^{n^k} 2^i (\frac{3}{2})$$

utilizziamo il risultato noto delle serie geometriche:

$$\sum_{k=m}^{n} x^k = \frac{x^m - x^{n+1}}{1 - x}$$

ottenendo:

$$\sum_{i=2}^{n^k} 2^i (\frac{3}{2}) = \frac{3}{2}(-4 + 2^{n^k+1}) = 3(2^{n^k} - 2) \sim \boldsymbol{O}(2^{n^k})$$

E questo conclude la nostra analisi. Abbiamo dimostrato che $f(n) = 2^{n^k}$ per ogni $k \in \mathbb{N}$ costante è una funzione time-constructible.

Soluzioni esercizi per casa

Esercizio 1:
1)
Vero, poiché $L_1^c, L_2^c \in \mathbf{NP} \Rightarrow L_1^c \cup L_2^c \in \mathbf{NP} \Rightarrow L_1 \cap L_2 = (L_1^c \cup L_2^c)^c \in \mathbf{coNP}$

2)
Vero, poiché $L_1 \in \mathbf{coNP} \Rightarrow L_1^c \in \mathbf{NP}$ e poiché $\mathbf{NP}$ è chiusa per l'operazione di intersezione, $L - L_1 = L \cap L_1^c \in \mathbf{NP}$

3)
Vero, poiché $\{xx : x \in L\}$ è chiaramente in $\mathbf{NP}$. Per prima cosa controlla che l'input sia della forma xx poi esegui la NDTM che accetta L su x. D'altra parte, possiamo esibire una riduzione da L, duplicando semplicemente l'input, quindi $\{xx : x \in L\} \in \mathbf{NPC}$.

4)
Osserviamo che l'operatore $\oplus$ può essere definito come segue:
$A \oplus B = (A \cap B^c) \cup (A^c \cap B)$.
Osserviamo esplicitamente che:
$L_1^c, L_2^c \in \mathbf{NP} \cup \mathbf{coNP} \Rightarrow L_1 \oplus L_2 = (L_1 \cap L_2^c) \cup (L_1^c \cap L_2)$ osserviamo che $L_1 \cap L_2^c, L_1^c \cap L_2 \in \mathbf{NP} \cap \mathbf{coNP} \Rightarrow L_1 \oplus L_2 \in \mathbf{NP} \cap \mathbf{coNP}$

Esercizio 2:
Claim: $\forall k \in \mathbb{N} \wedge \forall \langle a_0, a_2, \dots a_k \rangle$ tali che $\forall 0 \le i \le k[a_i \in \mathbb{N}]$, costanti , $f(n) = \sum_{i=0}^{k} a_i \; n_i$ è una funzione time-constructible.
Proof:
Forniamo una macchina di Turing che calcola $f(n)$, per comodità utilizzeremo il linguaggio *Pascal Minimo*.
Osserviamo esplicitamente che i valori $k, a_0, \dots a_k$ sono costanti, quindi possiamo codificarli negli stati della macchina di Turing che deve calcolare la funzione senza utilizzare i nastri.
Legenda:

- Con istruzioni del tipo $n_2 \leftarrow a_0$ ci riferiamo alla seguente sequenza di istruzioni della macchina di Turing:
    1. se nello stato $q_{a_0,0}$ la testina su n_2 legge un $\square$ allora scrive 1, si sposta a destra di una posizione ed entra nello stato $q_{a_0,1}$
    2. se nello stato $q_{a_0,1}$ la testina su n_2 legge un $\square$ allora scrive 1, si sposta a destra di una posizione ed entra nello stato $q_{a_0,2}$

3. ...
4. se nello stato q_{a_0,a_0-1} la testina su n_2 legge un $\square$ allora scrive 1, si sposta a destra di una posizione ed entra nello stato q_{a_0,a_0}

- n_i è il nastro i-esimo, semi-infinito dove le celle di n_i sono numerate a partire dalla posizione 1
- La variabile i corrisponde all'insieme di stati necessari a calcolare il monomio $a_i n^i$. L'istruzione $i \leftarrow i+1$ indica il passaggio del calcolo del monomio $a_{i+1}n^{i+1}$ (ovvero la transizione degli stati).
- La variabile j serve per enumerare gli stati $q_{a_i,0} \dots q_{a_i,a_i}$, l'istruzione $j \leftarrow j+1$ indica il passaggio della macchina dallo stato $q_{a_i,j}$ allo stato $q_{a_i,j+1}$
- L'operatore + applicato sui nastri $(n_j + n_h)$ indica l'operazione di concatenazione.
- La subroutine Calcola Potenza i-esima(n_h, n_j, i) scrive su n_j il valore corrispondente al valore di n_h elevato alla i.

Forniamo, ora, lo pseudocodice dell'algoritmo.

Algorithm 3 Calcola $f(n) = \sum_{i=0}^{k} a_i\, n_i$

$n_1 \leftarrow n$
$n_2 \leftarrow a_0$
$i \leftarrow 1$
while $(i \leq k)$ **do Begin**
    Calcola Potenza i-esima (n_2, n_3, i)
    $j \leftarrow 1$
    while $(j \leq a_i)$ **do Begin**
        $n_2 \leftarrow n_2 + n_3$
        $j \leftarrow j + 1$
    $i \leftarrow i + 1$
Output(n_2)

Mostriamo che tale macchina di Turing opera in tempo $\boldsymbol{O}(n^k)$.
Una generica invocazione i della subroutine Calcola Potenza i-esima (n_2, n_3, i) richiede tempo $t_i n^i$ per un opportuno t_i. Procediamo con l'analisi del ciclo **while** interno: in questo ciclo viene effettuata la concatenazione

del nastro n_2 ed n_3, questa operazione richiede tempo $a_i n^i$. Il ciclo **while** esterno richiede quindi:

$$\sum_{i=1}^{k} (t_i + a_i) n^i = \overbrace{\sum_{i=1}^{k} (t_i + a_i)}^{c} \underbrace{\sum_{i=1}^{k} n^i}_{n^k} \sim \boldsymbol{O}(n^k)$$

E questo conclude la nostra analisi. Abbiamo dimostrato che $f(n) = \sum_{i=0}^{k} a_i \; n_i$, $\forall k \in \mathbb{N} \wedge \forall \langle a_0, a_2, \ldots a_k \rangle$ tali che $\forall 0 \leq i \leq k [a_i \in \mathbb{N}]$, costanti è una funzione time-constructible.

Esercizio 3:
Claim: $2^{f(n)}$ è una funzione time-constructible.
Proof:
Forniamo una macchina di Turing che calcola $2^{f(n)}$, per comodità utilizzeremo il linguaggio *Pascal Minimo*.
Legenda:

- n_i è il nastro i-esimo, semi-infinito dove le celle di n_i sono numerate a partire dalla posizione 1
- La variabile i corrisponde alla posizione della testina sul nastro n_2.
- L'operatore + applicato sui nastri $(n_j + n_h)$ indica l'operazione di concatenazione.
- L'operatore + applicato alle testine $(i + 1)$ indica lo spostamento a destra della testina (in questo caso sul nastro n_2).

Segue lo pseudocodice:

Algorithm 4 Calcola $2^{f(n)}$

1: $n_1 \leftarrow n$
2: $n_2 \leftarrow f(n_1)$
3: $n_4 \leftarrow 1$
4: $i \leftarrow 1$
5: **while** $(i \leq n_2)$ **do Begin**
6: $\quad n_3 \leftarrow n_4$
7: $\quad n_4 \leftarrow n_4 + n_3$
8: $\quad i \leftarrow i + 1$
9: **Output**(n_4)

Mostriamo che tale macchina di Turing opera in tempo $\boldsymbol{O}(2^{f(n)})$.
Al passo 2 il calcolo di $f(n)$ richiede tempo $\boldsymbol{O}(f(n))$, poiché $f(n)$ è una funzione time-constructible. L'istruzione 3 richiede un numero costante di passi e l'istruzione 4 posiziona la testina sul primo carattere a sinistra di n_2 in $\boldsymbol{O}(f(n))$ passi. Il ciclo **while** viene ripetuto esattamente $f(n)$ volte e alla generica iterazione i il contenuto di n_4 viene copiato su n_3, in n_4 passi, e poi il contenuto di n_3 viene concatenato al contenuto di n_4 in n_3 passi riposizionando la testina su n_4 sul suo primo carattere in $2n_3$ passi, spostando, infine, la testina su n_2 di una posizione. Osserviamo che nella

i-esima iterazione viene calcolato il valore 2^i.
Analizziamo la complessità computazionale dell'algoritmo:

$$\boldsymbol{O}(f(n)) + \sum_{i=1}^{f(n)} (2^2 \cdot 2^{i-1} + 1) = \boldsymbol{O}(f(n)) + f(n) + 2\sum_{i=1}^{f(n)} 2^i = \boldsymbol{O}(f(n)) + 2\left(\frac{2 - 2^{f(n)+1}}{1-2}\right)$$
$$= \boldsymbol{O}(f(n)) + 2(2^{f(n)+1} - 2) = \boldsymbol{O}(f(n)) + 4 \cdot 2^{f(n)} - 4 \sim \boldsymbol{O}(2^{f(n)})$$

E quindi $2^{f(n)}$ è una funzione time-constructible.