Informatica Teorica 2022/2023 - Allenamento

Problema 1. Sia $t(n)$ una funzione con $t(n) \geq$. Dimostra che per ogni multi-tape TM che esegue in tempo $t(n)$ esiste una single-tape TM equivalente che esegue in tempo $O(t^2(n))$.

Idea Intuitiva. É possibile "convertire" ogni multi-tape TM in una single-tape TM che la simula (cf. Lezione 3, sl. 16ss.). Bisogna dunque analizzare la simulazione per capire quanto tempo essa richieda. Si puó mostrare che simulare ciascun passo della multi-tape TM richiede al piú $O(t(n))$ passi sulla corrispondente single-tape TM. Dunque, il tempo totale impiegato sará di $O(t^2(n))$ passi.

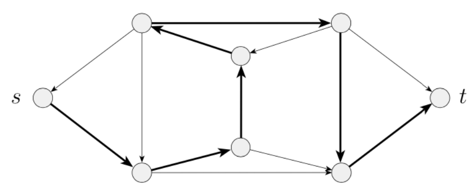
Dimostrazione. Sia M una k-tape TM che esegue in tempo $t(n)$. Mostriamo che possiamo costruire una single-tape TM M' che esegue in tempo $O(t^2(n))$.
La macchina M' opera simulando M. In particolare, M' impiega il suo nastro singolo per rappresentare il contenuto dei nastri di M. I nastri sono registrati consecutivamente, con le posizioni delle testina di M segnate sul riquadro appropriato.
Simulazione di multi-tape TM tramite single-tape TM. Inizialmente, il nastro di M' é posto in modo da rappresenta tutti i nastri di M. Dunque M' simula i passi di M. In particolare, per simulare uno step di M, M' scansiona tutte le informazioni immagazzinate sul nastro per determinare i simboli sotto le testine di M. In seguito, M' scansiona nuovamente il nastro per aggiornarne i contenuti e le posizioni della testina. Se una delle testine di M si muove a destra su una porzione del nastro non letta precedentemente, M' deve incrementare la quantitá di spazio allocato sul suo nastro. Questo é effettuato muovendo una porzione del suo nastro di una cella a destra.
Analisi della Simulazione. Si analizza dunque il tempo richiesto da tale simulazione. Per ogni passo della multi-tape M, la single-tape TM M' effettua due azioni sulla porzione attiva del suo nastro. La prima permette di ottenere le informazioni necessarie per determinare il passo successivo e la seconda compie effettivamente l'azione indicata. Il tempo richiesto per la scansione dipende dalla lunghezza della porzione attiva del nastro di M', quindi é necessario determinare un *upper bound* di questa lunghezza: consideriamo la somma delle lunghezza della porzione attiva dei k nastri di M. Ciascuna di queste porzioni attive ha lunghezza massima $t(n)$. Quindi, una scansione della porzione attiva del nastro di M' richiede $O(t(n))$ passi. Inoltre, per simulare ciascun passo di M, M' effettua due scansioni e fino a k spostamenti a destra. Ciascuno impiega tempo $O(t(n))$. Dunque, il tempo totale richiesto a M' per simulare un passo di M é $O(t(n))$.
Determiniamo ora un *bound* rispetto al tempo totale impiegato dalla simulazione. Lo stato iniziale, nel quale M' mette il nastro nella configurazione richiesta impiega $O(n)$ passi. Dopodiché, M' simula ciascuno dei $t(n)$ passi di M impiegando $O(t(n))$ passi. Questa parte della simulazione richiede $t(n) \times O(t(n)) = O(t^2(n))$ passi. Quindi, l'intera simulazione di M richiede $O(n) + O(t^2(n))$ passi. Abbiamo assunto che $t(n) \geq n$ (assunzione ragionevole in quanto M non puó neppure leggere l'intero input in tempo inferiore). Concludiamo che il tempo di esecuzione di M' é $O(t^2(n))$. □

Problema 2. Sia $t(n)$ una funzione con $t(n) \geq n$. Dimostra che, per ogni single-tape NTM che esegue in tempo $t(n)$ esiste una single-tape TM equivalente che esegue in tempo $2^{O(t(n))}$.

Dimostrazione. Sia N una NTM che esegue in tempo $t(n)$. Possiamo costruire una TM (deterministica) M che simula N ispezionando l'albero di computazione non deterministico di N (Cf. lezione 3, sl. 24ff.). Nuovamente dobbiamo analizzare questa simulazione.
Su input di lunghezza n ciascun ramo dell'albero di computazione (nondeterministica) di N ha lunghezza al piú $t(n)$. Ciascun nodo dell'albero puó avere al massimo m figli, dove m é il numero massimo di scelte accettabili date dalla funzione di transizione di N. Dunque, il numero totale di foglie nell'albero di computazione é al piú $m^{t(n)}$. La simulazione procede esplorando l'albero in larghezza, ovvero visitando ciascun nodo a profonditá l prima di passare ad altro noto a profonditá $l+1$. Il numero totale di nodi nell'albero é minore del doppio del numero massimo di foglie, dunque abbiamo *bound* $(O(m^{t(n)})$. Il tempo richiesto per iniziare dalla radice e propagarsi a un nodo é $O(t(n))$. Dunque il tempo di esecuzione di M é $O(t(n)m^{t(n)}) = 2^{O(t(n))}$.
La TM N usata per la data simulazione ha tre nastri (Cf. Lezione 3). Dunque il tempo di esecuzione della simulazione su single-tape TM M é $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$. □

Problema 4. Dato un grafo diretto G, un *Hamiltonian path* é un percorso diretto che attraversa ciascun nodo esattamene una volta. Consideriamo il problema di controllare se un grafo diretto contenga un *Hamiltonian path* che colleghi due nodi specificati. Sia

$$HP = \{\langle G, s, t\rangle \mid G\ grafo\ diretto\ con\ Hamiltonian\ path\ da\ s\ a\ t\}$$



Costruisci una (poly-time) NTM che decide HP.

Dimostrazione. Consideriamo una NTM che decide il problema HP in tempo polinomiale. (Ricorda che il tempo di esecuzione di una NTM é definito dal ramo piú lungo dell'albero di computazione.) Sia N definita come segue:

Su input $\langle G, s, t\rangle$, dove G é un grafo diretto con nodi s, t:

1. Scrive una lista di m numeri, $p_1, \ldots, p_m$, dove m é il numero dei nodi di G. Ciascun numero nella lista é selezionato *nondeterministicamente* tra 1 e m.
2. Controlla le ripetizioni nella lista. Se ne trova, rigetta.
3. Controlla se $s = p_1$ e $t = p_m$. Se una delle due fallisce, rigetta.
4. Per ogni i tra 1 e $m - 1$, controlla se (p_i, p_{i+1}) é un arco di G. Se qualcuno non lo é, rigetta. Se tutti i test risultano passati, accetta.

Per analizzare questo algoritmo e verificare che sia poly-time, consideriamo ciascuna sua fase. In 1., la selezione nondeterministica chiaramente esegue in poly-time. In 2. e 3., ciascuna parte é semplicemente un controllo dunque (insieme) eseguono in poly-time. Anche 4. chiaramente esegue in poly-time. Dunque, l'algoritmo esegue in (nondeterministic) poly-time. □

Problema 7. Mostra che SSUM é **NP**-completo.
Suggerimento. Considera che sappiamo che 3SAT é **NP**-completo e abbiamo giá dimostrato SSUM $\in$ **NP** (Problema 2, Esercitazione 3).

Dimostrazione. Abbiamo giá dimostrato SSUM $\in$ **NP** (Problema 2, Esercitazione 3). Vogliamo dimostrare che ciascun linguaggio in **NP** é poly-time riducibile a SSUM. In particolare, dimostriamo che 3SAT $\leq_P$ SSUM (sapendo 3SAT essere **NP**-completo).
Sia F una formula Booleana con variabili $x_1, \ldots, x_l$ e clausole $c_1, \ldots, c_l$. La riduzione converte F in un'istanza del problema SSUM $\langle S, t \rangle$, in cui gli elementi di S e il numero t sono righe della seguente tabella, espresse in notazione decimale.

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

Le righe sopra alla doppia linea sono etichettate $y_1, z_1, y_2, z_2, \ldots, y_l, z_l$ e $g_1, h_1, g_2, h_2, \ldots, g_k, h_k$ e costituiscono gli elementi di S. La riga sottostante la doppia linea é t. Dunque, S contiene una coppia di numeri y_i, z_i per ciascuna variabile x_i in F. La rappresentazione decimale di questi numeri é in due parti, come indicato dalla tabella. La parte sinistra comprende un 1 seguito da $l - i$ 0i. La parte destra contiene una cifra per ciascuna clausola, dove la cifra di y_i in colonna c_j é 1 se la clausola c_j contiene il letterale x_i e la cifra di z_i in colonna c_j é 1 se la clausola c_j contiene il letterale $\overline{x_i}$. Le cifre per cui non é specificato siano 1, sono 0. La tabella é parzialmente riempita per illustrare un'esempio di clasuole c_1, c_2 e c_k:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3 \vee \ldots) \wedge \cdots \wedge (\overline{x_3} \vee \cdots \vee \ldots).$$

Inoltre, S contiene una coppia di numeri g_j, h_k per ciascuna clausola c_j. Tali due numeri sono equivalenti e consistono di un 1 seguito da $k - j$ 0i. Infine, il numero target t – l'ultima linea della tabella – consta di l 1i seguiti da k 3.
Mostriamo ora che questa costruzione "funziona". Mostriamo in particolare che F é soddisfacibile se e solo se qualche sottoinsieme di S somma a t. Assummiamo F sia soddisfacibile. Costruiamo un sottoinsieme di S come segue. Selezioniamo y_i se a x_i é assegnato valore $\top$ (true) e z_i se a x_i é assegnato $\perp$ (false). Se aggiungiamo ció che abbiamo selezionato fino ad ora, otteniamo 1 per ciascuna delle prime l cifre, poiché abbiamo selezionato y_i o z_i per ogni i. Inoltre, ciascuna delle ultime k cifre é un numero tra 1 e 3 poiché ciascuna clausola é soddisfatta e quindi contiene tra

1 e 3 letterali veri. Inoltre, selezioniamo sufficienti g e h numeri per portare ciascuna delle ultime k cifre a 3, raggiungendo cosí il target. Supponiamo che un sottoinsieme di S sommi a t. Date le seguenti osservazioni, costruiamo un assegnamento che soddisfa F. Tutte le cifre nei membri di S sono 0 o 1. Inoltre, ciascuna colonna nella tabella che descrive S contiene al piú cinque 1i. Infine, lo spostamento nella colonna successiva non occorre quando un sotto-insieme di S é aggiunto. Per ottenere un 1 in ciascuna delle prime l colonne, il sottoinsieme deve avere y_i o z_i, per ciascun i, ma non per entrambi. Ora definiamo l'assegnamento. Se un sottoinsieme contiene y_i, assegnamo a x_i $\top$ (true); altrimenti, assegnamo a x_i $\perp$ (false). Questo assegnamento deve soddisfare F poiché in ciascuna delle k colonne finali, la somma é sempre 3. Nella colonna c_j, al piú 2 puó venire da g_j e h_j, quindi almeno 1 in questa colonna deve venire da qualche y_i o z_i nel sotto-insieme: se é y_i, allora x_i appare in c_j ed é assegnato valore $\top$ (true), dunque c_j é soddisfatto; se é z_i, allora $\overline{x_i}$ appare in c_j e a x_i é assegnato $\perp$ (false), dunque c_j é soddisfatto. Allora, ϕ é soddisfatto. Infine, basta assicurarsi che la riduzione possa essere effettuata in tempo polinomiale e tale é. $\square$