

1. (8 punti) Considera il linguaggio

$$L = \{0^m 1^n \mid m/n \text{ è un numero intero}\}.$$

Dimostra che L non è regolare.

Usiamo il Pumping Lemma per dimostrare che il linguaggio non è regolare.

Supponiamo per assurdo che L sia regolare:

- sia k la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 0^{k+1}1^{k+1}$, che è di lunghezza maggiore di k ed appartiene ad L perché $(k+1)/(k+1) = 1$;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq k$;
- poiché $|xy| \leq k$, allora x e y sono entrambe contenute nella sequenza di 0. Inoltre, siccome $y \neq \varepsilon$, abbiamo che $x = 0^q$ e $y = 0^p$ per qualche $q \geq 0$ e $p > 0$. z contiene la parte rimanente della stringa: $z = 0^{k+1-q-p}1^{k+1}$. Consideriamo l'esponente $i = 0$: la parola xy^0z ha la forma

$$xy^0z = xz = 0^q 0^{k+1-q-p} 1^{k+1} = 0^{k+1-p} 1^{k+1}.$$

Si può notare che $(k+1-p)/(k+1)$ è un numero strettamente compreso tra 0 e 1, e quindi non può essere un numero intero. Di conseguenza, la parola non appartiene al linguaggio L , in contraddizione con l'enunciato del Pumping Lemma.

2. (8 punti) Per ogni linguaggio L , sia $\text{prefix}(L) = \{u \mid uv \in L \text{ per qualche stringa } v\}$. Dimostra che se L è un linguaggio context-free, allora anche $\text{prefix}(L)$ è un linguaggio context-free.

Se L è un linguaggio context-free, allora esiste una grammatica G in forma normale di Chomski che lo genera. Possiamo costruire una grammatica G' che genera il linguaggio $\text{prefix}(L)$ in questo modo:

- per ogni variabile V di G , G' contiene sia la variabile V che una nuova variabile V' . La variabile V' viene usata per generare i prefissi delle parole che sono generate da V ;
- tutte le regole di G sono anche regole di G' ;
- per ogni variabile V di G , le regole $V' \rightarrow V$ e $V' \rightarrow \varepsilon$ appartengono a G' ;
- per ogni regola $V \rightarrow AB$ di G , le regole $V' \rightarrow AB'$ e $V' \rightarrow A'$ appartengono a G' ;
- se S è la variabile iniziale di G , allora S' è la variabile iniziale di G' .

3. (8 punti) Una Turing machine con alfabeto binario è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è $\Sigma = \{0, 1\}$ e l'alfabeto del nastro è $\Gamma = \{0, 1, _ \}$. Questo significa che la macchina può scrivere sul nastro solo i simboli 0, 1 e blank: non può usare altri simboli né marcare i simboli sul nastro.

Dimostra che le Turing machine con alfabeto binario riconoscono tutti e soli i linguaggi Turing-riconoscibili sull'alfabeto $\{0, 1\}$.

Per risolvere l'esercizio dobbiamo dimostrare che (a) ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è Turing-riconoscibile e (b) ogni linguaggio Turing-riconoscibile sull'alfabeto $\{0, 1\}$ è riconosciuto da una Turing machine con alfabeto binario.

- (a) Questo caso è semplice: una Turing machine con alfabeto binario è un caso speciale di Turing machine deterministica a nastro singolo. Quindi ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è anche Turing-riconoscibile.
- (b) Per dimostrare questo caso, consideriamo un linguaggio L Turing-riconoscibile, e sia M una Turing machine deterministica a nastro singolo che lo riconosce. Questa TM potrebbe avere un alfabeto del nastro Γ che contiene altri simboli oltre a 0, 1 e blank. Per esempio potrebbe contenere simboli marcati o separatori.

Per costruire una TM con alfabeto binario B che simula il comportamento di M dobbiamo come prima cosa stabilire una codifica binaria dei simboli nell'alfabeto del nastro Γ di M . Questa codifica è una funzione C che assegna ad ogni simbolo $a \in \Gamma$ una sequenza di k cifre binarie, dove k è un valore scelto in modo tale che ad ogni simbolo corrisponda una codifica diversa. Per esempio, se Γ contiene 4 simboli, allora $k = 2$, perché con 2 bit si rappresentano 4 valori diversi. Se Γ contiene 8 simboli, allora $k = 3$, e così via.

La TM con alfabeto binario B che simula M è definita in questo modo:

$B =$ "su input w :

1. Sostituisce $w = w_1w_2 \dots w_n$ con la codifica binaria $C(w_1)C(w_2) \dots C(w_n)$, e riporta la testina sul primo simbolo di $C(w_1)$.
 2. Scorre il nastro verso destra per leggere k cifre binarie: in questo modo la macchina stabilisce qual è il simbolo a presente sul nastro di M . Va a sinistra di k celle.
 3. Aggiorna il nastro in accordo con la funzione di transizione di M :
 - Se $\delta(r, a) = (s, b, R)$, scrive la codifica binaria di b sul nastro.
 - Se $\delta(r, a) = (s, b, L)$, scrive la codifica binaria di b sul nastro e sposta la testina a sinistra di $2k$ celle.
 4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M , allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."
4. (8 punti) Supponiamo che un impianto industriale costituito da m linee di produzione identiche debba eseguire n lavori distinti. Ognuno dei lavori può essere svolto da una qualsiasi delle linee di produzione, e richiede un certo tempo per essere completato. Il problema del bilanciamento del carico (LOADBALANCE) chiede di trovare un assegnamento dei lavori alle linee di produzione che permetta di completare tutti i lavori entro un tempo limite k .

Più precisamente, possiamo rappresentare l'input del problema con una tripla $\langle m, T, k \rangle$ dove:

- m è il numero di linee di produzione;
- $T[1 \dots n]$ è un array di numeri interi positivi dove $T[j]$ è il tempo di esecuzione del lavoro j ;
- k è un limite superiore al tempo di completamento di tutti i lavori.

Per risolvere il problema vi si chiede di trovare un array $A[1 \dots n]$ con gli assegnamenti, dove $A[j] = i$ significa che il lavoro j è assegnato alla linea di produzione i . Il tempo di completamento (o makespan) di A è il tempo massimo di occupazione di una qualsiasi linea di produzione:

$$\text{makespan}(A) = \max_{1 \leq i \leq m} \sum_{A[j]=i} T[j]$$

LOAD BALANCE è il problema di trovare un assegnamento con makespan minore o uguale al limite superiore k :

$$\text{LOADBALANCE} = \{ \langle m, T, k \rangle \mid \text{esiste un assegnamento } A \text{ degli } n \text{ lavori su } m \text{ linee di produzione tale che } \text{makespan}(A) \leq k \}$$

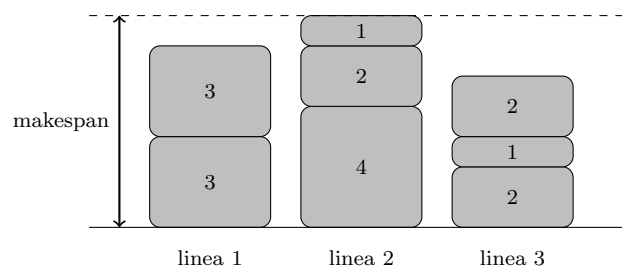


Figura 1: Esempio di assegnamento dei lavori $T = \{1, 1, 2, 2, 2, 3, 3, 4\}$ su 3 linee con makespan 7.

(a) Dimostra che LOADBALANCE è un problema NP.

(b) Dimostra che LOADBALANCE è NP-hard, usando SETPARTITIONING come problema NP-hard di riferimento.

(a) LOADBALANCE è in NP. L'array A con gli assegnamenti è il certificato. Il seguente algoritmo è un verificatore per LOADBALANCE:

$V =$ "Su input $\langle \langle m, T, k \rangle, A \rangle$:

1. Controlla che A sia un vettore di n elementi dove ogni elemento ha un valore compreso tra 1 e m . Se non lo è, rifiuta.
2. Calcola $\text{makespan}(A)$: se è minore o uguale a k accetta, altrimenti rifiuta."

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. La prima fase è un controllo sugli n elementi del vettore A , e quindi richiede un tempo polinomiale rispetto alla dimensione dell'input. Per calcolare il makespan, la seconda fase deve calcolare il tempo di occupazione di ognuna delle m linee e poi trovare il massimo tra i tempi di occupazione, operazioni che si possono fare in tempo polinomiale rispetto alla dimensione dell'input.

- (b) Dimostriamo che **LOADBALANCE** è NP-Hard per riduzione polinomiale da **SETPARTITIONING** a **LOADBALANCE**. La funzione di riduzione polinomiale f prende in input un insieme di numeri interi positivi $\langle T \rangle$ e produce come output la tripla $\langle 2, T, k \rangle$ dove k è uguale alla metà della somma dei valori in T :

$$k = \frac{1}{2} \sum_{1 \leq i \leq n} T[i]$$

Dimostriamo che la riduzione polinomiale è corretta:

- Se $\langle T \rangle \in \text{SETPARTITIONING}$, allora esiste un modo per suddividere T in due sottoinsiemi T_1 e T_2 in modo tale che la somma dei valori contenuti in T_1 è uguale alla somma dei valori contenuti in T_2 . Nota che questa somma deve essere uguale alla metà della somma dei valori in T , cioè uguale a k . Quindi assegnando i lavori contenuti in T_1 alla prima linea di produzione e quelli contenuti in T_2 alla seconda linea di produzione otteniamo una soluzione per **LOADBALANCE** con makespan uguale a k , come richiesto dal problema.
- Se $\langle 2, T, k \rangle \in \text{LOADBALANCE}$, allora esiste un assegnamento dei lavori alle 2 linee di produzione con makespan minore o uguale a k . Siccome ci sono solo 2 linee, il makespan di questa soluzione non può essere minore della metà della somma dei valori in T , cioè di k . Quindi l'assegnamento ha makespan esattamente uguale a k , ed entrambe le linee di produzione hanno tempo di occupazione uguale a k . Quindi, inserendo i lavori assegnati alla prima linea in T_1 e quelli assegnati alla seconda linea in T_2 otteniamo una soluzione per **SETPARTITIONING**.

La funzione di riduzione deve sommare i valori in T e dividere per due, operazioni che si possono fare in tempo polinomiale.