Esercitazioni III

Docente: Fabio Zanasi, Tutor: Melissa Antonelli

melissa.antonelli2@unibo.it

29 Marzo 2023

Prima di iniziare

- Esercizi scaricabili da virtuale e presentati in slide.
- Due blocchi di esercizi.
- Correzione in casse (lavagna e slide); dal pomeriggio soluzioni di tutti gli esercizi su virtuale.
- Per ogni dubbio (anche su esercizi precedenti o facoltativi) scrivetemi: `melissa.antonelli2@unibo.it`

Argomenti di oggi

- Time-complexity e notazione asintotica
- Classe **NP**
- Poly-riduzione ed **NP**-completezza
- SAT e teorema di Cook-Levin
- Space-Complexity e PSPACE

Sessione I

Problema 1

Problema 1.

Dato il linguaggio

$$A = \{\mathtt{0}^k\mathtt{1}^k \mid k \geq 0\}$$

considera la (multi-tape) TM M programmata come segue:

1. legge l'input e rigetta se trova 0 a destra di 1.
2. legge gli 0i sul nastro 1 e li copia sul nastro 2.
3. legge gli 1i su nastro 1 e per ogni 1 sul nastro 1 cancella uno 0 sul nastro 2; se tutti gli 0i sono cancellati prima che tutti gli 1i siano letti, rigetta.
4. se tutti gli 0i sono cancellati, accetta; se qualche 0 resta sul nastro 2 rigetta.

Domande:

a. Qual'é la *time complexity* di M?
b. Espressa in notazione asintotica?

Problema 1.

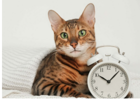
Dato il linguaggio

$$A = \{0^k 1^k \mid k \geq 0\}$$

considera (multi-tape) TM *M* programmata come segue:

1. legge l'input e rigetta se trova `0` a destra di `1`.
2. legge gli `0` sul nastro 1 e li copia sul nastro 2.
3. legge gli `1`i su nastro 1 e per ogni `1` sul nastro 1 cancella uno `0` sul nastro 2; se tutti gli `0`i sono cancellati prima che tutti gli `1`i siano letti, rigetta.
4. se tutti gli `0`i sono cancellati, accetta; se qualche `0` resta sul nastro 2 rigetta.

(a.) Qual'é la *time complexity* di *M*? (b.) Espressa in notazione asintotica?



Correzione tra 10/15 minuti.

Ricorda che...

Definizione (Time Complexity)

Sia M una TM che ferma su ogni input. La sua *time complexity* é definita come $f : \mathbb{N} \to \mathbb{N}$, dove $f(n)$ é il massimo numero di passi che M impiega a fermarsi su input arbitrario di lunghezza n.

Soluzione 1.a

Soluzione 1.a

Notiamo che, per M:

Soluzione 1.a

Notiamo che, per M:

1. leggere l'input richiede n **passi**; (eventualmente) rigettare richiede **1 passo**.

Soluzione 1.a

Notiamo che, per M:

1. leggere l'input richiede n **passi**; (eventualmente) rigettare richiede **1 passo**.
2. tornare all'inizio richiede n **passi**; copiare gli `0`i da nastro 1 a nastro 2 richiede n **passi**.

Soluzione 1.a

Notiamo che, per M:

1. leggere l'input richiede n **passi**; (eventualmente) rigettare richiede **1 passo**.
2. tornare all'inizio richiede n **passi**; copiare gli `0`i da nastro 1 a nastro 2 richiede n **passi**.
3. tornare all'inizio richiede n **passi**; per ciascun `1` sul nastro 1 cancellare uno `0` sul nastro 2 richiede al piú n **passi**; (eventualmente) rigettare richiede **1 passo**.

Soluzione 1.a

Notiamo che, per M:

1. leggere l'input richiede n **passi**; (eventualmente) rigettare richiede **1 passo**.
2. tornare all'inizio richiede n **passi**; copiare gli `0`i da nastro 1 a nastro 2 richiede n **passi**.
3. tornare all'inizio richiede n **passi**; per ciascun `1` sul nastro 1 cancellare uno `0` sul nastro 2 richiede al piú n **passi**; (eventualmente) rigettare richiede **1 passo**.
4. (eventualmente) accettare o rigettare richiede **1 passo**.

Soluzione 1.a

Notiamo che, per M:

1. leggere l'input richiede n **passi**; (eventualmente) rigettare richiede **1 passo**.
2. tornare all'inizio richiede n **passi**; copiare gli `0`i da nastro 1 a nastro 2 richiede n **passi**.
3. tornare all'inizio richiede n **passi**; per ciascun `1` sul nastro 1 cancellare uno `0` sul nastro 2 richiede al piú n **passi**; (eventualmente) rigettare richiede **1 passo**.
4. (eventualmente) accettare o rigettare richiede **1 passo**.

Quindi la procedura richiede al piú $n + 2n + 2n + 1 = 5n + 1$ passi.

Soluzione 1.b

In notazione asintotica:

Soluzione 1.b

In notazione asintotica:

1. leggere l'input; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.

Soluzione 1.b

In notazione asintotica:

1. leggere l'input; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.
2. tornare all'inizio; copiare gli 0i da nastro 1 a nastro 2 richiede complessivamente $O(n)$ **passi**.

Soluzione 1.b

In notazione asintotica:

1. leggere l'input; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.
2. tornare all'inizio; copiare gli `0`i da nastro 1 a nastro 2 richiede complessivamente $O(n)$ **passi**.
3. tornare all'inizio; per ciascun `1` sul nastro 1 cancellare uno `0` sul nastro 2; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.

Soluzione 1.b

In notazione asintotica:

1. leggere l'input; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.
2. tornare all'inizio; copiare gli `0`i da nastro 1 a nastro 2 richiede complessivamente $O(n)$ **passi**.
3. tornare all'inizio; per ciascun `1` sul nastro 1 cancellare uno `0` sul nastro 2; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.
4. accettare e rigettare richiede $O(1)$ **passi**.

Soluzione 1.b

In notazione asintotica:

1. leggere l'input; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.
2. tornare all'inizio; copiare gli `0`i da nastro 1 a nastro 2 richiede complessivamente $O(n)$ **passi**.
3. tornare all'inizio; per ciascun `1` sul nastro 1 cancellare uno `0` sul nastro 2; (eventualmente) rigettare richiede complessivamente $O(n)$ **passi**.
4. accettare e rigettare richiede $O(1)$ **passi**.

La complessitá complessiva della procedura é dunque

$$O(n) + O(n) + O(n) + O(1) = O(n).$$

Osservazione.

La *time complexity* richiesta per decidere un linguaggio puó dipendere dal modello scelto (in questo caso multi-tape TM).
Per esempio, confronta questo algoritmo con quello presentato in classe (Lezione 11), di complessitá $O(n^2)$.

Problema 2

Problema 2.

Considera il problema

$$SSUM = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\} \text{ \& esiste } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \sum y_i = t\}.$$

Esempio. $\langle\{4, 11, 12, 21, 28, 50\}, 25\rangle \in$ SSUM in quanto 4 + 21 = 24.
Dimostra SSUM $\in$ **NP**

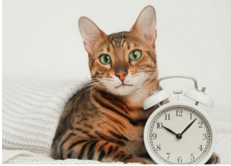
a. tramite (poly-time) NTM
b. traite verificatore.

Problema 2.

Considera il problema

$$SSUM = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\} \text{ \& esiste } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \sum y_i = t\}.$$

Dimostra SSUM $\in$ **NP** (a.) tramite (poly-time) NTM (b.) tramite verificatore.



Correzione tra 15/20 minuti.

Ricorda che...

Definizione (Classe **NP**)

Data una funzione $f : \mathbb{N} \to \mathbb{N}$ definiamo la classe di complessitá (di tempo) NTIME($t(n)$) come collezione di tutti i linguaggi decidibili da NTM (a un nastro) in tempo $O(t(n))$:

$$\mathbf{NP} = \bigcup_k NTIME(n^k)$$

Ricorda che...

Definizione (Classe **NP**)

Data una funzione $f : \mathbb{N} \to \mathbb{N}$ definiamo la classe di complessitá (di tempo) NTIME($t(n)$) come collezione di tutti i linguaggi decidibili da NTM (a un nastro) in tempo $O(t(n))$:

$$\mathbf{NP} = \bigcup_k NTIME(n^k)$$

Definizione (Classe **NP**, alternativa)

Un linguaggio L é *verificabile* se esiste una TM M (che termina sempre, accettando o rigettando), tale che:

$$w \in L \quad sse \quad esiste\ w' \ t.c.\ M\ accetta\ \langle w, w' \rangle$$

Intuitivamente c'é un *certificato* del fatto che w sia in L.

Soluzione 2.a

Definiamo poly-time NTM (ovvero un algoritmo non-deterministico che computa in tempo polinomiale) per SSUM come segue. Su input $\langle S, t\rangle$:

1. Seleziona non-deterministicamente un sottoinsieme S' di numeri di S.
2. Controlla se S' sia un insieme di numeri la cui somma ha valore t.
3. Se sí, accetta; altrimenti, rigetta.

Soluzione 2.b

Definiamo un verificatore *polinomiale* per SSUM. L'idea é che il certificato C sia il sottoinsieme di S i cui elementi sommati abbiano valore t. Su input $\langle\langle S, t\rangle, C\rangle$:

1. Controlla se S contenga tutti i numeri in C.
2. Controlla che C sia un insieme di numeri che sommati danno t.
3. Se entrambi i test hanno esito positivo, accetta; altrimenti, rifiuta.

Problema 3

Problema 3.

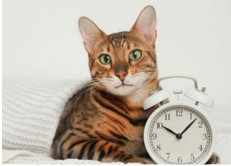
Se L é **NP**-completo, $L \leq_p L'$ e $L' \in$ **NP**, allora L' é **NP**-completo.

Problema 3.

Se L é **NP**-completo, $L \leq_p L'$ e $L' \in$ **NP**, allora L' é **NP**-completo.
Suggerimento. Ricorda che nella passata esercitazione abbiamo dimostrato che la mapping-reduction é transitiva.

Problema 3.

Se L é **NP**-completo, $L \leq_p L'$ e $L' \in$ **NP**, allora L' é **NP**-completo.
Suggerimento. Ricorda che nella passata esercitazione abbiamo dimostrato che la (mapping-)riduzione é transitiva.



Correzione tra 15/20 minuti.

Ricorda che...

Definizione (Poly-Riduzione)

Siano L, L' linguaggi su alfabeto Σ. Diciamo che L é *poly (mapping-)riducibile* a L', $L \leq_p L'$, se esiste una TM che computa in tempo polinomiale una funzione (totale) $f : \Sigma^* \to \Sigma^*$ tale che:

$$x \in L \quad sse \quad f(x) \in L'.$$

Detto altrimenti, $L \leq_p L'$ se $L \leq L'$ e la riduzione corrispondente é computabile in tempo polinomiale.

NP-Completezza

Un linguaggio L é **NP**-completo se é in **NP** e ogni altro linguaggio $L' \in$ **NP** é poly-riducibile ad esso.

Soluzione 3.

Per Df. di **NP**-completezza, per dimostrare che L' é **NP**-completo dobbiamo mostrare (i) $L' \in$ **NP** (ipotesi) e (ii) per ogni $L^* \in$ **NP**, $L^* \leq_p L'$.

Soluzione 3.

Per Df. di **NP**-completezza, per dimostrare che L' é **NP**-completo dobbiamo mostrare (i) $L' \in$ **NP** (ipotesi) e (ii) per ogni $L^* \in$ **NP**, $L^* \leq_p L'$.

Per dimostrare (ii) consideriamo generico $L'' \in$ **NP**.

Soluzione 3.

Per Df. di **NP**-completezza, per dimostrare che L' é **NP**-completo dobbiamo mostrare (i) $L' \in$ **NP** (ipotesi) e (ii) per ogni $L^* \in$ **NP**, $L^* \leq_p L'$.

Per dimostrare (ii) consideriamo generico $L'' \in$ **NP**. L é **NP**-completo (ipotesi), quindi essendo $L'' \in$ **NP**, $L'' \leq_p L$ (Df. **NP**-completezza).

Soluzione 3.

Per Df. di **NP**-completezza, per dimostrare che L' é **NP**-completo dobbiamo mostrare (i) $L' \in$ **NP** (ipotesi) e (ii) per ogni $L^* \in$ **NP**, $L^* \leq_p L'$.

Per dimostrare (ii) consideriamo generico $L'' \in$ **NP**. L é **NP**-completo (ipotesi), quindi essendo $L'' \in$ **NP**, $L'' \leq_p L$ (Df. **NP**-completezza). Inoltre, $L \leq_p L'$ (ipotesi).

Soluzione 3.

Per Df. di **NP**-completezza, per dimostrare che L' é **NP**-completo dobbiamo mostrare (i) $L' \in$ **NP** (ipotesi) e (ii) per ogni $L^* \in$ **NP**, $L^* \leq_p L'$.

Per dimostrare (ii) consideriamo generico $L'' \in$ **NP**. L é **NP**-completo (ipotesi), quindi essendo $L'' \in$ **NP**, $L'' \leq_p L$ (Df. **NP**-completezza).
Inoltre, $L \leq_p L'$ (ipotesi).
Poly-riduzione é un caso particolare (poly-time) di m-riduzione (Df. poly-riduzione); abbiamo visto che $\leq$ é transitiva; ripetendo la costruzione di tale prova con riduzione poly-time dimostriamo anche $\leq_p$ transitiva.

Soluzione 3.

Per Df. di **NP**-completezza, per dimostrare che L' é **NP**-completo dobbiamo mostrare (i) $L' \in$ **NP** (ipotesi) e (ii) per ogni $L^* \in$ **NP**, $L^* \leq_p L'$.

Per dimostrare (ii) consideriamo generico $L'' \in$ **NP**. L é **NP**-completo (ipotesi), quindi essendo $L'' \in$ **NP**, $L'' \leq_p L$ (Df. **NP**-completezza).
Inoltre, $L \leq_p L'$ (ipotesi).
Poly-riduzione é un caso particolare (poly-time) di m-riduzione (Df. poly-riduzione); abbiamo visto che $\leq$ é transitiva; ripetendo la costruzione di tale prova con riduzione poly-time dimostriamo anche $\leq_p$ transitiva. Dunque, dati $L'' \leq_p L$ e $L \leq_p L'$, concludiamo $L'' \leq_p L'$.

Soluzione 3.

Per Df. di **NP**-completezza, per dimostrare che L' é **NP**-completo dobbiamo mostrare (i) $L' \in$ **NP** (ipotesi) e (ii) per ogni $L^* \in$ **NP**, $L^* \leq_p L'$.

Per dimostrare (ii) consideriamo generico $L'' \in$ **NP**. L é **NP**-completo (ipotesi), quindi essendo $L'' \in$ **NP**, $L'' \leq_p L$ (Df. **NP**-completezza).
Inoltre, $L \leq_p L'$ (ipotesi).
Poly-riduzione é un caso particolare (poly-time) di m-riduzione (Df. poly-riduzione); abbiamo visto che $\leq$ é transitiva; ripetendo la costruzione di tale prova con riduzione poly-time dimostriamo anche $\leq_p$ transitiva. Dunque, dati $L'' \leq_p L$ e $L \leq_p L'$, concludiamo $L'' \leq_p L'$.
Poiché $L' \in$ **NP** e L'' é un linguaggio generico in **NP**, per ogni $L^* \in$ **NP**, $L^* \leq_p L'$, concludiamo che L' é **NP**-completo.

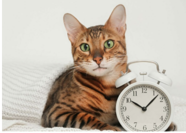
Problema 4.1

Problema 4.1

Sapendo che 3-SAT $\in$ **NP**, mostra (<u>ad alto livello</u>) 3-SAT essere **NP**-completo (usando il teorema di Cook-Levin).

Problema 4.1

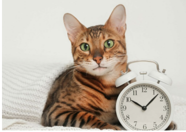
Sapendo che 3-SAT $\in$ **NP**, mostra (ad alto livello) 3-SAT essere **NP**-completo (usando il teorema di Cook-Levin).



Correzione tra 15 minuti (**suggerimento tra 5**/**10**).

Problema 4.1

Sapendo che 3SAT $\in$ **NP**, mostra (ad alto livello) 3SAT essere **NP**-completo (usando il teorema di Cook-Levin).
Suggerimento. Considera il problema precedente insieme alla definizione di **NP**-completezza e il teorema di Cook-Levin.



Correzione tra 15/20 minuti (suggerimento tra 5/10).

Ricorda che...

SAT e 3SAT

Una formula Booleana é ua *clausola* se é una disgiunzione di letterali. Una formula Booleana é in forma normale congiuntiva (CNF) se é una congiunzione di clausole. Una formula Booleana é in 3CNF se é in CNF e ogni clausola contiene esattamente tre letterali.

$$SAT = \{\langle F \rangle \mid F \textit{ formula Booleana soddisfacibile}\}$$

$$3SAT = \{\langle F \rangle \mid F \textit{ formula Booleana in } 3CNF \textit{ soddisfacibile}\}.$$

Soluzione 4.1

SAT é **NP**-completo (Teorema di Cook-Levin).

Soluzione 4.1

SAT é **NP**-completo (Teorema di Cook-Levin).
Inoltre se L é **NP**-completo e $L' \in$ **NP** é tale che $L \leq_p L'$, allora L' é **NP**-completo (Problema 3).

Soluzione 4.1

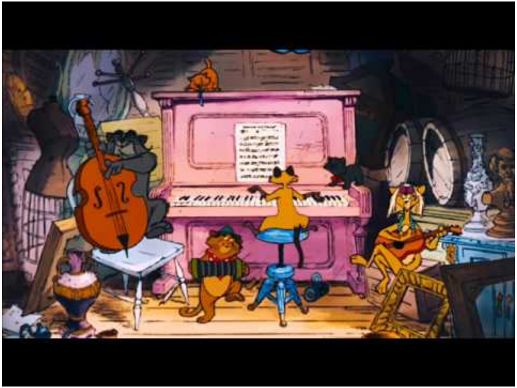
SAT é **NP**-completo (Teorema di Cook-Levin).
Inoltre se L é **NP**-completo e $L' \in$ **NP** é tale che $L \leq_p L'$, allora L' é **NP**-completo (Problema 3).
Per ipotesi 3SAT $\in$ **NP**.

Soluzione 4.1

SAT é **NP**-completo (Teorema di Cook-Levin).
Inoltre se L é **NP**-completo e $L' \in$ **NP** é tale che $L \leq_p L'$, allora L' é **NP**-completo (Problema 3).
Per ipotesi 3SAT $\in$ **NP**.
Dunque, per dimostrare 3SAT **NP**-completo basta dimostrare SAT $\leq_p$ 3SAT ovvero per Df. di poly-riduzione, che esiste funzione *poly-time* computabile f tale che:

$$x \in SAT \quad sse \quad f(x) \in 3SAT.$$

Breve Pausa di 5/10 minuti



Problema 5

Problema 5.1

Un *game* é una competizione tra opposti che tentano di raggiungere un obiettivo seguendo date regole.

Problema 5.1

Un *game* é una competizione tra opposti che tentano di raggiungere un obiettivo seguendo date regole.
Consideriamo *formula game* (fg). Sia F una QBF in PNF, con matrice G. Due giocatori, `A` e `E`, selezionano *a turno* i valori di veritá da attribuire a (gruppi) di variabili. In particolare, `A` assegna i valori alle variabili vincolate da $\forall$ e `E` a quelle vincolate da $\exists$.
Dati questi assegnamenti, se $G = 1$ vince `E`; se $G = 0$, vince `A`.
(a.) Data la formula

$$F_1 = \exists x_1 \forall x_2 \exists x_3 ((x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)),$$

il giocatore `E` assegna $x_1 = 1$; poi `A` assegna $x_2 = 0$; poi `E` assegna $x_3 = 1$. Chi vince?

Problema 5.1

Un *game* é una competizione tra opposti che tentano di raggiungere un obiettivo seguendo date regole.
Consideriamo *formula game* (fg). Sia F una QBF in PNF, con matrice G. Due giocatori, A e E, selezionano *a turno* i valori di veritá da attribuire a (gruppi) di variabili. In particolare, A assegna i valori alle variabili vincolate da $\forall$ e E a quelle vincolate da $\exists$. Dati questi assegnamenti, se $G = 1$ vince E; se $G = 0$, vince A.
(a.) Data la formula

$$F_1 = \exists x_1 \forall x_2 \exists x_3 ((x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)),$$

il giocatore E assegna $x_1 = 1$; poi A assegna $x_2 = 0$; poi E assegna $x_3 = 1$. Chi vince?

Diciamo che un giocatore ha una *winning strategy* se, seguendo tale strategia, vince per ogni scelta dell'altro giocatore.
(b.) Considera

$$F_2 = \exists x \forall y (x \vee \neg y).$$

Il giocatore E ha una *winning strategy* su F_2? Se sí, definiscila.
Considera

$$F_3 = \exists x_1 \forall x_2 \exists x_3 ((x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)).$$

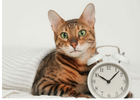
Il giocatore A ha una *winning strategy* per F_3? Se sí, definiscila.

Problema 5.1

Formula game (fg): sia F una QBF in PNF. I giocatori $\mathtt{A}$ e $\mathtt{E}$ selezionano *a turno* i valori di veritá da attribuire a (gruppi) di variabili: $\mathtt{A}$ per le variabili vincolate da $\forall$ e $\mathtt{E}$ le vincolate da $\exists$. Dati questi assegnamenti, se $G = 1$ vince $\mathtt{E}$; se $G = 0$, vince $\mathtt{A}$. Un giocatore ha una *winning strategy* se, seguendo tale strategia, vince per ogni scelta dell'altro giocatore.

a. Data $F_1 = \exists x_1 \forall x_2 \exists x_3 ((x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3))$, il giocatore $\mathtt{E}$ assegna $x_1 = 1$; poi $\mathtt{A}$ assegna $x_2 = 0$; poi $\mathtt{E}$ assegna $x_3 = 1$. Chi vince?

b. Considera $F_2 = \exists x \forall y (x \vee \neg y)$. Il giocatore $\mathtt{E}$ ha una *winning strategy* su F_2? Se sí, definiscila. Considera $F_3 = \exists x_1 \forall x_2 \exists x_3 ((x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_3))$. Il giocatore $\mathtt{A}$ ha una *winning strategy* per F_3? Se sí, definiscila.



Correzione tra 5/10 minuti.

Soluzione 5.1

(a.) Vince E. Infatti $[\![(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)]\!]_{\texttt{play}} = 1$, dove $[\![\cdot]\!]_{\texttt{play}}$ indica l'assegnamento dato.

Soluzione 5.1

(a.) Vince $\mathtt{E}$. Infatti $[\![(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)]\!]_{\mathtt{play}} = 1$, dove $[\![\cdot]\!]_{\mathtt{play}}$ indica l'assegnamento dato.

(b.) Sí $\mathtt{E}$ ha *winning strategy* su F_2: $x = 1$.

Soluzione 5.1

(a.) Vince $\mathtt{E}$. Infatti $[\![(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)]\!]_{\mathtt{play}} = 1$, dove $[\![\cdot]\!]_{\mathtt{play}}$ indica l'assegnamento dato.

(b.) Sí $\mathtt{E}$ ha *winning strategy* su F_2: $x = 1$.
Sí $\mathtt{A}$ ha una *winning strategy* su F_3: $x_2 = 0$, falsifica la matrice indipendentemente dalla scelta di $\mathtt{E}$.

Problema 5.2

Considera il problema di determinare se il giocatore E abbia una *winning strategy* in un *formula game* associato a una data formula F:

$$FG = \{\langle F \rangle \mid \texttt{E} \textit{ ha winning strategy nel fg associato a } F\}$$

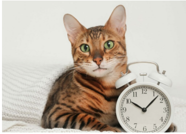
Mostra (anche informalmente) che FG é PSPACE-completo.

Problema 5.2

Considera il problema di determinare se il giocatore $\mathtt{E}$ abbia una *winning strategy* in un *formula game* associato a una data formula F:

$$FG = \{\langle F \rangle \mid \mathtt{E} \text{ } \textit{ha winning strategy nel fg associato a } F\}$$

Mostra (anche informalmente) che FG é PSPACE-completo.



Correzione tra 10/15 minuti.

Ricorda che...

Teorema (TQBF)

TQBF é PSPACE-completo.

Soluzione 5.2

Notiamo che FG altro non é che una diversa formulazione di TQBF. Infatti per ogni QBF:

$$F \textit{ vera} \quad \textit{sse} \quad \mathtt{E} \textit{ ha winning strategy su } F.$$

Soluzione 5.2

Notiamo che FG altro non é che una diversa formulazione di TQBF. Infatti per ogni QBF:

$$F \text{ vera} \quad \text{sse} \quad \mathtt{E} \text{ ha winning strategy su } F.$$

Abbiamo visto che TQBF é PSPACE-completo (Lezione 13).

Soluzione 5.2

Notiamo che FG altro non é che una diversa formulazione di TQBF. Infatti per ogni QBF:

$$F \textit{ vera} \quad \textit{sse} \quad \mathtt{E} \textit{ ha winning strategy su } F.$$

Abbiamo visto che TQBF é PSPACE-completo (Lezione 13).
Allora, chiaramente, anche FG é PSPACE-completo.

Problema 6

Problema 6.

Dato il linguaggio

$$U = \{w \in \Sigma^* \mid w \textit{ contiene eguale numero di } \mathtt{0} \textit{ e } \mathtt{1}\}$$

considera TM M su alfabeto $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ in grado di contrassegnare i simboli dell'alfabeto e tale che su input w:

1. Scansiona il nastro fino a primo bit (non contrassegnato) `0` o `1`

   Se non ne trova alcuno, accetta;

   Altrimenti, continua a scansionare fino al primo bit diverso (`1` e `0` resp.).
2. Se non ne trova, rigetta; altrimenti segna i due simboli e ripete la procedura.

Qual'é la *time complexity* di M in notazione asintotica?

Soluzione 6.

Considera che

1. Scansionare il nastro fino a primo bit (non contrassegnato) richiede **al piú** n **passi**

   (eventualmente) accettare richiede **1 passo**.

   Continuare a scansionare fino al primo bit diverso richiede **gli stessi** n **passi**.
2. (Eventualmente) rigettare richiede **1 passo**; tornare all'inizio richiede **al piú** n **passi**

La procedura complessiva viene ripetuta **al piú** $\frac{n}{2}$ **volte**.
(Nota che stiamo lavorando con un'astrazione della TM e ció che interessa é definire il *bound* al tempo di esecuzione.)

Soluzione 6.

In notazione asintotica

1. Scansionare il nastro fino a primo bit (non contrassegnato) richiede $O(n)$.

   (eventualmente) accettare richiede $O(1)$

   Continuare a scansionare fino al primo bit diverso richiede $O(n)$
2. (Eventualmente) rigettare richiede $O(1)$; tornare all'inizio richiede $O(n)$.

La procedura complessiva viene ripetuta $O(n)$ volte. Concludiamo che globalmente richiede al piú $O(n)O(n) = O(n^2)$ passi.

Problema 7

Problema 7.

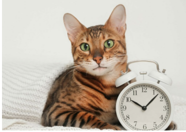
Dimostra PATH $\in$ **P** (senza consultare la slide).
Suggerimento. Ricorda che, dato un grafo diretto G con nodi s e t, il problema PATH consiste nel determinare se esiste un percorso diretto da s a t:

$$PATH = \{\langle G, s, t\rangle \mid G \text{ grafo diretto con percorso diretto da } s \text{ a } t\}.$$

Problema 7.

Dimostra PATH $\in$ **P** (senza consultare la slide).
Suggerimento. Ricorda che, dato un grafo diretto G con nodi s e t, il problema PATH consiste nel determinare se esiste un percorso diretto da s a t:

$$PATH = \{\langle G, s, t\rangle \mid G \text{ grafo diretto con percorso diretto da } s \text{ a } t\}.$$



Correzione tra 15 minuti.

Soluzione 7.

Presentiamo un algoritmo poly-time che decide PATH.

Soluzione 7.

Presentiamo un algoritmo poly-time che decide PATH.
Consideriamo il seguente algoritmo M per decidere PATH. Sia G un grado diretto di nodi s e t:

1. Contrassegna il nodo s.
2. Ripeti la seguente procedura finché nessun nuovo nodo é contrassegnato;

   Scansiona gli archi di G: se trova arco (a, b) da dono contrassegnato a a nodo non contrassegnato b, contrassegna b.
3. Se t é contrassegnato accetta; altrimenti rifiuta.

Soluzione 7.

Presentiamo un algoritmo poly-time che decide PATH.
Consideriamo il seguente algoritmo M per decidere PATH. Sia G un grado diretto di nodi s e t:

1. Contrassegna il nodo s.
2. Ripeti la seguente procedura finché nessun nuovo nodo é contrassegnato;

   Scansiona gli archi di G: se trova arco (a, b) da dono contrassegnato a a nodo non contrassegnato b, contrassegna b.
3. Se t é contrassegnato accetta; altrimenti rifiuta.

Mostriamo che la *time complexity* di tale algoritmo é polinomiale.

Soluzione 7.

Presentiamo un algoritmo poly-time che decide PATH.
Consideriamo il seguente algoritmo M per decidere PATH. Sia G un grado diretto di nodi s e t:

1. Contrassegna il nodo s.
2. Ripeti la seguente procedura finché nessun nuovo nodo é contrassegnato;

   Scansiona gli archi di G: se trova arco (a, b) da dono contrassegnato a a nodo non contrassegnato b, contrassegna b.
3. Se t é contrassegnato accetta; altrimenti rifiuta.

Mostriamo che la *time complexity* di tale algoritmo é polinomiale. 1. e 3. sono eseguiti una sola volta ed entrambi implementabili in tempo polinomiale.

Soluzione 7.

Presentiamo un algoritmo poly-time che decide PATH.
Consideriamo il seguente algoritmo M per decidere PATH. Sia G un grado diretto di nodi s e t:

1. Contrassegna il nodo s.
2. Ripeti la seguente procedura finché nessun nuovo nodo é contrassegnato;

   Scansiona gli archi di G: se trova arco (a, b) da dono contrassegnato a a nodo non contrassegnato b, contrassegna b.
3. Se t é contrassegnato accetta; altrimenti rifiuta.

Mostriamo che la *time complexity* di tale algoritmo é polinomiale. 1. e 3. sono eseguiti una sola volta ed entrambi implementabili in tempo polinomiale. 2. é eseguito al piú un numero di volte corrispondenti al numero dei nodi di G. La scansione dell'input e controllo se i nodi siano contrassegnati é implementabile in tempo polinomiale.

Soluzione 7.

Presentiamo un algoritmo poly-time che decide PATH.
Consideriamo il seguente algoritmo M per decidere PATH. Sia G un grado diretto di nodi s e t:

1. Contrassegna il nodo s.
2. Ripeti la seguente procedura finché nessun nuovo nodo é contrassegnato;

   Scansiona gli archi di G: se trova arco (a, b) da dono contrassegnato a a nodo non contrassegnato b, contrassegna b.
3. Se t é contrassegnato accetta; altrimenti rifiuta.

Mostriamo che la *time complexity* di tale algoritmo é polinomiale. 1. e 3. sono eseguiti una sola volta ed entrambi implementabili in tempo polinomiale. 2. é eseguito al piú un numero di volte corrispondenti al numero dei nodi di G. La scansione dell'input e controllo se i nodi siano contrassegnati é implementabile in tempo polinomiale. Dunque M descrive un algoritmo *poinomiale* per PATH.

Problema 4.2

Problema 4.2

Sia $F = \bigwedge_{j \in \{1,\dots,n\}} l_j$, costruisci una TM funzione f che computa in tempo polinomiale tale che:

$$F \in SAT \quad sse \quad f(F) \in 3SAT.$$

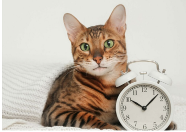
Suggerimento. Considera una funzione ausiliaria g tale che per ogni l, $g(l) = (l \vee x_1 \vee x_2) \wedge (l \vee x_1 \vee \overline{x_2}) \wedge (l \vee \overline{x_1} \vee x_2) \wedge (l \vee \overline{x_1} \vee \overline{x_2})$.

Problema 4.2

Sia $F = \bigwedge_{j \in \{1,\dots,n\}} l_j$, costruisci una TM funzione f che computa in tempo polinomiale tale che:

$$F \in SAT \quad sse \quad f(F) \in 3SAT.$$

Suggerimento. Considera una funzione ausiliaria g tale che per ogni l, $g(l) = (l \vee x_1 \vee x_2) \wedge (l \vee x_1 \vee \overline{x_2}) \wedge (l \vee \overline{x_1} \vee x_2) \wedge (l \vee \overline{x_1} \vee \overline{x_2})$.



Correzione tra 25 minuti.

Soluzione 4.2

Dimostriamo che per ogni letterale l e ogni valutazione v:

$$v(l) = 1 \quad sse \quad v(g(l)) = 1.$$

(Questa dimostrazione puó essere svolta in vari modi, e.g. sistema di prova o tavole di veritá.)

Soluzione 4.2

Dimostriamo che per ogni letterale l e ogni valutazione v:

$$v(l) = 1 \quad sse \quad v(g(l)) = 1.$$

(Questa dimostrazione puó essere svolta in vari modi, e.g. sistema di prova o tavole di veritá.)

Consideriamo una funzione f tale che, per ogni $F = \bigwedge_{j \in \{1,\dots,n\}} l_j$,

$$f(F) = \bigwedge_{j \in \{1,\dots,n\}} g(l_j).$$

(Chiaramente questo richiede tempo polinomiale.)

Soluzione 4.2

Dimostriamo che per ogni letterale l e ogni valutazione v:

$$v(l) = 1 \quad sse \quad v(g(l)) = 1.$$

(Questa dimostrazione puó essere svolta in vari modi, e.g. sistema di prova o tavole di veritá.)

Consideriamo una funzione f tale che, per ogni $F = \bigwedge_{j\in\{1,\dots,n\}} l_j$,

$$f(F) = \bigwedge_{j\in\{1,\dots,n\}} g(l_j).$$

(Chiaramente questo richiede tempo polinomiale.)

Congiunzioni di formule *logicamente equivalenti* sono equivalenti:

$$v(F) = 1 \quad sse \quad v(g(F)) = 1.$$

Soluzione 4.2

Dimostriamo che per ogni letterale l e ogni valutazione v:

$$v(l) = 1 \quad sse \quad v(g(l)) = 1.$$

(Questa dimostrazione puó essere svolta in vari modi, e.g. sistema di prova o tavole di veritá.)

Consideriamo una funzione f tale che, per ogni $F = \bigwedge_{j \in \{1,\dots,n\}} l_j$,

$$f(F) = \bigwedge_{j \in \{1,\dots,n\}} g(l_j).$$

(Chiaramente questo richiede tempo polinomiale.)

Congiunzioni di formule *logicamente equivalenti* sono equivalenti:

$$v(F) = 1 \quad sse \quad v(g(F)) = 1.$$

Quindi (i) $f(F)$ é in 3CNF e (ii) F é soddisfacibile sse $f(F)$ lo é.

Soluzione 4.2

Dimostriamo che per ogni letterale l e ogni valutazione v:

$$v(l) = 1 \quad sse \quad v(g(l)) = 1.$$

(Questa dimostrazione puó essere svolta in vari modi, e.g. sistema di prova o tavole di veritá.)

Consideriamo una funzione f tale che, per ogni $F = \bigwedge_{j \in \{1,\dots,n\}} l_j$,

$$f(F) = \bigwedge_{j \in \{1,\dots,n\}} g(l_j).$$

(Chiaramente questo richiede tempo polinomiale.)

Congiunzioni di formule *logicamente equivalenti* sono equivalenti:

$$v(F) = 1 \quad sse \quad v(g(F)) = 1.$$

Quindi (i) $f(F)$ é in 3CNF e (ii) F é soddisfacibile sse $f(F)$ lo é.
Concludiamo allora che, per ogni $F = \bigwedge_{j \in \{1,\dots,n\}} l_j$:

$$F \in SAT \quad sse \quad f(F) \in 3SAT.$$