1. (12 punti) Data una stringa w di 0 e 1, il *flip* di w si ottiene cambiando tutti gli 0 in w con 1 e tutti gli 1 in w con 0. Dato un linguaggio L, il flip di L è il linguaggio

$$\textit{flip}(L) = \{w \in \{0,1\}^* \mid \text{ il flip di } w \text{ appartiene ad } L\}.$$

Dimostra che la classe dei linguaggi regolari è chiusa rispetto all'operazione di flip.

FLIP → REGOLARE (?)

SE L REGOLARE → ∃ DFA D RICONOSCITORE DOVE

- INSIEME DI STATI $Q_D = Q$
- ALFABETO $\Sigma_D = \Sigma = \{0,1\}^*$
- δ (TRANSIZIONE) → INVERTE VALORI

→ $\delta_D(q_i, \emptyset) = ((q_i, 1), \delta_{D+1})$ [INVERSIONE $\emptyset$ CON 1] (e viceversa)

→ $\delta_D(q_i, 1) = ((q_i, \emptyset), \delta_{D+1})$

- $q_0^D = Q_0^L$ (STESSO ST. INIZIALE)
- $F_D = \{F_i \times F_P\}$

($\Longleftrightarrow$)

Per dimostrare che A' riconosce il linguaggio $\textit{flip}(L)$, dobbiamo considerare due casi.

- Se $w \in \textit{flip}(L)$, allora sappiamo che il flip di w appartiene ad L. Chiamiamo $\overline{w}$ il flip di w. Siccome A riconosce L, allora esiste una computazione di A che accetta $\overline{w}$:
$$s_0 \xrightarrow{\overline{w}_1} s_1 \xrightarrow{\overline{w}_2} \dots \xrightarrow{\overline{w}_n} s_n \qquad (0 \dots 1)$$
con $s_0 = q_0$ e $s_n \in F$. Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per A' sulla parola w. Di conseguenza, abbiamo dimostrato che $w \in L(A')$.
- Viceversa, se w è accettata dal nuovo automa A', allora esiste una computazione accettante che ha la forma (VICEVERSA)
$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$
con $s_0 = q_0$, $s_n \in F'$. Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per A sul flip di w. Di conseguenza, il flip di w appartiene ad L e abbiamo dimostrato che $w \in \textit{flip}(L)$.

2. **(9 punti)** Data una stringa $w \in \{0,1\}^*$, il *flip* di w si ottiene cambiando tutti gli 0 in w con 1 e tutti gli 1 in w con 0. Dato un linguaggio L, il flip di L è il linguaggio

$$\mathit{flip}(L) = \{w \in \{0,1\}^* \mid \text{ il flip di } w \text{ appartiene ad } L\}.$$

Dimostra che la classe dei *linguaggi context-free* è chiusa rispetto all'operazione di flip.

FLIP → CONTEXT-FREE ($\exists$ G CFG / PDA P)

✓

$\exists$ G CFG → G' EQUIVALENTE (CHOMSKY)

$G = (V, \Sigma, R, S) \Rightarrow G' = (V', \Sigma', R', S')$

- $\Sigma = \Sigma'$ → {0, 1}
- V (INS. DI VARIABILI) → R (REGOLE)
- S (STATO INIZIALE)

$$\left\{ \begin{array}{l} R_0 \to R_1 \mid F \\ R_1 \to R_0 \mid F \\ F \to f \end{array} \right\} \Rightarrow \text{REGOLE}$$

(⇔) (⇒) $\exists$ COMPUT. ACCETTANTE, UNA CFG CON INSIEME DI REGOLE R CHE "REALIZZANO" FLIP

(⇐) $\exists$ FLIP, $\exists$ COMPUT. ACCETTANTE (0/1) . . . . REGOLE

↓

CFG = G'!

3. (12 punti) Una grammatica context-free è *lineare* se ogni regola in R è nella forma $A \to aBc$ o $A \to a$ per qualche $a, c \in \Sigma \cup \{\varepsilon\}$ e $A, B \in V$. I linguaggi generati dalle grammatiche lineari sono detti *linguaggi lineari*. Dimostra che i linguaggi regolari sono un sottoinsieme proprio dei linguaggi lineari.

L

$\left[\begin{array}{l} A \to aBc \\ A \to a \end{array}\right]$ (POSSIBILE) REG. ⊂ LINEARI (SOTT. PROPRIO)

→ RIPETIZ. DI CARATTERI... $\not\subset$ (NO)

||

NFA CHE RICONOSCE L

① REGOLARI → LINEARI (OK)

② NON REG. ≡ LINEARE (!)

NFA $N \to (Q, q_0, \Sigma, \delta, F)$

CFG $G \to (V, \Sigma, R, S)$

$\left[\begin{array}{l} A \to aBc \\ (A \to a) \end{array}\right]$

NFA N ELABORA TUTTA LA COMPUTAZIONE

G

- NEL CASO $A \to a$ (REG. → STATO FINALE q_{END})

→ NEL CASO $A \to aBc$

$B \to bCd$

$a_1 \to a_2 \ldots a_n$ } FUNZIONE DI TRANSIZIONE (SCORRO TUTTI I CARATTERI)

$\delta(q, a) = (r, b, L)$

$\to \delta_1(q_1, a) = \{(q_2, b), \delta_2)$

⋮

$\delta_{n-1}(q_{n-2}, z) = ((a_n, \varepsilon), \delta_n)$

NFA → CFG (ALGORITMO DI ELIMINAZIONE STATI - THOMPSON)

(CHOMSKY)

↓

2n - 1 PASSI (FINITO !)

REG. → LINEARE

(~~←~~) → NON VICEVERSA

$L = \{0^n 1^n \mid n \geq 0\}$

$S \to 0S1 \mid \varepsilon$

- Dato un linguaggio regolare L, sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Inoltre, sappiamo che ogni DFA può essere convertito in una grammatica context-free dove le regole sono del tipo $R_i \to aR_j$ per ogni transizione $\delta(q_i, a) = q_j$ del DFA, e del tipo $R_i \to \varepsilon$ per ogni stato finale del q_i del DFA. Entrambi i tipi di regola rispettano le condizioni di linearità, quindi la grammatica equivalente al DFA è lineare, e questo implica che L è un linguaggio lineare.

1. (9 punti) Considera il linguaggio

$$L = \{x1^n \mid x \in \{0,1\}^* \text{ e } n \geq |x|\}.$$

Dimostra che L non è regolare.

L ⇒ NON REGOLARE !

PL ⇒ NON REGOLARE → PUMPING LENGTH "K"

$w = xyz$

$\underline{|xy| \leq z}$

$\emptyset \ldots$

$x = 0^q \quad q > 0$

$y = 0^p$

$z = 1^n 0^{k-p-q}$

1. (9 punti) Considera il linguaggio

$$L = \{x1^n \mid x \in \{0,1\}^* \text{ e } n \geq |x|\}.$$

Dimostra che L non è regolare.

$xy^0z = \cancel{0^p}\, 1^n\, 0^{\cancel{k-p}}$

$= \underline{1}^n\, 0^k$

$x = \varepsilon$ $\quad q > 0$

$y = 0^p$

$z = 1^n 0^{k-p}$

NUMERO DIVERSO RISPETTO A QUELLO INIZIALE

↓

NON REGOLARE

$n \geq |x|$

$k \geq |x|$ (ASSURDO)

2. (12 punti) Considera il linguaggio

$$L_2 = \{uvvu \mid u, v \in \{0,1\}^*\}.$$

Dimostra che L_2 non è regolare.

$w = xy^iz$, $i \geq 0$

$(|xy| \leq k)$

$w = 0110$

$x \quad y \quad z$

$x = 0^q$, $q > 0$

$y = 0^p$

$z = 1^k 0^{k-p-q}$

$xy^iz = xy^2z = 0^q\, 0^{2p}\, 1^k\, 0^{k-p-q}$

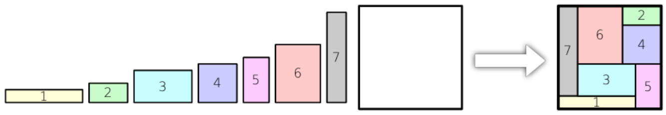
$= 1^k 0^{k+p}$ (SBILANCIATO → :))

- poiché $|xy| \leq k$, allora x e y sono entrambe contenute nella sequenza iniziale 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: $z = 0^{k-q-p}110^k$. Consideriamo l'esponente $i = 2$: la parola xy^2z ha la forma

$$xy^2z = 0^q0^{2p}0^{k-q-p}110^k = 0^{k+p}110^k$$

3. *Il problema* SetPartitioning *chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2. Sappiamo che questo problema è NP-hard.*

Il problema RectangleTiling *è definito come segue: dato un rettangolo grande e diversi rettangoli più piccoli, determinare se i rettangoli più piccoli possono essere posizionati all'interno del rettangolo grande senza sovrapposizioni e senza lasciare spazi vuoti.*



Un'istanza positiva di RectangleTiling.

Dimostra che RectangleTiling *è NP-hard, usando* SetPartitioning *come problema di riferimento.*

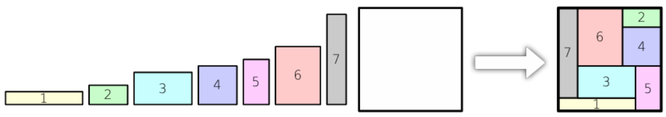
① RECTANGLETILING → NP
- ∃ VERIFICATORE
- CERTIFICATO (INPUT BUONO)

[RETTANGOLO = R
RETTANGOLI PICCOLI $\sum_{i=1}^{n} r_i = R$] → CERTIFICATO

VERIFICATORE V:
- $|R| = n$ ← CONTA IL TOTALE
- CONTA TUTTI I BLOCCHI $\sum_{i=1}^{n} r_i = R$
- SE TUTTO OK, RITORNA TRUE (YES)

3. *Il problema* SetPartitioning *chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2. Sappiamo che questo problema è NP-hard.*

Il problema RectangleTiling *è definito come segue: dato un rettangolo grande e diversi rettangoli più piccoli, determinare se i rettangoli più piccoli possono essere posizionati all'interno del rettangolo grande senza sovrapposizioni e senza lasciare spazi vuoti.*



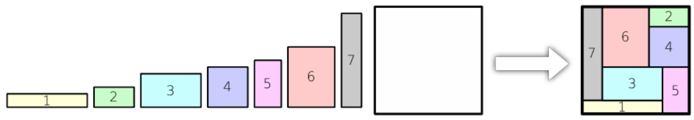
Un'istanza positiva di RectangleTiling.

Dimostra che RectangleTiling *è NP-hard, usando* SetPartitioning *come problema di riferimento.*

$A \le_m B \Rightarrow SP \le_m RT$

3. *Il problema* SetPartitioning *chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2. Sappiamo che questo problema è NP-hard.*

Il problema RectangleTiling *è definito come segue: dato un rettangolo grande e diversi rettangoli più piccoli, determinare se i rettangoli più piccoli possono essere posizionati all'interno del rettangolo grande senza sovrapposizioni e senza lasciare spazi vuoti.*

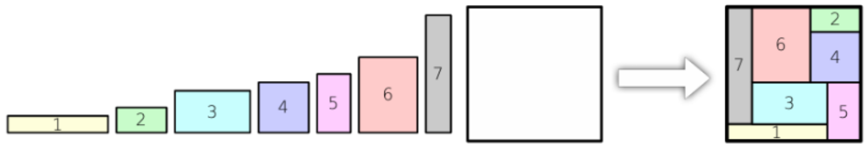


Un'istanza positiva di RectangleTiling.

Dimostra che RectangleTiling *è NP-hard, usando* SetPartitioning *come problema di riferimento.*

$\Rightarrow$ SP $\equiv S$, $S_1 \cup S_2 = S$

$S_1 \cap S_2 = \emptyset$



= BASE + ALTEZZA

PARTIAMO DA UN BLOCCO b

$\boxed{1}$ IF $(\underbrace{b_i}_{S_1} + \underbrace{b_j}_{S_2}) \leq S$ $(b_i + b_j) = b_k$

$S \to S_1 \leq \frac{S}{2}$, $S_2 \leq \frac{S}{2}$

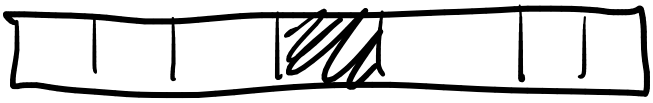
$\frac{S}{2} \to \frac{S}{4}$, $\frac{S}{4} \to \frac{S}{8}$

$RT \equiv \sum_{i=1}^{n} r_i = R$

ORDINO $\to$ [PICCOLI = GRANDI] = RIESCI A METTERE PIÙ BLOCCHI PICCOLI

$\left[S_1 \leq \frac{S}{2}, \quad S_2 \leq \frac{S}{2} \right] \to$ SEMPLICE

$\frac{S}{2}$; $\frac{3}{2}S$ [PARI ; DISPARI]



DISPARI = [illegible]

RETTANGOLO $\frac{S}{2}$ (ALTEZZA)

$3/2\ S$ (BASE)

$$\frac{S}{2} + \frac{3}{2}S = S$$

BASE ALTEZZA

$\frac{S}{2}$ $\frac{3}{2}$

$R = S$

Dimostriamo che RectangleTiling è NP-Hard per riduzione polinomiale da SetPartitioning. La funzione di riduzione polinomiale prende in input un insieme di interi positivi $S = \{s_1, \dots, s_n\}$ e costruisce un'istanza di RectangleTiling come segue:

- i rettangoli piccoli hanno altezza 1 e base uguale ai numeri in S moltiplicati per 3: $(3s_1, 1), \dots, (3s_n, 1)$;
- il rettangolo grande ha altezza 2 e base $\frac{3}{2}N$, dove $N = \sum_{i=1}^{n} s_i$ è la somma dei numeri in S.

Dimostriamo che esiste un modo per suddividere S in due insiemi S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 se e solo se esiste un tiling corretto:

- Supponiamo esista un modo per suddividere S nei due insiemi S_1 e S_2. Posizioniamo i rettangoli che corrispondono ai numeri in S_1 in una fila orizzontale, ed i rettangoli che corrispondono ad S_2 in un'altra fila orizzontale. Le due file hanno altezza 1 e base $\frac{3}{2}N$, quindi formano un tiling corretto.
- Supponiamo che esista un modo per disporre i rettangoli piccoli all'interno del rettangolo grande senza sovrapposizioni né spazi vuoti. Moltiplicare le base dei rettangoli per 3 serve ad impedire che un rettangolo piccolo possa essere disposto in verticale all'interno del rettangolo grande. Quindi il tiling valido è composto da due file di rettangoli disposti in orizzontale. Mettiamo i numeri corrispondenti ai rettangoli in una fila in S_1 e quelli corrispondenti all'altra fila in S_2. La somma dei numeri in S_1 ed S_2 è pari ad $N/2$, e quindi rappresenta una soluzione per SetPartitioning.

Per costruire l'istanza di RectangleTiling basta scorrere una volta l'insieme S, con un costo polinomiale.

RT è NP-COMPLETO = { NP, NP-HARD }

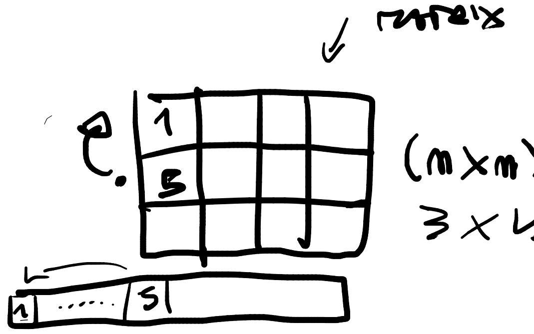
1. *Una* macchina di Turing bidimensionale *utilizza una griglia bidimensionale infinita di celle come nastro. Ad ogni transizione, la testina può spostarsi dalla cella corrente ad una qualsiasi delle quattro celle adiacenti. La funzione di transizione di tale macchina ha la forma*

$$\left(\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\uparrow, \downarrow, \rightarrow, \leftarrow\},\right) \rightarrow \text{F. DI TRANSIZIONE (a)}$$

dove le frecce indicano in quale direzione si muove la testina dopo aver scritto il simbolo sulla cella corrente.

Dimostra che ogni macchina di Turing bidimensionale può essere simulata da una macchina di Turing deterministica a nastro singolo.] (b)

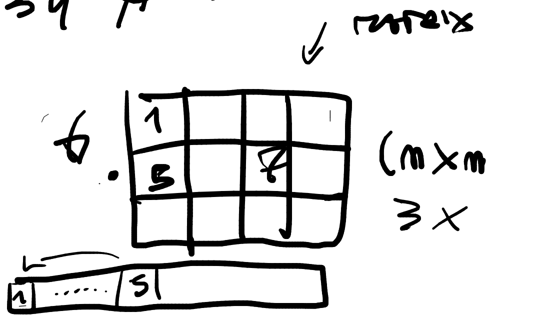
S = N. SINGOLO EQUIVALENTE



$\boxed{1}$ $\delta(q,a) = (r, b, \uparrow)$

NEL MOMENTO IN CUI SI HA ↑, LA MACCHINA SI SPOSTA FINO AL PRECEDENTE SIMBOLO "." DI UN NUM. CELLE EQUIVALENTE A m (N. DI RIGA). SE NON CI SONO ".", SI SPOSTA FINO A INIZIO NASTRO PRIMA DEL PRIMO #. LE RIGHE SONO SEPARATE DA #

\# 1 2 3 4 # 5 6 7 8 #....



$\boxed{2}$ $\delta(q,a) = (r, b, \downarrow)$

NEL MOMENTO IN CUI SI HA ↓, LA MACCHINA SI SPOSTA FINO AL SUCCESSIVO SIMBOLO "." DI UN NUM. CELLE EQUIVALENTE A m (N. DI RIGA). SE NON CI SONO ".", SI SPOSTA FINO A FINE NASTRO PRIMA DELL'ULTIMO #. LE RIGHE SONO SEPARATE DA #,

\# 1 2 3 4 # 5 6 7 8 #....

$\boxed{3}$ $\delta(q,a) = (r, b, \rightarrow)$ → SCRIVE R SUL NASTRO MUOVENDOSI REGOLARMENTE VERSO DESTRA

(←) = ANALOGAMENTE A SX...

SE Q END → END

→ TORNA AL PASSO $\boxed{1}$

$S =$ "su input w:

1. Sostituisce w con la configurazione iniziale $\#\#w\#\#$ e marca con $\frown$ il primo simbolo di w.
2. Scorre il nastro finché non trova la cella marcata con $\frown$.
3. Aggiorna il nastro in accordo con la funzione di transizione di B:
   - Se $\delta(r, a) = (s, b, \rightarrow)$, scrive b non marcato sulla cella corrente, sposta $\frown$ sulla cella immediatamente a destra. Poi sposta di una cella a destra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un $\#$, S scrive un blank marcato al posto del $\#$ e sposta il contenuto del nastro da questa cella fino al $\#\#$ finale, di una cella più a destra.
   - Se $\delta(r, a) = (s, b, \leftarrow)$, scrive b non marcato sulla cella corrente, sposta $\frown$ sulla cella immediatamente a sinistra. Poi sposta di una cella a sinistra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un $\#$, S scrive un blank marcato al posto del $\#$ e sposta il contenuto del nastro da questa cella fino al $\#\#$ iniziale, di una cella più a sinistra.
   - Se $\delta(r, a) = (s, b, \uparrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\frown$ sulla prima cella marcata con un pallino posta a sinistra della cella corrente. Se questa cella marcata non esiste, aggiunge una nuova riga composta solo da blank all'inizio della configurazione.
   - Se $\delta(r, a) = (s, b, \downarrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\frown$ sulla prima cella marcata con un pallino posta a destra della cella corrente. Se questa cella non esiste, aggiunge una nuova riga composta solo da blank alla fine della configurazione.

2. (12 punti) Considera il linguaggio

$$\text{FORTY-TWO} = \{\langle M, w\rangle \mid M \text{ termina la computazione su } w \text{ avendo solo 42 sul nastro}\}.$$

Dimostra che FORTY-TWO è indecidibile.

$A \leq_m B \Rightarrow A_{TM} \leq_m 42_{TM}$

$\overline{A_{TM}}$

E_{TM}

F = F. DI RIDUZIONE, SU INPUT $\langle M, w\rangle$:

1. COPIA TUTTO L'INPUT SUL NASTRO (W)
2. SIMULA M' SU X
3. SE X = 42, ACCETTA
4. SE X ≠ 42, VA IN LOOP CANCELLANDO TUTTA LA COMPUT. PRECEDENTE
5. RITORNA $\langle M, w\rangle$

$\mathrm{HALT}_{TM} / \leq_m 42_{TM}$

$E_{TM} \leq_m 42_{TM}$

```
M' = "Su input x:
1. Ignora completamente l'input x
2. Simula M₀ su w₀
3. Se M₀ termina (accetta o rifiuta):
   a. Cancella tutto il nastro
   b. Scrivi 42 sul nastro
   c. Termina nello stato di accettazione
4. Se M₀ non termina su w₀:
   - M' non termina"
```

M' = SU INPUT X:
- IGNORA L'INPUT X;
- SIMULA M' SUL NASTRO
- SE L'INPUT È VUOTO, ESEGUI M
- SE M SCRIVE 42, RIFIUTA
- SE M NON SCRIVE, ACCETTA

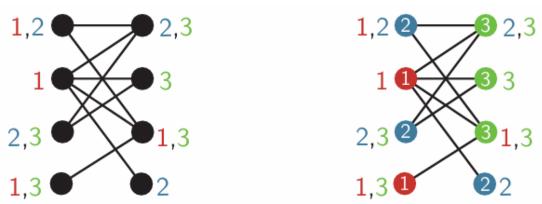
(Generi) $L = \emptyset \Rightarrow E_{TM}$

$\langle M, w\rangle \in A_{TM} / E_{TM} / \mathrm{HALT}_{TM} \iff \langle M', x\rangle \in 42_{TM}$

3. (12 punti) "Colorare" i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate "colori", ai vertici del grafo in modo tale che nessuna coppia di vertici adiacenti condivida lo stesso colore. Considera la seguente variante del problema chiamata LIST-COLORING. Oltre al grafo G, l'input del problema comprende anche una *lista di colori ammissibili* $L(v)$ per ogni vertice v del grafo.

Il problema che dobbiamo risolvere è stabilire se possiamo colorare il grafo G in modo che ogni vertice v sia colorato con un colore preso dalla lista di colori ammissibili $L(v)$.

$$\text{LIST-COLORING} = \{\langle G, L\rangle \mid \text{esiste una colorazione } c_1, \ldots, c_n \text{ degli } n \text{ vertici tale che } c_v \in L(v) \text{ per ogni vertice } v\}$$



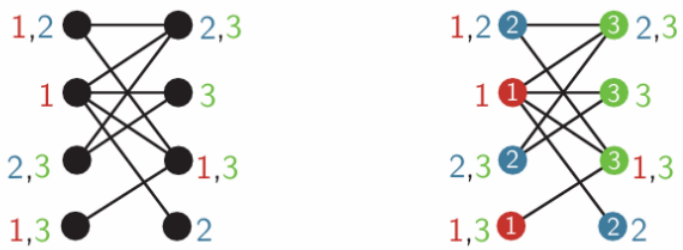
Esempio di istanza (a sinistra) e soluzione (a destra) di LIST-COLORING.

TRUE!

(a) Dimostra che LIST-COLORING è un problema NP. → $\exists\, c_1 \ldots c_n,\ \exists\, c_v \in L(v)$?

(b) Dimostra che LIST-COLORING è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

3-COLOR $\leq_m$ LIST-COLORING



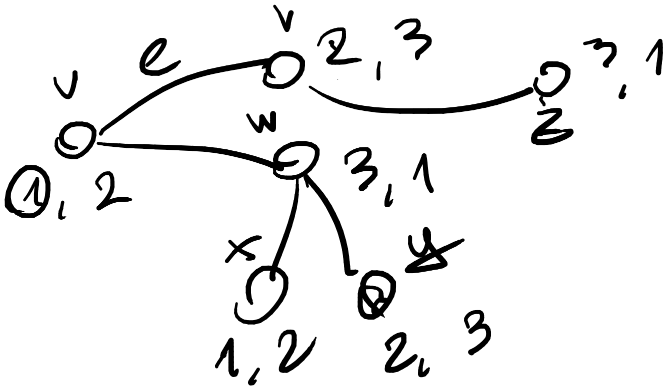
LIST-COLORING ⊲ 3-COLOR

L(v) = LISTA COLORI AMMISSIBILI

$C_1 \dots C_m$ = COLORAZIONE

G = GRAFO

∃ G ,



PARTIAMO DA UN VERTICE E GLI ASSEGNI ALTRI DUE COLORI → 1

AI VICINI ASSEGNO UN ALTRA COPPIA DI COLORI → 2

COPRO TUTTO IL GRAFO SAPENDO CHE ∃ $C_1 \dots C_m$ → TUTTI I COLORI → 3

3-COLOR ≤ | 1,2,3 > F| 4 → NP

Direzione (⇒): Se G è 3-colorabile

Esiste una colorazione valida c: V → {1, 2, 3} per G
La stessa colorazione c soddisfa:

c(v) ∈ L'(v) = {1, 2, 3} per ogni v ∈ V ✓
c(u) ≠ c(v) per ogni (u, v) ∈ E ✓ → ARCHI CON VERTICI CON COLORI DIVERSI

Quindi ⟨G', L'⟩ ∈ List-Coloring

Direzione (⇐): Se ⟨G', L'⟩ ∈ List-Coloring

Esiste una colorazione c: V → ℕ tale che:

c(v) ∈ L'(v) = {1, 2, 3} per ogni v ∈ V
c(u) ≠ c(v) per ogni (u, v) ∈ E

Quindi c: V → {1, 2, 3} è una 3-colorazione valida di G
G è 3-colorabile