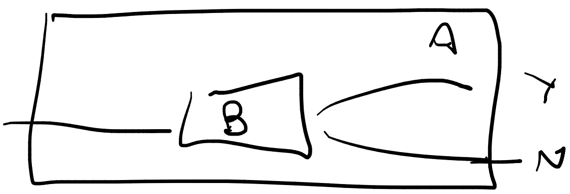
3. **(12 punti)** Un ristorante ha necessità di acquistare m ingredienti per preparare tutti i piatti che offre ai suoi clienti. Dato un insieme di n fornitori che forniscono tali ingredienti, si desidera determinare se esiste un piccolo insieme di fornitori che consentano di acquistare tutti gli ingredienti di cui il ristorante ha bisogno. Se ogni fornitore $i = 1, \dots, n$ fornisce un insieme $S_i \subseteq \{1, \dots, m\}$ di ingredienti, una *fornitura valida* è un insieme T di fornitori che, nel loro insieme, forniscono tutti gli m ingredienti: $\bigcup_{i \in T} S_i = \{1, \dots, m\}$. Definiamo il linguaggio

$$SUPPLY = \{\langle S_1, \dots, S_n, k \rangle \mid \text{esiste una fornitura valida } T \text{ di dimensione } |T| = k\}.$$

(a) Dimostra che *SUPPLY* è un problema NP.

(b) Una copertura di vertici di un grafo G è un insieme di vertici T tale che ogni arco del grafo ha almeno una estremità su un vertice in T. Sappiamo che il linguaggio $VERTEX\text{-}COVER = \{\langle G, k \rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$ è NP-completo. Dimostra che *SUPPLY* è NP-hard, usando *VERTEX-COVER* come problema NP-hard di riferimento.

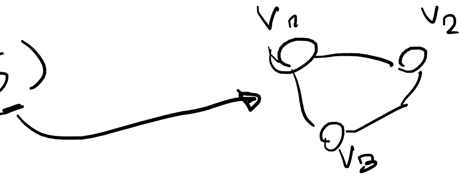
$(A \leq_m B)$

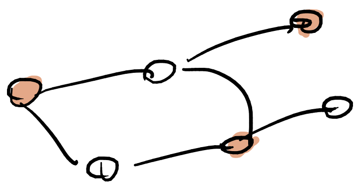


A INDEC. / ⇒ B

NP-HARD. UGUALE

$G = (V, E)$





← N. VERTICI SCONNESSI PER COPRIRE IL GRAFO

- VERTEX-COVER

- HAMILTONIAN PATH

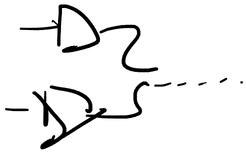
(K COLORI DIVERSI)

- K-COLOR

K-SAT

- SAT

CIRCUIT-SAT



TRUE = 1

CLAUSOLA → $(a \wedge b \vee c) \wedge (\dots) \dots = 1$

LETTERALE

3. (12 punti) Un ristorante ha necessità di acquistare m ingredienti per preparare tutti i piatti che offre ai suoi clienti. Dato un insieme di n fornitori che forniscono tali ingredienti, si desidera determinare se esiste un piccolo insieme di fornitori che consentano di acquistare tutti gli ingredienti di cui il ristorante ha bisogno. Se ogni fornitore $i = 1, \dots, n$ fornisce un insieme $S_i \subseteq \{1, \dots, m\}$ di ingredienti, una *fornitura valida* è un insieme T di fornitori che, nel loro insieme, forniscono tutti gli m ingredienti: $\bigcup_{i \in T} S_i = \{1, \dots, m\}$. Definiamo il linguaggio

$$SUPPLY = \{\langle S_1, \dots, S_n, k\rangle \mid \text{esiste una fornitura valida } T \text{ di dimensione } |T| = k\}.$$

100

TARGET

(a) → (a) Dimostra che *SUPPLY* è un problema NP.

(b) Una copertura di vertici di un grafo G è un insieme di vertici T tale che ogni arco del grafo ha almeno una estremità su un vertice in T. Sappiamo che il linguaggio *VERTEX-COVER* $= \{\langle G, k\rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$ è NP-completo. Dimostra che *SUPPLY* è NP-hard, usando *VERTEX-COVER* come problema NP-hard di riferimento.

RIDUZIONE

= CERTIFICATO / VERIFICATORE

(a) NP ↗ 1/2

INPUT VALIDO

MODELLO CHE DIMOSTRA CHE PUOI RISOLVERE IL PROBLEMA

3. (12 punti) Un ristorante ha necessità di acquistare m ingredienti per preparare tutti i piatti che offre ai suoi clienti. Dato un insieme di n fornitori che forniscono tali ingredienti, si desidera determinare se esiste un piccolo insieme di fornitori che consentano di acquistare tutti gli ingredienti di cui il ristorante ha bisogno. Se ogni fornitore $i = 1, \dots, n$ fornisce un insieme $S_i \subseteq \{1, \dots, m\}$ di ingredienti, una *fornitura valida* è un insieme T di fornitori che, nel loro insieme, forniscono tutti gli m ingredienti: $\bigcup_{i \in T} S_i = \{1, \dots, m\}$. Definiamo il linguaggio

$$SUPPLY = \{\langle S_1, \dots, S_n, k\rangle \mid \text{esiste una fornitura valida } T \text{ di dimensione } \boxed{|T| = k\}.}$$

(a) Dimostra che *SUPPLY* è un problema NP.

(b) Una copertura di vertici di un grafo G è un insieme di vertici T tale che ogni arco del grafo ha almeno una estremità su un vertice in T. Sappiamo che il linguaggio *VERTEX-COVER* $= \{\langle G, k\rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$ è NP-completo. Dimostra che *SUPPLY* è NP-hard, usando *VERTEX-COVER* come problema NP-hard di riferimento.

$\bigcup_{i \in T}, S_i = \{1 \dots m\}$

1 2 . . . m

(a) V = VERIFICATORE, SU INPUT $\bigcup_{i \in T}, S_i = \{1 \dots m\}$

[1] DATI GLI N FORNITORI, COMINCIO AD ASSEGNARE S_i

[2] CONTINUA FINCHÉ NON ARRIVI A K

[3] SE ARRIVI A K, DAI IN OUTPUT "SI" OPPURE "NO"

- LEGGI IL PROBLEMA / USA GLI INPUT MATEMATICI //

RISPETTA LA CONDIZIONE

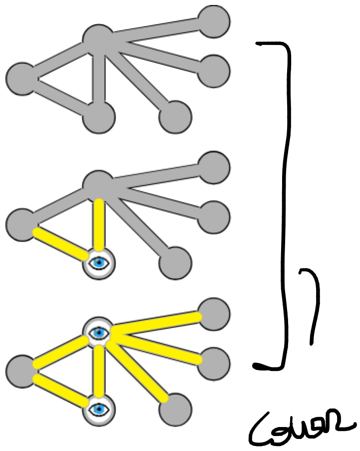
3. (12 punti) Un ristorante ha necessità di acquistare m ingredienti per preparare tutti i piatti che offre ai suoi clienti. Dato un insieme di n fornitori che forniscono tali ingredienti, si desidera determinare se esiste un piccolo insieme di fornitori che consentano di acquistare tutti gli ingredienti di cui il ristorante ha bisogno. Se ogni fornitore $i = 1, \dots, n$ fornisce un insieme $S_i \subseteq \{1, \dots, m\}$ di ingredienti, una *fornitura valida* è un insieme T di fornitori che, nel loro insieme, forniscono tutti gli m ingredienti: $\bigcup_{i \in T} S_i = \{1, \dots, m\}$. Definiamo il linguaggio

$$\mathit{SUPPLY} = \{\langle S_1, \dots, S_n, k\rangle \mid \text{esiste una fornitura valida } T \text{ di dimensione } |T| = k\}.$$

(a) Dimostra che *SUPPLY* è un problema NP.

→ (b) Una copertura di vertici di un grafo G è un insieme di vertici T tale che ogni arco del grafo ha almeno una estremità su un vertice in T. Sappiamo che il linguaggio $\mathit{VERTEX\text{-}COVER} = \{\langle G, k\rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$ è NP-completo. Dimostra che *SUPPLY* è NP-hard, usando *VERTEX-COVER* come problema NP-hard di riferimento.

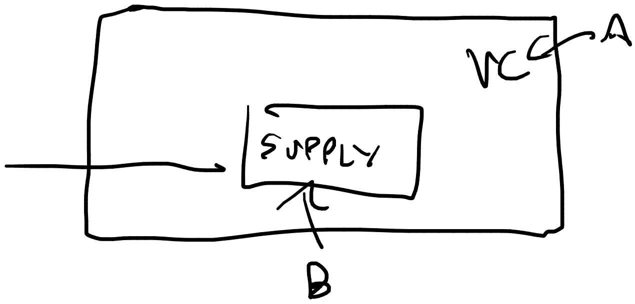
$A \leq_m B \iff \underline{VC \leq_m SUPPLY}$



NP-HARD

① SE VC, ALLORA SUPPLY

② SE SUPPLY, ALLORA VC



3. (12 punti) Un ristorante ha necessità di acquistare m ingredienti per preparare tutti i piatti che offre ai suoi clienti. Dato un insieme di n fornitori che forniscono tali ingredienti, si desidera determinare se esiste un piccolo insieme di fornitori che consentano di acquistare tutti gli ingredienti di cui il ristorante ha bisogno. Se ogni fornitore $i = 1, \dots, n$ fornisce un insieme $S_i \subseteq \{1, \dots, m\}$ di ingredienti, una *fornitura valida* è un insieme T di fornitori che, nel loro insieme, forniscono tutti gli m ingredienti: $\bigcup_{i \in T} S_i = \{1, \dots, m\}$. Definiamo il linguaggio

$$\mathit{SUPPLY} = \{\langle S_1, \dots, S_n, k\rangle \mid \text{esiste una fornitura valida } T \text{ di dimensione } |T| = k\}.$$

(a) Dimostra che *SUPPLY* è un problema NP.

(b) Una copertura di vertici di un grafo G è un insieme di vertici T tale che ogni arco del grafo ha almeno una estremità su un vertice in T. Sappiamo che il linguaggio $\mathit{VERTEX\text{-}COVER} = \{\langle G, k\rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$ è NP-completo. Dimostra che *SUPPLY* è NP-hard, usando *VERTEX-COVER* come problema NP-hard di riferimento.

- SUPPLY, $S_i \subseteq \{1 \dots m\}$

$\bigcup_{i \in T} S_i = \{1 \dots m\}$

$|T| = K$

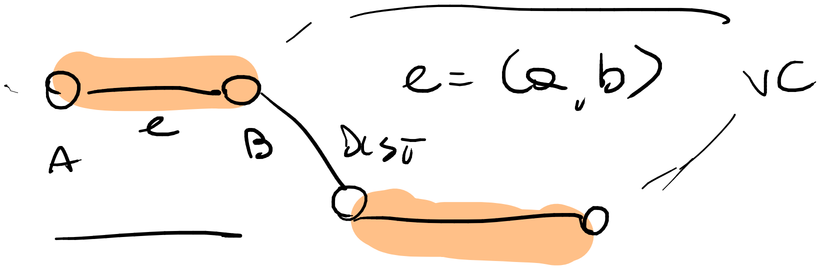
TARGET

∃! CERTIFICATO = SUPPLY = ALMENO UNA VOLTA SI RISOLVE

- VC, G HA COPERTURA $|K|$

① SE VC, ALLORA ESISTE UNA COPERTURA DI DIMENSIONE K

DATO IL GRAFO, PARTI DA UN VERTICE E PER OGNI COPPIA DI VERTICI, ∃ UN ARCO LI COLLEGA.



FORNITORE

PER COPRIRE IL GRAFO,
DOBBIAMO RAGGIUNGERE
ALTRI FORNITORI.

ESISTONO SICURAMENTE ALTRI ARCHI CHE COPRONO IL GRAFO. CONTINUA FINCHÉ $\forall i \in m \ldots$

$\bigcup_{i \in T} S_i \to |K|$ = TARGET

TOT INGREDIENTI DIVERSI

→ COVER

→ K =

S = INGREDIENTE

↓

FORNITORE

H

H

H .....

→ K

(2) . SUPPLY = $\{S_1 \ldots S_m\}$, $|T|$, K → n = INGREDIENTI

FORNITORI

→ ∃ SICURAMENTE LA COPERTURA DI FORNITORI

↓

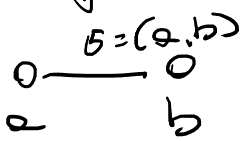
COPRE A AI INGREDIENTI

. VC = $\langle G, K \rangle$ → K = COPERTURA

$\forall i \in \{m\}$

FORNITORE

↓



∀ INGREDIENTE → FORNITORE

[SUPPLY → NP-COMPLETO { NP, NP-HARD]

{L R, SAVE, RESTORE}

1. **(12 punti)** Una macchina di Turing con "save e restore" (SRTM) è una macchina di Turing deterministica a singolo nastro, che può salvare la configurazione corrente per poi ripristinarla in un momento successivo. Oltre alle normali operazioni di spostamento a sinistra e a destra, una SRTM può effettuare l'operazione di SAVE, che salva la configurazione corrente, e l'operazione di RESTORE che ripristina la configurazione salvata. L'operazione di SAVE sovrascrive una eventuale configurazione salvata in precedenza. Fare il RESTORE in assenza di configurazione salvata non ha effetto: si mantiene inalterata la configurazione corrente.

SINGOLO/SEMI-INF/ MULTI-TAPE

   (a) Dai una definizione formale della funzione di transizione di una SRTM.

   (b) Dimostra che le SRTM riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

$T \times \ldots \longmapsto \{L, R, SAVE, RESTORE\}$

1. **(12 punti)** Una macchina di Turing con "save e restore" (SRTM) è una macchina di Turing deterministica a singolo nastro, che può salvare la configurazione corrente per poi ripristinarla in un momento successivo. Oltre alle normali operazioni di spostamento a sinistra e a destra, una SRTM può effettuare l'operazione di SAVE, che salva la configurazione corrente, e l'operazione di RESTORE che ripristina la configurazione salvata. L'operazione di SAVE sovrascrive una eventuale configurazione salvata in precedenza. Fare il RESTORE in assenza di configurazione salvata non ha effetto: si mantiene inalterata la configurazione corrente.

   (a) Dai una definizione formale della funzione di transizione di una SRTM.

   (b) Dimostra che le SRTM riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

{L, R}

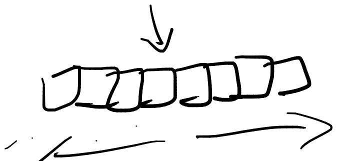
{L, R, SAVE, RESTORE}

↓ SINGOLO ↑ OVVIO

(b) SINGOLO -> L, R -> S = SIMULAZIONE

S = SU INPUT "w":

- LEFT, $\delta(a,b) = (r,b,L)$



- RIGHT

COPIO L'INPUT SUL NASTRO E MI SPOSTO OPPORTUNAMENTE A SX E DX SEGUENDO δ

FUNZ. DI TRANSIZIONE

- SAVE

QUANDO SI SALVA, LA TESTINA SI SPOSTA A DX DI UNA CELLA SCRIVENDO SUL NASTRO T UN (*)

REPLACE

NEW ASTERISK = CHECKPOINT

SHIFTARE TUTTO L'INPUT A DX DI UNA POSIZIONE E INSERISCO UN NUOVO * PER NUOVO "SALTO"

_ POSIZIONE

→ SE NO "*" ⇒ TESTINA RIMANE NELLA SUA POSIZIONE

→ SE SI "*"

→ SE RISCRIVO TUTTO L'INPUT FINO ALL'ASTERISCO E SOVRASCRIVO (PIÙ GENERALE / SEMI-INFINITO)

ALGORITMO USO I MARCATORI (*)

→ SE ACCETTA, ACCETTA

∪ UNIONE

2. (12 punti) Considera il seguente problema: dato un DFA D e un'espressione regolare R, il linguaggio riconosciuto da D è uguale al linguaggio generato da R?

(a) Formula questo problema come un linguaggio $EQ_{\text{DFA,REX}}$.

(b) Dimostra che $EQ_{\text{DFA,REX}}$ è decidibile. → ∃ ALGORITMO INT. FINITO?

↓

TM = MODELLO UTILE!

DFA
NFA
REX

$A_{DFA / NFA / REX}$

$E = EMPTY \rightarrow L(A) = \emptyset$

$EQ = \underbrace{A \cup B}_{\text{PROPRIETÀ}} = DFA \cup REX$

① $EQ_{DFA} = \{\langle M, w \rangle \mid M$ è UNA TM ~~tale che~~ w
è UN INPUT TALE CHE

$\{\langle D, R \rangle = |D| = |R|\}$

2. (12 punti) Considera il seguente problema: dato un DFA D e un'espressione regolare R, il linguaggio riconosciuto da D è uguale al linguaggio generato da R?

(a) Formula questo problema come un linguaggio $EQ_{\text{DFA,REX}}$.

(b) Dimostra che $EQ_{\text{DFA,REX}}$ è decidibile.

DFA = A_{DFA} = TM che riconosce DFA A

① COPIA TUTTO L'INPUT

② FINCHÉ CI SONO STATI CONTINUA A SCORRERE

③ SE FINISCE, ACCETTA

∪

LO STATO FINALE DI A_{DFA} È QUELLO INIZIALE DI REX

A_{REX} = TM che riconosce REX

① COPIA TUTTO L'INPUT

② FINCHÉ CI SONO STATI CONTINUA A SCORRERE

③ SE FINISCE, ACCETTA

$A_{REX} \cup A_{DFA} = \underline{EQ}$

(b) La seguente macchina N usa la Turing machine M che decide EQ_{DFA} per decidere $EQ_{\text{DFA,REX}}$:

N = "su input $\langle D, R \rangle$, dove D è un DFA e R una espressione regolare:

1. Converti R in un DFA equivalente D_R
2. Esegui M su input $\langle D, D_R \rangle$, e ritorna lo stesso risultato di M."

REX → NFA (1)
NFA → DFA (2)

STATO FINALE SSE ENTRAMBI TERMINANO...

Mostriamo che N è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Sappiamo che esiste un algoritmo per convertire ogni espressione regolare in un ε-NFA, ed un algoritmo per convertire ogni ε-NFA in un DFA. Il primo step di N si implementa eseguendo i due algoritmi in sequenza, e termina sempre perché entrambi gli algoritmi di conversione terminano. Il secondo step termina sempre perché sappiamo che EQ_{DFA} è un linguaggio decidibile. Quindi N termina sempre la computazione.

Vediamo ora che N dà la risposta corretta:

- Se $\langle D, R \rangle \in EQ_{\text{DFA,REX}}$ allora $L(D) = L(R)$, e di conseguenza $L(D) = L(D_R)$ perché D_R è un DFA equivalente ad R. Quindi $\langle D, D_R \rangle \in EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con accettazione. N ritorna lo stesso risultato di M, quindi accetta.
- Viceversa, se $\langle D, R \rangle \notin EQ_{\text{DFA,REX}}$ allora $L(D) \neq L(R)$, e di conseguenza $L(D) \neq L(D_R)$ perché D_R è un DFA equivalente ad R. Quindi $\langle D, D_R \rangle \notin EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con rifiuto. N ritorna lo stesso risultato di M, quindi rifiuta.

2. (9 punti) Dimostra che se B è **un linguaggio regolare**, allora il linguaggio

$$\text{PALINDROMIZE}(B) = \{ww^R \mid w \in B\}$$

è un linguaggio **context-free**.

$\{B \to \text{REGOLARE}\} \longrightarrow$ DFA/NFA/REX $\to \exists$ NFA CHE RICONOSCE B

$\downarrow$

$w \in L$

$(\text{PAL} \to ww^R \to \text{CF})$

$\uparrow$

$\exists$ G CHE RICONOSCE PAL IN CNF $\equiv \left\{ \begin{array}{l} A \to BC \\ A \to a \end{array} \right\}$

w

$\{w_1 \to w_2 \to \dots w_n\}$

SE TERMINA, $w \in L$ È REGOLARE.

10001 $\to \exists$ DFA $\to$ 10001

$\exists G'$, IN CNF CHE RICONOSCE w^R

$w^R = w$ — SÌ $(=)$ / NO $(\neq)$

$$\left[\left\{ \begin{array}{l} A \to aB \\ B \to C \\ C \to a \end{array} \right\} \; (w) \; = \; \left\{ \begin{array}{l} A \to Ca \\ C \to B \\ B \to a \end{array} \right\} \; (w^R) \right] \Rightarrow \text{PAL}$$

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

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