Argomenti trattati durante la lezione:

- Esercizi secondo compitino

(Commenti riassuntivi sulle singole categorie di esercizio:

- Varianti di Macchine di Turing

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}.$$

- Dimostra L decidibile

Usa una TM che descrive il problema oppure:

$A_{DFA} = \{\langle B, w\rangle \mid B \text{ è un DFA che accetta la stringa } w\}$

$A_{NFA} = \{\langle B, w\rangle \mid B \text{ è un } \varepsilon\text{-NFA che accetta la stringa } w\}$

$E_{DFA} = \{\langle A\rangle \mid A \text{ è un DFA e } L(A) = \emptyset\}$

$A_{REX} = \{\langle R, w\rangle \mid R \text{ è una espressione regolare che genera la stringa } w\}$

$A_{CFG} = \{\langle G, w\rangle \mid G \text{ è una CFG che genera la stringa } w\}$

$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ e } B \text{ sono DFA e } L(A) = L(B)\}$

$E_{CFG} = \{\langle G\rangle \mid A \text{ è una CFG ed } L(G) = \emptyset\}$

- Dimostra L indecidibile

Riduzione polinomiale usando:

$A_{TM} = \{\langle M, w\rangle \mid M \text{ è una TM che accetta la stringa } w\}$

$E_{TM} = \{\langle M\rangle \mid M \text{ è una TM tale che } L(M) = \emptyset\}$

$EQ_{TM} = \{\langle M_1, M_2 \mid M_1, M_2 \text{ TM tali che } L(M_1) = L(M_2)\}$

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ è una TM che si ferma su input } w\}$

- Dimostra L NP-Hard:

Riduzione polinomiale.

You need to find a polynomial time reduction from any known NP-hard problem to your target problem.

Let's call your problem "X"

From the pool of well known NP-hard problems you need to find a problem A and then a polynomial time reduction A<X

This means that:

1. You can take any instance of the known problem "A" and convert it to problem "X" and you only need polynomial time in the size of "A" to do it.
2. Solving problem X would then yield a solution to problem "A"

This proves "X" is NP-Hard because if you know how to solve "X" then you know how to solve "A" and since "A" is NP-Hard then automatically "X" must be NP-hard.

It's very very important to do reductions in the right order. The old known problem is always first and the target problem goes second.

Partiamo ora all'assalto con un po' di esercizi….

3. (12 punti) Considera le stringhe sull'alfabeto $\Sigma = \{1, 2, \ldots, 9\}$. Una stringa w di lunghezza n su Σ si dice *ordinata* se $w = w_1 w_2 \ldots w_n$ e tutti i caratteri $w_1, w_2, \ldots, w_n \in \Sigma$ sono tali che $w_1 \leq w_2 \leq \cdots \leq w_n$. Ad esempio, la stringa 1112778 è ordinata, ma le stringhe 5531 e 44427 non lo sono (la stringa vuota viene considerata ordinata). Diciamo che una Turing machine è *ossessionata dall'ordinamento* se ogni stringa che accetta è ordinata (ma non è necessario che accetti tutte queste stringhe). Considera il problema di determinare se una TM con alfabeto $\Sigma = \{1, 2, \ldots, 9\}$ è ossessionata dall'ordinamento.

(a) Formula questo problema come un linguaggio SO_{TM}.

(b) Dimostra che il linguaggio SO_{TM} è indecidibile.

Soluzione.

(a) $SO_{\text{TM}} = \{\langle M \rangle \mid M \text{ è una TM con alfabeto } \Sigma = \{1, 2, \ldots, 9\} \text{ che accetta solo parole ordinate}\}$

(b) La seguente macchina F calcola una riduzione $\overline{A_{TM}} \leq_m UA$:

F = "su input $\langle M, w \rangle$, dove M è una TM e w una stringa:

1. Costruisci la seguente macchina M':

   M' = "Su input x:
   1. Se $x = 111$, accetta.
   2. Se $x = 211$, esegue M su input w e ritorna lo stesso risultato di M.
   3. In tutti gli altri casi, rifiuta.

2. Ritorna $\langle M' \rangle$."

Mostriamo che F calcola una funzione di riduzione da $\overline{A_{TM}}$ a SO_{TM}, cioè una funzione tale che

$$\langle M, w \rangle \in \overline{A_{TM}} \text{ se e solo se } \langle M' \rangle \in SO_{\text{TM}}.$$

- Se $\langle M, w \rangle \in \overline{A_{TM}}$ allora la macchina M rifiuta o va in loop su w. In questo caso la macchina M' accetta la parola ordinata 111 e rifiuta tutte le altre, quindi è ossessionata dall'ordinamento e di conseguenza $\langle M' \rangle \in SO_{\text{TM}}$.
- Viceversa, se $\langle M, w \rangle \notin \overline{A_{TM}}$ allora la macchina M accetta w. Di conseguenza, la macchina M' accetta sia la parola ordinata 111 che la parola non ordinata 211, quindi non è ossessionata dall'ordinamento. Di conseguenza $\langle M' \rangle \notin SO_{\text{TM}}$.

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.

Soluzione.

(a) $EQ_{\text{DFA,REX}} = \{\langle D, R \rangle \mid D \text{ è un DFA, } R \text{ è una espressione regolare e } L(D) = L(R)\}$

(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."

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.

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.

2. **(12 punti)** Un linguaggio $B \subseteq \{0,1\}^*$ è *palindromo* se ogni stringa in B è palindroma, cioè se $w = w^R$ per ogni $w \in B$. Ad esempio, sia $\{00, 11011, 1001\}$ che $\emptyset$ sono linguaggi palindromi, mentre $\{00, 10\}$ non lo è. Considera il problema di determinare se il linguaggio di una TM M è palindromo.

   (a) Formula questo problema come un linguaggio PAL_{TM}.

   (b) Dimostra che il linguaggio PAL_{TM} è indecidibile.

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 $i = 1, \ldots, n$ fornisce un insieme $S_i \subseteq \{1, \ldots, m\}$ di ingredienti, una *fornitura valida* è un insieme T di fornitori che, nel loro insieme, forniscono tutti gli m ingredienti: $\bigcup_{i \in T} S_i = \{1, \ldots, m\}$. Definiamo il linguaggio

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

   (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 $VERTEX\text{-}COVER$ come problema NP-hard di riferimento.

Exercise 1: Save and Restore Turing Machines (SRTM)

(a) Formal definition of SRTM transition function

An SRTM is a single-tape deterministic Turing machine that extends the standard model with two additional operations: SAVE and RESTORE.

Formal Definition: An SRTM is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where:

- $Q, \Sigma, \Gamma, q_0, q_a, q_r$ have standard meanings
- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, SAVE, RESTORE\}$

The transition function δ now includes two special operations:

- **SAVE**: Stores the current configuration (state, tape contents, head position)
- **RESTORE**: Returns to the previously saved configuration, or no effect if no saved configuration exists

Configuration representation:

- Current configuration: (q, tape, head_position)
- Saved configuration: (q_s, $tape_s$, $head\_position_s$) or $\emptyset$

(b) Equivalence to standard Turing machines

Theorem: SRTMs recognize exactly the class of Turing-recognizable languages.

Proof sketch:

1. **SRTM ⊆ TM-recognizable:** Any SRTM can be simulated by a standard multi-tape TM using one tape to track the saved configuration.
2. **TM-recognizable ⊆ SRTM:** Any standard TM is trivially an SRTM (simply never use SAVE/RESTORE operations).

Implementation-level construction: Given SRTM M, construct equivalent 3-tape TM M':

- Tape 1: Working tape (simulates original tape)
- Tape 2: Saved state storage
- Tape 3: Saved tape contents

The simulation maintains the saved configuration on tapes 2-3 and implements SAVE/RESTORE by copying between tapes.

Exercise 2: Palindromic Language Problem

(a) Formulation as PAL_TM

Definition of PAL_TM:

```
PAL_TM = {⟨M⟩ | M is a TM and L(M) is palindromic}
```

where a language B is palindromic if ∀w ∈ B, w^R ∈ B (w^R is the reverse of w).

(b) Undecidability of PAL_TM

Theorem: PAL_TM is undecidable.

Proof by reduction from ATM: We reduce ATM = {⟨M,w⟩ | M accepts w} to PAL_TM.

Construction: Given ⟨M,w⟩, construct TM M' such that:

```
L(M') = {0^n1^n | n ≥ 0} ∪ {1^n0^n | n ≥ 0}  if M accepts w
L(M') = {0^n1^n | n ≥ 0}                      if M rejects w
```

Key insight:

- If M accepts w: L(M') is palindromic (contains both w and w^R for relevant strings)
- If M rejects w: L(M') is not palindromic (missing the reverses 1^n0^n)

Since this reduction is computable and ATM is undecidable, PAL_TM is undecidable.

Exercise 3: SUPPLY Problem

(a) SUPPLY ∈ NP

Verification algorithm: Given instance $(S_1,...,S_n,k)$ and certificate $T \subseteq \{1,...,n\}$ with $|T| = k$:

1. Verify $|T| = k$
2. For each ingredient $j \in \{1,...,m\}$, verify $\exists i \in T$ such that $j \in S_i$
3. Accept if all ingredients are covered

Complexity: O(mk) time, polynomial in input size.

Since SUPPLY has a polynomial-time verifier, SUPPLY ∈ NP.

(b) NP-hardness via VERTEX-COVER reduction

Theorem: SUPPLY is NP-complete.

Reduction: VERTEX-COVER $\leq_p$ SUPPLY

Given VERTEX-COVER instance (G=(V,E), k):

Construction:

- Create supplier S_v for each vertex $v \in V$
- Create ingredient e_i ∈for each edge (i,j) E
- Set $S_v = \{e_{ij} \mid v \in \{i,j\}\}$ (supplier v provides ingredients for incident edges)
- Use the same bound k

Correctness:

- Vertex cover of size k exists ⟺ Can cover all edges with k vertices
- ⟺ Can cover all ingredients (edges) with k suppliers (vertices)
- ⟺ SUPPLY instance has solution

Conclusion: Since VERTEX-COVER is NP-complete and SUPPLY ∈ NP, SUPPLY is NP-complete.

3. (12 punti) Una 3-colorazione di un grafo non orientato G è una funzione che assegna a ciascun vertice di G un "colore" preso dall'insieme $\{1, 2, 3\}$, in modo tale che per qualsiasi arco $\{u, v\}$ i colori associati ai vertici u e v sono diversi. Una 3-colorazione è *sbilanciata* se esiste un colore che colora più di metà dei vertici del grafo.

UNBALANCED-3-COLOR è il problema di trovare una 3-colorazione sbilanciata:

UNBALANCED-3-COLOR = $\{\langle G \rangle \mid G$ è un grafo che ammette una 3-colorazione sbilanciata$\}$

(a) Dimostra che UNBALANCED-3-COLOR è un problema NP
(b) Dimostra che UNBALANCED-3-COLOR è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

Soluzione.

(a) Il certificato è un vettore c dove ogni elemento $c[i]$ è il colore assegnato al vertice i. Il seguente algoritmo è un verificatore V per UNBALANCED-3-COLOR:

V = "Su input $\langle \langle G \rangle, c \rangle$:

1. Controlla se c è un vettore di n elementi, dove n è il numero di vertici di G.
2. Controlla se $c[i] \in \{1, 2, 3\}$ per ogni i.
3. Controlla se per ogni arco (i, j) di G, $c[i] \neq c[j]$.
4. Controlla se esiste un colore che compare in più di metà degli elementi di c.
5. Se tutte le condizioni sono vere, *accetta*, altrimenti *rifiuta*."

Ognuno dei passi dell'algoritmo si può eseguire in tempo polinomiale.

(b) Dimostriamo che UNBALANCED-3-COLOR è NP-Hard per riduzione polinomiale da 3-COLOR a UNBALANCED-3-COLOR. La funzione di riduzione polinomiale f prende in input un grafo $\langle G \rangle$ e produce come output un grafo $\langle G' \rangle$. Se n è il numero di vertici di G, G' è costruito aggiungendo $n + 1$ vertici isolati a G. Un vertice è isolato se non ci sono archi che lo collegano ad altri vertici.
Dimostriamo che la riduzione è corretta:

- Se $\langle G \rangle \in$ 3-COLOR, allora esiste un modo per colorare G con tre colori $1, 2, 3$. Sia n il numero di vertici di G. Posso costruire una colorazione sbilanciata per il grafo G' come segue:
  - i colori dei vertici di G' che appartengono anche a G sono colorati come in G;
  - i vertici isolati sono colorati tutti con il colore 1.

  In questo modo il colore 1 è assegnato ad almeno metà dei vertici di G' e la colorazione è sbilanciata.
- Se $\langle G' \rangle \in$ UNBALANCED-3-COLOR, allora esiste una colorazione sbilanciata di G'. Se elimino da G' i vertici isolati aggiunti dalla riduzione ottengo una 3-colorazione di G.

La funzione di riduzione deve contare i vertici del grafo G e aggiungere $n + 1$ vertici al grafo, operazioni che si possono fare in tempo polinomiale.

3. (9 punti) Chiamiamo k-PDA un automa a pila dotato di k pile. In particolare, uno 0-PDA è un NFA e un 1-PDA è un PDA convenzionale. Sappiamo già che gli 1-PDA sono più potenti degli 0-PDA (nel senso che riconoscono una classe più ampia di linguaggi). Mostra che i 2-PDA sono equivalenti alle Turing Machine.

Equivalenza 2-PDA e Turing Machine

Teorema: I 2-PDA sono equivalenti alle Turing Machine in termini di potenza computazionale.

Dimostrazione:

Direzione 1: TM → 2-PDA

Ogni Turing Machine può essere simulata da un 2-PDA.

Costruzione: Data una TM M, costruiamo un 2-PDA M' che la simula:

- **Pila 1:** Simula la parte sinistra del nastro (dalla posizione corrente della testina verso sinistra)
- **Pila 2:** Simula la parte destra del nastro (dalla posizione corrente della testina verso destra)

Simulazione delle operazioni TM:

- **Lettura:** Il simbolo corrente è sempre in cima a una delle due pile
- **Scrittura:** Sostituiamo il simbolo in cima alla pila appropriata
- **Movimento L:** Spostiamo un simbolo dalla Pila 1 alla Pila 2
- **Movimento R:** Spostiamo un simbolo dalla Pila 2 alla Pila 1

Direzione 2: 2-PDA → TM

Ogni 2-PDA può essere simulato da una Turing Machine.

Costruzione: Dato un 2-PDA M, costruiamo una TM M' con nastro diviso in sezioni:

- **Sezione 1:** Contenuto della Pila 1 (con marcatore di fondo)
- **Sezione 2:** Contenuto della Pila 2 (con marcatore di fondo)
- **Sezione 3:** Input string e stato corrente

La TM simula ogni transizione del 2-PDA manipolando appropriatamente le sezioni del nastro.

Conclusione: Poiché entrambe le direzioni sono dimostrabili, 2-PDA ≡ TM.

2. (12 punti) I *grawlix* sono sequenze di simboli senza senso che sostituiscono le parolacce nei fumetti.

Un linguaggio è *volgare* se contiene almeno un grawlix. Considera il problema di determinare se il linguaggio di una TM è volgare.

(a) Formula questo problema come un linguaggio $GROSS_{TM}$.
(b) Dimostra che il linguaggio $GROSS_{TM}$ è indecidibile.

Problema GROSS_TM

(a) Formulazione del problema GROSS_TM

Definizione:

`GROSS_TM = {⟨M⟩ | M è una TM e L(M) contiene almeno un grawlix}`

dove un grawlix è una sequenza di simboli come #$@*! che sostituisce le parolacce nei fumetti.

Precisazione formale: Assumiamo che l'insieme dei grawlix G sia fissato e finito (es. G = {#$@!, $#@!, ...}). Allora:

`GROSS_TM = {⟨M⟩ | M è una TM e ∃g ∈ G tale che g ∈ L(M)}`

(b) Indecidibilità di GROSS_TM

Teorema: GROSS_TM è indecidibile.

Dimostrazione per riduzione da A_TM:

Riduciamo A_TM = {⟨M,w⟩ | M accetta w} a GROSS_TM.

Costruzione: Dato ⟨M,w⟩, costruiamo una TM M' tale che:

```
M'(x):
  1. Se x è un grawlix fissato g₀ ∈ G:
     - Simula M su input w
```

- Se M accetta w, accetta x
- Se M rigetta w, rigetta x

2. Altrimenti, rigetta x

Analisi:

- **Se M accetta w:** L(M') = $\{g_0\}$, quindi M' ∈ GROSS_TM (contiene un grawlix)
- **Se M non accetta w:** L(M') = $\emptyset$, quindi M' ∉ GROSS_TM (non contiene grawlix)

Correttezza della riduzione: $\langle M,w \rangle \in A\_TM \Leftrightarrow \langle M' \rangle \in GROSS\_TM$

Poiché la costruzione di M' è computabile e A_TM è indecidibile, GROSS_TM è indecidibile.

Conclusione: Il problema di determinare se il linguaggio di una Turing Machine contiene almeno un grawlix è algoritmicamente irresolubile.

1. (12 punti) Una macchina di Turing "ecologica" (ETM) è uguale a una normale macchina di Turing deterministica a singolo nastro, ma può leggere e scrivere su entrambi i lati di ogni cella del nastro: fronte e retro. La testina di una TM ecologica può spostarsi a sinistra (L), a destra (R) o passare all'altro lato del nastro (F).

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

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

Soluzione.

(a) $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, F\}$

(b) Per dimostrare che TM ecologiche riconoscono la classe dei linguaggi Turing-riconoscibili dobbiamo dimostrare due cose: che ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM, e che ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

La prima dimostrazione è banale: le TM deterministiche a singolo nastro sono un caso particolare di ETM che usano solamente il lato di fronte del nastro e non effettuano mai la mossa F per passare dall'altro lato del nastro. Di conseguenza, ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM.

Per dimostrare che ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile, mostriamo come convertire una macchina di Turing ecologica M in una TM deterministica a due nastri S equivalente. Il primo nastro di S rappresenta il lato di fronte del nastro di M, il secondo nastro rappresenta il lato di retro.

S = "Su input w:

1. S usa lo stato per memorizzare lo stato di M ed il lato dove si trova la testina di M. All'inizio il lato corrente è "fronte" e lo stato di M è quello iniziale.

2. Se il lato corrente è “fronte”: leggi il simbolo sotto la testina del primo nastro per stabilire la mossa da fare. Se il lato corrente è “retro”, leggi il simbolo sotto la testina del secondo nastro per stabilire la mossa da fare.
3. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, L)$ scrive b sul primo nastro se il lato corrente è “fronte”, e scrive b sul secondo nastro se il nastro corrente è “retro”. Poi sposta entrambe le testine di una cella a sinistra.
4. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, R)$ scrive b sul primo nastro se il lato corrente è “fronte”, e scrive b sul secondo nastro se il nastro corrente è “retro”. Poi sposta entrambe le testine di una cella a destra.
5. Per simulare una mossa del tipo $\delta(q, a) = (r, b, F)$ la TM S scrive b sul primo nastro se il lato corrente è “fronte”, poi cambia il valore del lato corrente in “retro”. Se il lato corrente è “retro”, scrive b sul secondo nastro, poi cambia il valore del lato corrente in “fronte”. Sposta entrambe le testine di una cella a destra e poi una cella a sinistra, in modo da ritornare nella cella di partenza.
6. r diventa il nuovo stato corrente della simulazione. Se r è lo stato di accettazione di M, allora S termina con accettazione. Se r è lo stato di rifiuto di M, allora S termina con rifiuto. Negli altri casi continua la simulazione dal punto 2.”

Per concludere, siccome sappiamo che le TM multinastro riconoscono i linguaggi Turing-riconoscibili, allora abbiamo dimostrato che ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

1. **(12 punti)** Una *Macchina di Turing con stack di nastri* possiede due azioni aggiuntive che modificano il suo nastro di lavoro, oltre alle normali operazioni di scrittura di singole celle e spostamento a destra o a sinistra della testina: può salvare l’intero nastro inserendolo in uno stack (operazione di Push) e può ripristinare l’intero nastro estraendolo dallo stack (operazione di Pop). Il ripristino del nastro riporta il contenuto di ogni cella al suo contenuto quando il nastro è stato salvato. Il salvataggio e il ripristino del nastro non modificano lo stato della macchina o la posizione della sua testina. Se la macchina tenta di “ripristinare” il nastro quando lo stack è vuoto, la macchina va nello stato di rifiuto. Se ci sono più copie del nastro salvate nello stack, la macchina ripristina l’ultima copia inserita nello stack, che viene quindi rimossa dallo stack.

   Mostra che qualsiasi macchina di Turing con stack di nastri può essere simulata da una macchina di Turing standard. *Suggerimento:* usa una macchina multinastro per la simulazione.

Esercizio 1: Macchina di Turing con Stack di Nastri

Teorema: Qualsiasi macchina di Turing con stack di nastri può essere simulata da una macchina di Turing standard.

Dimostrazione tramite simulazione multinastro:

Costruiamo una TM multinastro M' che simula la TM con stack M:

Configurazione nastri di M':

- **Nastro 1:** Nastro di lavoro corrente (simula il nastro attivo di M)
- **Nastro 2:** Stack pointer e metadati dello stack
- **Nastri 3,4,5,...:** Copie salvate dei nastri (rappresentazione dello stack)

Simulazione delle operazioni:

1. **Operazioni normali:** Vengono eseguite direttamente sul Nastro 1
2. **Push (salvataggio):**
   - Copia l'intero contenuto del Nastro 1 sul prossimo nastro libero
   - Aggiorna il stack pointer sul Nastro 2
3. **Pop (ripristino):**

- Se stack non vuoto: copia il contenuto del nastro in cima allo stack sul Nastro 1
- Aggiorna il stack pointer
- Se stack vuoto: stato di rifiuto

Complessità: La simulazione introduce un overhead polinomiale, mantenendo la decidibilità.

Conclusione: Per il teorema di equivalenza delle TM multinastro con quelle standard, ogni TM con stack di nastri è equivalente a una TM standard.

3. (12 punti) In una delle storie delle Mille e una notte, Alì Babà, mentre viaggiava con il suo asino, trovò la grotta in cui i 40 ladroni avevano nascosto il loro bottino. Come cittadino rispettoso della legge, denunciò il fatto alla polizia, ma solo dopo aver tenuto il più possibile per sé. Il problema è che c'è troppo bottino e l'asino non può portarlo tutto: c'è un limite M al peso che l'asino può trasportare. Supponiamo che ognuno degli N oggetti rubati abbia un prezzo $P[i]$ e un peso $W[i]$. Alì Babà può caricare sull'asino un numero sufficiente di oggetti in modo che il prezzo totale sia almeno L?

Formalmente, possiamo rappresentare il problema che Alì Babà deve risolvere con il linguaggio

$$ALIBABA = \left\{ \langle N, P, W, M, L \rangle \;\middle|\; \text{esiste } B \subseteq \{1, \ldots, N\} \text{ tale che } \sum_{j \in B} W[j] \leq M \text{ e } \sum_{j \in B} P[j] \geq L \right\}.$$

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

(b) Sappiamo che il linguaggio

$$SUBSET\text{-}SUM = \left\{ \langle S, t \rangle \;\middle|\; S \text{ insieme di naturali, ed esiste } S' \subseteq S \text{ tale che } \sum_{x \in S'} x = t \right\}$$

è NP-completo. Dimostra che *ALIBABA* è NP-hard, usando *SUBSET-SUM* come problema NP-hard di riferimento.

(a) ALIBABA ∈ NP

Algoritmo di verifica:

- **Input:** Istanza (N, P, W, M, L) e certificato B ⊆ {1,...,N}
- **Verifica:**
  1. Calcola ∑_{j∈B} W[j] e verifica ≤ M
  2. Calcola ∑_{j∈B} P[j] e verifica ≥ L
  3. Accetta se entrambe le condizioni sono soddisfatte

Complessità: O(|B|) = O(N), tempo polinomiale.

Quindi ALIBABA ∈ NP.

(b) NP-completezza tramite riduzione da SUBSET-SUM

Riduzione: SUBSET-SUM ≤_p ALIBABA

Costruzione: Data istanza SUBSET-SUM (S, t) dove S = $\{s_1, s_2, ..., s_n\}$ e target t:

- N = n (stesso numero di elementi)
- ∀i: P[i] = W[i] = s_i (prezzo = peso = valore dell'elemento)
- M = L = t (capacità = valore target)

Correttezza:

- Esiste S' ⊆ S con ∑_{x∈S'} x = t

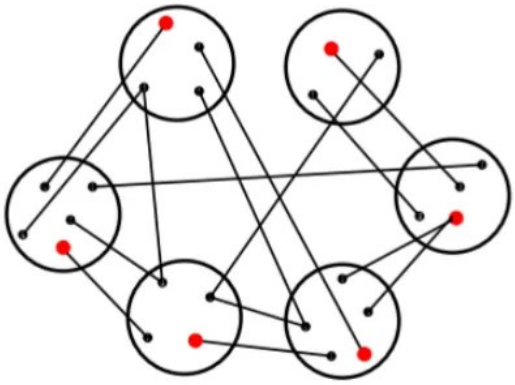
- ⟺ Esiste B ⊆ {1,...,n} con ∑_{j∈B} W[j] ≤ t e ∑_{j∈B} P[j] ≥ t
- ⟺ Esiste B con ∑_{j∈B} W[j] = ∑_{j∈B} P[j] = t (per la costruzione P[i] = W[i])

Conclusione: ALIBABA è NP-completo.

3. (12 punti) La Rettrice dell'Università di Padova vuole costituire una commissione selezionando un membro per ogni dipartimento dell'ateneo. Sappiamo che alcuni dei docenti si detestano a vicenda. Per evitare scontri, la Rettrice non vuole avere membri della commissione che si detestano tra di loro. Se ogni dipartimento è un insieme D_i di docenti, e se I è la relazione di inimicizia tra docenti, una *buona commissione* è un insieme C di docenti tali che:

- ogni dipartimento ha esattamente un rappresentante in commissione;
- non esistono coppie di docenti che si detestano.

La figura seguente mostra un esempio di istanza del problema dove i cerchi sono i dipartimenti, i punti sono i docenti e gli archi collegano docenti che si detestano. I docenti evidenziati in rosso sono i componenti di una buona commissione.



Definiamo il linguaggio

$$COMMITTEE = \{\langle D_1, \ldots, D_m, I\rangle \mid \text{esiste una buona commissione } C\}.$$

(a) Dimostra che *COMMITTEE* è un problema NP.
(b) Dimostra che *COMMITTEE* è NP-hard, usando *3SAT* come problema NP-hard di riferimento.

Esercizio 3: Problema COMMITTEE

(a) COMMITTEE ∈ NP

Algoritmo di verifica:

- **Input:** Istanza (D_1, ..., D_m, I) e certificato $C \subseteq \cup_i D_i$
- **Verifica:**
  1. ∀i ∈ {1,...,m}: verifica $|C \cap D_i| = 1$ (esattamente un rappresentante per dipartimento)
  2. ∀(d_1,d_2) ∈ I: verifica $\neg(d_1 \in C \wedge d_2 \in C)$ (nessuna coppia di nemici)

Complessità: O(m + |I|), tempo polinomiale.

Quindi COMMITTEE ∈ NP.

(b) NP-completezza tramite riduzione da 3SAT

Riduzione: 3SAT ≤_p COMMITTEE

Costruzione: Data formula 3SAT $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ con variabili x_1, ..., x_n:

Creazione dipartimenti:

- **Dipartimento variabile D□:** Per ogni variabile x_i, crea $\{x_i, \neg x_i\}$
- **Dipartimento clausola D'ⱼ:** Per ogni clausola $C_j = (l_1 \vee l_2 \vee l_3)$, crea $\{l_1, l_2, l_3\}$

Relazione di inimicizia I:

- $(x_i, \neg x_i)$ per ogni variabile x_i (literal opposti sono nemici)
- $(l, \neg l)$ per ogni literal l che appare in clausole diverse

Correttezza:

- **φ soddisfacibile → Esiste buona commissione:**
  Assegnamento soddisfacente → scegli il literal vero da ogni dipartimento variabile e un literal vero da ogni dipartimento clausola
- **Buona commissione → φ soddisfacibile:**
  Commissione valida → l'assegnamento indotto dai dipartimenti variabile soddisfa ogni clausola

Conclusione: COMMITTEE è NP-completo.

2. (12 punti) Considera il problema di determinare se i linguaggi di due DFA sono l'uno il complemento dell'altro.

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

(b) Dimostra che $COMPLEMENT_{\text{DFA}}$ è decidibile.

Soluzione.

(a) $COMPLEMENT_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ e } B \text{ sono DFA e } L(A) = \overline{L(B)}\}$

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

N = "su input $\langle A, B\rangle$, dove A e B sono DFA:

1. Costruisci l'automa $\overline{B}$ che riconosce il complementare del linguaggio di B
2. Esegui M su input $\langle A, \overline{B}\rangle$, e ritorna lo stesso risultato di M."

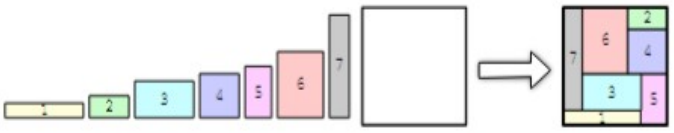
Mostriamo che N è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Sappiamo che esiste un algoritmo per costruire il complementare di un DFA (basta invertire stati finali e stati non finali nella definizione dell'automa). Di conseguenza, il primo step di N termina sempre. 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 $\langle A, B\rangle \in COMPLEMENT_{\text{DFA}}$ allora $L(A) = \overline{L(B)}$, e di conseguenza $L(A) = L(\overline{B})$ perché $\overline{B}$ è il complementare di B. Quindi $\langle A, \overline{B}\rangle \in EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con accettazione. N ritorna lo stesso risultato di M, quindi accetta.
- Viceversa, se $\langle A, B\rangle \notin COMPLEMENT_{\text{DFA}}$ allora $L(A) \neq \overline{L(B)}$, e di conseguenza $L(A) \neq L(\overline{B})$ perché $\overline{B}$ è il complementare di B. Quindi $\langle A, \overline{B}\rangle \notin EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con rifiuto. N ritorna lo stesso risultato di M, quindi rifiuta.

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.*

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

- i rettangoli piccoli hanno altezza 1 e base uguale ai numeri in S moltiplicati per 3: $(3s_1, 1), \ldots, (3s_n, 1)$;
- 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.