Argomenti trattati durante la lezione:

- Classi di complessità P ed NP. Tesi di Church computazionale
- Classi P/NP, Riduzioni + esercizi
- NP Completezza e problemi: SAT, CircuitSAT, £SAT, MaxIndSet, VertexCover

In this section we show that the phenomenon of undecidability is not confined to problems concerning automata. We give an example of an undecidable problem concerning simple manipulations of strings. It is called the ***Post Correspondence Problem***, or ***PCP***.

We can describe this problem easily as a type of puzzle. We begin with a collection of dominos, each containing two strings, one on each side. An individual domino looks like

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]$$

and a collection of dominos looks like

$$\left\{\left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right]\right\}.$$

The task is to make a list of these dominos (repetitions permitted) so that the string we get by reading off the symbols on the top is the same as the string of symbols on the bottom. This list is called a ***match***. For example, the following list is a match for this puzzle.

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{b}}{\mathtt{ca}}\right]\left[\frac{\mathtt{ca}}{\mathtt{a}}\right]\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{abc}}{\mathtt{c}}\right]$$

Reading off the top string we get abcaaabc, which is the same as reading off the bottom. We can also depict this match by deforming the dominos so that the corresponding symbols from top and bottom line up.

Dalle tessere al grafo

Definition (Grafo)

Un grafo (non orientato) G è una coppia (V, E) dove:

- $V = \{v_1, v_2, \ldots, v_n\}$ è un insieme finito e non vuoto di vertici;
- $E \subseteq \{\{u, v\} \mid u, v \in V\}$ è un insieme di coppie non ordinate, ognuna delle quali corrisponde ad un arco del grafo.

Grafo del domino

- Vertici: i numeri che si trovano sulle tessere
  - $V = \{\boxdot, \boxdot, \boxdot, \boxdot\}$
- Archi: le tessere del domino
  - $E = \{\boxdot, \boxdot, \boxdot, \boxdot, \boxdot\}$

Domino[1] è un problema su grafi!

- **Cammino Euleriano**: percorso in un grafo che attraversa tutti gli archi una sola volta

Il problema del Cammino Euleriano

$EULER = \{\langle G \rangle \mid G$ è un grafo che possiede un cammino Euleriano$\}$

- $EULER$ è un problema classico di teoria dei grafi
- Esistono algoritmi polinomiali per risolverlo

Algoritmo di Fleury

1. Scegliere un vertice con grado dispari (un vertice qualsiasi se tutti pari)
2. Scegliere un arco tale che sua cancellazione non sconnetta il grafo
3. Passare al vertice nell'altra estremità dell'arco scelto
4. Cancellare l'arco dal grafo
5. Ripetere i tre passi precedenti finche non eliminate tutti gli archi

Complessità

Su un grafo con n archi, l'algoritmo di Fleury impiega tempo $O(n^2)$

(Spiegazione in aula tramite riduzione) → D_1 <=_m EULER

Why have we been unsuccessful in finding polynomial time algorithms for these problems? We don't know the answer to this important question. Perhaps these problems have as yet undiscovered polynomial time algorithms that rest on unknown principles. Or possibly some of these problems simply cannot be solved in polynomial time. They may be intrinsically difficult.

A ***Hamiltonian path*** in a directed graph G is a directed path that goes through each node exactly once. We consider the problem of testing whether a directed graph contains a Hamiltonian path connecting two specified nodes, as shown in the following figure. Let

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}.$$

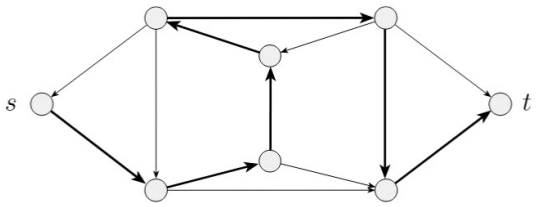


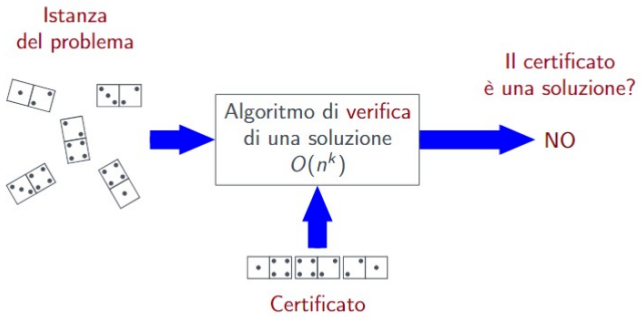
FIGURE **7.17**
A Hamiltonian path goes through every node exactly once

- Il problema del circuito Hamiltoniano è un problema classico di teoria dei grafi
- Un algoritmo polinomiale per risolverlo non è mai stato trovato
- Se qualcuno mi dà una possibile soluzione, è facile verificare se è corretta
  - I problemi per i quali esiste un algoritmo polinomiale vengono considerati trattabili
  - quelli che richiedono un algoritmo più che polinomiale sono detti intrattabili.
  - Sappiamo che ci sono problemi che non possono essere risolti da nessun algoritmo:
    - "Halting Problem" di Turing
  - Ci sono problemi che richiedono un tempo esponenziale:
    - il gioco della Torre di Hanoi

Stabilire con precisione qual'è il confine tra problemi trattabili ed intrattabili è piuttosto difficile

The *HAMPATH* problem has a feature called ***polynomial verifiability*** that is important for understanding its complexity. Even though we don't know of a fast (i.e., polynomial time) way to determine whether a graph contains a Hamiltonian path, if such a path were discovered somehow (perhaps using the exponential time algorithm), we could easily convince someone else of its existence simply by presenting it. In other words, *verifying* the existence of a Hamiltonian path may be much easier than *determining* its existence.

Domino[2] è in NP



Verificatori

Definition

Un verificatore per un linguaggio A è un algoritmo V tale che

$$A = \{w \mid V \text{ accetta } \langle w, c\rangle \text{ per qualche stringa } c\}$$

- il verificatore usa ulteriori informazioni per stabilire se w appartiene al linguaggio
- questa informazione è il certificato c

P vs NP

	P	NP
Facili da risolvere	✓	?
Facili da verificare	✓	✓
Esempi	Domino[1], Euler, ordinamento, ...	Domino[2], Hamilton, Sudoku, Protein folding, Crittografia, ...

- **P** è la classe dei linguaggi che possono essere decisi da una macchina di Turing deterministica che impiega tempo polinomiale.
- **NP** è la classe dei linguaggi che ammettono un verificatore che impiega tempo polinomiale.
- **Equivalente:** è la classe dei linguaggi che possono essere decisi da una macchina di Turing non deterministica che impiega tempo polinomiale.

Passiamo a degli esempi pratici...

$$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}.$$

For example, $\langle\{4, 11, 16, 21, 27\}, 25\rangle \in SUBSET\text{-}SUM$ because $4 + 21 = 25$. Note that $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_l\}$ are considered to be ***multisets*** and so allow repetition of elements.

THEOREM 7.25

SUBSET-SUM is in NP.

PROOF IDEA The subset is the certificate.

PROOF The following is a verifier V for *SUBSET-SUM*.

V = "On input $\langle\langle S, t\rangle, c\rangle$:

1. Test whether c is a collection of numbers that sum to t.
2. Test whether S contains all the numbers in c.
3. If both pass, *accept*; otherwise, *reject*."

ALTERNATIVE PROOF We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for *SUBSET-SUM* as follows.

N = "On input $\langle S, t\rangle$:

1. Nondeterministically select a subset c of the numbers in S.
2. Test whether c is a collection of numbers that sum to t.
3. If the test passes, *accept*; otherwise, *reject*."

A ***clique*** in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A ***k-clique*** is a clique that contains k nodes. Figure 7.23 illustrates a graph with a 5-clique.

The clique problem is to determine whether a graph contains a clique of a specified size. Let

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}.$$

THEOREM 7.24

CLIQUE is in NP.

PROOF IDEA The clique is the certificate.

PROOF The following is a verifier V for *CLIQUE*.

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

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, *accept*; otherwise, *reject*."

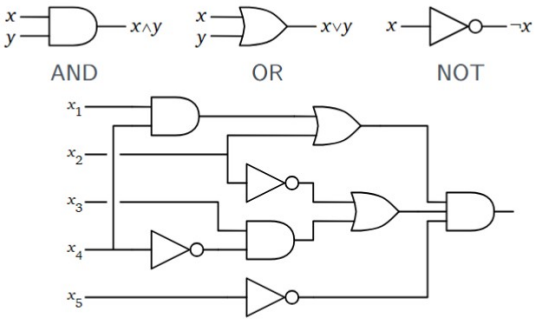
ALTERNATIVE PROOF If you prefer to think of NP in terms of nondeterministic polynomial time Turing machines, you may prove this theorem by giving one that decides *CLIQUE*. Observe the similarity between the two proofs.

N = "On input $\langle G, k\rangle$, where G is a graph:

1. Nondeterministically select a subset c of k nodes of G.
2. Test whether G contains all edges connecting nodes in c.
3. If yes, *accept*; otherwise, *reject*."

Prendiamo in considerazione un problema diverso…

Un problema NP



CircuitSAT: dato un circuito Booleano, esistono dei valori di input che permettono di ottenere output = 1?

The first NP-complete problem that we present is called the ***satisfiability problem***. Recall that variables that can take on the values TRUE and FALSE are called ***Boolean variables*** (see Section 0.2). Usually, we represent TRUE by 1 and FALSE by 0. The ***Boolean operations*** AND, OR, and NOT, represented by the symbols $\wedge$, $\vee$, and $\neg$, respectively, are described in the following list. We use the overbar as a shorthand for the $\neg$ symbol, so $\overline{x}$ means $\neg\, x$.

$0 \wedge 0 = 0$	$0 \vee 0 = 0$	$\overline{0} = 1$
$0 \wedge 1 = 0$	$0 \vee 1 = 1$	$\overline{1} = 0$
$1 \wedge 0 = 0$	$1 \vee 0 = 1$	
$1 \wedge 1 = 1$	$1 \vee 1 = 1$	

A ***Boolean formula*** is an expression involving Boolean variables and operations. For example,

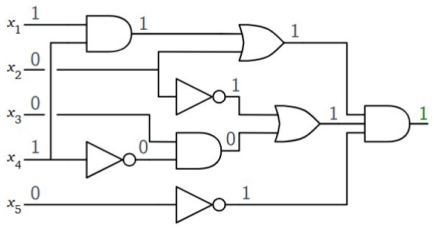
$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

is a Boolean formula. A Boolean formula is ***satisfiable*** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1. The preceding formula is satisfiable because the assignment $x = 0$, $y = 1$, and $z = 0$ makes ϕ evaluate to 1. We say the assignment *satisfies* ϕ. The ***satisfiability problem*** is to test whether a Boolean formula is satisfiable. Let

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}.$$

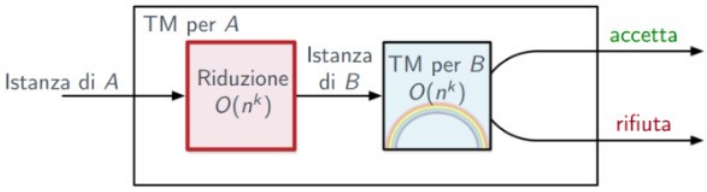
Verifica del certificato

Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



5 Calcola l'output dell'intero circuito: se = 1, SI, altrimenti NO

Se $A \leq_P B$, e $B \in P$, allora $A \in P$:



Language A is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

Ma anche…

DEFINITION 7.34

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

- Un problema è NP-hard se l'esistenza di un algoritmo polinomiale per risolverlo implica l'esistenza di un algoritmo polinomiale per ogni problema in NP.
- Se siamo in grado di risolvere un problema NP-hard in modo efficiente, allora possiamo risolvere in modo efficiente ogni problema di cui possiamo verificare facilmente una soluzione, usando la soluzione del problema NP-hard come sottoprocedura.
- Un problema è NP-completo se è sia NP-hard che appartenente alla classe NP (o "NP-easy").
  - Esempio: **CircuitSAT**!

Ogni dimostrazione di NP-completezza di compone di due parti:

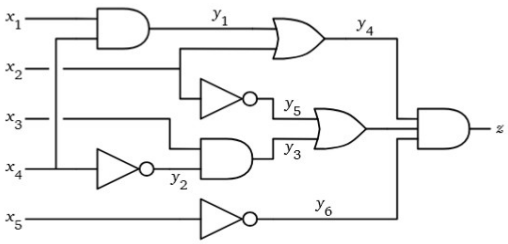
1. dimostrare che il problema appartiene alla classe NP;
2. dimostrare che il problema è NP-hard.

- Dimostrare che un problema è in NP vuol dire dimostrare l'esistenza di un verificatore polinomiale.
- Le tecniche che si usano per dimostrare che un problema è NP-hard sono fondamentalmente diverse.

Un esempio diretto…

Riduzione di CircuitSAT a SAT

1. Dare un nome agli output delle porte logiche:



2. Scrivere le espressioni booleane per ogni porta logica e metterle in and logico, aggiungendo "AND z" alla fine:

$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge (y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile se e solo se la formula risultante Φ è soddisfacibile.
Dimostriamo questa affermazione in due passaggi:

- $\Rightarrow$ Dato un insieme di input che rende vero il circuito K, possiamo ottenere i valori di verità per le variabili nella formula Φ calcolando l'output di ogni porta logica di K.
- $\Leftarrow$ Dati i valori di verità delle variabili nella formula Φ, possiamo ottenere gli input del circuito semplicemente ignorando le variabili delle porte logiche interne y_i e la variabile di uscita z.

L'intera trasformazione da circuito a formula può essere eseguita in tempo lineare. Inoltre, la dimensione della formula risultante cresce di un fattore costante rispetto a qualsiasi ragionevole rappresentazione del circuito.

Lanciamoci subito ad un esempio pratico:

Esercizio 1

Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Un *circuito quasi Hamiltoniano* in un grafo G è un ciclo che attraversa esattamente una volta tutti i vertici del grafo *tranne uno*. Il *problema del circuito quasi Hamiltoniano* è il problema di stabilire se un grafo contiene un circuito quasi Hamiltoniano.

- Dimostra che il problema del circuito quasi Hamiltoniano è in NP
- Dimostra che il problema del circuito quasi Hamiltoniano è NP-hard, usando il problema del circuito Hamiltoniano come problema di riferimento.

1.

Per mostrare che QHAM ∈ NP, basta osservare che una **certificazione** di positività è semplicemente la sequenza di vertici (v1,v2,…,vn−1,v1)(v_1,v_2,\dots,v_{n-1},v_1) che descrive il ciclo.

> **Verificatore (polinomiale).** Dato in input:
>
> 1. il grafo GG;
> 2. una lista C=[v1,v2,…,vn−1]C=[v_1,v_2,\dots,v_{n-1}] di n−1n-1 vertici distinti di VV.
>
> Il verificatore controlla in O(n)O(n) tempo se:
>
> - i vertici in CC sono tutti distinti;
> - {vi,vi+1}∈E\{v_i,v_{i+1}\}\in E per i=1,…,n−2i=1,\dots,n-2;
> - {vn−1,v1}∈E\{v_{n-1},v_1\}\in E.
>
> Se tutte le condizioni sono soddisfatte, CC descrive un ciclo che visita n−1n-1 vertici → **accetta**, altrimenti → **rifiuta**.

Tutti i controlli possono essere effettuati in tempo polinomiale in |V|+|E||V|+|E|. **Quindi QHAM ∈ NP.**

2.

Per ridurre HAM a QHAM costruiamo una trasformazione polinomiale $f : G \mapsto G'$ tale che

$$G \in \text{HAM} \iff G' \in \text{QHAM}.$$

Costruzione di G'

- Partiamo da $G = (V, E)$, con $|V| = n$.
- Aggiungiamo un **nuovo vertice** v^* di **grado 1**, connesso a un qualunque vertice $u \in V$:

$$V' = V \cup \{v^*\}, \quad E' = E \cup \{\{v^*, u\}\}.$$

- Chiaramente $|V'| = n + 1$.

Correttezza della riduzione

1. (⇒) Se G ha un circuito Hamiltoniano, cioè un ciclo H di lunghezza n che visita tutti i vertici di V, allora H è un ciclo in G' che **non usa** v^* (poiché v^* ha grado 1 e non può far parte di un ciclo). Quindi H visita esattamente $n = |V'| - 1$ vertici di V', cioè è un **circuito quasi-Hamiltoniano** in G'.
2. (⇐) Se G' ha un circuito quasi-Hamiltoniano Q, esso deve visitare esattamente $(n+1) - 1 = n$ vertici di V'. Poiché v^* ha grado 1, non può essere incluso in alcun ciclo; pertanto Q è interamente contenuto in G e visita tutti i n vertici di V. Ne segue che Q è un circuito Hamiltoniano di G.

La costruzione di G' richiede tempo $O(n + 1 + |E| + 1)$, cioè polinomiale.

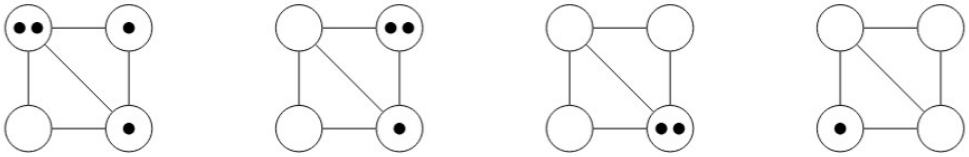
Conclusione. Abbiamo mostrato una riduzione polinomiale da HAM a QHAM:

$$G \text{ in HAM} \quad \Longleftrightarrow \quad f(G) = G' \text{ in QHAM}.$$

Poiché HAM è NP-completo, ne consegue che QHAM è NP-hard.

Esercizio 4

Pebbling è un solitario giocato su un grafo non orientato G, in cui ogni vertice ha zero o più ciottoli. Una mossa del gioco consiste nel rimuovere due ciottoli da un vertice v e aggiungere un ciottolo ad un vertice u adiacente a v (il vertice v deve avere almeno due ciottoli all'inizio della mossa). Il problema PebbleDestruction chiede, dato un grafo $G = (V, E)$ ed un numero di ciottoli $p(v)$ per ogni vertice v, di determinare se esiste una sequenza di mosse che rimuove tutti i sassolini tranne uno.



Una soluzione in 3 mosse di PebbleDestruction.

Dimostra che PebbleDestruction è NP-hard usando il problema del Circuito Hamiltoniano come problema NP-hard noto (un circuito Hamiltoniano è un ciclo che attraversa ogni vertice di G esattamente una volta).

Costruzione della riduzione

Dato un grafo G = (V, E) istanza di HC, costruiamo un'istanza di PEBBLEDESTRUCTION come segue:

1. **Grafo**: Utilizziamo lo stesso grafo G = (V, E)
2. **Assegnazione ciottoli**: Per ogni vertice v ∈ V, poniamo p(v) = deg(v), dove deg(v) è il grado del vertice v

Correttezza della riduzione

Lemma 1: Se G ha un ciclo hamiltoniano, allora l'istanza di PEBBLEDESTRUCTION ammette soluzione.

Dimostrazione: Sia C = (v_1, v_2, ..., v_n, v_1) un ciclo hamiltoniano. Definiamo la seguente sequenza di mosse:

- Per i = 1, 2, ..., n-1: rimuoviamo 2 ciottoli da v_i e aggiungiamo 1 ciottolo a v_{i+1}
- Rimuoviamo 2 ciottoli da v_n e aggiungiamo 1 ciottolo a v_1

Osserviamo che:

- Ogni vertice v_i ha inizialmente deg(v_i) ciottoli
- Nel ciclo hamiltoniano, ogni vertice ha esattamente 2 archi incidenti
- Dopo n mosse, ogni vertice perde 2 ciottoli e ne guadagna 1, risultando in deg(v) - 2 + 1 = deg(v) - 1 ciottoli
- Il numero totale di ciottoli si riduce da Σ deg(v) = 2|E| a |E|
- Continuando questo processo, possiamo ridurre i ciottoli fino ad averne uno solo.

Lemma 2: Se l'istanza di PEBBLEDESTRUCTION ammette soluzione, allora G ha un ciclo hamiltoniano.

Dimostrazione: Supponiamo che esista una sequenza di mosse che riduce tutti i ciottoli a uno solo.

Consideriamo il flusso di ciottoli: ogni mossa trasferisce effettivamente 1 ciottolo da un vertice a un vertice adiacente (rimuovendo 2 e aggiungendo 1). Per raggiungere la configurazione finale con un solo ciottolo, il flusso netto deve essere tale che ogni vertice eccetto uno perda tutti i suoi ciottoli.

La struttura ottimale per questo trasferimento richiede che ogni vertice sia attraversato da un percorso che lo connetta al resto del grafo in modo efficiente. Poiché ogni vertice inizia con un numero di ciottoli pari al suo grado, e deve perdere tutti i ciottoli tranne eventualmente uno, la topologia del flusso deve necessariamente formare un ciclo che attraversi tutti i vertici, cioè un ciclo hamiltoniano.

Commentiamo anche…

Altri esempi di dualità

Problemi NP-Completi

- circuito Hamiltoniano
  Trovare un ciclo che visita ogni vertice esattamente una volta
- Copertura di vertici
  Trovare il minimo sottoinsieme S di vertici tali che ogni arco ha almeno un'estremità in S
- 3-colorazione di un grafo
  Trovare un modo per colorare i vertici di un grafo con tre colori tale che vertici adiacenti sono di colore diverso

Problemi in P

- Circuito Euleriano
  Trovare un ciclo che visita ogni arco esattamente una volta
- Copertura di archi
  Trovare il minimo sottoinsieme M di archi tali che ogni vertice è adiacente ad un arco in M
- 2-colorazione di un grafo
  Trovare un modo per colorare i vertici di un grafo con due colori tale che vertici adiacenti sono di colore diverso

Come scegliere il problema giusto

- Se il problema richiede di assegnare bit agli oggetti: **SAT**
- Se il problema richiede di assegnare etichette agli oggetti prese un piccolo insieme, o di partizionare gli oggetti in un numero costante di sottoinsiemi: **3Color** o ***k*Color**
- Se il problema richiede di organizzare un insieme di oggetti in un ordine particolare: **Circuito Hamiltoninano**
- Se il problema richiede di trovare un piccolo sottoinsieme che soddisfi alcuni vincoli: **MinVertexCover**
- Se il problema richiede di trovare un sottoinsieme grande che soddisfi alcuni vincoli: **MaxIndependentSet**
- Se il numero 3 appare in modo naturale nel problema, provare **3SAT** o **3Color** (No, questo non è uno scherzo.)
- Se tutto il resto fallisce, prova **3SAT** o anche **SAT**!

Passiamo ora ad altri esercizi…

5.4
If A $\leq_m$ B and B is a regular language, does that imply that A is a regular language? Why or why not?

No, that does not imply that A is regular.
For example, $\{a^n b^n \mid n \geq 0\} \leq_m \{a^n \mid n \geq 0\}$.
The reduction first tests whether its input is a member of $\{a^n b^n \mid n \geq 0\}$. If so, it outputs the string a, and if not it outputs the string b.

5.12 Let S=$\{\langle M\rangle \mid$ M is a TM that accepts w^R whenever it accepts $w\}$.
Show that S is undecidable.

We show that $A_{TM} \leq_m S$ by mapping $\langle M, w\rangle$ to $\langle M'\rangle$ where M' is the following TM:
On input x:
1. If $x = 01$ then accept.
2. If $x \neq 10$ then reject.
3. If $x = 10$, then simulate M on w. If M accepts w then accept. If M halts and rejects w, then reject.

If $\langle M, w\rangle \in A_{TM}$, then M accepts w, and L(M') = $\{01,10\}$, so $\langle M'\rangle \in S$.
If $\langle M, w\rangle \notin A_{TM}$, then M rejects w, and L(M') = $\{01\}$, so $\langle M'\rangle \notin S$.
Therefore, $\langle M, w\rangle \in A_{TM} \Leftrightarrow \langle M'\rangle \in S$.
Since A_{TM} is undecidable, so is S.

Let $INFINITE_{PDA} = \{\langle M\rangle \mid$M is a PDA and L(M) is an infinite language$\}$

Want to show that $INFINITE_{PDA}$ is decidable.
$\Rightarrow$ Construct a halting TM M that decides INFINITE$_{PDA}$

Note: Given the PDA M, we can convert it into an equivalent CFG.
We can then convert the CFG into an equivalent CFG G' in Chomsky Normal Form.
We can check CFG G' if there exists a derivation $A \Rightarrow uAv$ where $u, v \in \Sigma^*$

If $A \Rightarrow uAv$ is a derivation in CFG G' then L(M) is an infinite language.
If $A \Rightarrow uAv$ is not a derivation in CFG G' then L(M) is not an infinite language.

The following TM M decides $INFINITE_{PDA}$:

M = "On input $\langle M\rangle$ where M is a PDA:
1. Convert M into an equivalent CFG G
2. Convert G into an equivalent CFG G' in Chomsky Normal Form.
3. If G' includes the derivation A $\Rightarrow uAv$, accept. Else, reject."

$\Rightarrow$INFINITE$_{PDA}$ is decidable

3. Say that string x is a *prefix* of string y if a string z exists where $xz = y$, and say that x is a *proper prefix* of y if in addition $x \neq y$. A language is *prefix-free* if it doesn't contain a proper prefix of any of its members. Let

$$\mathsf{PrefixFree}_{\mathsf{REX}} = \{R | R \text{ is a regular expression where } L(R) \text{ is prefix-free}\}$$

Show that $\mathsf{PrefixFree}_{\mathsf{REX}}$ is decidable. **[15 points]**

SOLUTION:We construct a TM that decides $\mathsf{PrefixFree}_{\mathsf{REX}}$ as follows[1]. On input R, reject if R is not a valid regular expression. Otherwise, construct a DFA D for the language $L(R)$ (refer to chapter 1 of Sipser for the algorithm that constructs an equivalent NFA for $L(R)$ from R, and for the algorithm that converts an NFA to a DFA). By running a DFS starting from q_0, we can remove all states that are not reachable from q_0 from the automaton.

Finally, for each accept state q, we run a DFS starting from q and check if another accept state (not equal to q) is reachable from q, or if there is a loop from q to itself. If any such paths or loops are found, reject. Otherwise, accept. Note that it is first required to remove all the states (actually, just accepting states) not reachable from q_0 as these states cannot lead to any string being in the language.

2. (**Eco-friendly TM**) An *eco-friendly* Turing machine (ETM) is the same as an ordinary (deterministic) one-tape Turing machine, but it can read and write on both sides of each tape square: front and back.

   At the end of each computation step, the head of the eco-friendly TM can move left (L), move right (R), or flip to the other side of the tape (F).

   (a) Give a formal definition of the syntax of the transition function of an eco-friendly TM. (Modify Part 4 of Definition 3.3 on page 168 of the textbook.)

   (b) Show that eco-friendly TMs recognize the class of Turing-recognizable languages. That is, use a simulation argument to show that they have exactly the same power as ordinary TMs.

(a) Definizione formale della funzione di transizione

Una funzione di transizione per una ETM può essere definita come: $\delta: Q \times \Gamma \times \{\text{front, back}\} \rightarrow Q \times \Gamma \times \{L, R, F\}$

dove:

- Q è l'insieme degli stati
- Γ è l'alfabeto del nastro
- {front, back} indica il lato del quadrato del nastro
- {L, R, F} rappresenta le possibili direzioni: sinistra, destra o flip

(b) Dimostrazione della potenza equivalente

Per dimostrare che le ETM riconoscono esattamente le stesse lingue delle TM ordinarie:

1. Una ETM può simulare una TM ordinaria ignorando il lato posteriore dei quadrati del nastro.
2. Una TM ordinaria può simulare una ETM codificando il contenuto di entrambi i lati di ciascun quadrato. Per ogni simbolo Γ, definiamo una coppia (γ_front, γ_back) sull'alfabeto $\Gamma \times \Gamma$. L'operazione "flip" viene simulata con uno spostamento zero e scambio della rappresentazione interna.

2. (**Enthusiastic TM**) Consider the problem of determining whether a given TM ever[1] writes "332" on three adjacent squares of its tape. You may assume that the input alphabet of this TM is $\{0, 1\}$ and the tape alphabet is $\{0, 1, 2, \ldots, 9\}$.

   (a) Formulate this problem as a language $ENTHUSIASTIC_{\text{TM}}$.

   (b) Show $ENTHUSIASTIC_{\text{TM}}$ is undecidable.

(a) Formulazione del linguaggio ENTHUSIASTIC_TM

ENTHUSIASTIC_TM = {⟨M⟩ | M è una TM che nel corso della sua computazione scrive "332" su tre quadrati adiacenti del nastro}

dove ⟨M⟩ è la codifica della macchina di Turing M.

(b) Dimostrazione dell'indecidibilità

Per dimostrare che ENTHUSIASTIC_TM è indecidibile, riduciamo il problema dell'arresto (HALT_TM) a ENTHUSIASTIC_TM.

Data una TM M e un input w, costruiamo una TM M' che:

1. Simula M su w
2. Se M si ferma su w, M' scrive "332" in tre celle adiacenti

5.24 Let $J = \{w |$ either $w = 0x$ for some $x \in A_{\text{TM}}$, or $w = 1y$ for some $y \in \overline{A_{\text{TM}}}\}$. Show that neither J nor $\overline{J}$ is Turing-recognizable.

Quindi, M si ferma su w se e solo se M' appartiene a ENTHUSIASTIC_TM. Poiché HALT_TM è indecidibile, anche ENTHUSIASTIC_TM deve essere indecidibile.

Sia J = {w| w = 0x per qualche x ∈ A_TM, oppure w = 1y per qualche y ∈ Ā_TM}.

Per dimostrare che né J né J̄ sono Turing-riconoscibili:

1. Supponiamo che J sia Turing-riconoscibile. Allora potremmo decidere A_TM nel seguente modo:
   - Per ogni stringa z, calcoliamo 0z
   - Se 0z ∈ J, allora z ∈ A_TM
   - Se 1z ∈ J, allora z ∉ A_TM Questo è impossibile perché A_TM è indecidibile.
2. Supponiamo che J̄ sia Turing-riconoscibile. Allora:
   - Per ogni stringa z, calcoliamo 0z
   - Se 0z ∈ J̄, allora z ∉ A_TM
   - Se 1z ∈ J̄, allora z ∈ A_TM Nuovamente, questo permetterebbe di decidere A_TM, che è impossibile.

Quindi, né J né il suo complemento sono Turing-riconoscibili.

4. (9 punti) Una Turing Machine *somma correttamente* se, dati in input due numeri binari separati da #, termina la computazione con la loro somma (in binario) sul nastro. (Non importa cosa fa sugli altri input.) Considera il problema di determinare se una TM somma correttamente.

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

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

(a) Formalizzazione di SUM_{tm}

Sia $\Sigma = \{0, 1, \#\}$. Denotiamo con

$$\texttt{bin}(n) \;=\; \text{stringa binaria di } n \geq 0, \text{ senza zeri iniziali (eccetto “0”).}$$

Definiamo il linguaggio

$$\text{SUM}_{\text{TM}} = \big\{ \langle M \rangle \;\big|\; M \text{ è una TM totale su } \Sigma^* \text{ tale che } \forall u, v \in \{0,1\}^* : \\ M, \text{ partendo da } u\#v\,, \text{ termina in uno stato accettazione con sul nastro esattamente } \texttt{bin}\big(\text{val}_2(u) + \text{val}_2(v)\big) \big\},$$

dove val_2 interpreta la stringa binaria come numero naturale.

In altre parole,

$$\langle M \rangle \in \text{SUM}_{\text{TM}} \iff M \begin{cases} \text{per ogni input of the form } u\#v \text{ (con } u, v \in \{0,1\}^*), \\ \text{termina accettando e lascia sul nastro } \texttt{bin}(\text{val}_2(u) + \text{val}_2(v)); \\ \text{su altri input non importa.} \end{cases}$$

Richiamiamo

$$A_{\text{TM}} = \big\{ \langle N, w \rangle \;\big|\; \text{la TM } N \text{ accetta } w \big\},$$

noto indecidibile. Costruiamo in tempo polinomiale una TM M' da $\langle N, w \rangle$ in modo che

$$\langle N, w \rangle \in A_{\text{TM}} \iff \langle M' \rangle \in \text{SUM}_{\text{TM}}.$$

Costruzione di M':

Su input qualunque $x \in \{0, 1, \#\}^*$:

1. **Parse**: se x non ha la forma $u\#v$, **loop** (non importa).
2. **Simula** N su w.
   - Se N **non** accetta w, **loop**.
   - Se N **accetta** w, calcola “in colonna” la somma binaria di u e v, scrive il risultato in forma $\texttt{bin}\big(\text{val}_2(u) + \text{val}_2(v)\big)$ e **accetta**.

- Se N accetta w, allora M' su **ogni** $u\#v$ termina con la somma corretta $\Rightarrow \langle M' \rangle \in \text{SUM}_{\text{TM}}$.
- Se N non accetta w, allora M' **non** è totale sui $\{u\#v\}$ (loopa sempre) $\Rightarrow \langle M' \rangle \notin \text{SUM}_{\text{TM}}$.

La trasformazione $\langle N, w \rangle \mapsto \langle M' \rangle$ è chiaramente polinomiale.

Quindi $A_{\text{TM}} \leq_m \text{SUM}_{\text{TM}}$ e **SUM_{tm} è indecidibile.** ■

3. (12 punti) Data una Turing Machine M, considera il problema di determinare se esiste un input tale che M scrive "$xyzzy$" su cinque celle adiacenti del nastro. Puoi assumere che l'alfabeto di input di M non contenga i simboli x, y, z.

(a) Formula questo problema come un linguaggio $MAGIC_{\text{TM}}$.

(b) Dimostra che il linguaggio $MAGIC_{\text{TM}}$ è indecidibile.

Soluzione.

(a) $MAGIC_{\text{TM}} = \{\langle M \rangle \mid M \text{ è una TM che scrive } xyzzy \text{ sul nastro per qualche input } w\}$

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

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

1. Verifica che i simboli x, y, z non compaiano in w, né nell'alfabeto di input o nell'alfabeto del nastro di M. Se vi compaiono, sostituiscili con tre nuovi simboli X, Y, Z nella parola w e nella codifica di M.
2. Costruisci la seguente macchina M':

   M' = "Su input x:
   1. Simula l'esecuzione di M su input w, senza usare i simboli x, y, z.
   2. Se M accetta, scrivi $xyzzy$ sul nastro, altrimenti rifiuta senza modificare il nastro.
3. Ritorna $\langle M' \rangle$."

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

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

- Se $\langle M, w \rangle \in A_{\text{TM}}$ allora la macchina M accetta w. In questo caso la macchina M' scrive $xyzzy$ sul nastro per tutti gli input. Di conseguenza $\langle M' \rangle \in MAGIC_{\text{TM}}$.
- Viceversa, se $\langle M, w \rangle \notin A_{TM}$ allora la macchina M rifiuta o va in loop su w. Per tutti gli input, la macchina M' simula l'esecuzione di M su w senza usare i simboli x, y, z (perché sono stati tolti dalla definizione di M e di w se vi comparivano), e rifiuta o va in loop senza scrivere mai $xyzzy$ sul nastro. Di conseguenza $\langle M' \rangle \notin MAGIC_{\text{TM}}$.