

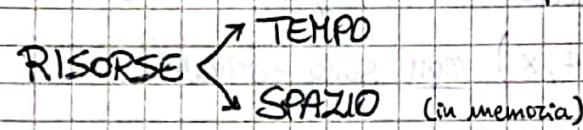
Secondo teorema di incompletezza di Gödel:

Nessun sistema formale sufficientemente potente che sia consistente può dimostrare la sua stessa consistenza.

23/05/2020

COMPLESSITÀ COMPUTAZIONALE

In questa parte del corso vedremo quante risorse servono per eseguire un algoritmo che risolve un problema in un tempo finito.



Ci concentreremo innanzitutto sulla complessità temporale degli algoritmi.

Il TEMPO può essere visto come il NUMERO DI PASSI (transizioni) che una macchina di Turing deve effettuare prima di fermarsi.

In particolare, è opportuno considerare una funzione f che leggi la dimensione dell'input con il numero di passi della TM:

$$\underbrace{\text{Dimensione dell'input}}_n \xrightarrow{f} \underbrace{\#\text{ passi della TM}}_{f(n)}$$

Tuttavia, la complessità temporale di un algoritmo non dipende solo dalla dimensione dell'input, ma anche dalla forma dell'input stesso (ci sono infatti casi in cui la TM rifiuta subito senza aver prima leggato tutti gli n bit della stringa).

Perciò, data la lunghezza dell'input, si possono effettuare più analisi:

→ ANALISI DEL CASO PEGGIORE (che noi approfondiremo)

→ ANALISI DEL CASO MEDIO

Definizione:

Sia M una TM che termina su qualunque input. Il TEMPO DI ESECUZIONE o COMPLESSITÀ TEMPORALE è la funzione $f: \mathbb{N} \rightarrow \mathbb{N}$, dove $f(n)$ è il numero massimo di passi che M effettua su qualunque input.

i dimensione n .

Si dice quindi che "M gira in tempo $f(n)$ ".

Quello che in particolare ci interessa fare è l'ANALISI ASINTOTICA di f !

Introduciamo delle notazioni a riguardo:

Siano $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$

$\rightarrow f(n) = O(g(n))$ se $\exists c > 0, n_0 > 0$ t.c. $\forall n \geq n_0 \quad f(n) \leq cg(n)$

NB: $2^{O(\log n)} = 2^{c \cdot \log_b n} = 2^{c \cdot \log_b n / \log_2 b} = m^{c \log_b b} = m^d = O(n^d)$ per qualche d

Definizione:

Una funzione f è POLINOMIALMENTE LIMITATA se $f(n) = O(n^d)$ per qualche $d > 0$

(o equivalentemente $f(n) = 2^{O(\log n)}$).

Definizione:

Una funzione f è ESPOENZIALMENTE LIMITATA se $f(n) \leq 2^{n^d}$ per qualche $d > 0$

$\rightarrow f(n) = o(g(n))$ se $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

$f(n) = O(g(n)) \Rightarrow \forall c > 0 \exists n_0 > 0$ t.c. $\forall n \geq n_0 \quad f(n) < cg(n)$

$f(n) = O(g(n)) \Rightarrow f(n) = O(g(n))$.

Definizione:

Sia $t : \mathbb{N} \rightarrow \mathbb{R}^+$ una funzione. La CLASSE DI COMPLESSITÀ TEMPORALE TIME($t(n)$) è l'insieme di tutti i linguaggi che sono decidibili da una DTM singletape in $O(t(n))$ passi (ovvero con tempo di esecuzione pari a $O(t(n))$).

Esempio:

Consideriamo il linguaggio decidibile $A = \{0^k 1^k \mid k \geq 0\}$ e costruiamo una DTM singletape M_1 che lo decide:

Sull'input w :

1. Scansiona il mastro e rifiuta se trovi uno 0 alla destra di un 1. $O(2n)$

2. Finché sia uno 0 sia un 1 rimangono sul mastro: $O(n^{1/2})$

3. Scansiona il nastro e sbarra un simbolo 0 e un simbolo 1. $O(2m)$
4. Se non rimane né uno 0 né un 1 sul nastro, accetta; altrimenti rifiuta. $O(n)$
- COMPLESSITÀ TEMPORALE DELL'ALGORITMO: $O(2m) + O(\frac{m}{2} \cdot 2m) + O(m) = O(m^2)$
- $\Rightarrow A \in \text{TIME}(m^2)$

In realtà si può fare di meglio: costruiamo un'altra DTM singletape M_1 che decida lo stesso linguaggio A :

Sull'input w :

1. Scansiona il nastro e rifiuta se trovi uno 0 alla destra di un 1. $O(m)$
2. Finché sia uno 0 sia un 1 rimangono sul nastro: $O(\log m)$
3. Scansiona il nastro e calcola il numero totale di 0 e di 1 sul nastro; se è dispari, rifiuta. $O(m)$
4. Scansiona il nastro e sbarra uno 0 sì e uno no partendo dal primo. $O(m)$
5. Continua la scansione e sbarra un 1 sì e uno no partendo dal primo. $O(m)$
6. Se non rimane né uno 0 né un 1 sul nastro, accetta; altrimenti rifiuta. $O(n)$

COMPLESSITÀ TEMPORALE DELL'ALGORITMO: $O(m) + O(\log m) O(m) \cdot 2 + O(m) = O(m \log m)$

$\Rightarrow A \in \text{TIME}(m \log m)$

Con una DTM singletape non si può fare ancora meglio; ciò è giustificato dal seguente teorema.

Teorema:

Qualunque linguaggio deciso da una DTM singletape in tempo $O(m \log m)$ è regolare.

→ E, come sappiamo, il linguaggio A in esame non è regolare

Tuttavia, A può essere deciso in tempo lineare da una DTM con 2 nastri.
Infatti, possiamo scrivere il seguente algoritmo:

Sull'input w :

1. Scansiona il nastro e rifiuta se trovi uno 0 alla destra di un 1. $O(m)$

2. Scansiona gli 0 e copiali sul secondo nastro. $O(m)$
3. Scansiona gli 1 e, corrispondentemente, muovi la testina del secondo nastro a sinistra; se non trovi alcuno 0, rifiuta (caso in cui ci sono più 1 che 0). $O(m)$
4. Se ci sono altri 0 alla sinistra della testina del secondo nastro, rifiuta (caso in cui ci sono più 0 che 1). $O(m)$
5. Accetta. $O(1)$

Teorema:

Sia $t(n) > m$. Allora, qualunque DTM multtape con tempo di esecuzione pari a $t(n)$ ha una DTM singletape equivalente che gira in $O(t(n)^2)$ passi.

Dimostrazione:

Di partire da una DTM M con K nastri, realizziamo una DTM S singletape equivalente e vediamo qual è il lavoro che deve fare:

mo. 1) Inizialmente S modifica il suo nastro così da codificare il primo nastro di M con l'input e $K-1$ nastri vuoti.

$$w_1 w_2 \dots w_n \implies \hat{w}_1 \hat{\square} \dots \hat{\square} \underbrace{w_2 \square \dots \square}_{K-1} \dots \underbrace{w_m \square \dots \square}_{K-1} \dots$$

→ i "cappellini" su simboli indicano i caratteri puntati dalle varie testine della DTM multtape

2) Per ogni passo di M , S : $t(n)$

2.1) Scansiona il nastro per scoprire le posizioni delle testine. $O(Kt(n))$

2.2) Scansiona il nastro per aggiornare tutti i nastri di M . $O(Kt(n))$

2.3) Se qualunque dei K nastri di M deve essere ampliato, sposta il contenuto del nastro a destra. $K \cdot O(Kt(n)) = O(K^2t(n))$

COMPLESSITÀ COMPUTAZIONALE: $t(n) \cdot (O(Kt(n)) + O(K^2t(n)) + O(Km)) = O(K^2t^2(n)) = O(t(n)^2)$

Definizione:

Il TEMPO DI ESECUZIONE di un decisore non deterministico (che si forma su qualche ramo della storia della computazione) è la funzione $f: N \rightarrow N$

che dà luogo al massimo numero $f(m)$ di passi effettuati in qualunque percorso dalla radice a una qualsiasi foglia dell'albero della computazione.

Teorema:

Sia $t(m) \geq n$. Allora, qualunque TM singletape non deterministica con tempo di esecuzione pari a $t(n)$ ha una DTM singletape equivalente che gira in $O(t(m))$ passi.

Dimostrazione:

Poiché abbiamo imposto che la NTM si ferma sul qualunque ramo della storia della computazione, la DTM equivalente può anche fare una visita in profondità dell'albero della computazione.

Poiché per ipotesi il tempo di esecuzione del ramo più grande è $t(n)$, posto $b = \#$ massimo di ramificazioni a partire da un qualunque nodo dell'albero, il numero massimo possibile di percorsi di computazione della NTM è $b^{t(m)}$.

Perciò, il tempo di esecuzione della DTM equivalente, la quale deve simulare gli i vari percorsi di computazione è $O(b^{t(m)} \cdot t(m))$.

Ma, abbiamo dimostrato a suo tempo che la DTM equivalente a una NTM, per effettuare la visita dell'albero della computazione senza fare particolari altre operazioni, deve disporre di tre nastri. Perciò:

TEMPO DI ESECUZIONE DI UNA DTM CON 3 NASTRI:

$$O(b^{t(m)} \cdot t(m))$$

TEMPO DI ESECUZIONE DI UNA DTM CON 1 NASTRO:

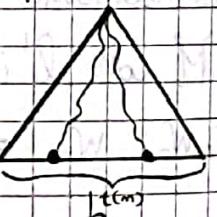
$$O(b^{2t(m)} \cdot t^2(m)) =$$

$$= O(b^{2t(m) + 2\log_b t(m)}) = O(b^{t(m)}) = 2^{O(t(m))}$$

Definizione:

La classe di linguaggi che sono decidibili in un tempo polinomiale dalla DTM è $P = \bigcup_{k \geq 0} \text{TIME}(n^k)$.

Più informalmente potremmo dire che $P \approx$ "classe dei problemi che possono essere risolti realisticamente (cioè in tempi umani) e in maniera effi-



male (ovvero senza approssimazioni) sui computer moderni".

Osservazione:

P è invariante rispetto a tutti i modelli di calcolo che sono polynomialmente equivalenti:

- DTM singletape vs multtape

- LR-move vs LRS-move

- Nastro semi-infinito vs nastro doppiamente infinito

MA ANCHE

- Python

- Java

- C++

- C

- ...

NB: affinché il discorso non sia falso, è necessario che la codifica dell'input sia ragionevole. Ad esempio, dato un valore $m \in \mathbb{N}$, la codifica binaria $m = \underbrace{11\dots11}_m$ non è ragionevole. Infatti:

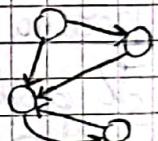
→ CODIFICA IN BASE 2: $| \langle m \rangle_2 | \approx \log_2 m$ } C'è un divario esponenziale tra le due codifiche!
→ CODIFICA IN BASE 1: $| \langle m \rangle_1 | = m$

I grafi:

NODI → ARCHI

Sono delle coppie $G = (V, E)$, dove $|V| =: m$, $|E| =: n$,

$$E \subseteq V^2 \Rightarrow m \leq n^2$$



I grafi possono essere codificati in più modi:

→ MATRICE DI ADIACENZA $n \times n$, le cui componenti (i, j) valgono 1 se esiste un arco che collega il nodo i col nodo j , valgono 0 altrimenti; in particolare, se gli archi non sono orientati, $(i, j) = 1 \Leftrightarrow (j, i) = 1$. Questa codifica ha una grandezza pari a $O(n^2)$.

→ LISTA DI ARCHI, preceduta da un valore che indica il numero dei nodi del grafo:

$$\underbrace{|V|}_{\log m \text{ bit}}, \underbrace{(i, j), \dots}_{|E|}$$

Questa codifica ha una grandezza al massimo pari a $O(\log m + m \cdot 2 \cdot \log m) = O(m^2 \log m)$

In ogni caso, possiamo assumere tranquillamente che $|G| = O(n^3)$

Teorema:

Consideriamo il problema $\text{PATH} = \{ \langle G, s, t \rangle \mid G \text{ è un grafo diretto che ha un percorso diretto da } s \text{ a } t \}$. $\text{PATH} \in P$

Dimostrazione:

Costruiamo una TM che decida il problema PATH e che quindi abbia il seguente algoritmo:

Sull'input $\langle G, s, t \rangle$, dove G è un grafo diretto e $s, t \in V(G)$:

1. Marca il nodo s . $O(1)$
2. Finché non possono essere marcati ulteriori nodi: $O(m)$
3. Scansiona gli archi e marca ogni nodo non marcato che ha un arco entrante da un nodo già marcato. $O(m) \cdot O(|G|)$
4. Se t è stato marcato, accetta; altrimenti, rifiuta. $O(1)$

COMPLESSITÀ TEMPORALE: $O(m \cdot m^2 \cdot m^2) = O(m^5)$

Teorema:

Il problema $\text{RELPRIME} = \{ \langle x, y \rangle \mid x, y \in \mathbb{N} \text{ e } x, y \text{ sono relativamente primi} \}$ $\in P$.

Dimostrazione:

$$\langle x, y \rangle \in \text{RELPRIME} \iff \text{MCD}(x, y) = 1$$

Sfruttiamo l'algoritmo di Euclide E:

Sull'input $\langle x, y \rangle$, dove x, y sono numeri naturali:

1. Finché $y \neq 0$:
2. $x \leftarrow x \text{ mod } y$
3. $x \leftarrow y$ (scambia i valori di x, y)
4. Dai x in output.

Se operazioni all'interno del ciclo sono efficienti, quindi il problema è

solo analizzare quante volte deve essere rientrato il ciclo.
 Ma possiamo affermare che $x \bmod y < \frac{x}{2}$, poiché:

$$\rightarrow \text{Se } \frac{x}{2} \geq y \Rightarrow x \bmod y < y \leq \frac{x}{2} \quad \checkmark$$

$$\rightarrow \text{Se } \frac{x}{2} < y \Rightarrow x \bmod y = x - y < x - \frac{x}{2} = \frac{x}{2} \quad \checkmark$$

Per cui, praticamente, ciascuna delle due variabili ogni due cicli viene più che dimezzata. In definitiva, il numero di cicli effettuati è pari a $\min(2 \log_2 x, 2 \log_2 y)$, per cui si può concludere che l'algoritmo di Euclide ^E è polinomiale.

Realizziamo adesso un algoritmo ^R che decida il problema REPRIME:

Sull'input $\langle x, y \rangle$, dove x, y sono numeri naturali:

1. Esegui E sull'input $\langle x, y \rangle$.
2. Se $E(\langle x, y \rangle) = 1$, allora accetta; altrimenti, rifiuta.

È evidente che la complessità temporale di R coincide con quella di E
 $\Rightarrow \text{REPRIME} \in \mathbf{P}$. ■

Teorema:

ogni linguaggio libero dal contesto è in \mathbf{P} .

Idea della dimostrazione:

Utilizzare, come algoritmo di riferimento, l'Earley Parser, atto a riconoscere le CFG.

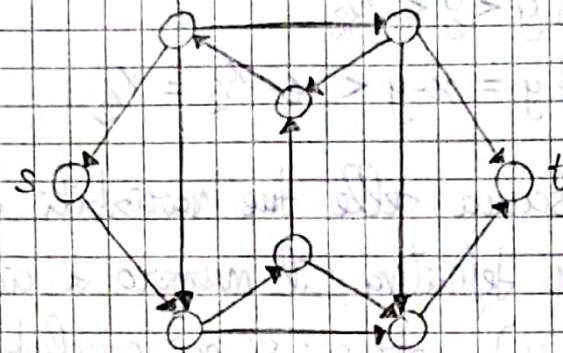
Ricordiamoci che, su un input di lunghezza n , la complessità temporale dell'Earley Parser è:

- $O(n^3)$ per grammatiche ambigue
- $O(n^2)$ per grammatiche non ambigue
- $O(n)$ per DCFG e LR(K)

26/05/2020

Consideriamo il problema HAMPATH = $\{ \langle G, s, t \rangle \mid G \text{ è un grafo diretto ed esiste un percorso hamiltoniano da } s \text{ a } t, \text{ st} \in V(G) \}$.

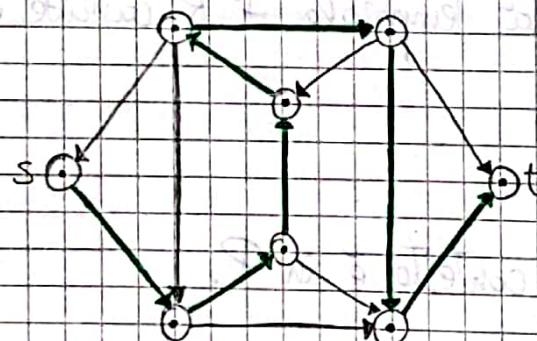
PERCORSO HAMILTONIANO = percorso diretto che tocca ogni nodo esattamente una volta.



Costruiamo una TM M con il seguente algoritmo:

Sull'input $\langle G, s, t \rangle$, dove G è un grafo diretto e $s, t \in V(G)$:

1. Per tutti i possibili percorsi diretti da s a t :
2. Controlla se il percorso passa attraverso ogni nodo esattamente una volta; se sì, accetta.
3. Rifiuta.



L'algoritmo non ci dimostra $\neq P$ che HAMPATH $\in P$, poiché il numero di tutti i possibili percorsi da s a t potrebbe essere esponenziale. Ed è proprio per questo motivo che la parte difficile del problema è scoprire l'eventuale percorso da s a t che, tra i tanti, sia hamiltoniano. D'altra parte, una volta che questo percorso è stato evidenziato, è immediato verificare che effettivamente passa per tutti i nodi una volta sola. Vediamo di formalizzare questa idea.

Definizione:

Un problema è POLINOMIALMENTE VERIFICABILE se un "certificato" di un qualunque istanza sì del problema può essere verificato da una DTM

ca in un tempo polinomiale nella dimensione dell'istanza.

Nel nostro esempio, un certificato è il percorso evidenziato in verde nella pagina precedente.

Affinché il certificato possa essere verificato in tempo polinomiale, è ovvio che la sua stessa dimensione sia polinomiale, altrimenti la macchina non avrebbe neanche il tempo di leggerlo. Ma, tornando all'esempio, poiché il certificato è un sottoinsieme di un grafo (che è rappresentabile polinomialmente) e poiché la verifica sul certificato stesso può essere effettuata banalmente in pochi passi, si può concludere che HAMPATH è polinomialmente verificabile.

Consideriamo ora il problema COMPOSITE = $\{ \langle x \rangle \mid x \in \mathbb{N} \wedge \exists p, q \in \mathbb{N}, p, q > 1 \text{ t.c. } pq = x \}$.

COMPOSITE è polinomialmente verificabile.

Infatti, un certificato per $\langle x \rangle$ è un divisore p per il numero x . Calcolare x/p può essere fatto in tempo polinomiale rispetto a $|\langle x \rangle| = O(\log x)$.

Recentemente è stato anche dimostrato che COMPOSITE ∈ P.

Osservazione:

Se un problema è polinomialmente verificabile, non è affatto detto che anche il suo complementare lo sia. Per esempio, consideriamo il complementare di HAMPATH ($\text{HAMPATH}^c = \overline{\text{HAMPATH}}$); le sue istanze si consistano nel-

le coppie di nodi s, t di un grafo tali che non esiste alcun percorso hamiltoniano da s a t. Perciò, i certificati di tali istanze si devo-

no contenere tutti i percorsi diretti da s a t e mostrare che tutti que-

sti percorsi o non attraversano un determinato nodo, o lo attraversano più volte. Ma, poiché in generale tutti i percorsi da un nodo a un altro sono di numero esponenziale, possiamo concludere che HAMPATH^c

NON è polinomialmente verificabile.

Definizione:

DETERMINISTICO

Un VERIFICATORE di un linguaggio A è un algoritmo V tale che
 $A = \{w \mid V \text{ accetta } \langle w, c \rangle \text{ per qualche stringa } c \text{ (che decodifica un certificato) }\}$

Definizione:

Un VERIFICATORE A TEMPO POLINOMIALE è un verificatore che gira in tempo polinomiale rispetto alla lunghezza di w (NON rispetto alla lunghezza di $\langle w, c \rangle$)

Definizione:

NP è la classe dei linguaggi che hanno verificatori a tempo polinomiale.

→ HAMPATH ∈ NP , COMPOSITE ∈ NP

→ NP = Non-deterministicamente polinomiale → comprende quei problemi che possono essere risolti con macchine di turing non deterministiche in tempo polinomiale

Costruiamo una NTM che risolva il problema HAMPATH in tempo polinomiale:

Sull'input $\langle G, s, t \rangle$, dove G è un grafo diretto, $s, t \in V(G)$:

1. Intervista non-deterministicamente e scrivi sul nastro una lista di n nodi p_1, \dots, p_m , dove $m = |V(G)|$.
2. Rifiuta se un qualunque nodo delle liste è ripetuto.
3. Rifiuta se $p_1 \neq s$ o $p_n \neq t$.
4. Per $i = 1 \rightarrow n-1$:
 5. Controlla se $(p_i, p_{i+1}) \in E(G)$. Se no, rifiuta.
 6. Accetta.

Teorema:

Un linguaggio A è in NP \iff è deciso da qualche NTM che gira in tempo polinomiale.

Idea della dimostrazione:

Convertire un verificatore in un decisore non deterministico e viceversa.

Dimostrazione:

$\Rightarrow A \in NP \Rightarrow V$ verifica A in tempo polinomiale :

$$A = \{w \mid V(\langle w, c \rangle) \text{ accetta in tempo } t < n^k\}$$

costruiamo una NTM N col seguente algoritmo:

Sull'input w :

1. Intovina non-deterministicamente una stringa c con lunghezza $|c| < m^k$.
2. Esegui V sull'input $\langle w, c \rangle$.
3. Se $V(\langle w, c \rangle)$ accetta, allora accetta; altrimenti, rifiuta.

$\Rightarrow N$ decide il linguaggio A in un tempo non-deterministicamente polinomiale.

$\Rightarrow A$ ha una NTM N che la decide in tempo polinomiale. Allora possiamo costruire un verificatore V col seguente algoritmo:

Sull'input $\langle w, c \rangle$:

1. Simula N su w , utilizzando i simboli di c per scegliere il percorso non deterministico da attraversare all'interno dell'albero di computazione.
2. Se la simulazione di N su w fatta da c accetta, allora accetta; altrimenti, rifiuta.

$\Rightarrow V$ è un verificatore a tempo polinomiale $\Rightarrow A \in NP$

Definizione:

$NTIME(t(m)) := \{L \mid L \text{ è un linguaggio deciso da una NTM in un tempo } O(t(m))\}$

$\rightarrow NP = \bigcup_{k \geq 0} NTIME(m^k)$

Teorema:

Il problema CLIQUE = $\{\langle G, k \rangle \mid G \text{ è un grafo non diretto con una K-clique}\}$ appartiene a NP.

$\rightarrow K\text{-CLIQUE} :=$ SOTTOGRAFO COMPLETO DI K NODI (IN CUI ESISTE UN ARCO PER OGNI COPPIA DI NODI)

Idea della dimostrazione:

Sfruttare un certificato che potrebbe essere una lista di K nodi che formano un sottografo completo.

Dimostrazione:

Costruiamo un verificatore V col seguente algoritmo:

Sull'input $\langle\langle G, K \rangle, c \rangle$, dove G è un grafo, $K \in \mathbb{N}$, c è una stringa:

1. Verifica se c codifica un sottografo di G con K nodi.
2. Verifica se G contiene tutti gli archi tra i nodi di c .
3. Se entrambi i controlli sono andati a buon fine, accetta; altrimenti, rifiuta.

$\Rightarrow V$ è un verificatore a tempo polinomiale $\Rightarrow \text{CLIQUE} \in \text{NP}$

Dimostrazione alternativa:

Va bene anche costruire una NTM N col seguente algoritmo:

Sull'input $\langle G, K \rangle$, dove G è un grafo e $K \in \mathbb{N}$:

1. Indovina non-deterministicamente un sottoinsieme c di K nodi di G .
2. Verifica se G contiene tutti gli archi tra i nodi di c .
3. Se il controllo è andato a buon fine, accetta; altrimenti, rifiuta.

$\Rightarrow N$ è una NTM che decide CLIQUE in tempo polinomiale $\Rightarrow \text{CLIQUE} \in \text{NP}$

Teorema:

Il problema SUBSET-SUM = $\{ \langle S, t \rangle \mid S \text{ è un multi-insieme } \{x_1, \dots, x_n\}, x_i \in \mathbb{N} \text{ e, per qualche multi-insieme } Y = \{y_1, \dots, y_e\} \subseteq S, \sum_{j=1}^e y_j = t \}$ appartiene a NP.

\leadsto UN MULTI-INSIEME CONTIENE ELEMENTI CHE POSSONO ESSERE RIPETUTI.

Idea della dimostrazione:

Sfruttare un certificato che potrebbe essere un sottoinsieme di elementi di S la cui somma è uguale a t .

Dimostrazione:

Costruiamo un verificatore V col seguente algoritmo:

Sull'input $\langle\langle S, t \rangle, c \rangle$, dove S è un multi-insieme, $t \in \mathbb{N}$, c è una stringa:

1. Verifica se c costituisce un sottoinsieme di S .
2. Verifica se la somma degli elementi di c è uguale a t .
3. Se entrambi i controlli sono andati a buon fine, accetta; altrimenti, rifiuta.

$\Rightarrow V$ è un verificatore a tempo polinomiale $\Rightarrow \text{SUBSET-SUM} \in \text{NP}$

Dimostrazione alternativa:

Va bene anche costruire una NTM N col seguente algoritmo:

Sull'input $\langle S, t \rangle$, dove S è un multi-insieme e $t \in \mathbb{N}$:

1. Indovina non-deterministicamente un sottoinsieme c di S .
2. Verifica se la somma degli elementi di c è uguale a t .
3. Se il controllo è andato a buon fine, accetta; altrimenti, rifiuta.

$\Rightarrow N$ è una NTM che decide SUBSET-SUM in tempo polinomiale $\Rightarrow \text{SUBSET-SUM} \in \text{NP}$

Definizione:

$\text{coNP} := \{ L \mid L \text{ è un linguaggio t.c. } \overline{L} \in \text{NP} \}$

$\rightarrow \text{CLIQUE} \in \text{coNP}$

SUBSET-SUM $\in \text{coNP}$

Quesione aperta:

? $\text{coNP} = \text{NP}$?

? Sono possibili due scenari ad oggi:

• $\text{coNP} = \text{NP}$

• $\text{coNP} \neq \text{NP}$ con $\text{coNP} \cap \text{NP} \neq \emptyset$

Definizione:

$\text{EXPTIME} := \bigcup_{k \geq 0} \text{TIME}(2^{m^k})$

Per ora sappiamo che:

$P \subseteq \text{NP} \subseteq \text{EXPTIME}$

La questione aperta più importante dell'ingegneria informatica:

$P = ? \text{NP}$

Dovuto al fatto che ogni NTM che gira in tempo $O(t(m))$ è equivalente a una DTM che gira in tempo $2^t(m)$

2

{ MA }

Non sappiamo se $\text{NP} = \text{EXPTIME}$

Definizione:

Una funzione $f: \Sigma^* \rightarrow \Sigma^*$ è una FUNZIONE CALCOLABILE IN TEMPO POLINOMIALE se esiste una DTM che lavora in tempo polinomiale che termina sull'input w lasciando sul nastro la stringa $f(w)$.

Definizione:

Un linguaggio A si dice "POLYNOMIAL TIME MAPPING REDUCIBLE" (o "POLYNOMIAL TIME MANY-ONE REDUCIBLE" o "POLYNOMIAL TIME REDUCIBLE") a un linguaggio B ($A \leq_p B$) se esiste una funzione calcolabile in tempo polinomiale f tale che $w \in A \iff f(w) \in B$.

Teorema:

Se $A \leq_p B$, $B \in P \Rightarrow A \in P$

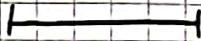
Dimostrazione:

A partire da una DTM M che decide il linguaggio B in tempo polinomiale e da una DTM T che calcola la funzione calcolabile dalla riduzione polinomiale tra A e B , costruiamo una DTM N col seguente algoritmo:

Sull'input w :

1. Esegui T su w e lascia $f(w)$ sul nastro.
2. Esegui M su $f(w)$. → poiché T lavora in tempo polinomiale, $f(w)$ ha lunghezza polinomiale rispetto a $w \Rightarrow M$ gira in tempo polinomiale.
3. Se $\cancel{N} \rightarrow M(f(w))$ accetta, allora accetta. Altrimenti, rifiuta.

$\Rightarrow N$ decide A in tempo polinomiale $\Rightarrow A \in P$



→ Una VARIABILE BOOLEANA può assumere solo i valori VERO (1) e FALSO (0)

→ Una FORMULA BOOLEANA è una formula che contiene variabili booleane combinate con gli operatori $\wedge \vee \neg$

→ Una formula booleana si dice SODDISFACIBILE se esiste una combinazione di valori da assegnare alle variabili booleane che renda la formula vera

Teorema:

Il problema di soddisficiabilità delle formule booleane $SAT = \{ \langle F \rangle \mid F \text{ è una formula booleana soddisfacibile} \}$ appartiene a NP.

Dimostrazione:

Costruiamo un verificatore V col seguente algoritmo:

Sull'input $\langle F, c \rangle$, dove F è una formula booleana, c è una stringa:

1. Se c non è una lista di K valori di verità, rifiuta.
2. Se il numero delle variabili in F non è K , rifiuta.
3. Se c rende vera F , accetta; altrimenti, rifiuta.

$\Rightarrow V$ è un verificatore a tempo polinomiale $\Rightarrow SAT \in NP$



\rightarrow LETTERALE := variabile | variabile

\rightarrow CLAUSOLA := (letterale $\vee \dots \vee$ letterale)

\rightarrow FORMA NORMALE CONGIUNTIVA DI UNA FORMULA BOOLEANA (CNF) := clausola $\wedge \dots \wedge$ clausola

\rightarrow 3CNF := CNF in cui ogni clausola ha 3 letterali

Affiancando una riduzione polinomiale ($SAT \leq_p 3SAT$) si può benissimo dimostrare che il problema $3SAT := \{ \langle F \rangle \mid F \text{ è una formula booleana soddisfacibile in 3CNF} \}$ appartiene alla classe NP.

30/05/2020

Teorema:

$\text{3SAT} \leq_p \text{CLIQUE}$

Tea della dimostrazione:

Trasformare un'istanza di 3SAT in un'istanza di CLIQUE.

Quindi, a partire da una formula $F = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \dots \wedge (a_n \vee b_n \vee c_n)$, vogliamo realizzare un grafo G che ha una K -clique $\Leftrightarrow F$ è soddisfacibile.

Dimostrazione:

Definiamo come devono essere fatti i nodi $V(G)$ e gli archi $E(G)$ del grafo G che stiamo andando a costruire:

$V(G)$: per ogni clausola (tripletta) di F , esistono 3 nodi in G etichettati con i letterali corrispondenti.

$E(G)$: per ogni coppia di nodi, esiste sempre un arco che li collega a meno che si verifica una delle seguenti condizioni:

- 1) I nodi appartengono alla stessa clausola.
- 2) I nodi hanno etichette opposte.

Allora:

• F è soddisfacibile $\Rightarrow \exists$ assegnazione dei valori alle variabili tale che, in ogni clausola, \exists letterale che è vero \Rightarrow se consideriamo i nodi corrispondenti ai letterali veri di ogni clausola, formano una K -clique (infatti, questi K nodi provengono tutti da K clausole diverse e non sono mai uno l'opposto dell'altro: se un letterale è vero, non può essere vera anche la sua negazione affinché la formula sia soddisfacibile!) $\Rightarrow (G, K) \in \text{CLIQUE}$

• $(G, K) \in \text{CLIQUE} \Rightarrow \exists K$ nodi in G che formano un sottografo completo \Rightarrow questi K nodi, per forza di cose, devono provenire da K clausole diverse e non possono contenere letterali opposti $\Rightarrow \exists$ assegnazione dei valori alle variabili tale che, in ogni clausola, \exists letterale che è vero \Rightarrow la formula F è soddisfacibile

In definitiva, F è soddisfacibile $\iff G$ ha una K -clique
Abbiamo quindi provato che $3SAT \leq_p CLIQUE$, anche perché la costruzione del
grafo G è polinomiale rispetto all'istanza F (c'è infatti una corrispondenza
tra i letterali di F e i nodi di G). ■

Definizione:

Un linguaggio B si dice "NP-COMPLETO" se valgono le seguenti condizioni:

- (1) $B \in NP$ (2) $\forall A \in NP \quad A \leq_p B$

Teorema:

Se un linguaggio B fosse NP-completo e $B \in P \implies P = NP$

Dimostrazione:

Se $A \in NP \implies A \leq_p B$

Ma, per ipotesi, $B \in P \implies A \in P \implies NP \subseteq P$

Già sappiamo che $P \subseteq NP$, quindi otterremmo che $P = NP$ ■

Teorema:

Se B è NP-completo, $B \leq_p C$ e $C \in NP \implies C$ è NP-completo

Dimostrazione:

" \leq_p " è una relazione transitiva. Perciò:

$$\forall A \in NP \quad A \leq_p B \leq_p C \implies A \leq_p C$$

Definizione:

Un linguaggio B si dice "NP-HARD" se $\forall A \in NP \quad A \leq_p B$ (non sta-
mo facendo nulla riguardo l'appartenenza di B alla classe NP).

Corollario 1:

Se un linguaggio B fosse NP-hard e $B \in P \implies P = NP$

Corollario 2:

Se B è NP-hard e $B \leq_p C \implies C$ è NP-hard

Ma esistono effettivamente dei problemi NP-completi?

Teorema di Cook - Levin:

SAT è un problema NP-completo.

Corollario:

$$SAT \in P \iff P = NP$$

Idea della dimostrazione del Teorema:

Abbiamo già provato che $SAT \in NP$. Ci resta da dimostrare che $\forall A \in NP \quad A \leq_p SAT$.

Se $A \in NP \Rightarrow \exists \text{NTM } M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ che decide A in un tempo $t < n^k$ per qualche K .

A partire dalla NTM M e da un'istanza w , noi vogliamo costruire una formula booleana ϕ composta dagli operatori \wedge, \vee, \neg tale che $M(w)$ accetta $\iff \phi$ è soddisfacibile.

Dimostrazione formale:

Costruiamo un tableau (una matrice) fatto così:

Il tableau è relativo a un unico ramo di computazione della NTM M

n^k colonne

#	$q_0 w_1 w_2 \dots w_m L L \dots$	L	#
#			#
#			#

← configurazione iniziale di $M(w)$

Non appartiene a Γ

cella $[i, j] \in C := \Gamma \cup Q \cup \{\#\}$

Indice riga che indica lo step in cui $M(w)$ si trova all'interno del ramo di computazione corrispondente al tableau

Indice colonna che indica la posizione all'interno del nastro

Il tableau deve essere chiaramente fatto in modo tale che $\forall i \exists$ mossa legale in S che porta dalla configurazione descritta nell' i -esima riga alla configurazione descritta nell' $(i+1)$ -esima riga.

Il tableau si dice ACCETTANTE se \exists almeno una configurazione al suo inizio

terno che contiene lo stato q di accettazione q_A .

Il nostro scopo è costruire una formula ϕ che sia soddisfacibile $\iff \exists$ tableau accettante legale per $M(w) \iff w \in A$.

Φ sarà fatta nel seguente modo:

$$\Phi = \Phi_{\text{cell}} \wedge \Phi_{\text{start}} \wedge \Phi_{\text{move}} \wedge \Phi_{\text{accept}}$$

Le variabili booleane su cui si fonda questa formula sono del tipo

$x_{i,j,s}$, con $1 \leq i, j \leq n^k$, $s \in C$. In particolare:

$$x_{i,j,s} = 1 \text{ (VERO)} \iff \text{cella } [i,j] = s$$

Vediamo come devono essere fatte le quattro sottoformule che costituiscono Φ :

1) Φ_{cell} : "Ogni cella del tableau contiene esattamente un simbolo di C ".

$$\Phi_{\text{cell}} := \bigwedge_{1 \leq i, j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s, t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

↑ ↑ ↑ ↑
Per ogni cella c'è un simbolo e non più di un simbolo

2) Φ_{start} : "La prima riga del tableau è la configurazione iniziale di $M(w)$, $w = w_1 \dots w_m$ ".

$$\Phi_{\text{start}} := x_{1,1,*} \wedge x_{1,2,w_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,n+2,w_m} \wedge$$

↑
 $n+3 \leq j \leq m-1$ $x_{1,j,w_j} \wedge x_{1,m,*}$

3) Φ_{accept} : "Almeno una riga del tableau include q_A ".

$$\Phi_{\text{accept}} := \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_A}$$

4) Φ_{move} : "Ogni riga del tableau è seguita da una riga che corrisponde a una configurazione che può essere raggiunta da M in un unico passo".

In realtà, per effettuare questo controllo, non serve confrontare tra loro le righe per intero, ma basta prendere una finestra composta da sei celle

(con due righe e tre colonne) ed esaminerà tutte le celle del tableau a gruppi di sei. Se tutti i controlli hanno successo, possiamo concludere che nessuna riga è tale da non seguire in maniera lecita quella presente e, quindi, che l'intero tableau è lecito.

Esempio:

$$\delta(q_1, a) = \{(q_1, b, R)\}$$

$$\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$$

ESEMPI DI FINESTRE LECITE:

a	q ₁	b
q ₂	a	c

a	q ₁	b
a	a	q ₂

a	a	q ₁
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q ₂

b	b	b
c	b	b

ESEMPI DI FINESTRE NON LECITE:

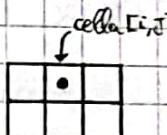
a	b	a
a	a	a

a	q ₁	b
q ₂	a	a

In particolare, per ogni NTM esiste una collezione di finestre lecite su cui ci si basa quando si effettua il controllo sul se un dato tableau è lecito o no.

Ora siamo in grado di definire meglio Φ_{move} :

$$\Phi_{\text{move}} := \bigwedge_{1 \leq i, j \leq m^k} \left\{ \text{La finestra relativa a cella } [i, j] \text{ è legale} \right\} \equiv$$



$$\equiv \bigwedge_{1 \leq i, j \leq m^k} \left\{ \bigvee \begin{array}{c} \begin{array}{|c|c|c|} \hline a_1 & a_2 & a_3 \\ \hline a_4 & a_5 & a_6 \\ \hline \text{legale} \\ \hline \end{array} \\ \wedge X_{i, j-1, a_1} \wedge X_{i, j, a_2} \wedge X_{i, j+1, a_3} \\ \wedge X_{i+1, j-1, a_4} \wedge X_{i+1, j, a_5} \wedge X_{i+1, j+1, a_6} \end{array} \right\}$$

A questo punto rimane solo da verificare che la costruzione della formula Φ è polinomiale:

$$|\Phi| = |\Phi_{\text{cell}}| + |\Phi_{\text{start}}| + |\Phi_{\text{move}}| + |\Phi_{\text{accept}}| + O(1)$$

Dovuto agli and logici che legano questi quattro sottosimboli

$$\rightarrow |\Phi_{\text{cell}}| = O((m^k)^2 \cdot |C|^2 \cdot \log n) = O(n^{2k+4})$$

↓ Dovuto ai simboli p_i
 ↓ Dovuto ai simboli i, j
 ↓ Dovuto ai simboli s, t
 ↓ Dovuto alla rappresentazione
 dei parametri i, j

Perché C dipende solo dalla configurazione della TM M e non dall'input, è da considerarsi una costante.

$$\rightarrow |\Phi_{\text{start}}| = O(n^k \cdot \log n) = O(n^{k+1})$$

Dovuto al ciclo del parametro j

Dovuto alla rappresentazione
del parametro j

$$\rightarrow |\Phi_{\text{accept}}| = O((n^k)^2 \cdot \log n) = O(n^{2k+1})$$

Dovuto ai cicli dei parametri i, j

Dovuto alla rappresentazione
dei parametri i, j

$$\rightarrow |\Phi_{\text{move}}| = O((n^k)^2 \cdot \log n) = O(n^{2k+1})$$

Dovuto ai cicli dei parametri i, j

Dovuto alla rappresentazione
dei parametri i, j .

Non compaiono altri fattori perché la complessità del ciclo sulle finestre legali dipende esclusivamente dalla NTH M e non dall'input; perciò, è da considerarsi costante

$$\begin{aligned} \Rightarrow |\Phi| &= |\Phi_{\text{cell}}| + |\Phi_{\text{start}}| + |\Phi_{\text{move}}| + |\Phi_{\text{accept}}| + O(1) = \\ &= O(n^{2k+1}) + O(n^{k+1}) + O(n^{2k+1}) + O(n^{2k+1}) + O(1) = O(n^{2k+1}) \end{aligned}$$

Poiché K è un'altra costante che dipende solo dalla NTH M, possiamo concludere che la costruzione della formula Φ è polinomiale e, perciò, che SAT è un problema NP-completo. ■

Lemma:

CNF-SAT \leq_p 3SAT , dove CNF-SAT := { $\langle F \rangle$ | F è una formula booleana soddisfacibile in CNF}.

Dimostrazione:

Sia F una CNF $\Rightarrow F = \bigwedge (l_1 \vee l_2 \vee \dots \vee l_k)$

Sono possibili tre scenari per ciascuna clausola di F :

1) LA CLAUSOLA HA 3 LETTERALI \rightarrow OK!

2) LA CLAUSOLA HA MENO DI 3 LETTERALI; facciamo un esempio:

$$(l_1 \vee l_2) \equiv (l_1 \vee l_2 \vee l_2)$$

3) LA CLAUSOLA HA PIÙ DI ~~MASSIMO~~ 3 LETTERALI; facciamo un esempio:

$$(l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5) \equiv (l_1 \vee l_2 \vee z_1) \wedge (\bar{z}_1 \vee l_3 \vee z_2) \wedge (\bar{z}_2 \vee l_4 \vee l_5)$$

\rightsquigarrow I letterali $z_1, \bar{z}_1, z_2, \bar{z}_2$ servono a garantire che ciascuna delle tre clausole della nuova formula contenga almeno un letterale vero nel caso in cui ~~anche~~ solo uno dei letterali l_1, l_2, l_3, l_4, l_5 sia impostato a vero.

Corollario:

3SAT è un problema NP-completo.

Dimostrazione:

Se sfruttiamo il lemma che abbiamo enunciato poc' anzi, ci basta provare che CNF-SAT è NP-completo. Per far ciò, riprendiamo la formula booleana $\Phi = \Phi_{\text{cell}} \wedge \Phi_{\text{start}} \wedge \Phi_{\text{move}} \wedge \Phi_{\text{accept}}$ e verifichiamo che si possa trasformare in una CNF in tempo polinomiale. Poiché sono semplicemente separate da and logici (\wedge), è sufficiente analizzare le quattro sottiformule singolarmente.

$$\rightarrow \Phi_{\text{cell}} = \wedge [(\vee) \wedge (\wedge (\vee))] \equiv [\wedge (\vee)] \wedge [\wedge (\vee)] \equiv \wedge (\vee) \quad \rightsquigarrow \text{È un AND di OR} \Rightarrow \text{è già una CNF} \quad \checkmark$$

$$\rightarrow \Phi_{\text{start}} = \wedge (\text{clausole lunghe 1}) \quad \rightsquigarrow \text{È già una CNF} \quad \checkmark$$

$$\rightarrow \Phi_{\text{accept}} = \vee (\text{letterali}) \quad \rightsquigarrow \text{È un'unica clausola} \Rightarrow \text{è già una CNF} \quad \checkmark$$

$$\rightarrow \Phi_{\text{move}} = \wedge \{\vee [\wedge]\} \quad \rightsquigarrow \text{Non è una CNF}$$

Per trasformare Φ_{move} in una CNF, applichiamo una delle due proprietà di distributività:

$$Pv(Q \wedge R) \equiv (PvQ) \wedge (PvR)$$

Bisogna però prestare attenzione: se le clausole ~~fussero~~ un po' più lunghe, come nel caso di Φ_{move} , con la trasformazione esse aumenterebbero esponenzialmente. Facciamo un esempio:

$$(a \wedge b \wedge c) \vee (d \wedge e \wedge f) = ((a \wedge b \wedge c) \vee d) \wedge ((a \wedge b \wedge c) \vee e) \wedge ((a \wedge b \wedge c) \vee f) \equiv (d \vee a) \wedge (d \vee b) \wedge (d \vee c) \wedge (e \vee a) \wedge (e \vee b) \wedge (e \vee c) \wedge (f \vee a) \wedge (f \vee b) \wedge (f \vee c)$$

→ È UNA CNF MA È ESADENZIALMENTE PIÙ LUNGA RISPETTO ALLA FORMULA DI PARTENZA!

Ma questo non è un problema: tutti i letterali che si trovano nell'AND più esterno sono di un numero costante rispetto alla dimensione dell'input (infatti, il ciclo dell'OR è su tutte le finestre legali che sono costanti, e all'interno

no di questo OR ci sono sempre sei variabili poste in AND, quindi ancora un numero costante)! Perciò, la crescita esponenziale viene ridotta solo da un valore costante, mentre la complessità totale di Φ_{mire} trasformata in CNF rimane esattamente $O(n^{2k+1})$ e, quindi, polinomiale.

Possiamo perciò concludere che CNF-SAT è un problema NP-completo $\Rightarrow \Rightarrow 3\text{SAT}$ è NP-completo.

Cordilloro:

CLIQUE è un problema NP-completo.

Dimostrazione:

Abbiamo già provato che:

- CLIQUE \in NP
- $3\text{SAT} \leq_p \text{CLIQUE}$
- 3SAT è NP-completo

} \Rightarrow CLIQUE è NP-completo

05/06/2020

Il problema Vertex Cover (VC):

ISTANZA: un grafo indiretto G e un numero $K \in \mathbb{N}$.

DOMANDA: esiste un vertex cover di dimensione K nel grafo G ?

Un vertex cover è un sottoinsieme $C \subseteq V(G)$ di nodi tale che ogni arco in $E(G)$ è adiacente ad almeno uno dei nodi in C .

Teorema:

VC è un problema NP-completo.

Idea della dimostrazione:

Provare che VC \in NP, $3\text{SAT} \leq_p \text{VC}$.

Dimostrazione:

Un certificato per VC è un sottoinsieme $C \subseteq V(G)$ di dimensione K e che è un vertex cover. Il verificatore per questo certificato deve solo assicurarsi che C contenga K nodi e che tutti gli archi di G siano adiacenti a almeno uno dei nodi in C .

ti ad almeno uno di questi nodi; perciò, è facile convincersi che il verificatore lavora in tempo polinomiale rispetto all'input $\Rightarrow VC \in NP$

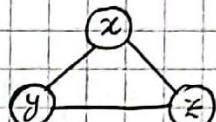
Dimostriamo adesso che $3SAT \leq VC$.

Consideriamo una formula 3CNF F con le variabili v_1, \dots, v_m e le clausole c_1, \dots, c_l

\forall variabile v_i : G ha una coppia di nodi:



\forall clausola $c_i = (x \vee y \vee z)$ G ha una tripletta di nodi:



Ogni nodo in una tripletta è connesso al nodo corrispondente nelle coppie (per esempio, se $x = \bar{v}_i \Rightarrow$ l'arco che collega x con \bar{v}_i)

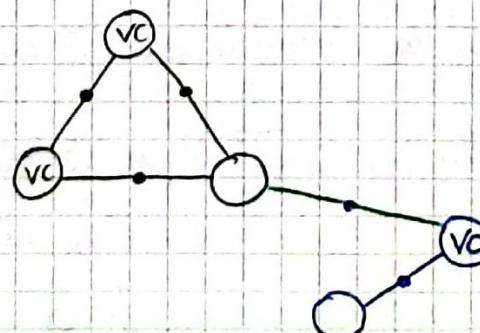
Dimostriamo la seguente affermazione:

Se F è soddisfacibile $\Rightarrow \exists$ VC di dimensione $K = m + 2l$.

Assegniamo un valore di verità a ciascuna variabile e mettiamo nel VC i nodi delle coppie che corrispondono ai letterali VERO; abbiamo così inserito i primi m nodi nel VC .

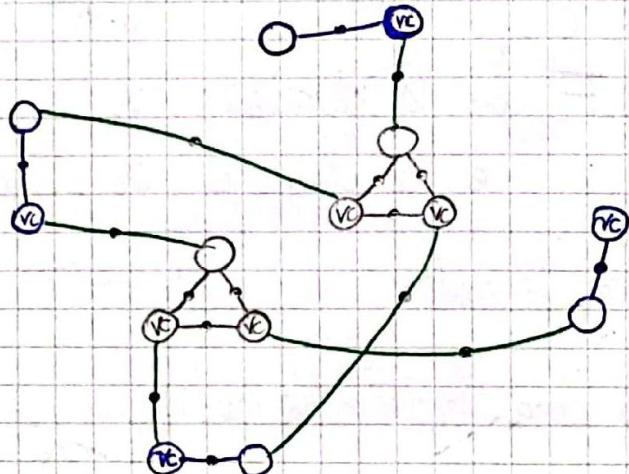
Poiché F è soddisfacibile per ipotesi, e quindi in ogni clausola c'è sempre almeno un letterale VERO, allora in ogni tripletta c'è sempre almeno un nodo collegato a un altro nodo di una coppia già facente parte del VC .

Perciò, per ricoprire tutti e tre gli archi della tripletta, bastano gli altri due nodi della tripletta stessa; in pratica, è sufficiente inserire nel VC due nodi per ogni clausola, e, quindi, altri $2l$ nodi che, sommati a quelli aggiunti precedentemente, formano un totale di $m + 2l$.



Dimostriamo ora il viceversa:

Se \exists VC di dimensione $m+2l \Rightarrow F$ è soddisfacibile.



Un VC esattamente di dimensione $m+2l$ contiene sicuramente un nodo per ogni coppia e due nodi per ogni clausola. In particolare, poiché per ipotesi tutti gli archi sono coperti dal VC, è inevitabile che in ogni tripletta ci sia almeno un nodo x connesso a un altro nodo v di una coppia appartenente al VC. Se assumiamo che x corrisponde a un letterale VERO della clausola, ed estendiamo l'ipotesi su tutto il grafo, possiamo concludere che la formula F è soddisfacibile. \square

In definitiva, poiché F è soddisfacibile $\Leftrightarrow \exists$ VC di dimensione $m+2l$, e la costruzione del grafo G è polinomiale, l'asserto $3SAT \leq_p VC$ è dimostrato. \blacksquare

Teorema:

HAMPATH è un problema NP-completo.

Dimostrazione:

Abbiamo già provato che HAMPATH $\in NP$; se riusciamo a provare anche che $3SAT \leq_p HAMPATH$, allora la dimostrazione è completata.

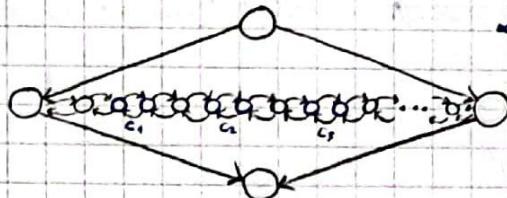
Consideriamo una formula 3CNF F con le variabili x_1, \dots, x_n e le clausole C_1, \dots, C_k .

A partire da F vogliamo costruire un grafo G con le seguenti regole:

\forall clausola C_j G ha un nodo:

C_j

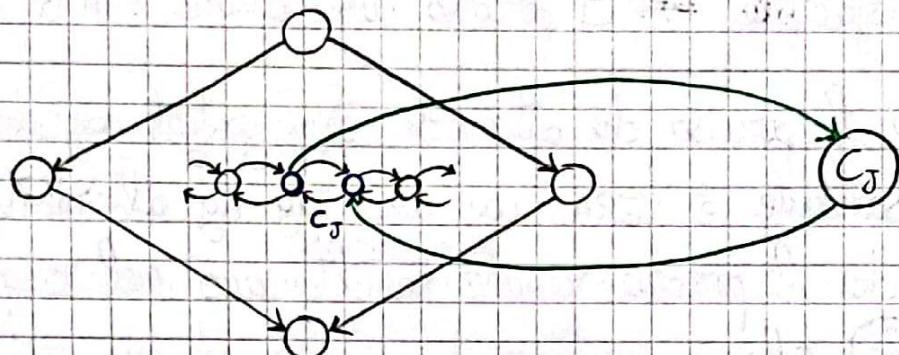
\forall variabile x_i G ha un gadget:



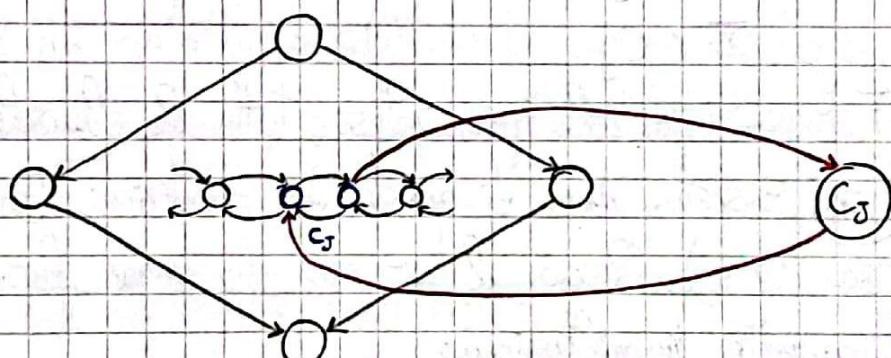
← Ciascuna coppia di nodi
che corrisponde a una
clausola

Inoltre, bisogna aggiungere degli archi che connettono i nodi C_j ai gadget. Fundamentalmente ci sono due casi:

1) $x_i \in C_j$

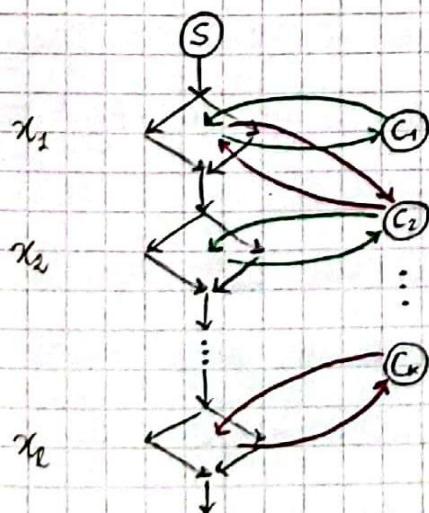


2) $\bar{x}_i \in C_j$

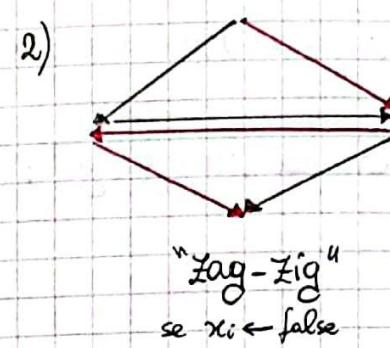
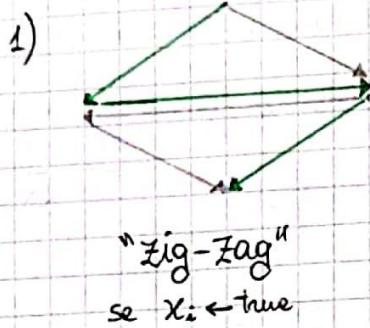


Poiché ciascuna clausola contiene tre letterali, ogni nodo C_j sarà collegato con tre gadget mediante questa costruzione.

Vediamo ora una rappresentazione stilizzata e globale del nostro grafo G :



In particolare, i gadget possono essere percorsi in due maniere diverse, che ve =
stremo nella pagina seguente.



Dimostriamo ora la seguente affermazione:

Se F è soddisfacibile $\Rightarrow \exists$ percorso hamiltoniano $\$$ in G dal nob s al nob t.

Consideriamo il percorso che attraversa ogni gadget con uno zig-zag se la variabile corrispondente è VERA, con uno Zag-Zig altrimenti.

Osserviamo che il percorso rimane ^{potenzialmente} hamiltoniano nel ricoprire anche un nob-clausula C_j solo in due casi:

$\sim x_i = \text{true}$ e $x_i \in C_j$

$\sim x_i = \text{false}$ e $\bar{x}_i \in C_j$

Ma, poiché F è soddisfacibile per ipotesi, tutte le clausole hanno almeno una variabile che rispetta una di queste due condizioni (ovvero che rende la clausola stessa vera). Perciò, il percorso che viene fuori dalla nostra costruzione è effettivamente hamiltoniano. \square

Dimostriamo anche il viceversa:

Se \exists percorso hamiltoniano dal nob s al nob t $\Rightarrow F$ è soddisfacibile.

Se il percorso attraversa i gadget usando gli zig-zag e gli zag-zig descritti in precedenza, allora è facile convincersi che l'assunzione di verità corrispondente alle variabili (e, appunto, descrivibile con gli zig-zag e gli zag-zig) è le vera l'intera formula.

Ma la nostra costruzione ti G regge soltanto se qualunque altro tipo di percorso (per esempio si potrebbe passare da un nob-clausula a un gadget diverso da quello da cui si proviene) NON è hamiltoniano. E fortunatamente le cose vanno proprio così. \square

In definitiva, poiché F è soddisfacibile $\Leftrightarrow \exists$ percorso hamiltoniano dal nodo s al nodo t , e la costruzione del grafo G è polinomiale, l'asserto $\text{3SAT} \leq_p \text{HAMPATH}$ è dimostrato. ■

Il problema UHAMPATH:

ISTANZA: Un grafo INDIRETTO G e $s, t \in V(G)$, $s \neq t$.

DOMANDA: esiste un percorso hamiltoniano in G dal nodo s al nodo t ?

Teorema:

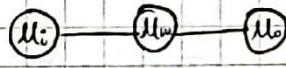
UHAMPATH è un problema NP-completo.

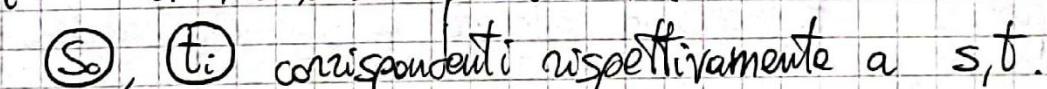
Dimostrazione:

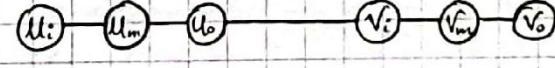
Un certificato per UHAMPATH è una sequenza di nodi e il suo verificatore non fa altro che controllare che tra questi nodi esista sempre un arco in modo tale che si possa andare da s a t attraversando tutti i nodi una volta sola. Si tratta quindi di un controllo assolutamente paragonabile a quello su un certificato per HAMPATH e, perciò, si può concludere facilmente che il problema UHAMPATH è NP.

Dimostriamo adesso che $\text{HAMPATH} \leq_p \text{UHAMPATH}$.

A partire da un'istanza (G, s, t) di HAMPATH costruiamo l'istanza corrispondente (G', s_0, t_0) di UHAMPATH con le seguenti regole:

$\forall u \in V(G), u \neq s, t \quad G'$ ha una tripletta di nodi: 

G' ha anche i nodi  corrispondenti rispettivamente a s, t .

$\forall (u, v) \in E(G) \quad (u_0, v_0) \in E(G')$ 

Dimostriamo la seguente affermazione:

$(G, s, t) \in \text{HAMPATH} \Rightarrow (G', s_0, t_0) \in \text{UHAMPATH}$

Molto banalmente, il percorso hamiltoniano viene convertito in

$s \rightarrow a \rightarrow b \rightarrow \dots \rightarrow z \rightarrow t$

$s_0 - a_0 - a_m - a_o - b_0 - b_m - b_o - \dots - z_i - z_m - z_o - t_0$

che è a sua volta un percorso hamiltoniano.

Dimostriamo ora il viceversa:

$$(G', s_0, t_0) \in \text{UHAMPATH} \Rightarrow (G, s, t) \in \text{HAMPATH}$$

Se tutte le istanze si di UHAMPATH fossero del tipo $\dots - u_i - u_m - u_o - \dots$ senza uscire dallo schema che ci siamo prefissati, allora la dimostrazione si verrebbe praticamente banale, ed effettivamente le cose vanno proprio così: per costruzione, G' è fatto in modo tale che i nodi di tipo u_o siano connessi esclusivamente ai nodi di tipo u_m (se $u \neq s, t$) e w_i , e così via.

In definitiva, poiché $(G, s, t) \in \text{HAMPATH} \Leftrightarrow (G', s_0, t_0) \in \text{UHAMPATH}$, e la costruzione del grafo G' è polinomiale, l'asserto $\text{HAMPATH} \Leftrightarrow \text{UHAMPATH}$ è dimostrato. \square

Teorema:

SUBSET SUM è un problema NP-completo.

Dimostrazione:

Abbiamo già provato che SUBSET SUM $\in \text{NP}$; se riusciamo a provare anche che $\text{3SAT} \leq_p \text{SUBSET SUM}$, allora la dimostrazione è completa.

Consideriamo una formula 3CNF F con le variabili x_1, \dots, x_e e le clausole c_1, \dots, c_k .

A partire da F vogliamo costruire la seguente tabella:

	1	2	3	...	l	c_1	c_2	c_3	...	c_k	
$x_1 \{$	1	0	0	...	0	0	Una cella (y_i, c_j)				
	z ₁	1	0	0	...	0	0	0	0	0	
$x_2 \{$	y ₂	0	1	0	...	0	0	0	0	0	
	z ₂	0	1	0	...	0	0	0	0	0	
\vdots											
$x_e \{$	y _e	0	0	0	...	0	1	vale 1 $\Leftrightarrow x_i \in c_j$			
	z _e	0	0	0	...	0	1	0	0	0	
\vdots											
$c_1 \{$	g ₁					1	0	0	...	0	
	h ₁					1	0	0	...	0	
\vdots											
$c_k \{$	g _k					0	0	0	...	0	
	h _k					0	0	0	...	0	
t	1	1	1	...	1	1	3	3	3	...	3

N.B.: in questa zona della tabella, ogni sottosezione contiene esattamente tre 1 perché siano partiti da una formula in 3CNF.

Il nostro multinsieme S sarà
 $S = \{ \{ y_1, z_1, \dots, y_e, z_e, g_1, h_1, \dots, g_k, h_k \} \}$
mentre il nostro valore target t sarà
 $t = \underbrace{1 \dots 1}_{i} \underbrace{3 \dots 3}_{j} \dots \underbrace{3 \dots 3}_{k}$

In pratica, la tabella è composta da vari numeri che vanno letti per righe e che vanno considerati in BASE 10.

Dimostriamo ora la seguente affermazione:

Se F è soddisfacibile $\Rightarrow (S, t) \in \text{SUBSET SUM}$.

Cominciamo con l'osservare che la somma delle varie righe non porta mai risparmio: ciascuna colonna non contiene mai più di cinque 1.

Prendiamo un sottoinsieme $T \subseteq S$ di righe, in cui $y_i \in T$ se $x_i = \text{true}$ e $z_i \in T$ se $x_i = \text{false}$. In questo modo è inevitabile che, nella parte sinistra della tabella, la somma delle righe di T dà luogo a tutte cifre pari a 1.

Inoltre, F è soddisfacibile per ipotesi, per cui tutte le clausole hanno almeno un letterale VERO e, di conseguenza, tutte le colonne della parte destra della sottotabella formata dalle righe di T contengono da una a tre cifre pari a 1. Perciò, in tutti i casi è possibile scegliere a piacimento delle righe della parte inferiore della tabella da aggiungere al sottoinsieme T in modo tale che, nella parte destra della tabella, la somma delle righe di T dà luogo a tutte cifre pari a 3, mentre alla parte sinistra non vengono aggiunti contributi. □

Dimostriamo anche il viceversa:

Se $(S, t) \in \text{SUBSET SUM} \Rightarrow F$ è soddisfacibile.

Per ipotesi abbiamo che $\exists T \subseteq S$ (sottoinsieme di righe) tale che la somma di queste righe dà luogo a t . Ma allora, analizzando la parte sinistra della tabella, abbiamo che o $y_i \in T$, oppure $z_i \in T \forall i$. Quindi possiamo considerare la seguente assegnazione dei valori di verità alle variabili:

$x_i \leftarrow \text{true}$ se $y_i \in T$; $x_i \leftarrow \text{false}$ se $z_i \in T$

D'altra parte, se guardiamo nella zona in basso a destra della tabella, notiamo che ogni sottocolonna contiene al massimo due 1, il che vuol dire che tutte le colonne di destra devono contenere ~~minimo~~ almeno un 1 nella par-

te superiori. Questo implica che tutte le clausole hanno almeno un letterale impostato a VERO, per cui la formula F è soddisfacibile.

In definitiva, poiché F è soddisfacibile $\Leftrightarrow (S, t) \in \text{SUBSET SUM}$, e la costruzione della tabella è polinomiale (infatti ha dimensione $O((K+t)^2)$), l'asserto $3\text{SAT} \leq \text{SUBSET SUM}$ è dimostrato. □

Il problema Independent Set (IS):

ISTANZA: un grafo indiretto G e un numero $K \in \mathbb{N}$.

DOMANDA: esiste un sottoinsieme $V' \subseteq V(G)$ di nodi con $|V'| = K$ tale che nessuna coppia di nodi ha un arco che li connette?

Effettivamente IS è un problema NP-completo poiché $\text{VP} \leq_p \text{IS}$ e, anti, vale il seguente teorema:

Teorema:

Per ogni grafo $G = (V, E)$ e per ogni sottoinsieme $V' \subseteq V(G)$ di nodi, le seguenti affermazioni sono equivalenti:

- 1) V' è un VC in G .
- 2) $V \setminus V'$ è un IS in G .
- 3) $V \setminus V'$ è una clique in G^c , dove:

$$G^c := (V, E^c), \quad E^c = \{(u, v) \mid u, v \in V \wedge (u, v) \notin E\}.$$